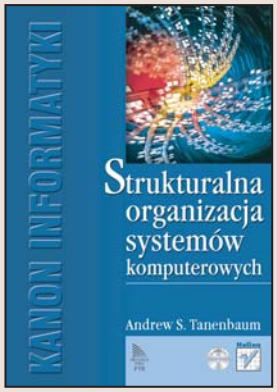
Strukturalna organizacja systemów komputerowych. Wydanie V



Autor: Andrew S. Tanenbaum
Tłumaczenie: Andrzej Grażyński (rozdz. 6–9, dod. A– C), Paweł Koronkiewicz (przedmowa, rozdz. 1–5)
ISBN: 83-246-0238-0
Tytuł oryginału: Structured Computer Organization (5th Edition)
Format: B5, stron: 864

Doskonałe omówienie zasad działania współczesnych komputerów

- Dowiedz się, jak działają procesory i magistrale
- Poznaj reguły algebry logiki
- Odkryj tajemnice współczesnych systemów operacyjnych

Dla większości użytkowników komputerów, nawet tych, dla których komputer jest narzędziem pracy, wiedza o tym urządzeniu kończy się na umiejętności instalowania i uruchamiania aplikacji. Współczesne, intuicyjne systemy operacyjne, technologie maksymalnie upraszczające pracę z komputerem, łatwe w obsłudze aplikacje — wszystko to powoduje, że znajomość zasad funkcjonowania komputerów wydaje się nam niepotrzebna. Tymczasem taka wiedza może okazać się przydatna nie tylko specjaliście, ale również zwykłemu użytkownikowi. Dzięki niej twórcy aplikacji są w stanie zoptymalizować ich działanie i zrozumieć przyczyny błędów, projektanci urządzeń peryferyjnych wybrać najlepszy sposób komunikacji swojego produktu z komputerem, a osoby zainteresowane kupnem nowego sprzętu dokonać świadomego wyboru.

W książce „Strukturalna organizacja systemów komputerowych. Wydanie V" zaprezentowano system komputerowy w ujęciu hierarchicznym, jako zespół zależnych od siebie warstw. Poznajemy go, poczynając od poziomu „logiki cyfrowej", poprzez mikroarchitekturę i poziom maszynowy, aż do poziomu systemu operacyjnego i języka asemblera. Książka przedstawia również historię rozwoju komputerów, zadania systemów operacyjnych, zasady programowania w języku maszynowym oraz architektury najpopularniejszych procesorów.

- Procesory i pamięć operacyjna
- Wykonywanie rozkazów języka maszynowego
- Operacje wejścia i wyjścia
- Arytmetyka cyfrowa
- Magistrale ISA, PCI oraz PCI Express
- Przetwarzanie danych przez procesory
- Programowanie w języku asemblera
- Działanie systemów operacyjnych
- Przetwarzanie rozproszone i obliczenia równoległe

Dowiedz się jak działa Twój komputer

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

SPIS TREŚCI

2.

Organizacja systemów komputerowych 71

3.

Poziom układów logicznych 153

5.

Konwencjonalny poziom maszynowy (ISA) 349

7.

Poziom języka asemblera 543

8.

Architektury komputerów równoległych 587

9.

Literatura uzupełniająca i bibliografia 703

A

Liczby dwójkowe 727

B

Liczby zmiennopozycyjne 741

C

Programowanie w języku asemblera 751

8

Architektury komputerów równoległych

Chociaż komputery wciąż stają się coraz szybsze, stawiane przed nimi wymagania rosną jeszcze szybciej — apetyt zwiększa się w miarę jedzenia. Astronomowie pragną poznać — poprzez symulację — całą historię wszechświata, od Wielkiego Wybuchu aż do chwil ostatecznych; w badaniach farmaceutycznych, w dążeniu do wynajdywania leków na wszelkie dolegliwości, z pewnością bardziej humanitarne jest drenowanie procesorów niż poświęcanie co rusz kolejnych legionów szczurów, myszy i innych przemiłych zwierzątek. W badaniach nad coraz bardziej ekonomicznymi silnikami do samolotów (i rakiet) modelowanie komputerowe jest szybsze i tańsze niż budowanie wymyślnych tuneli aerodynamicznych. A fizycy różnych specjalności potrafią „zarżnąć" każdy komputer…

Mimo iż szybkość procesorów z roku na rok staje się ewidentnie coraz większa, nie może ona przecież rosnąć nieograniczenie: skończona prędkość światła stanowi nieubłaganą granicę, a chłodzenie nowoczesnych procesorów staje się stopniowo dziedziną sztuki inżynierskiej. W dodatku na drodze do zmniejszania rozmiarów tranzystora (do wielkości porównywalnych z rozmiarami atomów) coraz wyraźniejszą przeszkodą okazują się zjawiska kwantowe, z zasadą nieoznaczoności Heisenberga na czele.

W dążeniu do zapewnienia coraz większej szybkości obliczeń należy zatem sięgnąć po alternatywę, jaką jest konstruowanie komputerów równoległych. Choć trudno wyobrazić sobie dziś procesor o cyklu rozkazowym 0,001 nanosekundy, całkiem realne jest zbudowanie komputera złożonego z tysiąca procesorów o cyklu 1 nanosekundy, tańszego i dającego porównywalną moc obliczeniową.

Zrównoleglenie może być wprowadzane na różnych poziomach konstrukcyjnych. Na najniższym poziomie — układu scalonego procesora (*on-chip*) — realizowane jest ono bądź to w sposób ukryty, za pomocą pracy potokowej (*pipelining*) i technik superskalarnych z wieloma podzespołami funkcjonalnymi, bądź też w sposób jawny, przez pakowanie wielu rozkazów w długie słowa maszynowe. Ponadto, w procesory można wbudowywać specjalne mechanizmy umożliwiające realizację kilku wątków

równocześnie, bądź nawet umieszczać kilka kompletnych procesorów w ramach pojedynczego układu scalonego. W chwili obecnej techniki te pozwalają na uzyskiwanie wydajności dziesięciokrotnie (w przybliżeniu) większej od tej oferowanej przez rozwiązania czysto sekwencyjne.

Kolejny poziom realizacji zrównoleglenia to różnej maści układy rozszerzające, wykonujące rozmaite funkcje specjalizowane: przetwarzanie pakietów sieciowych, szyfrowanie i deszyfracja, obsługa strumieni multimedialnych itp. Układy takie umożliwiają pięcio-, a nawet dziesięciokrotne zwiększenie wydajności[1].

Pięć czy dziesięć razy szybciej — to swoją drogą i tak interesujące; jeśli jednak rozważamy „prawdziwe" zrównoleglenie zdolne poprawić wydajność przetwarzania setki, tysiące a nawet miliony razy, konieczne jest zwielokrotnienie kompletnych procesorów i zapewnienie ich efektywnej współpracy. Pomysł ten realizowany jest w postaci komputerów wieloprocesorowych oraz systemów wielokomputerowych (klastrów). Nie trzeba dodawać, że zapewnienie efektywnego współdziałania kilku tysięcy procesorów samo z siebie jest niebagatelnym wyzwaniem rodzącym mnóstwo problemów do rozwiązania.

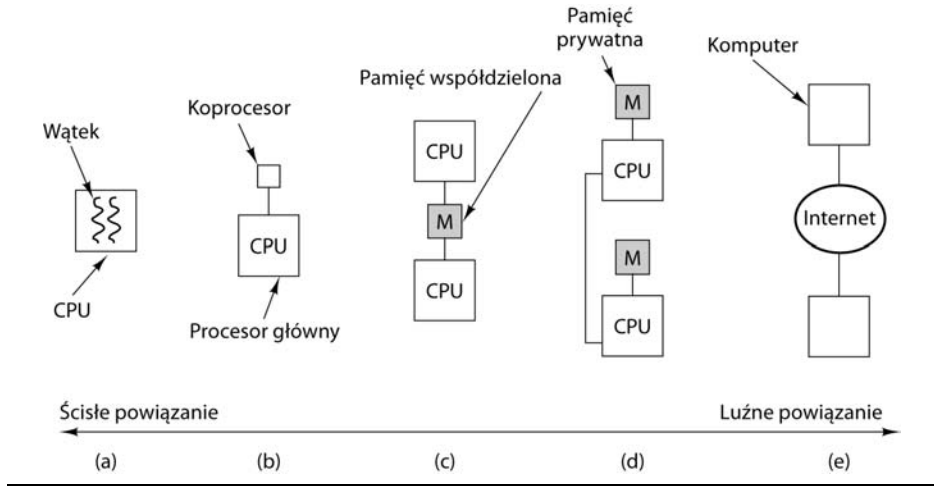
Upowszechnienie się internetu stwarza kolejną okazję w postaci możliwości tworzenia olbrzymich sieci połączonych ze sobą komputerów. Pełne wykorzystanie tej możliwości to jednak melodia bliższej i dalszej przyszłości.

W opisanych wariantach zrównoleglenia poszczególne elementy systemu w różny sposób muszą być ze sobą powiązane. Jeśli elementy te znajdują się blisko siebie, są od siebie wysoce uzależnione obliczeniowo, a komunikacja między nimi odbywa się z dużą prędkością i małymi opóźnieniami, mamy do czynienia z **powiązaniem ścisłym** (*tight couple*); dla odmiany, duże oddalenie geograficzne, wąskie pasmo transmisji, stosunkowo duże opóźnienie i ograniczona zależność obliczeniowa to znamiona **powiązania luźnego** (*loose couple*). W niniejszym rozdziale zajmiemy się analizą rozmaitych rozwiązań zarówno realizujących w pełni opisane cechy, jak i reprezentujących koncepcje pośrednie między tymi dwiema skrajnościami, jak pokazano to schematycznie na rysunku 8.1: rozpoczniemy od systemów najbardziej ściśle powiązanych, realizujących zrównoleglenie ma poziomie układu scalonego, przechodząc stopniowo do systemów powiązanych coraz luźniej, a na obliczeniach siatkowych (*grid computing*) kończąc.

8.1. ZRÓWNOLEGLENIE NA POZIOMIE UKŁADU SCALONEGO

Oczywistym sposobem przyspieszania wszelkich działań jest wykonywanie kilku rzeczy równocześnie. W niniejszym podrozdziale pokażemy, jak pomysł ten realizowany jest na poziomie konstrukcji układu scalonego, poprzez (między innymi) zrównoleglenie wykonywania rozkazów, wielowątkowość i integrowanie kilku (wielu) procesorów w pojedynczym układzie scalonym. Techniki te, jakkolwiek różnią się między sobą, wszystkie stanowią wariant wykonywania kilku rzeczy w tym samym czasie.

[1] W porównaniu z implementacjami czysto programowymi — *przyp. tłum.*



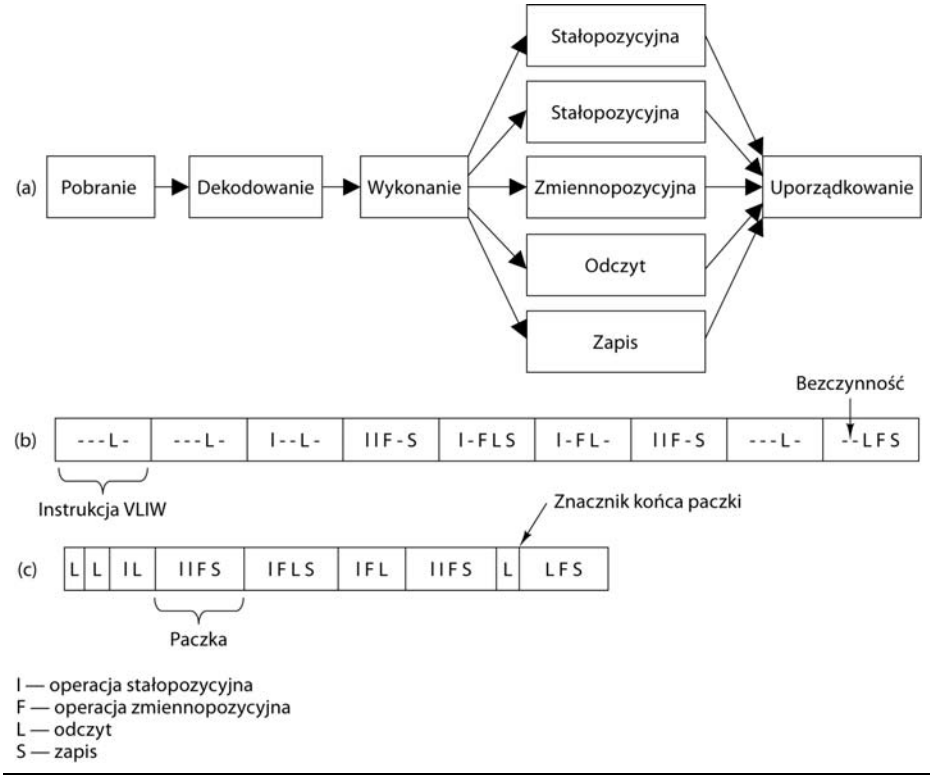
RYSUNEK 8.1. Różne stopnie powiązań komponentów systemów równoległych: (a) zrównoleglenie na poziomie układu scalonego; (b) koprocesor; (c) komputer wieloprocesorowy; (d) wielokomputer; (e) siatka obliczeniowa

8.1.1. Zrównoleglone wykonywanie rozkazów

Jednym ze sposobów osiągnięcia równoległego przetwarzania na najniższym poziomie jest wykonywanie kilku rozkazów maszynowych w jednym cyklu zegara. Zadanie to zrealizowane zostało w praktyce na dwa sposoby: w postaci architektury superskalarnej oraz przez użycie długiego słowa rozkazowego. Wspominaliśmy już o tym wcześniej w niniejszej książce, nie zaszkodzi jednak dla przypomnienia powrócić w tym miejscu do faktów najważniejszych.

Architekturę procesora superskalarnego widzieliśmy już na rysunku 2.5. W większości przypadków, na pewnym stopniu potoku rozkaz jest gotowy do wykonania. Procesory superskalarne posiadają możliwość jednoczesnego realizowania kilku rozkazów, każdego na innym etapie obróbki; liczba rozkazów znajdujących się jednocześnie w „obróbce" zależy zarówno od cech projektowych konkretnego procesora, jak i bieżących uwarunkowań. Maksymalna liczba jednocześnie realizowanych rozkazów zdeterminowana jest przez sprzęt i zazwyczaj waha się w granicach od dwóch do sześciu; oczywiście, wykonywanie rozkazu nie może być kontynuowane, jeśli musi on skorzystać z niedostępnego aktualnie bloku funkcjonalnego bądź też wykorzystać wynik poprzednich obliczeń, który to wynik nie jest jeszcze gotowy.

Inny wariant równoległego wykonywania rozkazów zrealizowany został pod postacią architektury opartej na długich słowach rozkazowych, oznaczanej skrótem **VLIW** (*Very Long Instruction Word* — bardzo długie słowo rozkazowe). W swej oryginalnej formie komputery VLIW faktycznie posiadają długie słowa rozkazowe zawierające rozkazy odwołujące się do wielu różnych bloków funkcjonalnych. Spójrzmy dla przykładu na potok, przedstawiony na rysunku 8.2 (a), gdzie komputer posiada pięć bloków funkcjonalnych i wykonywać może jednocześnie dwie operacje stałopozycyjne, jedną zmiennopozycyjną, jeden odczyt i jeden zapis. Rozkaz tego komputera powinien więc zawierać pięć kodów operacji i pięć operandów — po jednej parze „kod-operand"



RYSUNEK 8.2. Architektura VLIW: (a) potok procesora; (b) sekwencja rozkazów VLIW; (c) strumień rozkazów ze znacznikami paczek

dla każdej jednostki funkcjonalnej. Jeśli przeznaczymy po 6 bitów na każdy kod operacji, po 5 na numer rejestru i po 32 bity na każdy adres pamięci, potrzebujemy ni mniej ni więcej tylko 134 bitów, by to wszystko pomieścić — słowo rozkazowe jest więc istotnie bardzo długie.

Projekt ten okazuje się jednak zbyt prymitywny, często się bowiem zdarza, że rozkaz nie jest w stanie skorzystać ze wszystkich potrzebnych bloków funkcjonalnych, co w praktyce prowadzi do wielu stanów bezczynności, pełniących rolę „wypełniaczy", jak zilustrowano to na rysunku 8.2 (b). W konsekwencji nowoczesne procesory VLIW dysponują funkcją grupowania rozkazów w paczki, przy czym koniec paczki zaznaczany jest explicite np. za pomocą specjalnego bitu, jak na rysunku 8.2 (c); procesor pobiera do wykonania całą paczkę równocześnie.

Sama funkcja grupowania „kompatybilnych" rozkazów w paczki zrzucona została na barki kompilatorów, co nie tylko przyczyniło się do uproszczenia sprzętu i zwiększenia jego wydajności, lecz także otworzyło drogę do znacznie lepszego grupowania rozkazów w paczki: wszak kompilator optymalizujący może wykonywać swą pracę tak długo, jak będzie to konieczne, w przeciwieństwie do sprzętu, którego operacje są silnie uwarunkowane czasowo. Tak się niestety składa, że radykalne zmiany w architekturze procesorów z trudem torują sobie drogę na rynku, czego jednym z przejawów jest stosunkowo chłodne przyjęcie nowego procesora Itanium.

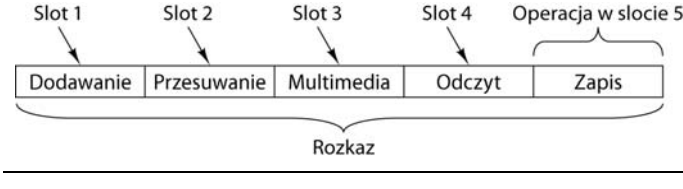
Należy w tym miejscu zaznaczyć, że równoległe wykonywanie rozkazów nie jest bynajmniej jedynym wariantem zrównoleglenia niskopoziomowego; inny wariantem jest zrównoleglenie pamięciowe, w ramach którego wykonywanych jest jednocześnie kilka operacji pamięciowych (Chou i in., 2004).

Przykład architektury VLIW — procesor TriMedia

W rozdziale piątym poznaliśmy już jeden procesor o architekturze VLIW — Itanium-2; obecnie zaprezentujemy zupełnie inny przykład tej architektury — procesor **TriMedia**, zaprojektowany i wyprodukowany przez holenderską firmę Philips, słynącą m.in. z wynalezienia płyty kompaktowej i dysku CD-ROM. Procesor TriMedia przeznaczony jest do pracy głównie w warunkach intensywnego przetwarzania masowych sygnałów audio i wideo, a więc przede wszystkim w odtwarzaczach CD, DVD i MP3, nagrywarkach CD-R(W) i DVD, kamerach cyfrowych, kamkoderach i systemach telewizji interaktywnej. Nic więc dziwnego, że konstrukcyjnie różni się w sposób zasadniczy od procesora Itanium-2, zaprojektowanego przede wszystkim jako procesor uniwersalny, na potrzeby wydajnych serwerów.

Każdy rozkaz procesora TriMedia zawiera do pięciu **operacji**; w szczególnie sprzyjających warunkach możliwe jest przekazanie do wykonania jednego rozkazu i równoległa realizacja czterech innych w jednym cyklu zegara. Częstotliwość zegara taktującego wynosi 266 lub 300 MHz, lecz ze względu na opisane zwielokrotnienie rzeczywista wydajność procesora może być nawet pięciokrotnie większa. W dalszym opisie posłużymy się modelem TM3260 procesora TriMedia; inne modele różnią się od niego stosunkowo niewiele.

Format typowego rozkazu procesora pokazano na rysunku 8.3. Rozkazy są zróżnicowane, od 8-, 16- i 32-bitowych rozkazów całkowitoliczbowych, poprzez rozkazy zmiennopozycyjne zgodne ze standardem IEEE 754, aż do równoległych rozkazów multimedialnych. W konsekwencji przy pięciu równolegle wykonywanych rozkazach i zrównolegleniu operacji multimedialnych procesor TriMedia jest wystarczająco szybki do programowego dekodowania cyfrowego strumienia wideo kamkodera z pełną częstotliwością i przy pełnym rozmiarze ramki.



RYSUNEK 8.3. Typowy rozkaz procesora TriMedia, z pięcioma możliwymi operacjami

Procesor TriMedia stosuje bajtowe adresowanie pamięci, z rejestrami wejścia-wyjścia mapowanymi w adresy pamięciowe. Adresy półsłów (16-bitowych) i słów (32-bitowych) muszą być wyrównane na naturalnych granicach — odpowiednio wielokrotności 2 i wielokrotności 4 bajtów. Jeśli chodzi o kolejność („starszeństwo") bajtów w słowie wielobajtowym, to procesor pracować może zarówno w konwencji *little-endian*, jak i *big-endian,* zależnie od ustawienia stosownego bitu w słowie PSW, modyfikowalnego jedynie przez system operacyjny. Rozróżnienie między *little-endian* a *big-endian* istotne jest tylko dla rozkazów pobierających dane z pamięci do

rejestrów i zapisujących zawartość rejestrów w pamięci. Procesor wspomagany jest 8-kanałową pamięcią podręczną o organizacji zespołowej (*8-way associative-set cache*) osobno dla kodu (64 KB) i danych (16 KB); w obydwu przypadkach długość wiersza pamięci wynosi 64 bajty.

Procesor posiada 128 rejestrów 32-bitowych ogólnego przeinaczenia. Rejestr `R0` ma ustaloną zawartość 0, której zmienić nie można, podobnie rejestr `R1` ma ustaloną zawartość 1. Pozostałe rejestry (126) są funkcjonalnie równoważne i mogą być wykorzystywane do rozmaitych celów. Dodatkowo w procesorze istnieją cztery specjalne rejestry 32-bitowe: licznik rozkazów, słowo stanu programu (PSW — *Program Status Word*) i dwa rejestry związane z przerwaniami. Ponadto w specjalnym 64-bitowym rejestrze zliczane są zegarowe cykle pracy procesora od jego ostatniego zresetowania; przy częstotliwości zegara 300 MHz potrzeba prawie 1950 lat, by rejestr ten przepełnić.

Procesor TriMedia TM3260 posiada 11 różnych jednostek funkcjonalnych wykonujących operacje arytmetyczne i logiczne, organizujących przepływ sterowania i nadzorujących cache'owanie zawartości (tej ostatniej funkcji nie będziemy tu omawiać). Jednostki te scharakteryzowane zostały pokrótce w tabeli 8.1. Pierwsze dwie kolumny zawierają nazwy[2] i krótkie opisy jednostek, w trzeciej wymieniono liczbę sprzętowych egzemplarzy każdej jednostki, w czwartej podano natomiast wartość opóźnienia (w cyklach zegara), jakiego dana jednostka wymaga do wykonania swej funkcji: należy bowiem pamiętać, że wszystkie wymienione jednostki (oprócz `FP sqrt/div`) są jednostkami potokowymi i podana wartość jest odstępem czasowym między wprowadzeniem informacji do potoku a uzyskaniem wyniku. Niezależnie od opóźnień poszczególnych jednostek funkcjonalnych, w każdym cyklu zegara może być inicjowane wykonywanie nowego rozkazu, tak więc na przykład każdy z trzech kolejnych rozkazów może zawierać dwie operacje odczytu, w rezultacie czego w danej chwili może być równolegle wykonywanych *sześć* operacji odczytu, każda na innym etapie realizacji.

Pięć ostatnich kolumn tabeli zawiera informację o tym, które ze slotów rozkazu (patrz rysunek 8.3) mogą mieć związek z poszczególnymi jednostkami funkcjonalnymi — przykładowo operacja porównywania zmiennopozycyjnego może być kodowana wyłącznie w slocie nr 3.

Jednostka `Constant` zajmuje się operacjami natychmiastowymi, między innymi pobieraniem stałych kodowanych w rozkazach jako argumenty natychmiastowe (*immediate*). Jednostka `Integer ALU` wykonuje dodawanie i odejmowanie całkowitoliczbowe, podstawowe operacje boolowskie oraz operacje pakowania i rozpakowywania operandów. Jednostka `Shifter` odpowiedzialna jest za przesuwanie zawartości rejestrów we wskazanym kierunku, o wskazaną liczbę bitów.

Operacje pobierania danych z pamięci do rejestrów i zapisywania zawartości rejestrów w pamięci spoczywają w gestii jednostki `Load/Store`. Pod względem konstrukcyjnym TriMedia jest rozbudowanym procesorem RISC, w związku z czym wszelkie operacje wykonywane są na rejestrach, zaś jedynymi operacjami pamięciowymi są operacje odczytu i zapisu — operacje arytmetyczne i logiczne nie odwołują się do pamięci. Dane wymieniane między rejestrami a pamięcią mogą być przesyłane w porcjach 8-, 16- lub 32-bitowych.

W jednostce `Int/FP Mul` wykonywane jest mnożenie, zarówno całkowitoliczbowe jak i zmiennopozycyjne. Trzy kolejne jednostki odpowiedzialne są za dodawanie i odejmowanie zmiennopozycyjne, porównania zmiennopozycyjne oraz dzielenie zmiennopozycyjne i obliczanie zmiennopozycyjnego pierwiastka kwadratowego.

[2] Zachowano nazwy oryginalne — *przyp. tłum.*

TABELA 8.1.
Jednostki funkcjonalne procesora TriMedia TM3260

Jednostka	Wykonywane funkcje	Liczba egzemplarzy	Opóźnienie	Sloty				
				1	2	3	4	5
Constant	Operacje natychmiastowe.	5	1	×	×	×	×	×
Integer ALU	32-bitowa arytmetyka całkowitoliczbowa, operacje boolowskie.	5	1	×	×	×	×	×
Shifter	Wielobitowe przesunięcia.	2	1	×	×	×	×	×
Load/Store	Odczyt i zapis z (do) pamięci.	2	3				×	×
Int/FP Mul	Mnożenie całkowitoliczbowe (32-bitowe) i zmiennopozycyjne.	2	3		×	×		
FP ALU	Arytmetyka zmiennopozycyjna.	2	3	×			×	
FP Compare	Porównania zmiennopozycyjne.	1	1			×		
FP sqrt/div	Dzielenie zmiennopozycyjne i obliczanie zmiennopozycyjnego pierwiastka kwadratowego.	1	17		×			
Branch	Sterowanie przepływem (skoki i rozgałęzienia).	3	3		×	×	×	
DSP ALU	Arytmetyka multimedialna, dwukrotna 16-bitowa i czterokrotna 8-bitowa.	2	3	×		×		×
DSP MUL	Mnożenie multimedialne, dwukrotne 16-bitowe i czterokrotne 8-bitowe.	2	3		×	×		

Operacje rozgałęzień sterowania wykonywane są w ramach jednostki Branch. Z każdym rozgałęzieniem związane jest 3-cyklowe opóźnienie, w ramach którego inicjowane są trzy następne rozkazy (mogące zawierać łącznie do 15 operacji) bez względu na kierunek rozgałęzienia, a nawet przy skoku bezwarunkowym.

Przedrostek „DSP” w nazwach dwóch ostatnich jednostek to skrót od *Digital Signal Processor* — „procesor sygnałów cyfrowych” lub krótko „procesor sygnałowy”; to specjalizowany procesor przeznaczony do cyfrowej obróbki sygnałów. Pierwszy uważany za przedstawiciela tej klasy procesor 2920 wyprodukowany został w roku 1979; procesory sygnałowe to prekursorzy obecnych rozwiązań multimedialnych i stąd wspomniany przedrostek w nazwach jednostek funkcjonalnych o wybitnie multimedialnym charakterze. Jedną z najistotniejszych cech przetwarzania multimedialnego jest tzw. **arytmetyka nasyceniowa** (*saturated arithmetic*), różniąca się od klasycznej arytmetyki dwójkowej wymuszonym korygowaniem wyników wykraczających poza

dozwolone wartości, czyli „wtłaczaniem”[3] wyniku operacji w zakres dozwolony dla liczb o danym rozmiarze. Jeżeli mianowicie spróbujemy dodać do siebie 8-bitowe wartości 130 i 140, to w klasycznej arytmetyce dwójkowej otrzymamy bądź to wyjątek nadmiaru stałopozycyjnego, bądź też bezsensowny wynik 14, stanowiący rezultat obcięcia prawdziwego wyniku 270 do ośmiu najmłodszych bitów. W arytmetyce nasyceniowej wartość 270 zostanie natomiast skorygowana („strimowana”) do największej dopuszczalnej wartości 255.

Ponieważ — jak widać w tabeli 8.1 — przyporządkowywanie operacji do slotów nie jest całkowicie dowolne (pewne operacje nie mogą pojawiać się w niektórych slotach), nie we wszystkich rozkazach kodowana jest maksymalna liczba (5) operacji. Zakodowana w slocie operacja może zajmować 26, 34 lub 42 bity, natomiast niewykorzystywane sloty kompaktowane są tak, by minimalizować zajmowane miejsce. Zależnie od liczby faktycznie kodowanych operacji i przy uwzględnieniu stałych elementów rozkazu jego długość może zmieniać się od 2 do 28 bajtów.

Procesor TriMedia nie wykonuje kontroli kompatybilności grupowanych ze sobą operacji; przy braku takiej kompatybilności rozkaz wykonywany jest jak gdyby nigdy nic, a wynik tego wykonania jest na ogół bezwartościowy. Rezygnacja z kontroli kompatybilności zgrupowanych rozkazów pozwoliła na znaczne uproszczenie konstrukcji procesora i zwiększenie szybkości jego pracy. Dla odmiany, w procesorach Pentium **jest** przeprowadzana kontrola kompatybilności operacji superskalarnych, za cenę jednakże znacznego zwiększenia liczby tranzystorów, znacznej komplikacji układu i wydłużenia czasu wykonywania rozkazu; konstruktorzy procesora TriMedia uniknęli tych kosztów, zrzucając odpowiedzialność za kompatybilne grupowanie operacji na kompilatory, niepodlegające wyśrubowanym ograniczeniom czasowym i zdolne do starannego optymalizowania przekładu.

Podobnie jak w procesorze Itanium-2, wykonywanie operacji procesora TriMedia podlega specjalnemu uwarunkowaniu, zwanemu predykcją. W każdej operacji (z dwoma małymi wyjątkami) określony jest mianowicie rejestr, zwany rejestrem predykcyjnym, którego zawartość decyduje o tym, czy operacja ta zostanie wykonana, czy też nie: operacja wykonywana jest tylko wtedy, gdy najmniej znaczący bit tego rejestru ma wartość 1. Na przykład, operacja określona jako:

```
IF R2 IADD R4, R5 -> R8
```

powodująca zapisanie w rejestrze `R8` sumy zawartości rejestrów `R4` i `R5` zostanie wykonana tylko wtedy, gdy najmłodszy bit rejestru `R2` będzie miał wartość 1. Każda z operacji kodowanych w rozkazie podlega niezależnej predykcji. Ze względu na specyficzną zawartość rejestrów `R0` i `R1`, wyspecyfikowanie rejestru `R1` w roli rejestru predykcyjnego oznacza bezwarunkowe wykonanie operacji, natomiast wyspecyfikowanie w tym charakterze rejestru `R0` — jej bezwarunkowe pominięcie.

[3] W literaturze anglojęzycznej wtłaczanie to nazywane jest „obcinaniem” (*clipping*), co jest trochę mylące, bowiem w odniesieniu do pozycyjnego (bitowego) zapisu liczby może kojarzyć się z obcinaniem wartości do ustalonej liczby najmniej znaczących bitów (i ignorowaniem bitów najbardziej znaczących). W odniesieniu jednak do **analogowego** rozumienia wartości (na przykład jako długości odcinka) korygowanie wartości do dopuszczalnych granic jest niczym innym jak właśnie ich obcinaniem; mimo to w dalszym ciągu rozdziału, dla uniknięcia opisanego nieporozumienia, będziemy konsekwentnie używać terminu „nasycanie” zamiast „obcinanie” — *przyp. tłum.*

Operacje multimedialne procesora TriMedia podzielić można na 15 grup, wymienionych w tabeli 8.2. Wiele z wymienionych operacji wiąże się z nasycaniem wyniku, czyli korygowaniem go do jawnie lub niejawnie określonych granic. Owe „granice" to nic innego jak graniczne wartości słów 8-, 16- i 32-bitowych w konwencji „ze znakiem" (*signed*) albo „bez znaku" (*unsigned*). Jeżeli na przykład wynik odnoszony jest do bezznakowego bajtu, granicami tymi są wartości 0 i 255; wyniki 140 i 340 będą więc mieć po skorygowaniu wartości (odpowiednio) 140 i 255; gdyby wyniki te odnieść do bajtu ze znakiem, po skorygowaniu miałyby jednak (obydwa) wartość 127.

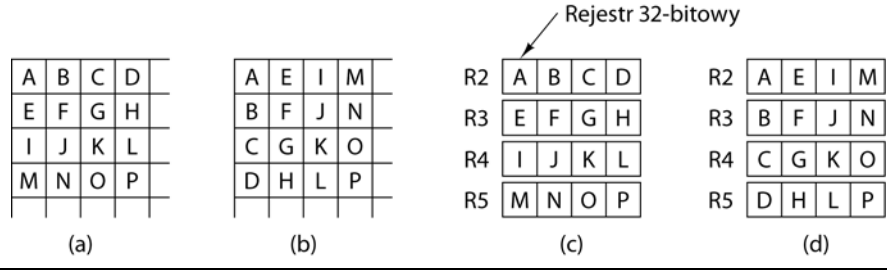
TABELA 8.2.
Podstawowe grupy operacji multimedialnych procesora TriMedia

Grupa	Operacje
Nasycanie	Nasycanie czterech wartości bajtowych lub dwóch wartości dwubajtowych.
Wartość bezwzględna DSP	Nasycona wartość bezwzględna, ze znakiem.
Dodawanie DSP	Nasycone dodawanie ze znakiem.
Odejmowanie DSP	Nasycone odejmowanie ze znakiem.
Mnożenie DSP	Nasycone mnożenie ze znakiem.
Minimum i maksimum	Wartości minimalne i maksymalne z każdej z czterech par wartości bajtowych.
Porównanie	Porównywanie odpowiadających sobie bajtów z dwóch rejestrów.
Przesunięcie	Przesuwanie pary operandów 16-bitowych.
Suma produktów	Suma, ze znakiem, produktów 8- lub 16-bitowych.
Łączenie, pakowanie, wymiana	Manipulowanie bajtami i półsłowami dwubajtowymi.
Uśrednianie czwórek bajtowych	Obliczanie średniej z czwórki bajtowej.
Uśrednianie bajtowe	Bajtowe uśrednianie czterech elementów.
Mnożenie bajtów	Bezznakowe mnożenie dwóch bajtów.
Szacowanie ruchu	Bezznakowa suma wartości bezwzględnych różnic elementów bajtowych.
Różne	Pozostałe operacje arytmetyczne.

Pierwsza z wymienionych grup obejmuje korygowanie (nasycanie) wartości, cztery następne grupy obejmują operacje arytmetyczne połączone z nasycaniem. Operacje z grupy „Minimum i maksimum" traktują swe 32-bitowe operandy jako czterobajtowe ciągi i dla każdej pary odpowiadających sobie bajtów wyliczają wartość większą i mniejszą z tej pary. Podobnie operacje z grupy „Porównanie" porównują ze sobą odpowiadające sobie bajty w poszczególnych rejestrach.

Operacje multimedialne rzadko wykonywane są na operandach 32-bitowych, ponieważ większość obrazów składa się z pikseli określonych za pomocą 24-bitowych wartości RGB reprezentujących bądź to 8-bitowe składowe kolorów czerwonego (*red*), zielonego (*green*) i niebieskiego (*blue*), bądź też odpowiadające im składowe YUV (zajmiemy się nimi w dalszej części rozdziału). W konsekwencji znakomita większość obliczeń multimedialnych wykonywana jest na prostokątnych macierzach 8-bitowych wartości bez znaku.

Procesor TriMedia wyposażony jest w obfity repertuar efektywnych operacji na takich właśnie operandach. W charakterze prostego przykładu rozpatrzmy lewy górny narożnik takiej macierzy, której bajtowe elementy zgrupowane są w czterobajtowe słowa zgodnie z konwencją *big-endian*, jak pokazano to na rysunku 8.4 (a). Bajtowe



RYSUNEK 8.4. Przetwarzanie bloku elementów 8-bitowych: (a) oryginalna macierz elementów; (b) macierz elementów po transpozycji; (c) zawartość oryginalnej macierzy po załadowaniu do czterech rejestrów; (d) zawartość rejestrów odpowiadająca macierzy transponowanej

elementy kwadratowego bloku o rozmiarze 4 × 4 oznaczone zostały literami od `A` do `P`. Przypuśćmy, że blok ten chcemy poddać transpozycji, czyli przekształcić go do postaci widocznej w części (b) rysunku, w której wiersze oryginału stają się kolumnami.

Najbardziej oczywistym sposobem wykonania transpozycji jest zamiana elementów w każdej z 6 par: (`E`, `B`), (`I`, `C`), (`M`, `D`), (`J`, `G`), (`N`, `H`) i (`O`, `L`). Wymaga to 12 osobnych operacji odczytu z pamięci i takiej samej liczby operacji zapisu, czyli 24 kosztowych operacji pamięciowych.

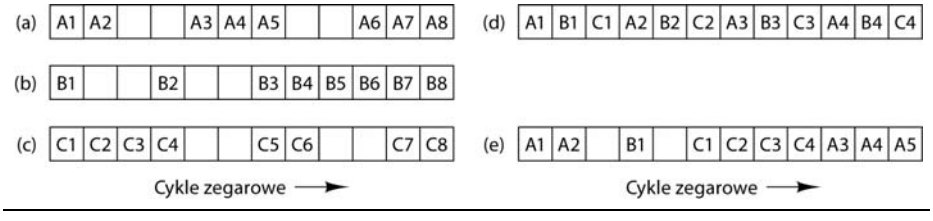
Nieco bardziej efektywne podejście polega na załadowaniu poszczególnych wierszy macierzy do czterech rejestrów 32-bitowych, kolejno od `R2` do `R5`, jak pokazano to w części (c) rysunku. Za pomocą szeregu operacji bitowych i operacji przesuwania można rejestry te doprowadzić do stanu, w którym ich zawartość odpowiada poszczególnym kolumnom, jak widać to w części (d) rysunku. Ostatecznie zwartość rejestrów należy kolejno zapisać do pamięci. Mimo iż liczba operacji pamięciowych została w ten sposób zredukowana z 24 do 8, wadą tego rozwiązania jest duży koszt operacji związanej z przekształcaniem zawartości rejestrów.

TriMedia oferuje znacznie efektywniejsze rozwiązanie. Oczywiście nie unikniemy czterech operacji odczytu i czterech operacji zapisu. Ponieważ rozkazy odczytu i zapisu mogą być kodowane tylko w slotach 4 i 5 (patrz tabela 8.1), dla czterech operacji odczytu potrzebujemy dwóch rozkazów: w pierwszym z nich zakodowany będzie odczyt do rejestrów `R2` i `R3`, w drugim — odczyt do rejestrów `R4` i `R5`. Sloty 1 – 3 obydwu rozkazów można przeznaczyć na inne operacje. Opisaną wcześniej transformację zawartości rejestrów da się przeprowadzić za pomocą ośmiu operacji multimedialnych, które można upakować w dwa następne rozkazy. Do zakodowania czterech operacji zapisu potrzebujemy dwóch kolejnych rozkazów — i tak oto cała transpozycja macierzy zakodowana zostaje w sześciu rozkazach, w których wykorzystano tylko 4+8+4=16 slotów. 14 slotów pozostaje niewykorzystane i można je przeznaczyć do zakodowania innych operacji. Statystycznie cała transpozycja zajmuje więc tylko odpowiednik $\frac{16}{30} \approx 5{,}3$ rozkazu. Przykład ten jest doskonałą ilustracją efektywności operacji multimedialnych procesora TriMedia.

8.1.2. Wielowątkowość na poziomie układu scalonego

Konstruktorzy nowoczesnych procesorów potokowych nieodmiennie zmagają się z jednym podstawowym problemem: jeżeli odwołanie do pamięci związane jest z chybieniem cache na poziomie 1 lub 2 — czyli gdy w pamięci podręcznej nie jest reprezentowany żądany adres — konieczne jest przeprowadzenie fizycznego odczytu danych z pamięci (i przy okazji uaktualnienie zawartości cache); jest operacja stosunkowo czasochłonna, a w czasie jej wykonywania potok zostaje zatrzymany. Jednym ze sposobów pokonania tej trudności jest realizowanie przez procesor kilku potoków równocześnie: jeśli potok 1 zmuszony będzie do oczekiwania, być może realizować będzie można potok 2, dzięki czemu dostępny sprzęt wykorzystany będzie możliwie jak najefektywniej. Rozwiązanie takie nazywamy **wielowątkowością na poziomie układu scalonego** (*on-chip multithreading*).

Ta koncepcyjnie prosta technika realizowana jest w kilku wariantach. Jednym z nich jest **wielowątkowość drobnoziarnista** (*fine-grained multithreading*), której istotę zilustrowano na rysunku 8.5. W częściach (a) – (c) rysunku widzimy trzy wątki *A*, *B* i *C* na przestrzeni 12 cykli maszynowych (procesor zdolny jest do inicjowania jednego rozkazu w jednym cyklu maszynowym). W pierwszym cyklu inicjowany jest rozkaz *A*1 i w tym samym cyklu zostaje zakończony, zatem w drugim cyklu inicjowany jest rozkaz *A*2; niestety, jego wykonanie powoduje chybienie cache na poziomie 1, w związku z czym konieczne jest odczekanie dwóch cykli w celu pobrania żądanej zawartości z cache poziomu 2. Wątek *A* będzie więc kontynuowany dopiero od cyklu piątego. Z podobnej przyczyny wstrzymywane są wątki *B* i *C*; generalnie, wstrzymanie jednego rozkazu powoduje wstrzymanie całego wątku, w kontekście którego jest on wykonywany.



RYSUNEK 8.5. Wielowątkowość na poziomie procesora: (a) – (c) trzy wątki, puste okienka oznaczają oczekiwanie podczas cykli maszynowych; (d) potok generowany przez wielowątkowość drobnoziarnistą; (e) potok generowany przez wielowątkowość gruboziarnistą

W modelu wielowątkowości drobnoziarnistej wstrzymywanie wątku jest więc maskowane poprzez realizację kilku (trzech) wątków według algorytmu karuzelowego (*round-robin*): w kolejnych cyklach realizowane są naprzemiennie rozkazy z kolejnych wątków, jak pokazano to w części (d) rysunku 8.5. Ponieważ w danym wątku kolejne rozkazy inicjowane są co trzy cykle, a maksymalny czas wstrzymania wątku wynosi dwa cykle, więc istnieje gwarancja, że przed pobraniem kolejnego rozkazu zakończy się wykonanie poprzedniego w tym samym wątku, a więc np. rozkaz *A*2 może zostać zainicjowany nawet wtedy, gdy korzysta z wyników dostarczonych przez rozkaz *A*1. W efekcie, niezależnie od ewentualnego wstrzymywania wątków, wykonywanie rozkazów odbywa się nieprzerwanie. Gdyby wstrzymanie wątku mogło rozciągać się na trzy cykle, dla zapewnienia tej ciągłości potrzebowalibyśmy czwartego wątku.

Ponieważ poszczególne wątki realizowane są niezależnie od siebie, każdy z nich wymaga odrębnego, własnego zbioru rejestrów. Wraz z zainicjowaniem kolejnego rozkazu musi zostać przekazany wskaźnik do właściwego mu zbioru rejestrów, tak by wszelkie odwołania do rejestrów mogły być kierowane do właściwego ich zbioru. Tym samym liczba rejestrów procesora determinuje już na etapie jego konstrukcji maksymalną liczbę wątków, jaką może on realizować.

Operacje pamięciowe nie są jedyną przyczyną wstrzymywania wątków; często zainicjowanie kolejnego rozkazu musi zostać wstrzymane w oczekiwaniu na wyniki dostarczane przez poprzedni rozkaz. Podobnie kłopotliwe są rozkazy następujące bezpośrednio po rozkazach skoku warunkowego: wykonanie takiego rozkazu nie może zostać rozpoczęte wcześniej, niż znany będzie kierunek wspomnianego skoku. Generalnie, jeśli głębokość potoku wynosi k, a procesor realizuje co najmniej k wątków równocześnie według algorytmu karuzelowego, w danej chwili w potoku obecny jest co najwyżej jeden rozkaz z każdego wątku i konflikty między rozkazami z tego samego wątku są wykluczone. Wykonywanie rozkazów odbywa się przy pełnym wykorzystaniu mocy procesora, bez wstrzymywania jego pracy.

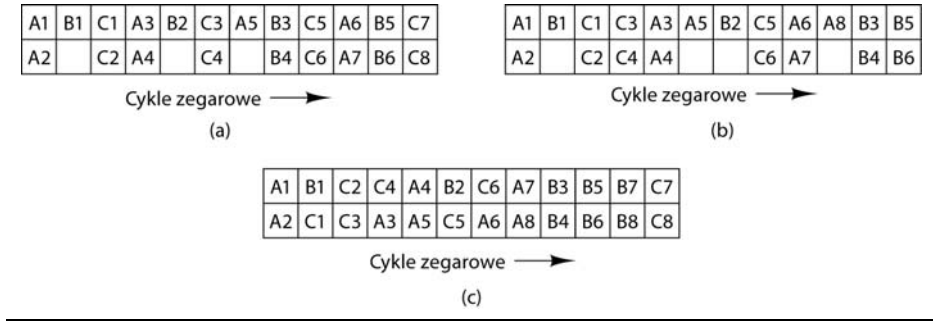
Zapewnienie dostatecznej liczby wątków — w stopniu rekompensującym głębokość potoku — jest posunięciem kosztownym, dlatego też niektórzy projektanci preferują inny wariant wielowątkowości, zwany **wielowątkowością gruboziarnistą** (*coarse-grained multithreading*). Jego istotę zilustrowano w części (e) rysunku 8.5. Rozkazy wątku *A* wykonywane zostają aż do jego wstrzymania. Po jednym cyklu oczekiwania następuje przełączenie na wątek *B*, który wstrzymany zostaje na pierwszym rozkazie, w związku z czym (znowu po odczekaniu jednego cyklu) następuje przełączenie na wątek *C*. Poszczególne wątki nadal więc realizowane są w sposób cykliczny, jednakże przełączanie między nimi odbywa się nie w sposób drobnoziarnisty, czyli co kolejny cykl zegara, lecz dopiero w momencie wstrzymywania wątków; czas pomiędzy kolejnymi przełączeniami może więc być bardzo długi, stąd dygresja do gruboziarnistości. Ze względu na tracenie jednego cyklu w momencie wstrzymywania wątku, ten wariant wielowątkowości jest potencjalnie mniej efektywny (w porównaniu z drobnoziarnistym odpowiednikiem) z punktu wykorzystania mocy procesora, lecz ma też cenną zaletę, jaką jest utrzymanie zajętości procesora przy niewielkiej liczbie wątków.

Opisane marnotrawienie jednego cyklu zegara w momencie wstrzymywania wątku można wyeliminować, decydując się na przełączenie wątku nie w momencie wystąpienia konieczności jego wstrzymania, lecz już na etapie dekodowania rozkazu, na postawie jego charakteru. Wiedząc o tym, że najczęstszą przyczyną wstrzymywania są rozkazy odczytu z pamięci, zapisu do pamięci i skoku, można podjąć decyzję o przełączeniu wątku natychmiast po rozpoczęciu wykonywania takiego rozkazu — nawet jeśli w rzeczywistości nie wymaga on wstrzymania wątku, co jednak okazuje się nieco później. Przełączenie odbywa się więc nie w momencie wstrzymywania, lecz w momencie stwierdzenia, że **być może** wstrzymanie takie okaże się konieczne — zgodnie z zasadą „wykonuj rozkazy tak długo, aż mogą pojawić się problemy”. W ten sposób gruboziarnisty charakter wielowątkowości zostaje nieco złagodzony, bo przełączenia między wątkami odbywają się wyraźnie częściej.

Niezależnie od konkretnego wariantu wielowątkowości, niezbędne jest zapewnienie kontroli nad przynależnością poszczególnych operacji do poszczególnych wątków. W warunkach wielowątkowości drobnoziarnistej jedynym rozsądnym rozwiązaniem jest związanie z każdym rozkazem identyfikatora wątku i wysyłania go wraz z tym rozkazem do potoku. W wariancie gruboziarnistym możliwe jest inne rozwiązanie:

gdy następuje przełączenie wątku, wykonanie jego pierwszego rozkazu wstrzymywane jest aż do momentu opróżnienia potoku[4]. W ten oto prosty sposób unikamy mieszania w potoku rozkazów pochodzących z różnych wątków i problem powiązania rozkazów z wątkami rozwiązuje się samoczynnie. Oczywiście opisana technika ma sens jedynie wówczas, gdy średni czas między przełączeniem wątków jest znacznie dłuższy od czasu potrzebnego na opróżnienie (zrealizowanie) potoku.

Jak dotąd zakładaliśmy, że procesor może inicjować tylko jeden rozkaz w jednym cyklu zegara; jak wiadomo, założenie to jest nieprawdziwe w odniesieniu do większość nowoczesnych procesorów. W sytuacji przedstawionej na rysunku 8.6 w jednym cyklu mogą być inicjowane dwa rozkazy (tzw. procesor dwupotokowy), choć nadal obowiązuje zasada, że wstrzymanie jednego rozkazu wstrzymuje także inicjowanie rozkazów następujących po nim. W części (a) rysunku pokazano, jak w tym warunkach realizowana jest wielowątkowość drobnoziarnista: dwa pierwsze rozkazy wątku *A* (*A1* i *A2*) zainicjowane zostają w pierwszym cyklu, lecz w przypadku wątku *B* występuje konieczność wstrzymania na rozkazie *B*1, więc rozkaz *B*2 nie jest już inicjowany.



RYSUNEK 8.6. Wielowątkowość w procesorze dwupotokowym: (a) drobnoziarnista; (b) gruboziarnista; (c) współbieżna

W części (b) rysunku 8.6 widzimy realizację gruboziarnistego wariantu wielowątkowości w procesorze dwupotokowym; specjalny statyczny planista (*scheduler*) *nie* wprowadza martwych cykli zegarowych w przypadku wstrzymania rozkazu. W ramach wątku inicjowane są dwa rozkazy w każdym cyklu zegara, aż do momentu, gdy któryś z rozkazów wymagać będzie wstrzymania; na początku kolejnego cyklu zegara następuje przełączenie wątków.

W procesorach superskalarnych realizowany jest jeszcze jeden wariant wielowątkowości, zwany **wielowątkowością współbieżną** (*simultaneous multithreading*) i zilustrowany w części (c) rysunku 8.6. Jest to w istocie ulepszony wariant wielowątkowości gruboziarnistej, a istotą ulepszenia jest to, że w przypadku wstrzymania wątku zaczyna się natychmiastowe pobieranie rozkazów z kolejnego wątku tak, by zapewnić ciągłą zajętość procesora. Pomaga to także w pełnym wykorzystaniu jednostek funkcjonalnych procesora: gdy dany rozkaz nie może zostać zainicjowany ze względu na niedostępność (zajętość) któreś jednostki, wybierany jest do wykonania rozkaz z innego wątku. W przykładzie pokazanym na rysunku rozkaz *B*8 zostaje wstrzymany w cyklu 11, wskutek czego w cyklu 12 inicjowany jest rozkaz *C*7.

[4] Czyli wykonania wszystkich operacji znajdujących się aktualnie w potoku — *przyp. tłum.*

Czytelników zainteresowanych bardziej szczegółową dyskusją na temat wielowątkowości w procesorach odsyłamy do prac (Dean, 2004), (Kalla i in., 2004) i (Kapil i in., 2004). Połączenie wielowątkowości z wykonywaniem spekulatywnym dyskutowane jest w pracy (Sohi i Roth, 2001).

Hiperwątkowość w Pentium 4

Po omówieniu wielowątkowości w ujęciu abstrakcyjnym, przyjrzyjmy się teraz jej konkretnej realizacji zastosowanej w procesorze Pentium 4. Krótko po tym, jak procesor ten skierowany został do produkcji, architekci z firmy Intel poszukiwać zaczęli sposobów na zwiększenie jego wydajności bez zmiany interfejsu programisty, traktowanego przez większość programistów jak wręcz nietykalna świętość, na modyfikacje której żadną miarą zgody wyrazić niepodobna. W grę wchodziło więc pięć na wskroś oczywistych zabiegów:

(1) zwiększenie częstotliwości zegara taktującego,
(2) zintegrowanie dwóch procesorów w jednym układzie,
(3) dodanie nowych jednostek funkcjonalnych,
(4) wydłużenie potoku,
(5) zastosowanie wielowątkowości.

Zwiększanie częstotliwości taktowania jest najbardziej oczywistym i najmniej skomplikowanym sposobem zwiększania wydajności procesora, tak więc kolejne modele procesorów bywają zwykle nieco szybsze od swych poprzedników. Wyższa częstotliwość to jednak większy pobór energii, co okazuje się być problemem w przypadku laptopów i zasilanych z baterii urządzeń przenośnych; to także większa ilość rozpraszanego ciepła i większe problemy z chłodzeniem.

Integrowanie dwóch procesorów we wspólnym układzie także nie jest zbyt skomplikowane, lecz podnosi koszty samego układu w związku ze zdwojeniem zapotrzebowania na powierzchnię przeznaczoną na pamięć cache, osobną dla każdego procesora. Można tego problemu uniknąć, stosując pojedynczą pamięć cache współdzieloną przez obydwa procesory, co jednak z perspektywy konkretnego procesora zmniejsza dwukrotnie efektywny rozmiar *jego* pamięci podręcznej, a to natychmiast odbija się na wydajności. Samo zdublowanie procesorów we wspólnym układzie nie gwarantuje bynajmniej efektywnego wykorzystania tego tandemu, do tego bowiem konieczne są aplikacje napisane w odpowiedni sposób — te jednak częściej spotkać można w serwerach z górnej półki niż w aplikacjach biurowych.

Rozszerzanie architektury procesora o nowe jednostki funkcjonalne też nie należy do zabiegów zbyt skomplikowanych, nie może być jednak rozważane w oderwaniu od całej architektury: pożytek z dziesięciu jednostek ALU będzie raczej nikły, jeśli procesor nie nadąży z napełnianiem potoku rozkazami w tempie gwarantującym pełne wykorzystanie tych jednostek.

Dłuższe potoki, z większą liczbą etapów, na których wykonywane są prostsze operacje, przyczyniają się co prawda do zwiększania wydajności, lecz jednocześnie zwiększają prawdopodobieństwo występowania sytuacji niepożądanych, jak błędne przewidywania rozgałęzień, chybienia pamięci cache, przerwania i w ogóle wszelkie inne zjawiska zakłócające normalną pracę w potoku. Ponadto, w celu pełnego wykorzystania możliwości oferowanych przez dłuższe potoki należało by jednocześnie zwiększyć częstotliwość ich taktowania, a to rodzi problemy opisane w jednym z wcześniejszych akapitów.

Pozostaje więc zastosowanie wielowątkowości i tu perspektywy okazują się wielce obiecujące, bowiem jak wykazały liczne eksperymenty, zwiększenie o 5% powierzchni układu w związku ze wsparciem wielowątkowości może poprawić wydajność niektórych aplikacji nawet o 25%. Pierwszym procesorem Intela wykorzystującym mechanizm wielowątkowości był Xeon wypuszczony na rynek w 2002 roku, lecz wielowątkowość dodana została także do procesora Pentium 4, w modelach taktowanych zegarem 3,06 GHz i szybszych. Wielowątkowość w wersji obecnej w Pentium 4 firma Intel określa mianem **hiperwątkowości** (*hyperthreading*).

Podstawowa idea hiperwątkowości sprowadza się do jednoczesnej realizacji dwóch wątków lub dwóch procesów — z perspektywy procesora wątek nie różni się bowiem niczym od procesu. Z perspektywy systemu operacyjnego hiperwątkowy Pentium 4 postrzegany jest jako dwa procesory współdzielące pamięć operacyjną i (wspólną) pamięć cache. System operacyjny gwarantuje niezależność wątków (w ubieganiu się o czas procesora); nic nie stoi więc na przeszkodzie, by każdej z dwóch realizowanych równocześnie aplikacji przydzielić po jednym ze wspomnianych procesorów — na przykład program-demon odbierający (wysyłający) w tle pocztę elektroniczną może być realizowany niezależnie od programu interakcyjnego, z którym użytkownik aktualnie pracuje.

W szczególności aplikacje napisane w sposób wielowątkowy mogą być przez system traktowane w taki sposób, że różne wątki realizowane będą na różnych procesorach. Przykładowo, programy do obróbki materiałów multimedialnych wykorzystują niezliczone filtry w odniesieniu do ramek z pewnego zakresu: filtry te dokonują modyfikacji jasności, kontrastu, głębi kolorów, gammy itp. dla każdej z ramek. Jeśli filtr taki napisany zostanie tak, że dwa różne jego wątki dokonują obróbki ramek o numerach (odpowiednio) parzystych i nieparzystych, to każdy z tych wątków może zostać powierzony (przez system operacyjny) odrębnemu procesorowi.

Ponieważ obydwa wątki sprzętowe — w dalszym ciągu będziemy je nazywać **hiperwątkami** — współdzielą rozmaite elementy sprzętowe, konieczne jest wypracowanie określonej strategii zarządzania tym współdzieleniem. W związku w hiperwątkowością firma Intel zdefiniowała cztery takie strategie: powielanie (duplikowanie) zasobów, partycjonowanie zasobów, współdzielenie progowe i współdzielenie pełne.

Niewątpliwie **powielony** zostać musi licznik rozkazów, ponieważ każdy z hiperwątków posiada swój własny przepływ sterowania. Jako że każdy hiperwątek korzysta z rejestrów, zduplikowany musi zostać ich zestaw, a dokładniej — tablica dokonująca mapowania (przemianowywania) rejestrów logicznych (`EAX`, `EBX` itd.) w rejestry sprzętowe. Konieczne jest także powielenie kontrolerów przerwań, ponieważ każdy z hiperwątków może być niezależnie przerywany.

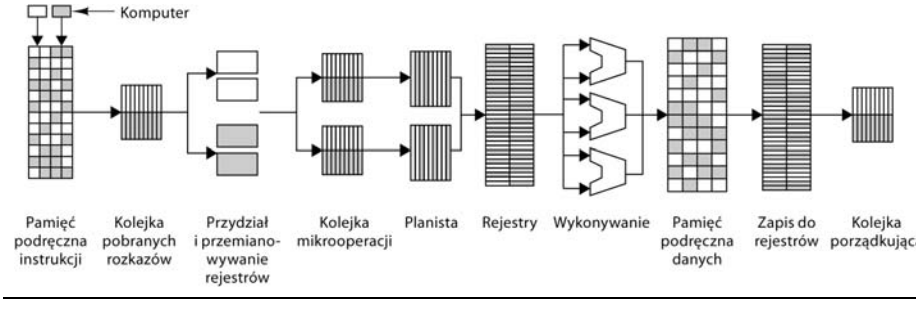
Partycjonowanie zasobów polega na ich (w miarę) równomiernym rozdzieleniu między obydwa hiperwątki. Jeżeli na przykład procesor wyposażony jest w kolejkę rozkazów czekających na umieszczenie w potoku, poszczególne elementy tej kolejki przydzielane są obydwu hiperwątkom naprzemiennie. Da się to zrealizować dość łatwo, zapewnia niezależność hiperwątków, a jeżeli zasoby są rozdzielone prawdziwie równomiernie, mamy w istocie do czynienia z dwoma oddzielnymi procesorami. Partycjonowanie zasobów ma jednak podstawową wadę: mimo że zasoby przydzielone do jednego hiperwątku nie są aktualnie wykorzystywane, nie można ich przydzielić drugiemu hiperwątkowi, który mógłby być wykonywany, a tak po prostu musi czekać.

Przeciwieństwem partycjonowania zasobów jest ich **pełne współdzielenie**. Hiperwątki ubiegają się o zasoby dynamicznie i każdy z nich otrzymuje to, czego żąda na zasadzie „pierwszy żąda — pierwszy obsłużony”. Likwiduje to marnotrawienie zasobów

„leżących odłogiem", za to stwarza problemy innego rodzaju. Wyobraźmy sobie mianowicie współdziałanie dwóch hiperwątków: jeden, szybki, wykonuje głównie operacje dodawania i odejmowania, drugi, wolny, naszpikowany jest operacjami mnożenia i dzielenia. Jeśli pobranie rozkazu z pamięci trwa krócej niż wykonanie mnożenia czy dzielenia, rozkazy czekające na wprowadzenie do potoku zajmują dynamicznie kolejne elementy kolejki i po krótkiej chwili zajmują całą kolejkę; szybki hiperwątek, także ubiegający się o dostęp do tej kolejki, może funkcjonować jedynie w tych krótkich momentach, gdy wolny hiperwątek zwalnia pojedyncze elementy kolejki (i konsekwentnie zajmuje wszystko, co zwolni hiperwątek szybszy). Współdzielona kolejka staje się przysłowiowym wąskim gardłem.

Można opisanego efektu uniknąć, stosując **współdzielenie progowe**. Podobnie jak przy współdzieleniu pełnym zasoby przydzielane są dynamicznie, ale przydzielane są tylko do pewnej ustalonej granicy. Jeżeli w przypadku wątków opisanych w poprzednim akapicie narzucimy ograniczenie, że każdy z hiperwątków może zająć nie więcej jak $\frac{3}{4}$ kolejki, nie wystąpi ryzyko „zagłodzenia" (*starvation*) hiperwątku szybszego, ponieważ co najmniej $\frac{1}{4}$ kolejki będzie dostępna tylko dla niego.

Oczywiście to, która strategia jest najlepsza, zależne jest od specyfiki konkretnego zasobu. Hiperwątkowość w Pentium 4 wykorzystuje różną strategie dla różnych zasobów, w celu uniknięcia problemów wyżej opisanych. Duplikowaniu podlegają zasoby permanentnie wykorzystywane przez obydwa hiperwątki, czyli licznik rozkazów, mapa rejestrów i kontroler przerwań. Ich powielenie wymaga zwiększenia powierzchni układu o 5%, co stanowi umiarkowaną cenę za zwiększoną wydajność. Pełnemu współdzieleniu podlegają zasoby występujące w takiej obfitości, że nie istnieje zagrożenie, iż któryś hiperwątek zagarnie je wszystkie dla siebie — na przykład wiersze pamięci podręcznej (cache). Zasoby sterujące operacjami w potoku, jak rozmaite kolejki, są natomiast partycjonowane — każdemu hiperwątkowi przydzielana jest połowa egzemplarzy każdego zasobu, na przykład połowa elementów danej kolejki. Na rysunku 8.7 przedstawiono schematycznie mikroarchitekturę Netburst wykorzystywaną w Pentium 4, z podziałem zasobów między dwa hiperwątki oznakowane (odpowiednio) białymi i szarymi prostokątami.



RYSUNEK 8.7. Podział zasobów między hiperwątki w mikroarchitekturze Netburst procesora Pentium 4

Na rysunku widać wyraźnie podział każdej kolejki między hiperwątki (statystycznie po połowie elementów) — każdy hiperwątek ma teraz do dyspozycji po połowie z każdej kolejki i nie grozi mu zagłodzenie przez partnera. Podzielona jest też pula przydzielanych rejestrów i pula rejestrów podstawianych w procesie ich przemianowywania. Elementy planisty przydzielane są dynamicznie do pewnej granicy, co zapobiega zawłaszczeniu planisty przez któryś z hiperwątków. Pozostałe elementy potoku podlegają pełnemu przydziałowi dynamicznemu.

Mimo iż hiperwątkowość pomyślana została jako środek zwiększenia wydajności procesora, to jednak niekiedy jej obecność może przynosić skutki wręcz odwrotne. Wyobraźmy sobie dwa wątki pewnego procesu (w sensie wątków systemu operacyjnego, nie hiperwątków — patrz rozdział 6.) takie, że każdy z nich może działać w pełni efektywnie, mając do dyspozycji co najmniej $\frac{3}{4}$ pamięci podręcznej, w przeciwnym razie częste chybienia adresów w tej pamięci będą jego wydajność znacząco obniżać. Nie jest problemem spełnienie tego warunku w przypadku realizacji wspomnianego procesu na „zwykłym" procesorze, oczywiście pod kontrolą systemu operacyjnego zapewniającego obsługę wątków. W warunkach hiperwątkowości, gdy obydwa wątki realizowane będą jako hiperwątki procesora, każdy z wątków będzie mieć do dyspozycji **tylko połowę** pamięci podręcznej; zysk wynikający z potraktowania obu wątków jako hiperwątki zostanie zniweczony przez straty wynikające ze zbyt małej pamięci podręcznej. Nie można także zapominać o komplikacjach wynikających z rozmaitych strategii zarządzania przydziałem zasobów na potrzeby hiperwątków: podczas gdy partycjonowanie zasobów nie wydaje się trudne w obsłudze, to już dynamiczny przydział zasobów (zwłaszcza przydział limitowany) wymaga odpowiednich mechanizmów ewidencjonujących.

Czytelnikom zainteresowanym hiperwątkowością procesora Pentium 4 polecamy jako lekturę uzupełniającą prace (Gerber i Binstock, 2004), (Koufaty i Marr, 2003) oraz (Tuck i Tullsen, 2003).

8.1.3. Wieloprocesory jednoukładowe

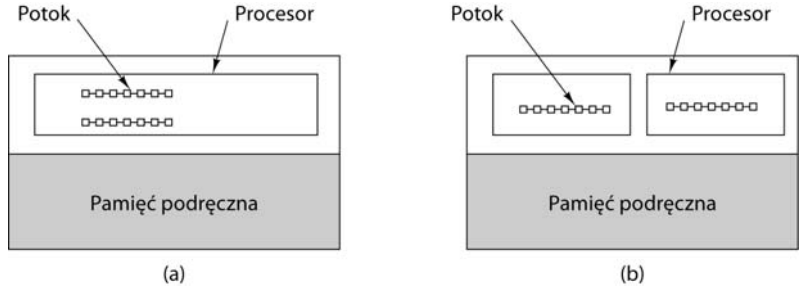
Mimo iż wielowątkowość sprzętowa zdolna jest znacząco zwiększyć wydajność procesora za umiarkowaną cenę, to jednak dla wielu aplikacji potrzebny jest sprzęt znacznie bardziej wydajny. Na potrzeby takich aplikacji opracowano układy scalone, zawierające dwa lub więcej procesorów; dwoma głównymi obszarami zastosowań takich układów są wydajne serwery oraz elektronika użytkowa (*consumer electronics*) — omówimy je krótko w dalszej części rozdziału.

Wieloprocesory jednorodne

Dzięki osiągnięciom technologii VLSI możliwe jest pakowanie dwóch lub więcej wydajnych procesorów w jednym układzie scalonym. Ponieważ wszystkie procesory upakowane w takim układzie współdzielą pamięć podręczną poziomu 1 i 2 oraz pamięć operacyjną, wspomniany układ zaliczany jest do wieloprocesorów, zgodnie z klasyfikacją dokonaną w rozdziale 2. Typowym obszarem zastosowań takich wieloprocesorów są duże „farmy serwerów" WWW; przez wyposażenie serwera w kilka procesorów

współdzielących nie tylko pamięć, lecz także dyski i interfejs sieciowy, można niemal podwoić jego wydajność, bez konieczności podwajania kosztu — koszt układu z upakowanymi procesorami stanowi bowiem zwykle drobny ułamek kosztu całego wyposażenia serwera.

Wieloprocesory jednoukładowe małej skali projektowane są w dwóch wariantach, przedstawionych na rysunku 8.8. W pierwszym wariancie, widocznym w części (a), w pojedynczym układzie zintegrowane są dwa potoki, co podwaja szybkość wykonywania rozkazów. W wariancie drugim, widocznym w części (b), układ zawiera dwa odrębne **rdzenie**, z których każdy zawiera kompletny procesor. Pod pojęciem „rdzenia" (*core*) rozumiemy tu złożony obwód — procesor, kontroler wejścia-wyjścia, pamięć cache — który może być wbudowywany w układ w sposób modularny, najczęściej w towarzystwie innych rdzeni.



RYSUNEK 8.8. Wieloprocesor jednoukładowy: (a) z dualnym potokiem; (b) z dwoma rdzeniami

Pierwszy z wymienionych wariantów umożliwia współdzielenie zasobów — na przykład jednostek funkcjonalnych — między procesory, co pozwala jednemu procesorowi na wykorzystywanie zasobów niepotrzebnych innym procesorom; rozwiązanie to z trudem poddaje się jednak skalowaniu przy zwiększaniu liczby procesorów (powyżej dwóch). Dla odmiany, upakowanie w jednym układzie kilku rdzeni będących kompletnymi procesorami da się zrealizować względnie prosto.

Do wieloprocesorów powrócimy w dalszej części rozdziału; ponieważ nawiązywać będziemy często do wieloprocesorów obejmujących układy zawierające pojedyncze procesory, większość z omawianych zagadnień dotyczyć będzie także układów z wieloma procesorami.

Wieloprocesory niejednorodne (heterogeniczne)

Zupełnie innym obszarem zastosowań jednoukładowych wieloprocesorów są systemy wbudowane, między innymi w urządzenia elektroniki użytkowej: telewizory, odtwarzacze DVD, kamkodery, konsole gier, telefony komórkowe itp. Systemy te charakteryzują się wysokimi wymogami pod względem wydajności oraz rozmaitymi ograniczeniami, na przykład gabarytowymi. Mimo oczywistych różnic w wyglądzie i zastosowaniach wspomnianych urządzeń, wszystkie one są w istocie małymi komputerami, z jednym lub kilkoma procesorami, pamięciami, kontrolerami wejścia-wyjścia i innymi specyficznymi urządzeniami: przykładowo, telefon komórkowy jest (z grubsza biorąc) komputerem integrującym w sobie procesor, pamięć, miniaturową klawiaturę, mikrofon, głośnik i bezprzewodowe łącze sieciowe.

Spójrzmy z kolei na inne urządzenie elektroniki użytkowej — przenośny odtwarzacz DVD; znajdujący się w nim komputer wykonywać musi między innymi następujące zadania:

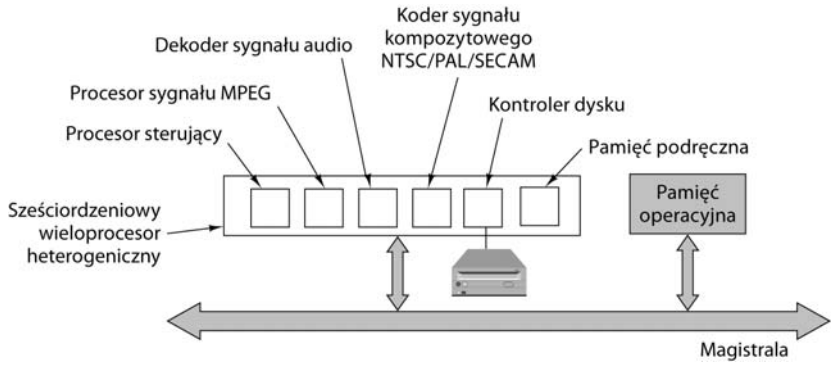
- sterowanie tanim i zawodnym serwomechanizmem wyboru i śledzenia ścieżek,
- konwersja analogowo-cyfrowa,
- korekcja błędów,
- deszyfracja i zarządzanie cyfrowymi prawami autorskimi,
- dekompresja strumienia MPEG-2,
- dekompresja audio,
- kodowanie sygnału dla telewizorów pracujących z systemach NTSC, PAL lub SECAM.

Realizacja wymienionych zadań podlega silnym uwarunkowaniom czasowym oraz określonym wymogom w zakresie jakości, oszczędności energii, chłodzenia, a także ograniczeniom pod względem rozmiarów, ciężaru i ceny.

Dysk DVD zawiera pojedynczą spiralną ścieżkę podobną do ścieżki płyty CD widocznej na rysunku 2.20 Podczas wirowania dysku ścieżka ta musi być dokładnie śledzona przez głowicę odczytującą; ponieważ ze względu na ograniczenia cenowe serwomechanizm głowicy z konieczności musi być tani — a więc raczej zawodny — pozycjonowanie głowicy musi być w dużym stopniu kontrolowane przez oprogramowanie. Sygnał pochodzący z głowicy jest sygnałem analogowym, który w celu jakiejkolwiek obróbki przekształcony musi zostać na postać cyfrową. Po uzyskaniu formy cyfrowej sygnał ten staje się przedmiotem intensywnej korekcji błędów — których multum pojawia się podczas wytłaczania płyt DVD i które korygowane muszą być przez oprogramowanie. Sygnał wideo jest sygnałem skompresowanym według standardu MPEG-2, którego dekompresja wymaga złożonej obróbki (m.in. wyliczania transformat Fouriera) i koniecznej do tego dużej mocy obliczeniowej. Podobnie sygnał audio skompresowany jest przy użyciu modelu psychoakustycznego i jego dekompresja wymaga równie skomplikowanych obliczeń. Zdekompresowanie sygnałów audio i wideo to dopiero połowa roboty, muszą one być bowiem następnie zakodowane w sposób umożliwiający ich odtworzenie w odbiorniku telewizyjnym działającym w określonym standardzie — NTSC, PAL lub SECAM, zależnie od kraju, w którym odtwarzacz DVD jest sprzedawany. Powinno być oczywiste, iż wykonanie tych wszystkich funkcji w czasie rzeczywistym, przez oprogramowanie pracujące na tanim, uniwersalnym procesorze, jest po prostu niemożliwe. Potrzebne są niejednorodne (heterogeniczne) wieloprocesory zawierające kilka rdzeni wyspecjalizowanych w konkretnych zadaniach. Schemat funkcjonalny przykładowego odtwarzacza DVD przedstawiliśmy na rysunku 8.9.

Poszczególne rdzenie wieloprocesora użytego w tym odtwarzaczu zaprojektowane zostały do spełniania swych funkcji szczególnie starannie i przy możliwie najniższym koszcie. Przykładowo, strumień wideo DVD kompresowany jest według algorytmu oznaczanego akronimem **MPEG-2**, od nazwy wynalazców — **Motion Picture Expert Group**. Ogólnie rzecz biorąc, algorytm ten dokonuje podziału każdej ramki na bloki pikseli i każdy z tych bloków poddaje skomplikowanym transformacjom[5].

[5] Bardziej szczegółowy opis algorytmu MPEG znajduje się w książce A.S. Tanenbauma *Sieci komputerowe*, wyd. Helion, 2004 (*http://helion.pl/ksiazki/siecik.htm*) — *przyp. tłum.*



RYSUNEK 8.9. Struktura logiczna prostego odtwarzacza DVD zawierającego heterogeniczny wieloprocesor wielordzeniowy

Każdą ramkę można następnie wyrazić w postaci sekwencji tak przetransformowanych bloków; zamiast oryginalnego bloku może się jednak pojawić w sekwencji informacja, że bieżący blok jest niemal identyczny (z wyjątkiem niewielkiej liczby pikseli) z blokiem w innej ramce, zajmującym (w tej ramce) pozycję przesuniętą o wektor (Δx, Δy) w stosunku do pozycji bieżąco analizowanej (i takie właśnie sytuacje w największym stopniu wpływają na efektywność kompresji). Dekompresja tak zakodowanego strumienia (nie mówiąc już o kompresji) w sposób programowy byłaby niesamowicie powolna[6], można jednak skonstruować dekoder sprzętowy wykonujący tę dekompresję znacznie szybciej. Podobnie ma się rzecz z dekodowaniem sygnału audio i kodowaniem sygnału kompozytowego przeznaczonego do odtworzenia w zwykłym telewizorze, zgodnie z jednym z przyjętych standardów. Przesłanki te odprowadziły do skonstruowania heterogenicznych, wielordzeniowych wieloprocesorów, z myślą o ich zastosowaniu głownie w aplikacjach przeznaczonych do obróbki materiałów audiowizualnych. Ponieważ jednak procesor sterujący takiego wieloprocesora jest zwykłym procesorem ogólnego przeznaczenia, można ów wieloprocesor zastosować także w podobnych warunkach, na przykład w odtwarzaczu DVD.

Innym urządzeniem powszechnego użytku, w którym znalazł zastosowanie wieloprocesor, jest telefon komórkowy. Dzisiejsze telefony komórkowe nie są li tylko telefonami sensu stricte, rozszerzane są bowiem o dodatkowe funkcje aparatów cyfrowych, kamer wideo, gier, przeglądarek WWW, klientów poczty elektronicznej, a nawet GPS, wykorzystując jako środek łączności bądź to rodzime technologie telefonii komórkowej (CDMA lub MSG), bądź też bezprzewodowy internet (IEEE 802.11, bardziej znany pod nazwą WiFi). Sprawna realizacja wszystkich tych funkcji, przy zachowaniu niewielkich gabarytów i niewielkiego ciężaru (jak na urządzenia mobilne przystało), kwalifikuje je w sam raz do realizacji przez specjalizowane rdzenie heterogenicznych wieloprocesorów.

Współczesne układy scalone składają się z kilkuset milionów tranzystorów. Tak wielka ich liczba eliminuje a priori możliwość projektowania takiego układu metodą „bramka po bramce, ścieżka po ścieżce": czasochłonność takiego projektowania

[6] Faktycznie, jeszcze dziesięć lat temu można było napisać, że „wymogi dekompresji strumienia MPEG-2 potrafią rzucić na kolana każdy odtwarzacz programowy" (to cytat z jednej z książek wydanej niegdyś przez Helion), dziś jednak programowe odtwarzacze płyt DVD są powszechnie spotykane we wszystkich (niemal) komputerach — *przyp. tłum.*

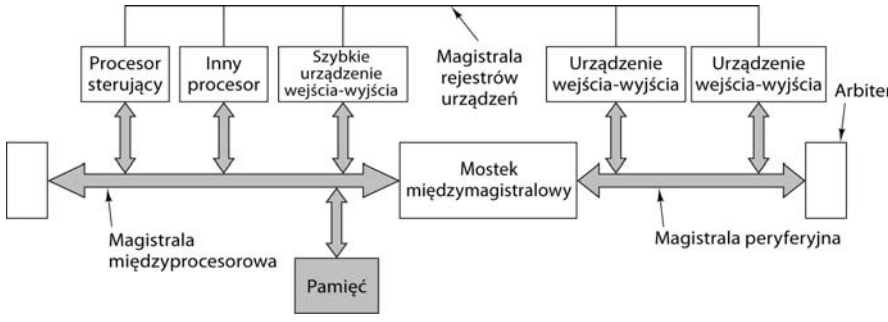
sprawiałaby, że w momencie ukończenia projekt ten byłby już projektem przestarzałym. Jedyną alternatywą jest wykorzystanie gotowych rdzeni, które w analogii do oprogramowania mogą być uważane jako coś na kształt gotowych bibliotek. Główne zadanie projektanta sprowadza się do znalezienia złotego środka między wykorzystywaniem procesorów uniwersalnych a specjalizowanych rdzeni. Przerzucenie większości funkcji na oprogramowanie pracujące na kontrolnym procesorze uniwersalnym czyni projektowany system wolniejszym, lecz za to umożliwia zmniejszenie rozmiarów całego układu i tym samym obniżenie jego ceny; użycie specjalizowanych procesorów (rdzeni) wykonujących skomplikowaną obróbkę strumieni multimedialnych wymaga większego układu i pociąga za sobą większe koszty, pozwala jednak osiągnąć żądaną wydajność przy mniejszej częstotliwości zegara i wynikającego stąd mniejszego zużycia energii oraz mniej intensywnego wydzielania ciepła. Konieczność rozstrzygania takich „makroskopowych" kompromisów wypiera niegdysiejszą (mikroskopową) troskę projektantów o to, gdzie najkorzystniej byłoby ulokować ten czy inny tranzystor.

Zastosowania audiowizualne wiążą się z przetwarzaniem ogromnych ilości danych, nic więc dziwnego, że 50% do 75% procent powierzchni typowego układu przeznaczonego do takich zastosowań stanowią różnego rodzaju pamięci i że odsetek ten wciąż rośnie. Stawia to przed projektantami kolejne wyzwania: ile poziomów pamięci cache należałoby użyć? Czy powinny być one monolityczne, czy też podzielone? Jak duża powinna być każda z tych pamięci? Jak szybka? Czy umieścić wewnątrz układu także pamięć operacyjną? W jakiej wielkości? W jakiej technologii — SRAM czy SDRAM? Konkretna odpowiedź na każde z postawionych pytań będzie miała swe konsekwencje w postaci określonej wydajności układu, określonego zużycia przezeń energii i określonych problemów związanych z jego chłodzeniem.

W związku z projektowaniem specjalizowanych rdzeni i integrowaniem ich (wraz z innymi komponentami) w pojedynczych układach scalonych, pojawia się kolejny problem projektowy — jak to wszystko połączyć ze sobą? W małych systemach wystarczająca może okazać się pojedyncza magistrala, lecz w systemach bardziej rozbudowanych niechybnie stałaby się ona wąskim gardłem. Alternatywnym rozwiązaniem może być sieć magistrali albo pierścień łączący ze sobą wszystkie rdzenie. W tym ostatnim przypadku problem konfliktów w dostępie do pierścienia może być rozwiązany poprzez cykliczne przekazywanie między rdzeniami małego pakietu zwanego **żetonem** (*token*) — rdzeń może wysłać dane do pierścienia dopiero po otrzymaniu żetonu (który to żeton zobowiązany jest niezwłocznie przekazać do następnego rdzenia).

Za przykład połączenia komponentów układu scalonego może posłużyć rozwiązanie firmy IBM o nazwie **CoreConnect**, którego schemat przedstawiono na rysunku 8.10. W pewnym sensie CoreConnect spełnia dla układu scalonego rolę podobną do tej, jaką dla procesora Pentium pełni magistrala PCI, z tą jednak różnicą, że CoreConnect zaprojektowane zostało bez jakichkolwiek wymogów zachowania kompatybilności ze starszymi rozwiązaniami i bez krępujących ograniczeń właściwych magistralom „na płycie", na przykład wynikających z ustalonej liczby styków w gnieździe.

CoreConnect składa się z trzech magistral. **Magistrala międzyprocesorowa** jest szybką, synchroniczną potokową magistralą z 32, 64 lub 128 liniami danych taktowanych z częstotliwością 66, 133 lub 183 MHz, co daje maksymalną przepustowość 23,4 Gb/s (w porównaniu z 4,2 Gb/s jako maksimum dla magistrali PCI). Potokowy charakter magistrali umożliwia rdzeniom inicjowanie kolejnych transferów przed zakończeniem poprzednich oraz równoczesny dostęp różnych rdzeni do różnych linii, podobnie jak w przypadku PCI. Magistrala międzyprocesorowa zoptymalizowana została dla szybkich transferów blokowych, przeznaczona jest bowiem do łączenia szybkich rdzeni, jak procesory, dekodery MPEG-2, szybkie sieci i tym podobne komponenty.



RYSUNEK 8.10. Przykład zastosowania architektury IBM CoreConnect

Rozciągnięcie magistrali międzyprocesorowej na cały układ spowodowałoby spadek jego ogólnej wydajności, dlatego też konieczna jest druga magistrala łącząca wolne urządzenia wejścia-wyjścia, jak UART, zegary, kontrolery USB, urządzenia transmisji szeregowej itp. Magistrala ta, zwana **magistralą peryferyjną**, pomyślana została jako prosty interfejs dla 8-, 16- i 32-bitowych urządzeń zewnętrznych, zbudowany na bazie jedynie kilkuset bramek. Jest ona magistralą synchroniczną o maksymalnej przepustowości 300 Mb/s. Obydwie magistrale — międzyprocesorowa i peryferyjna — połączone są ze sobą mostkiem podobnym do tego łączącego niegdyś magistrale PCI i ISA.

Trzecia magistrala to **magistrala rejestrów urządzeń** — bardzo powolna magistrala asynchroniczna z negocjowaniem dostępu, umożliwia procesorom łączność z rejestrami wszystkich urządzeń peryferyjnych w celu sterowania tymi urządzeniami i kontrolowania ich pracy. Przeznaczona jest do sporadycznych, kilkubajtowych transferów.

Dzięki standardowym magistralom, interfejsowi i środowisku, IBM spodziewa się stworzyć miniaturę magistrali PCI, do której przyłączać będzie można łatwo procesory i kontrolery wytwarzane przez różnych producentów. Za ową miniaturyzacją skrywa się także ważna różnica praktyczna. Oryginalna magistrala PCI, jako składnik płyt głównych, dostarczana jest sprzedawcom i klientom bezpośrednio przez producentów. W przypadku CoreConnect jest inaczej: najpierw projektanci pojedynczych rdzeni dostarczają swe projekty (jako licencjonowaną własność intelektualną) firmom produkującym sprzęt elektroniki użytkowej. Firmy te, na podstawie uzyskanych licencji oraz na bazie własnych rozwiązań, sporządzają projekty kompletnych mikrokomputerów. Ponieważ wytwarzanie wielokomputerów wymaga niezbędnego zaplecza technicznego, wytwórcy elektroniki użytkowej poprzestają jedynie na ich projektowaniu, zlecając ich produkcję wyspecjalizowanym wytwórcom układów scalonych. W ten oto sposób powstają rdzenie wielu procesorów (ARM, MIPS, PowerPC i innych), jak również dekoderów MPEG, procesorów sygnałowych i standardowych kontrolerów wejścia-wyjścia.

CoreConnect nie jest jedynym obecnym na rynku przedstawicielem magistrali wewnątrzukładowej. Równie rozpowszechnione jest rozwiązanie o nazwie **AMBA** (**Advanced Microcontroller Bus Architecture**, „architektura zaawansowanej magistrali mikrokontrolerów"). Wśród rozwiązań mniej popularnych wymienić należy przede wszystkim **VCI** (**Virtual Component Interconnect**) (Flynn, 1997) i **OCP-IP** (**Open Core Protocol-International Partnership**) (Kogel i Meyr, 2004 oraz Ouadjaout i Houzet, 2004). Magistrale wewnątrzukładowe to jednak dopiero początek, bowiem melodią niedalekiej przyszłości są kompletne sieci komputerowe ukryte wewnątrz układu scalonego (Benini i De Micheli, 2002).

Ze względu na nieustannie rosnącą częstotliwość taktowania i związane z tym coraz bardziej intensywne wydzielanie ciepła, projektowanie wielokomputerów jednoukładowych samo staje się gorącym tematem. Zainteresowanych Czytelników odsyłamy do prac (Claasen, 2003), (Jerraya i Wolf, 2005), (Kumar i in., 2004), (Lavagno, 2002), (Lines, 2004) oraz (Ravikumar, 2004).

8.2. KOPROCESORY

Po zapoznaniu się z kilkoma metodami realizacji zrównoleglenia na poziomie architektury pojedynczego układu scalonego, przenieśmy się teraz o poziom wyżej i popatrzmy, jak można przyspieszyć komputer, wyposażając go w drugi, specjalizowany procesor. Te wspomagające procesory, zwane popularnie **koprocesorami**, mają długą historię i występują w różnych odmianach, także gabarytowych. We flagowym systemie komputerowym lat 60. i 70. ubiegłego wieku — IBM/360 — i w systemach będących jego następcami obecne są specjalizowane układy, zwane **kanałami wejścia-wyjścia** i realizujące operacje wejścia-wyjścia niezależnie od procesora centralnego. Z kolei jednostce centralnej komputera CDC 6600 towarzyszy 10 niezależnych **procesorów peryferyjnych**, zajmujących się realizacją wejścia-wyjścia i spełniających inne funkcje administracyjne. Także w komputerach PC od początku obecne były koprocesory realizujące arytmetykę zmiennopozycyjną i zaawansowane operacje graficzne — a nawet kontroler DMA może być uważany za koprocesor. Zadania spełniane przez koprocesory bywają zróżnicowane: niekiedy jest to wykonywanie *ad hoc* pojedynczych rozkazów zlecanych przez procesor centralny, niekiedy jednak koprocesor działa (całkowicie lub częściowo) niezależnie od procesora centralnego.

Zróżnicowane bywają także rozmiary koprocesorów. Kanały wejścia-wyjścia systemu IBM/360 miały rozmiary porównywalne z segmentami meblościanek, podczas gdy koprocesor zmiennopozycyjny był małym układem scalonym, niewiele większym od procesora centralnego PC[7]. Procesory sieciowe są natomiast typowymi kartami rozszerzającymi. We wszystkich tych przypadkach koprocesor jest jednak wyraźnie oddzielony od procesora głównego i spełnia w stosunku do niego rolę pomocniczą. Przyjrzymy się nieco dokładniej trzem obszarom potencjalnych zastosowań koprocesorów: przetwarzaniu pakietów sieciowych, aplikacjom multimedialnym oraz kryptografii.

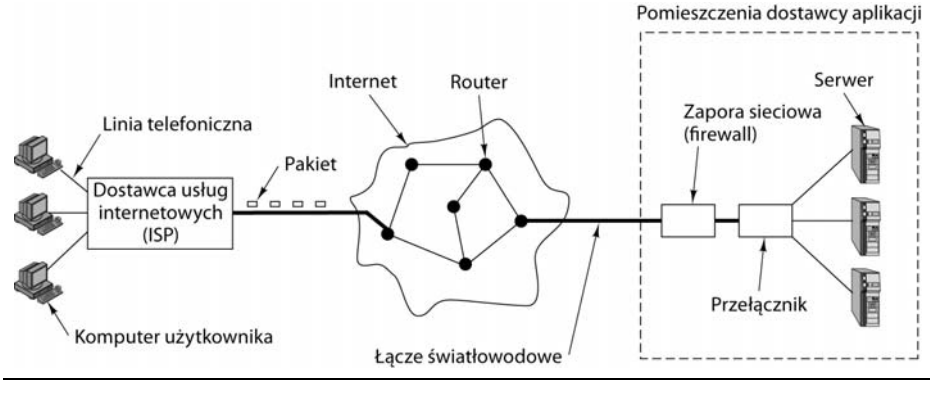
8.2.1. Procesory sieciowe

Większość dzisiejszych komputerów podłączona jest do internetu lub do innych sieci. W rezultacie postępu technologicznego w dziedzinie sprzętu sieciowego same sieci stały się dziś tak szybkie, że czysto programowe zarządzanie wysyłanymi i otrzymywanymi danymi staje się coraz trudniejsze. Lekarstwem na tę trudność są specjalne, sprzętowe procesory sieciowe i w wielu komputerach z górnej półki spotkać można co najmniej jeden z nich. Po krótkim wprowadzeniu w tematykę sieci komputerowych przyjrzymy się więc dokładniej działaniu przykładowego procesora sieciowego.

[7] Począwszy od 80486DX, jednostka zmiennopozycyjna jest zintegrowana z procesorem głównym — *przyp. tłum.*

Wprowadzenie do sieci komputerowych

Sieci komputerowe podzielić można zasadniczo na dwa rodzaje: **sieci lokalne** (**LAN**, **Local Area Network**) znajdujące się wewnątrz biura, budynku czy osiedla oraz **sieci rozległe** (**WAN**, **Wide Area Network**) obejmujące swym zasięgiem całe miasta, kraje, kontynenty, a nawet — jak internet — całą kulę ziemską. Sieci lokalne budowane są najczęściej w technologii **Ethernet** — oryginalny Ethernet miał postać grubego kabla, do którego przyłączane były przewody biegnące od poszczególnych komputerów; przyłączenie to realizowane było za pomocą mechanicznego przecinania się przez izolację kabla, za pomocą ostrza zwanego pieszczotliwie **wampirzym ząbkiem** (*vampire tap*). We współczesnej wersji Ethernetu komputery podłączone są do wspólnego **przełącznika** (*switch*) — jak w prawej części rysunku 8.11. W oryginalnym Ethernecie dane przesyłane były z prędkością 3 Mb/s, lecz w pierwszej komercyjnej wersji prędkość ta wynosiła 10 Mb/s. Niedługo potem pojawiły się sieci o prędkości 100 Mb/s oraz „Ethernet gigabitowy" o prędkości najpierw 1 Gb/s, później 10 Gb/s. Za horyzontem czai się już prędkość 40 Gb/s.



RYSUNEK 8.11. Sposób połączenia komputerów użytkowników z serwerem internetowym

Sieci rozległe skonstruowane są w całkiem inny sposób. Zawierają one specjalizowane komputery zwane **routerami**, połączone ze sobą za pomocą kabli miedzianych lub światłowodowych, jak w centralnej części rysunku 8.11. Porcje danych o wielkości 64 do 1500 bajtów, zwane **pakietami**, przesyłane są od komputera źródłowego, poprzez jeden lub więcej routerów aż do komputera stanowiącego ich przeznaczenie. Pakiet, dotarłszy do routera, zapisywany jest w jego pamięci, po czym przesyłany do kolejnego routera na trasie, gdy dostępna staje się prowadząca do niego linia transmisyjna. Każdy „przeskok" (*hop*) pakietu polega więc na jego zapamiętaniu i przekazaniu, a opisana technika nazywa się **komutacją pakietów z zapamiętywaniem i przekazywaniem** (*store-and-forward packet switching*).

Chociaż zwykło się uważać internet za pojedynczą sieć rozległą, w istocie jest on kolekcją wielu połączonych ze sobą sieci WAN; z perspektywy niniejszych rozważań różnica ta jest jednak nieistotna: na rysunku 8.11 przedstawiono internet (jako całość) z punktu widzenia użytkownika domowego. Jego komputer połączony jest zazwyczaj z serwerem WWW za pośrednictwem linii telefonicznej z modemem albo cyfrowej linii abonenckiej (ADSL), którą omawialiśmy w rozdziale 2. (możliwe jest także połączenie przez kabel telewizyjny, wówczas w lewej części rysunku rolę dostawcy

usług spełnia operator telewizji kablowej). Dane w komputerze użytkownika dzielone są na pakiety i wysyłane do **dostawcy usług internetowych** (*ISP — Internet Service Provider*), czyli firmy zapewniającej swoim klientom dostęp do internetu. Firma ta połączona jest (zwykle za pośrednictwem światłowodu) z jedną z sieci regionalnych lub szkieletowych (*backbone*) tworzących internet. Pakiety użytkownika docierają, przeskok po przeskoku, do wyznaczonego serwera.

Większość firm świadczących usługi WWW posiada specjalne komputery zwane **zaporami sieciowymi** lub **firewallami** (*firewalls*), których zadaniem jest filtrowanie przychodzących pakietów i eliminowanie pakietów niepożądanych, na przykład generowanych w wyniku destrukcyjnej działalności hakera. Zapora sieciowa podłączona jest do sieci lokalnej, zwykle za pośrednictwem przełącznika ethernetowego, kierującego pakiety do odpowiedniego serwera. Oczywiście to tylko opis ogólnej zilustrowanej na rysunku 8.11 idei, za którą kryją się skomplikowane szczegóły techniczne.

Oprogramowanie sieciowe tworzone jest w oparciu o wiele **protokołów**, z których każdy jest zbiorem definicji formatów, sekwencji poleceń oraz znaczeń nadawanych poszczególnym pakietom. Jeżeli na przykład użytkownik chce pobrać (dla przeglądarki) stronę z serwera WWW, przeglądarka ta wysyła pakiet zawierający żądanie `GET PAGE` za pośrednictwem protokołu **HTTP** (**HyperText Transfer Protocol**) do serwera, który zawiera oprogramowanie zdolne to żądanie zinterpretować. Wykorzystywane protokoły są często ze sobą łączone na sposób warstwowy — czyli w ten sposób, że pakiety wyjściowe jednego protokołu stają się pakietami wejściowymi drugiego. Przykładowo, po przetworzeniu w górnej warstwie pakiety kierowane są do warstwy dolnej, która dokonuje ich wysłania; w komputerze odbierającym kolejność przechodzenia pakietów przez warstwy jest odwrotna.

Ponieważ realizacja protokołów sieciowych stanowi sedno działania procesorów sieciowych, zobaczmy najpierw, czym w istocie są owe protokoły. Powróćmy do żądania `GET PAGE` — w jaki sposób jest ono przesyłane do serwera? Otóż przeglądarka (w komputerze użytkownika) nawiązuje najpierw połączenie z serwerem według protokołu **TCP** (**Transmission Control Protocol**, protokół sterowania transmisją). Implementujące ten protokół oprogramowanie odpowiedzialne jest za poprawne dostarczenie wszystkich pakietów, w prawidłowej kolejności. Jeśli dany pakiet zostanie zagubiony po drodze, protokół TCP zapewni jego retransmisję (być może wielokrotną, aż do skutku).

Żądanie `GET PAGE` formatowane jest do postaci komunikatu HTTP i przekazywane protokołowi TCP. Oprogramowanie tego protokołu dodaje do tego komunikatu (na początku) nagłówek zawierający numer sekwencyjny i inne informacje techniczne. Nagłówek ten nazywamy jest oczywiście **nagłówkiem TCP**.

Tak obudowany komunikat przekazywany jest do obróbki przez inną porcję oprogramowania, implementującą **protokół IP** (*Internet Protocol*). Oprogramowanie to dodaje na początku swój własny nagłówek zawierający adres komputera-nadawcy (tego, w którym oryginalny komunikat został utworzony), adres komputera docelowego (który jest przeznaczeniem komunikatu), maksymalną dopuszczalną liczbę przeskoków (w celu zapobieżenia krążeniu zagubionego pakietu w nieskończoność po sieci), sumę kontrolną (umożliwiającą wykrycie ewentualnych błędów w czasie transmisji) i inne informacje.

Tak utworzony pakiet, zawierający oryginalny komunikat `GET PAGE` poprzedzony dwoma nagłówkami, kierowany jest do warstwy łącza danych, gdzie poprzedzony zostaje kolejnym nagłówkiem, ponadto na końcu pakietu dodawana jest suma kontrolna obliczona według kodu **CRC** (**Cyclic Redundancy Code**, redundancyjny kod kontroli

cyklicznej). Obecność tej sumy kontrolnej może wydawać się zbędna, bowiem podobna suma kontrolna znajduje się już w nagłówku IP. To prawda, lecz obecność dwóch sum[8] kontrolnych zwiększa niezawodność transmisji. Na każdym przeskoku sprawdzana jest prawidłowość obydwu sum — po zdjęciu nagłówka (nagłówków) wartość sumy obliczana jest na nowo i porównywana z wartością zapisaną w nagłówku; ewentualna różnica jest świadectwem przekłamania w transmisji. Strukturę pakietu po wzbogaceniu go o nagłówek łącza danych przedstawiono na rysunku 8.12; jeżeli nośnikiem danych jest nie Ethernet, a na przykład linia telefoniczna, należy odpowiednio zmienić nazwę nagłówka. Zarządzanie nagłówkami jest jedną w podstawowych funkcji procesorów sieciowych. Rzecz jasna przedstawiony tu opis jest z konieczności bardzo skrótowy; Czytelników zainteresowanych szczegółami funkcjonowania poszczególnych warstw modelu sieciowego odsyłamy do książki (Tanenbaum, 2003).

RYSUNEK 8.12. Postać pakietu transmitowanego poprzez internet

Wprowadzenie do procesorów sieciowych

Do sieci komputerowych przyłączane są różne rodzaje urządzeń — przede wszystkim komputery (biurkowe i notebooki), lecz także konsole gier, palmtopy (PDA) i telefony komórkowe. Końcowymi urządzeniami sieci są także zazwyczaj serwery korporacyjne. W sieciach funkcjonują oczywiście także urządzenia pośrednie: routery, przełączniki, zapory sieciowe, serwery *proxy,* balansery obciążeń i wiele innych. Interesujące jest to, że właśnie owe pośrednie urządzenia obciążone są najbardziej, bowiem to one przetwarzać muszą najwięcej pakietów w ciągu sekundy.

Zależnie od konkretnej sieci i od konkretnego pakietu, pakiet ten wymagać może różnego rodzaju przetworzeń, zanim dostarczony zostanie aplikacji docelowej lub wysłany na linię transmisyjną. Przetworzenia te polegać mogą np. na podziale zbytu dużego pakietu na mniejsze pakiety (nazywa się to fragmentacją pakietu) lub (wręcz przeciwnie) na odtwarzaniu oryginalnego pakietu z pakietów będących jego fragmentami (defragmentacja). Oprogramowanie routera może także sprawować zarządzanie jakością usługi (szczególnie w przypadku strumieni audio i wideo), zapewniać odpowiedni stopień bezpieczeństwa (przez szyfrowanie i deszyfrację pakietów), dokonywać kompresji i dekompresji, itd.

W sieci LAN o szybkości transmisji 40 Gb/s i rozmiarze pakietów 1KB, urządzenia pośrednie zmuszone są przetwarzać ok. 5 milionów pakietów w ciągu sekundy; przy pakietach 64-bajtowych liczba pakietów przetwarzanych w ciągu sekundy przekroczyć może 80 milionów. Daje to od 12 do 200 nanosekund na jeden pakiet i to przy założeniu, że nie jest konieczne wysyłanie wielu kopii tego samego pakietu. Przekracza to zdecydowanie możliwości obsługi czysto programowej i jest nie do zrealizowania bez wydatnego wsparcia ze strony sprzętu.

[8] Nie należy ponadto zapominać o tym, że suma kontrolna nagłówka IP weryfikuje tylko sam nagłówek IP, podczas gdy ethernetowa suma kontrolna weryfikuje poprawność całego pakietu — *przyp. tłum.*

Jednym ze sposobów realizacji tego wsparcia jest wykorzystywanie **specjalizowanych układów scalonych**, wykonujących funkcje zaszyte „na sztywno" w ich pamięci. Układy takie, oznaczane potocznie akronimem **ASIC** (**Application-Specific Integrated Circuit**, „układ scalony do specyficznych zastosowań"), spotykane są m.in. w wielu współczesnych routerach. Układy ASIC nastręczają jednak całą gamę problemów: po pierwsze, ze względu na ich specyfikę, ich projektowanie jest procesem długotrwałym, podobnie jak samo ich wytwarzanie; ich zawartość kodowana jest na stałe w procesie produkcyjnym, gdy więc potrzebne są nowe elementy funkcjonalne, konieczne jest zaprojektowanie i wyprodukowanie nowych układów. Jako że sama czynność programowania obarczona jest przyrodzonym piętnem pomyłek i przeoczeń, więc koszmarem stają się wszelkiego rodzaju błędy w oprogramowaniu zaszytym w układzie: jedynym sposobem „poprawienia" błędu jest zaprojektowanie, wyprodukowanie i zainstalowanie nowego układu. Jako że układy ASIC wytwarzane są (ze względu na specyfikę zastosowania) w małych seriach, a proces ich projektowania i wytwarzania jest sam w sobie kosztowny, jest oczywiste, że ich koszt jednostkowy musi być stosunkowo wysoki[9].

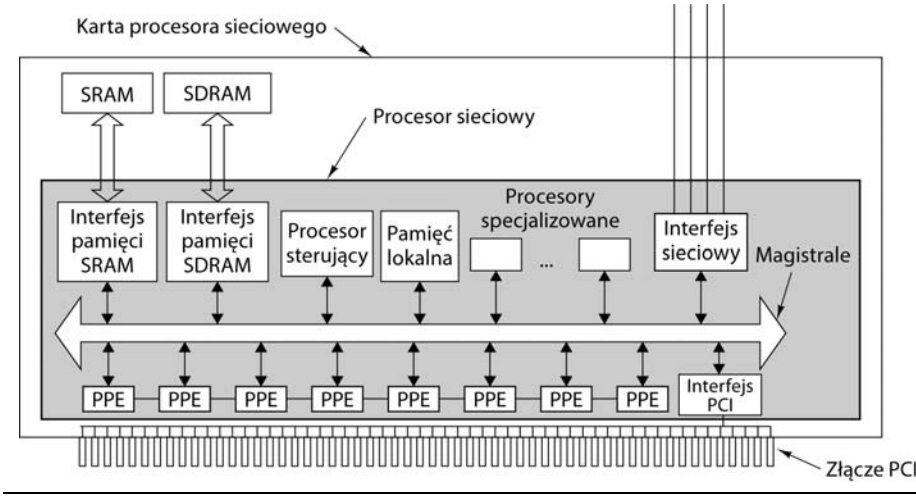
Rozwiązaniem bardziej elastycznym od ASIC są **programowalne macierze bramkowe** (**FPGA**, **Field Programmable Gate Arrays**), stanowiące kolekcje bramek, które można organizować w żądane obwody poprzez zmianę struktury połączeń. W przeciwieństwie do układów ASIC, można je (zgodnie z nazwą) przeprogramowywać za pomocą specjalnych urządzeń, a ich projektowanie i wytwarzanie jest szybsze i prostsze. Mimo niższego kosztu jednostkowego i tak są jednak drogie, skomplikowane, wolne (zwykle wolniejsze od ASIC), co czyni je przydatnymi jedynie w niszowych zastosowaniach.

I tak oto dochodzimy do **procesorów sieciowych**, zdolnych przetwarzać nadchodzące i wychodzące pakiety „z szybkością przewodu", czyli w czasie rzeczywistym. Konstrukcyjnie procesor sieciowy jest zwykle kartą rozszerzającą, zawierającą układ scalony integrujący procesor, pamięć i wspomagające układy logiczne. Jedna lub więcej linii przyłączonych do komputera przekierowywanych jest do procesora sieciowego, dokonującego ekstrakcji i obróbki pakietów, a następnie wysyłania ich na odpowiednie linie lub przekazywania na magistralę PCI komputera, w celu dostarczenia ich do aplikacji docelowej. Architektura przykładowego procesora sieciowego przedstawiona została na rysunku 8.13.

Na karcie procesora obecne są zazwyczaj obydwa typy pamięci — SRAM i SDRAM — wykorzystywane do różnych celów. Pamięć SRAM, jako szybsza, lecz droższa, występuje w mniejszej ilości i wykorzystywana jest do przechowywania m.in. tablic routingu i innych kluczowych struktur, podczas gdy w wolniejszej, lecz tańszej (i przez to obfitszej) pamięci SDRAM magazynowane są przetwarzane pakiety. Dzięki umieszczeniu obydwu pamięci poza układem procesora, projektant ma dużą swobodę w decydowaniu o ich ilości na karcie, wskutek czego tanie karty z jedną linią sieciową (dla komputerów PC i serwerów) zawierają tej pamięci znacznie mniej niż karty „z górnej półki" przeznaczone dla dużych routerów.

Procesory sieciowe optymalizowane są pod kątem szybkiego przetwarzania olbrzymich ilości przychodzących i wysyłanych pakietów. Milion pakietów w ciągu sekundy na jednej tylko linii sieciowej to rzecz zwyczajna, a router może obsługiwać i pół tuzina linii. Aby procesory sieciowe mogły sprostać tak wygórowanym wymaganiom,

[9] Zazwyczaj jednak jest on i tak niższy od kosztu realizacji danego rozwiązania za pomocą układów ogólnego przeznaczenia, a układy ASIC są zwykle mniejsze i tańsze w eksploatacji od takich „uniwersalnych" odpowiedników — *przyp. tłum.*



RYSUNEK 8.13. Karta i układ scalony typowego procesora sieciowego

muszą charakteryzować się dużym stopniem zrównoleglenia — i faktycznie każdy z nich wyposażony jest w wiele rdzeni nazywanych **silnikami przetwarzania pakietów** (**PPE**, **Packet Processing Engine** lub **Protocol Processing Engine** lub **Programmable Processing Engine**, co kto woli), będących w istocie procesorami RISC ze szczątkowymi pamięciami przechowującymi wykonywany program i niewielką liczbę zmiennych.

Silniki PPE mogą być zorganizowane na dwa różne sposoby. W prostszym wariancie wszystkie PPE są identycznie zaprogramowane, a pakiet — nadchodzący lub wysyłany na magistralę — przetwarzany jest przez dowolny z oczekujących PPE; jeśli wszystkie PPE są zajęte, pakiet umieszczany jest w kolejce w pamięci SDRAM i tak oczekuje na zwolnienie któregoś z nich. W tym wariancie *nie* występują poziome połączenia między PPE, widoczne na rysunku 8.13, ponieważ poszczególne PPE nie komunikują się ze sobą w żaden sposób.

W wariancie bardziej skomplikowanym PPE uszeregowane są w potok, a każdy z nich wykonuje ściśle określoną funkcję. Każdy PPE, po przetworzeniu pakietu, przekazuje wskaźnik tego pakietu następnemu w kolejności PPE — w ten oto sposób PPE podobne są do uszeregowanych w potok procesorów, o czym pisaliśmy w rozdziale 2. Oczywiście w obydwu wariantach organizacyjnych wszystkie PPE są w pełni programowalne.

W rozwiązaniach bardziej zaawansowanych PPE wyposażone są w mechanizmy wielowątkowości, czyli w kilka zestawów rejestrowych wraz z niezbędnymi układami zarządzającymi ich wykorzystywaniem. Jeżeli jeden wątek danego PPE zajęty jest oczekiwaniem na pobranie danych z pamięci SDRAM (co wymaga kilku cykli zegarowych), PPE uruchamia kolejny wątek, osiągając w ten sposób maksymalną wydajność nawet w przypadku częstych dostępów do pamięci lub innych operacji zewnętrznych wymagających oczekiwania.

Każdy procesor sieciowy, poza rdzeniami PPE, zawiera także standardowy procesor RISC ogólnego przeznaczenia, wykonujący rozmaite operacje sterujące niezwiązane bezpośrednio z obróbką pakietów, na przykład uaktualnianie tablic routingu. Program i dane dla tego procesora znajdują się w pamięci lokalnej procesora sieciowego.

W wielu procesorach sieciowych obecne są także procesory specjalizowane (jeden lub więcej), wykonujące rozmaite operacje krytycznie uwarunkowane czasowo, na przykład dopasowywanie wzorców; są to najczęściej procesory ASIC, najlepiej spisujące się w przypadku prostych operacji, jak poszukiwanie adresu docelowego w tablicach routingu. Wszystkie komponenty procesora sieciowego komunikują się ze sobą poprzez magistrale równoległe, z szybkością sięgającą wielu gigabitów na sekundę.

Przetwarzanie pakietów

Nadchodzący do procesora sieciowego pakiet poddawany jest wieloetapowej obróbce niezależnie od tego, czy sam procesor ma organizację równoległą, czy potokową. Wiele procesorów sieciowych rozdziela tę obróbkę na dwa stadia: typowe dla pakietów przychodzących (stadium to zwane jest **przetwarzaniem ingresowym**) oraz typowe dla pakietów wysyłanych (zwane **przetwarzaniem egresowym**). Mimo istnienie tego rozróżnienia trudno tak naprawdę powiedzieć, w którym momencie kończy się przetwarzanie ingresowe, a zaczyna przetwarzanie egresowe, bowiem pewne operacje — jak zbieranie statystyki ruchu pakietów — wykonywane są w obydwu tych stadiach obróbki.

Poniżej prezentujemy przykładowe wyodrębnienie ważniejszych etapów wspomnianej obróbki; ma ono jedynie charakter poglądowy — nie wszystkie z wymienionych etapów występują w przypadku obróbki wszystkich pakietów i wiele innych podobnych podziałów na etapy byłoby w tym miejscu równie dobrych.

(1) **Weryfikacja sumy kontrolnej**. Gdy pakiet nadchodzi z Ethernetu, obliczana jest wartość jego kodu CRC i porównywana z wartością zawartą w pakiecie. Jeśli porównywane wartości są równe (bądź jeżeli w pakiecie sumy kontrolnej nie zapisano), w podobny sposób weryfikowana jest suma kontrolna IP — weryfikacja ta ma na celu zapewnienie, że zawartość pakietu nie uległa przekłamaniu już po obliczeniu sumy kontrolnej, na przykład wskutek błędnej zmiany bitu w pamięci routera. Gdy wynik weryfikacji jest pozytywny, pakiet przekazywany jest do dalszej obróbki, w przeciwnym razie jest odrzucany przez router.

(2) **Wyodrębnianie pól**. W tej fazie z nagłówków ekstrahowane są poszczególne pola; w przełączniku ethernetowym sprawdzany jest jedynie nagłówek Ethernetu, natomiast w routerze IP analizowana jest zawartość nagłówka IP. W przypadku równoległej organizacji PPE zawartość poszczególnych pól zapisywana jest w rejestrach, w przypadku organizacji potokowej zapisywana jest ona w pamięci SRAM.

(3) **Klasyfikacja pakietów.** Na podstawie serii testów (zdeterminowanych w oprogramowaniu) dokonywana jest klasyfikacja pakietów według odpowiednich kryteriów. W najprostszym przypadku kryterium to może polegać na odróżnianiu pakietów sterujących od pakietów z danymi, lecz stosowane w praktyce kryteria są zwykle bardziej zaawansowane.

(4) **Wybór ścieżki przetwarzania.** Większość procesorów sieciowych stosuje specjalną „szybką ścieżkę" przetwarzania dla pewnych podstawowych formatów pakietów, traktując odmiennie pozostałe pakiety, na przykład przez kierowanie ich do obróbki przez procesor sterujący. Aby istnienie owej szybkiej ścieżki mogło dać zadowalające rezultaty, konieczne jest zapewnienie efektywnego wyboru konkretnej ścieżki dla danego pakietu.

(5) **Określenie sieci docelowej.** Pakiety IP zawierają adresy 32-bitowe; nie jest ani możliwe, ani pożądane utrzymywanie tablic o 2^{32} pozycjach w celu określenia docelowej sieci przetwarzanego pakietu. Najbardziej znacząca część adresu IP to właśnie numer sieci, pozostała część adresu to numer urządzenia w tej właśnie sieci. Problem w tym, że granica między tymi dwiema częściami nie jest ustalona — numer sieci może mieć różną długość i w związku z tym możliwe są różne dopasowania; należy spośród nich wybrać dopasowanie najdłuższe. Czynność tę powierza się najczęściej układom ASIC.

(6) **Wybór trasy.** Gdy znany jest numer sieci docelowej, w tablicach routingu (przechowywanych w pamięci SRAM) poszukiwana jest odpowiadająca mu linia wyjściowa routera. Ponownie, funkcja ta wykonywana jest najczęściej przez układy ASIC.

(7) **Fragmentacja i składanie pakietów.** Programiści lubią wykorzystywać duże pakiety w warstwie TCP w celu zredukowania częstotliwości odwołań do systemu, lecz TCP, IP oraz Ethernet narzucają swe ograniczenia na maksymalną wielkość pakietu; by pogodzić ze sobą te sprzeczności, konieczne jest dzielenie dużego pakietu na mniejsze pakiety przed wysłaniem i ponowne składanie oryginalnego pakietu w komputerze docelowym. To jedna z typowych funkcji wykonywanych przez procesory sieciowe.

(8) **Obliczenia.** Niekiedy zawartość pakietu jest materiałem wejściowym do skomplikowanych obliczeń, na przykład kompresji (dekompresji) lub szyfrowania (deszyfracji). To także jedna z typowych funkcji procesorów sieciowych.

(9) **Zarządzanie nagłówkami.** Obróbka pakietu często wiąże się z koniecznością „zdejmowania" nagłówków i ponownego ich „nakładania". Jest tak w sytuacji, gdy w ramach tej obróbki ulega modyfikacji treść komunikatu zawartego w pakiecie lub zawartość któregoś z pól nagłówka — przykładem takiego pola jest pole określające maksymalną dopuszczalną liczbę przeskoków, dekrementowane przy każdym przeskoku. To kolejna typowa funkcja procesorów sieciowych.

(10) **Zarządzanie kolejkami.** Nadchodzące pakiety często muszą być umieszczane w kolejkach w oczekiwaniu na przetworzenie i podobnie ma się rzecz z pakietami oczekującymi na wysłanie. Wiele aplikacji multimedialnych celowo wprowadza przerwy w wysyłaniu pakietów w celu zredukowania błędów opóźnień (*jitters*); zapory sieciowe i routery dokonują często dystrybucji otrzymywanych pakietów pomiędzy różne linie wejściowe, zgodnie z określonymi regułami. Wszystkie te zadania mogą być wykonywane przez procesory sieciowe.

(11) **Obliczanie sum kontrolnych.** Każdy wysyłany pakiet musi być opatrzony sumą kontrolną. Suma kontrolna IP może być obliczana przez procesor sieciowy, natomiast ethernetowy kod kontrolny CRC generowany jest przez sprzęt.

(12) **Ewidencjonowanie i rozliczanie.** W pewnych sytuacjach — na przykład w przypadku usług komercyjnych, gdy pakiety przesyłane są między różnymi sieciami — ewidencjonowana musi być ilość i rodzaj przesyłanych pakietów, w celach rozliczeniowych. Ewidencjonowanie to może być wykonywane przez procesory sieciowe.

(13) **Zbieranie statystyk.** Wiele firm, niezależnie od ewentualnych rozliczeń, kolekcjonuje rozmaite statystyki związane z generowanym ruchem sieciowym. Zadanie to może być powierzone procesorom sieciowym.

Zwiększanie wydajności

Wydajność — to kryterium numer 1 procesorów sieciowych; cóż jednak określenie to oznacza w praktyce i w jakich kategoriach można je mierzyć? Dobrą miarą może być liczba pakietów przetwarzanych w ciągu sekundy, a wcale nie gorszą — liczba bajtów, które w ciągu sekundy przepłynąć muszą przez procesor sieciowy. Istnieją różne miary, które okazują się adekwatne w przypadku małych pakietów i zawodzą w przypadku pakietów znacznie większych — i vice versa: przykładowo, zwiększenie częstotliwości analizowania linii wyjściowych w tablicach routingu może okazać się wielce pożyteczne w przypadku małych pakietów, lecz zupełnie nieodczuwalne w przypadku dużych.

Najbardziej oczywistym sposobem zwiększenia wydajności procesora sieciowego jest zwiększenie szybkości zegara. Ponieważ szybkość zegara nie jest jedynym czynnikiem wpływającym na wydajność — która jest silnie uzależniona także m.in. od długości cyklu pamięci — jest oczywiste, iż wydajność *nie jest liniową funkcją* szybkości zegara. Szybszy zegar to ponadto intensywniejsze wydzielanie ciepła, które trzeba skutecznie odprowadzać.

Dobre wyniki daje zwiększanie liczby rdzeni PPE, zwłaszcza w organizacji liniowej. Pomocne może okazać się także wydłużenie potoku, lecz tylko w tych zadaniach, które wyrazić można jako sekwencję prostych etapów.

Wydajność zyskuje generalnie wskutek przerzucania rozmaitych operacji z oprogramowania na specjalizowany sprzęt (na przykład układu ASIC), zwłaszcza w przypadku prostych, lecz często wykonywanych operacji jak przeglądanie tablic, obliczanie sum kontrolnych, szyfrowanie itp.

Wreszcie, potencjalna rezerwa tkwi w samych magistralach wewnętrznych — dodawanie nowych magistral i poszerzanie istniejących skutkuje szybszym przepływem pakietów. Wymiana części pamięci SDRAM na szybszą pamięć SRAM też jest krokiem w kierunku zwiększenia wydajności — oczywiście krokiem dość kosztownym.

Ograniczyliśmy się z konieczności do wybranych zagadnień związanych z procesorami sieciowymi; spośród bogatej literatury uzupełniającej możemy polecić Czytelnikom prace (Comer, 2005), (Crowley i in., 2002), (Lekkas, 2003) i (Papaefstathiou i in., 2004).

8.2.2. Procesory multimedialne

Drugim ważnym obszarem zastosowań koprocesorów są aplikacje przetwarzające grafikę wysokiej rozdzielczości oraz strumienie audio i wideo. W chwili obecnej procesory uniwersalne nie są w stanie w pełni sprostać wymogom efektywności, z jaką przetwarzanie to musi się odbywać, wskutek czego większość obecnych komputerów (i zapewne wszystkie komputery w przyszłości) wyposażana będzie w procesory multimedialne przejmujące na siebie znaczną część pracy, która bez ich obecności musiałaby być wykonywana w całości przez programowanie.

Wieloprocesor multimedialny Nexperia

Przyjrzyjmy się nieco bliżej konkretnemu przedstawicielowi coraz popularniejszej platformy procesorów multimedialnych, produkowanych dla różnych szybkości zegara — wieloprocesorowi **Nexperia**. Jest on jednoukładowym procesorem heterogenicznym, w sensie architektury przedstawionej na rysunku 8.9. Zawiera wiele rdzeni, z których

jednym jest — co ciekawe — procesor VLIW TriMedia, pełniący rolę procesora sterującego; towarzyszą mu liczne rdzenie przeznaczone do obróbki obrazów, audio, wideo i przetwarzania sieciowego. Nexperia może pełnić rolę samodzielnego procesora w odtwarzaczu (nagrywarce) CD, DVD lub MP3, telewizorze, kamerze wideo itp., lecz może być także elementem komputera PC jako koprocesor zajmujący się przetwarzaniem grafiki i strumieni multimedialnych. W obydwu przypadkach sterowany jest własnym systemem operacyjnym czasu rzeczywistego.

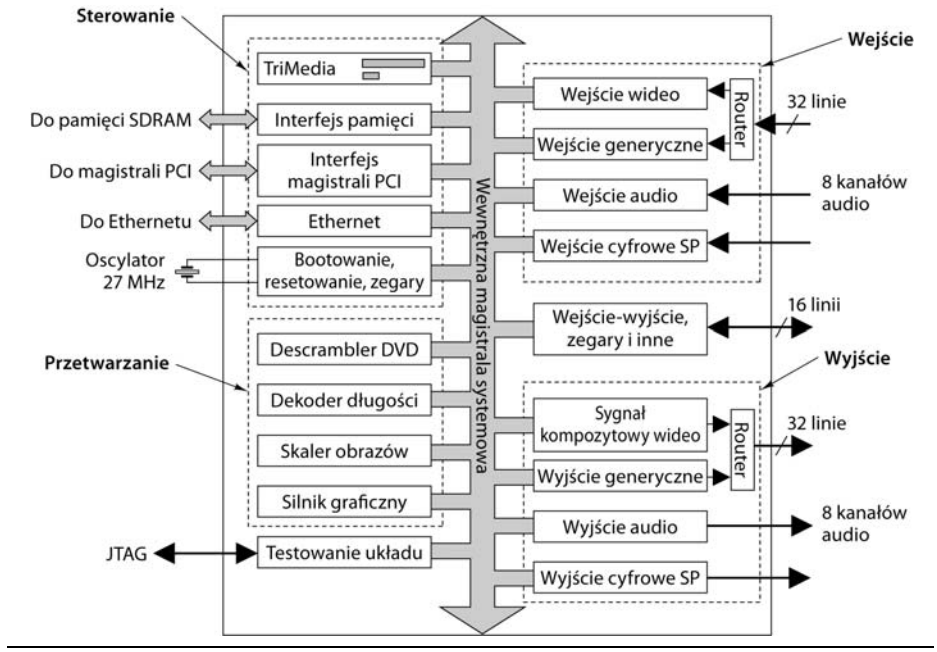
Wieloprocesor Nexperia wykonywać może zadania trojakiego rodzaju: przechwytywanie strumieni multimedialnych i konwertowanie ich do postaci struktur danych zapisywanych w pamięci operacyjnej, przetwarzanie tych struktur oraz tworzenie odpowiadających tym strukturom strumieni zdatnych do odtworzenia lub przetworzenia w rozmaitych urządzeniach. I tak na przykład, gdy komputer PC wykorzystywany jest w roli odtwarzacza DVD, Nexperia może być zaprogramowany do odczytu skompresowanego strumienia z dysku DVD, jego deszyfracji i dekompresji, i wreszcie przekształcenia do postaci umożliwiającej wyświetlenie w oknie aplikacji odtwarzającej. Wszystkie te czynności mogą być wykonywane w tle, bez angażowania procesora centralnego od momentu, gdy procesor centralny dokona odpowiedniego zaprogramowania swego koprocesora Nexperia.

Wszystkie nadchodzące dane magazynowane są w pamięci w celu dalszego przetwarzania — nie ma bezpośredniego połączenia urządzeń wejściowych z urządzeniami wyjściowymi. Przechwytywanie strumieni wejściowych polega na ich dekodowaniu z rozmaitych formatów i rozmiarów wideo (m.in. MPEG-1, MPEG-2 i MPEG-4) oraz formatów audio (m.in. AAC, Dolby i MP3) i konwertowaniu ich w struktury stanowiące materiał wyjściowy do dalszego przetwarzania. Owe strumienie wejściowe mogą pochodzić z magistrali PCI, Ethernetu lub dedykowanych kanałów, na przykład mikrofonu czy też kompletnego zestawu stereo przyłączonego do komputera, a dokładniej — bezpośrednio do koprocesora. Układ Nexperia posiada 456 wyprowadzeń („nóżek” — *pins*) — niektóre z nich przeznaczone są do bezpośredniego przyłączania do źródeł strumieni multimedialnych.

Przetwarzanie danych sterowane jest przez centralny procesor TriMedia, który zaprogramować można właściwie dowolnie. Najczęściej wykonywanymi zadaniami są usuwanie przeplotu (*deinterlacing*) w celu poprawienia czystości i wyrazistości obrazu, korekcja jasności, kontrastu i kolorów, skalowanie rozmiarów obrazu, konwersja między różnymi formatami wideo i redukowanie szumów. Procesor centralny działa zwykle jako centralny koordynator, rozdzielający większość pracy pomiędzy specjalizowane rdzenie.

Funkcjonalność układu Nexperia jako procesora wyjściowego obejmuje m.in. kodowanie struktur danych do postaci odpowiedniej dla określonych urządzeń zewnętrznych, łączenie danych z kilku źródeł (wideo, audio, obrazy, grafika 2D) i sterowanie urządzeniami zewnętrznymi. Podobnie jak w przypadku strumieni wejściowych, dane wyjściowe mogą być kierowane na magistralę PCI, do Ethernetu lub na dedykowane linie wyjściowe (na przykład do głośników lub wzmacniaczy).

Na rysunku 8.14 widoczny jest schemat blokowy układu Nexperia PNX 1500; inne modele tego układu różnią się nieznacznie między sobą, każdorazowo więc, gdy w dalszym ciągu rozdziału używać będziemy terminu „Nexperia”, będziemy mieć na myśli jego model PNX 1500. Model ten składa się z czterech zasadniczych sekcji: sterującej, wejściowej, przetwarzającej i wyjściowej. Jego procesorem sterującym jest 32-bitowy procesor VLIW TriMedia, omawiany w punkcie 8.1.1, taktowany zegarem 300 MHz. Wykonywany przez niego program, napisany zwykle w języku C lub C++, determinuje całą funkcjonalność układu.



RYSUNEK 8.14. Heterogeniczny wieloprocesor jednoukładowy Nexperia

Nexperia nie posiada własnej pamięci, jeśli nie liczyć dwóch pamięci cache wewnątrz procesora TriMedia. W zamian posiada interfejs pamięci zewnętrznej, umożliwiający podłączenie od 8 do 256 MB pamięci DDR SDRAM, co dla zastosowań multimedialnych jest w zupełności wystarczające. Przy częstotliwości zegara 200 MHz dane wymieniane są z pamięcią z szybkością 1,6 GB/s.

Nexperia posiada także interfejs magistrali PCI, zapewniający 8-, 16- lub 32-bitowy transfer z częstotliwością 33 MHz. Gdy wykorzystywany jest jako samodzielny procesor w urządzeniu zewnętrznym (na przykład odtwarzaczu DVD), interfejs ten może służyć jako arbiter magistrali, zapewniający kontakt z napędem DVD.

Dedykowany rdzeń zapewnia także bezpośrednie połączenie z Ethernetem, z szybkością 10 Mb/s lub 100 Mb/s. Dzięki temu zbudowany na bazie Nexperia kamkoder umożliwia wysyłanie przez Ethernet cyfrowego sygnału wideo, do urządzenia przechwytującego lub telewizora.

Kolejny rdzeń umożliwia bootowanie i resetowanie, dostarcza sygnałów zegarowych i wykonuje kilka jeszcze pomniejszych funkcji. Podanie sygnału „1” na określoną nóżkę układu powoduje jego zresetowanie. Rdzeń ten może także zostać zaprogramowany jako wykrywacz bezczynności (*dead man's switch*) — jeśli procesor centralny komputera nie odpowiada na sygnały testujące (*ping*) przez określony czas, wspomniany rdzeń interpretuje to jako zawieszenie systemu i powoduje jego „przeładowanie” (*reboot*). W urządzeniach samodzielnych przeładowywanie może zostać przeprowadzone w oparciu o pamięć *flash*.

Pisany rdzeń taktowany jest wytwarzanym przez oscylator sygnałem o częstotliwości 27 MHz, który pomnożony przez 64 daje częstotliwość efektywną 1,728 GHz sygnału taktującego cały układ. Przez zmianę wymienionego mnożnika można redukować zużycie energii przez układ. Normalnie procesor centralny pracuje z pełną częstotliwością, zaś poszczególne rdzenie — z częstotliwością umożliwiającą wykonywanie

przypisanych im funkcji. Możliwe jest jednak zredukowanie częstotliwości procesora centralnego oraz wprowadzenie układu w tryb oszczędzania energii (*sleep mode*) w którym wyłączona jest większość jego funkcji. Ma to szczególne znaczenie w przypadku urządzeń mobilnych zasilanych z baterii.

W opisywanym rdzeniu znajduje się także 16 „semaforów", które można wykorzystywać do synchronizacji współpracy urządzeń. Gdy wartość semafora wynosi zero, próba jej zmiany na niezerową powiedzie się; do semafora z niezerową wartością można wpisać jedynie wartość zero — próba wpisania niezerowej wartości jest odrzucana i wartość semafora nie zmienia się. Ponieważ w danej chwili tylko jeden rdzeń ma dostęp do magistrali wewnętrznej, więc próba zapisu i sprawdzenie wartości semafora mogą być wykonane jako sekwencja niepodzielna (atomowa) co stanowi narzędzie wystarczające do realizacji wzajemnego wykluczania. Jeżeli rdzeń zamierza uzyskać dostęp do określonego zasobu chronionego przez semafor, próbuje wpisać do semafora charakterystyczną dla siebie niezerową wartość, po czym sprawdza (testując wartość semafora), czy próba ta się udała; jeśli tak, może rozpocząć wykonywanie operacji na zasobie, a po ich zakończeniu wpisać do semafora wartość zero. Jeśli opisana próba rezerwacji zasobu nie powiedzie się, rdzeń powinien ją cyklicznie ponawiać aż do skutku. Zauważmy, że przy nieudanej próbie rezerwacji proces rdzenia **nie jest zawieszany** — opisane „semafory" nie są więc typowymi semaforami, o których pisaliśmy w rozdziale 6.

Spójrzmy teraz na sekcję wejściową. Rdzeń wejścia wideo odbiera 10-bitowy sygnał wideo i za pomocą specjalnego algorytmu „wygładzającego" konwertuje go na sygnał 8-bitowy i zapisuje w zewnętrznej pamięci SDRAM. W większości przypadków ów 10-bitowy sygnał jest sygnałem wyjściowym z konwertera analogowo-cyfrowego, przetwarzającego na postać cyfrową analogowy sygnał telewizyjny, pochodzący z anteny naziemnej lub z kabla.

Rdzeń wejścia generycznego jest w stanie przyjmować niestrukturalne sygnały 32-bitowe z dowolnego źródła danych, z częstotliwością do 100 MHz, i zapisywać te sygnały w zewnętrznej pamięci SDRAM. Może on także przechwytywać sygnały strukturalne, czyli zawierające sygnały wyznaczające granice rekordów. Poprzedzający obydwa wejścia router wykonuje demultipleksowanie sygnałów wejściowych i może także dokonywać „w locie" niektórych transformacji wideo. Demultipleksowanie to jest konieczne, ponieważ te same „nóżki" układu używane są do przyłączania zarówno wejścia wideo, jak i wejścia generycznego.

Rdzeń wejścia audio zdolny jest przyjmować sygnały muzyki stereo lub głosu z maksymalnie ośmiu kanałów z 8-, 16- lub 32-bitową precyzją i częstotliwością do 96 kHz i zapisywać te sygnały w pamięci SDRAM. Może ona także dekodować dane ze skompresowanych formatów, miksować kanały, zmieniać częstotliwość próbkowania (*sampling rate*) sygnałów i filtrować dane — wszystko to wykonywane jest „w locie", przed zapisaniem danych w pamięci SDRAM.

Wejście cyfrowe SP przeznaczone jest dla sygnałów audio spełniających standard dźwięku cyfrowego Sony-Philips (IEC 1937). Sygnały takie mogą być przenoszone między urządzeniami bez utraty jakości.

Wprowadzone do układu sygnały podlegają naturalną koleją rzeczy przetwarzaniu, co jest domeną następnej sekcji. Filmy DVD podlegające licencjonowaniu (sprzedawane lub wypożyczane) są zaszyfrowane w celu ochrony przed kopiowaniem. Układ descramblera rozszyfrowuje zakodowany strumień do postaci oryginalnego strumienia MPEG-2. Rozszyfrowywanie to odbywa się całkowicie w pamięci: zakodowany strumień wejściowy znajduje się w jednym buforze, a wynik deszyfracji zapisywany jest w drugim.

Istnienie rdzenia **dekodera długości** ma związek ze specyfiką algorytmu kodowania MPEG. Teoretycznie rzecz biorąc, cały proces dekodowania mógłby zostać wykonany programowo przez procesor TriMedia. Projektanci uznali jednak, iż procesor ten nie jest zbyt dobrze przystosowany do dwóch początkowych etapów tego dekodowania: dekompresji Huffmana i tzw. dekompresji długości serii (RLE, *Run-Length Encoding*), i postanowili powierzyć te dwie operacje specjalizowanemu rdzeniowi, uzyskując w ten sposób znaczny przyrost wydajności kosztem zaledwie dodatkowych kilku milimetrów kwadratowych powierzchni układu. Będący produktem tej wstępnej dekompresji strumień jest następnie ostatecznie dekompresowany przez procesor TriMedia do postaci zwykłej mapy pikseli.

Owa mapa pikseli może przyjmować jeden z trzech zasadniczych formatów, z których każdy występuje w czterech wariantach różniących się rozmiarem i parametrami. Pierwszym ze wspomnianych formatów jest format **koloru indeksowanego** (*indexed color*), w którym każdy kolor wyrażony jest jako indeks do specjalnej tablicy (**CLUT**, **Color LookUp Table**), zawierającej 24-bitowe kody kolorów oraz tzw. 8-bitową **maskę kanału alfa** (*alpha channel mask*) decydującą o przezroczystości pikseli w przypadku nakładania się na siebie kilku warstw obrazu. W drugim formacie — **RGB** (*Red-Green-Blue*), wykorzystywanym w monitorach komputerowych, kolor każdego piksela kodowany jest w postaci trzech niezależnych, 8-bitowych składowych kolorów czerwonego, zielonego i niebieskiego; trzeci format — **YUV** — zaprojektowany na potrzeby telewizji, specyfikuje każdy piksel w postaci trzech składowych: luminancji oraz dwóch wartości chrominancji; w sygnale tym więcej pasma przydzielane jest dla luminancji niż dla chrominancji[10], co czyni sygnał bardziej odpornym na zakłócenia w czasie transmisji. Format ten doskonale nadaje się do wszelkich aplikacji przetwarzających sygnały telewizyjne. Dzięki ograniczeniu formatu przechowywania bitmapy do niektórych tylko opcji, każdy rdzeń zdolny jest odczytywać sygnały produkowane przez inne rdzenie.

Skaler obrazów odczytuje listę zadań skalujących i kolejno wykonuje te zadania, z szybkością dochodzącą do 120 milionów pikseli na sekundę. Zadania te obejmują między innymi:

- usuwanie przeplotu (*deinterlacing*),
- skalowanie poziome i pionowe,
- liniową i nieliniową konwersję proporcji obrazu (*aspect ratio*),
- konwersję między różnymi formatami pikseli,
- tworzenie kolekcji histogramów luminancji,
- redukcję migotania.

Wysyłane przez stację nadawczą sygnały telewizyjne są sygnałami **z przeplotem** (*interlace*) co oznacza, że dla każdej ramki, składającej się z 525 linii (w systemie PAL i SECAM — z 625 linii) najpierw transmitowane są linie o numerach parzystych, a następnie linie o numerach nieparzystych. Likwidacja przeplotu daje w rezultacie sygnał **ze skanowaniem progresywnym** (*progressive scan*), w którym linie wyświetlane są w naturalnej kolejności, a odświeżanie obrazu odbywa się dwukrotnie częściej niż przy przeplocie (29,97 ramek na sekundę w systemie NTSC i 25 ramek na sekundę w systemach PAL i SECAM). Skalowanie obrazu (poziome lub pionowe)

[10] Ze względu na większą wrażliwość oka na zmiany luminancji niż na zmiany chrominancji — *przyp. tłum.*

polega na zwiększaniu lub zmniejszaniu (poziomego lub pionowego) rozmiaru obrazu, powstającego zazwyczaj w wyniku wycinania lub obcinania. Proporcje obrazu tradycyjnej telewizji wynoszą 4:3, zaś telewizji szerokoekranowej 16:9 — te ostatnie bliższe są proporcji 3:2 charakterystycznej dla filmów na taśmie 35 mm. Skaler umożliwia przeprowadzanie konwersji między tymi proporcjami, zarówno według algorytmów liniowych, jak i nieliniowych. Oferuje on także konwersję między różnymi formatami bitmap — indeksowym, RGB i YUV — oraz kolekcjonowanie histogramów luminancji, pomocnych w polepszaniu jakości produkowanego obrazu. Wreszcie, za pomocą specjalnych transformacji sygnału możliwe jest zredukowanie migotania obrazu.

Silnik graficzny wykonuje dwuwymiarowe renderowanie obrazu na podstawie opisu obiektów, rozpoznaje on także granice zamkniętych konturów w celu ich wypełniania, jak również wykonuje operacje graficzne typu **bitblt**, polegające na składaniu kolorów dwóch prostokątnych bitmap według różnych funkcji boolowskich[11], m.in. `AND`, `OR` i `XOR`.

Nexperia *nie* posiada specjalizowanych rdzeni dla przetwarzania sygnałów audio — są one przetwarzane bezpośrednio na wejściu oraz w sposób programowy przez procesor TriMedia; ze względu na niewielką (stosunkowo) ilość danych związanych z sygnałami audio, ich programowa obróbka nie stanowi żadnego problemu, poza tym wiele aplikacji w ogóle nie wymaga przetwarzania tych sygnałów z wyjątkiem być może ich konwersji między różnymi formatami.

Rdzeń **testujący** pomaga projektantom i programistom w testowaniu poprawności funkcjonowania sprzętu i oprogramowania. Udostępnia on interfejs do narzędzi i przyrządów systemu **JTAG** (**Joint Test Action Group**) zdefiniowanego w standardzie IEEE 1149.1.

W sekcji wyjściowej dane zgromadzone w pamięci przetwarzane są na sygnały wyjściowe. Rdzeń **sygnału kompozytowego** dokonuje normalizacji pikseli reprezentowanych w strukturach danych; dla bitmap zapisanych w formacie indeksowym ustalane są wartości poszczególnych pikseli, a niekompatybilne formaty konwertowane są według potrzeby. W rdzeniu tym przeprowadzana jest także niezbędna korekcja jasności, kontrastu i koloru, jak również realizacja tzw. **kluczowania kolorów** (*chroma keying*) którego przykładem jest popularny *bluebox*: aktor lub spiker filmowany jest na niebieskim tle i scena ta nakładana jest na inną pochodzącą z drugiego źródła. W miejscu, gdzie w pierwszej scenie występują niebieskie piksele tła, widoczne są w rzeczywistości piksele z drugiej sceny. W ten oto sposób spiker może „znaleźć się" w plenerze, nie wychodząc w ogóle ze studia. Na podobnej zasadzie uzyskiwany jest efekt nakładania ruchomych postaci na statyczne lub przewijane („scrollowane") tło — efekt często spotykany w grach i filmach rysunkowych. Ostatnim etapem obróbki sygnału jest jego konwersja do postaci wymaganej przez konkretny standard telewizyjny (NTSC, PAL lub SECAM) wraz z impulsami synchronizacji pionowej i poziomej.

Ze względu na umiarkowaną cenę układu Nexperia, oczekuje się, że większość urządzeń, w których będzie on montowany, zdolna będzie do automatycznego dostosowywanie się do lokalnego standardu sygnału, dzięki czemu możliwe będzie używanie takich samych egzemplarzy telewizorów, odtwarzaczy itp. na całym świecie. Z analogicznych powodów dodanie nowego formatu, jakim jest **telewizja o dużej rozdzielczości** (**HDTV**, **High Definition TeleVision**) wymagać będzie tylko niewielkich zmian w oprogramowaniu w związku z dodatkowymi transformacjami danych zapisywanych w pamięci.

[11] Obliczanych niezależnie dla każdej pary odpowiadających sobie bitów dwóch pikseli — *przyp. tłum.*

Rdzeń **wyjścia generycznego** dokonuje przesyłania 8-, 16- i 32-bitowych danych z częstotliwością 100 MHz, co daje maksymalną szybkość przesyłu 3,2 Gb/s. Podłączając generyczne wyjście jednego układu Nexperia do generycznego wejścia drugiego, wykonywać można transfer plików z prędkością przekraczającą prędkość „gigabitowego Ethernetu" (1 Gb/s). Na generycznym wyjściu produkować można programowo dowolne sygnały, stosownie do specyficznych zastosowań.

Router wyjściowy wykonuje multipleksowanie danych z wyjścia generycznego i wyjścia kompozytowego, wykonując przy okazji pewne operacje pomocnicze, jak odświeżanie panelu wyświetlacza TFT o rozmiarach do 1280 × 768 pikseli z częstotliwością 60 Hz lub odświeżanie obrazu telewizyjnego wyświetlanego w trybie progresywnym lub z przeplotem. Multipleksowanie to konieczne jest z tego względu, że wejściu kompozytowemu i generycznemu przyporządkowane są te same nóżki układu.

Rdzeń **wyjścia audio** może produkować sygnał dźwiękowy stereo (do 8 kanałów) z precyzją 32 bitów i częstotliwością próbkowania do 96 kHz. Zazwyczaj sygnał ten przyłączany jest do wejścia konwertera cyfrowo-analogowego. Sygnał z **cyfrowego wyjścia SP** może być podłączany do wejścia SP urządzeń obsługujących cyfrowy standard Sony-Philips.

Ostatni z omawianych rdzeni (nieprzyporządkowany do żadnej sekcji) zarządza operacjami wejścia-wyjścia o charakterze ogólnym. 16 dostępnych nóżek może być wykorzystywanych w dowolny sposób — przykładowo, można je łączyć z przyciskami, przełącznikami lub diodami LED w celu programowego sterowania nimi, a nawet można je wykorzystywać w charakterze realizowanego programowo protokołu sieciowego o umiarkowanej prędkości (20 Mb/s). Rozmaite zegary, liczniki i procedury obsługi zdarzeń także są implementowane w tym rdzeniu.

Reasumując, wieloprocesor Nexperia dostarczyć może olbrzymiej mocy obliczeniowej aplikacjom multimedialnym, odciążając w ten sposób (podobnie jak procesory sieciowe) procesor centralny komputera. W rzeczywistości wspomniana moc obliczeniowa może okazać się większa, niż może się to wydawać w pierwszej chwili, bowiem poszczególne rdzenie mogą funkcjonować niezależnie od siebie i równolegle z procesorem sterującym TriMedia. To wszystko uzyskuje się za stosunkowo niską cenę — jednostkowa cena układu sprzedawanego w ilościach hurtowych wynosi zaledwie kilka dolarów. Daje to wystarczające wyobrażenie o roli i użyteczności koprocesorów we współczesnych systemach komputerowych i innych urządzeniach elektronicznych.

Czytelnikom zainteresowanym wykorzystaniem analogicznych koprocesorów w telefonii polecamy pracę (Nickolls i in., 2003).

8.2.3. Kryptoprocesory

Trzecim obszarem, w którym koprocesory znalazły powszechne zastosowanie, jest bezpieczeństwo, zwłaszcza bezpieczeństwo sieci komputerowych. Gdy nawiązywane jest połączenie między klientem a serwerem, zazwyczaj wymaga ono obustronnego uwierzytelnienia; uwierzytelnienie to musi jednak zostać dokonane w ramach połączenia — koło się zatem zamyka, chyba że połączenie uwierzytelniające będzie połączeniem bezpiecznym, szyfrowanym, czyniącym bezowocnymi ewentualne próby podsłuchu.

Podstawowym środkiem zapewnienia bezpieczeństwa połączenia jest jego szyfrowanie. Wymaga ono zastosowania technik kryptograficznych, te natomiast wymagają znacznych ilości obliczeń. Zależnie od wzajemnej relacji między kluczem szyfrującym a deszyfrującym, wszystkie techniki kryptograficzne podzielić można na dwie grupy: kryptografię z **kluczami symetrycznymi** oraz kryptografię z **kluczami publicznymi**. Techniki należące do pierwszej z wymienionych grup polegają na intensywnym żonglowaniu bitami, prowadzącym do starannego ich wymieszania; istotą technik drugiej grupy jest wykonywanie złożonych operacji mnożenia i potęgowania dużych (na przykład 1024-bitowych liczb), co oczywiście pochłania mnóstwo mocy obliczeniowej.

By sprostać wymaganiom w zakresie tej mocy, wielu wytwórców opracowało różnorodne koprocesory kryptograficzne, najczęściej w formie kart PCI, wykonujących niezbędne obliczenia znacznie wydajniej, niż zdolne byłyby to uczynić procesory ogólnego przeznaczenia. Pominiemy szczegółową dyskusję na temat tych koprocesorów z jednego zasadniczego względu: wymagałaby ona uprzedniego wprowadzenia w matematyczne podstawy kryptografii, to zaś wykraczałoby poza tematykę niniejszej książki. Zainteresowanych Czytelników odsyłamy do prac (Daneshbeh i Hasan, 2004) oraz (Lutz i Hasan, 2004).

8.3. WIELOPROCESORY ZE WSPÓŁDZIELONĄ PAMIĘCIĄ

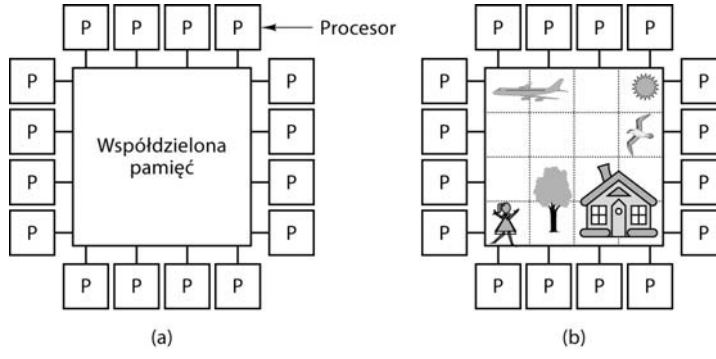
Posuwamy się konsekwentnie na zewnątrz — po zrównolegleniu wewnątrz układu i koprocesorach wspomagających pora teraz na systemy złożone z kompletnych procesorów odpowiednio ze sobą połączonych. Systemy takie podzielić można na dwie grupy: wieloprocesory i wielokomputery. Po przedstawieniu podstawowych kryteriów tego podziału zajmiemy się bardziej szczegółowo obydwiema tymi kategoriami.

8.3.1. Wieloprocesory a wielokomputery

Niezależnie od sposobu połączenia, procesory tworzące system muszą się ze sobą w jakiś sposób komunikować. Konkretny sposób tego komunikowania jest jednym z kluczowych zagadnień projektowych, a konkretne pomysły w tym zakresie i zaimplementowane rozwiązania zaliczyć można do dwóch grup — wieloprocesorów albo wielokomputerów. Podstawowym kryterium wyraźnie wyznaczającym ten podział jest obecność (albo nieobecność) współdzielonej pamięci, przesądzającej równocześnie o sposobie projektowania, budowania i programowania systemów obydwu grup, jak również o ich skali i cenie.

Wieloprocesory

Komputer równoległy, w którym wszystkie procesory współdzielą tę samą pamięć — co schematycznie przedstawiono na rysunku 8.15 — nazywamy **wieloprocesorem**.



RYSUNEK 8.15. Zasada funkcjonowania wieloprocesora: (a) wieloprocesor złożony z 16 procesorów współdzielących pamięć operacyjną; (b) obraz podzielony na 16 fragmentów, z których każdy analizowany jest przez inny procesor

Wszystkie procesy pracujące na takim komputerze współdzielą tę samą przestrzeń adresową[12]. Każdy z procesorów może bez ograniczeń odczytywać i zapisywać słowa w tej pamięci za pomocą zwykłych rozkazów w stylu LOAD i STORE — nic więcej nie jest potrzebne, wszystkie wynikające stąd problemy rozstrzygane są automatycznie przez sprzęt. Dzięki temu dwa procesy mogą się ze sobą komunikować w najbardziej elementarny sposób, zapisując i odczytując dane w uzgodnionym obszarze wspólnej pamięci operacyjnej.

Właśnie owa łatwość komunikacji międzyprocesowej jest podstawowym czynnikiem przesądzającym o dużej popularności wieloprocesorów. Dla programistów stanowią one prosty model pojęciowy, nadający się znakomicie do rozwiązywania sporej gamy problemów. W charakterze przykładowego zastosowania rozpatrzmy program analizujący podaną bitmapę w celu wyodrębnienia reprezentowanych na niej obiektów. Jak pokazano to w części (b) rysunku 8.15, w pamięci operacyjnej znajduje się pojedynczy egzemplarz bitmapy, podzielony na 16 fragmentów, z których każdy przydzielony jest do analizy innemu procesorowi; każdy procesor, niezależnie od powierzonego mu fragmentu, ma jednak dostęp do całej bitmapy. Nie byłoby w tym nic nadzwyczajnego, gdyby nie fakt, że poszczególne obiekty mogą przekraczać granice wyznaczonych fragmentów i potencjalnie każdy z procesorów może być zmuszony do wejścia na teren fragmentu powierzonemu partnerowi. W związku z tym konieczne jest zapewnienie jakiejś synchronizacji między procesami — ten sam obiekt może być bowiem wykryty przez dwa różne procesory i w sytuacji przedstawionej na rysunku 8.15 (b) konieczne jest zapobieżenie wykryciu np. dwóch samolotów (lub czterech domków) na bitmapie.

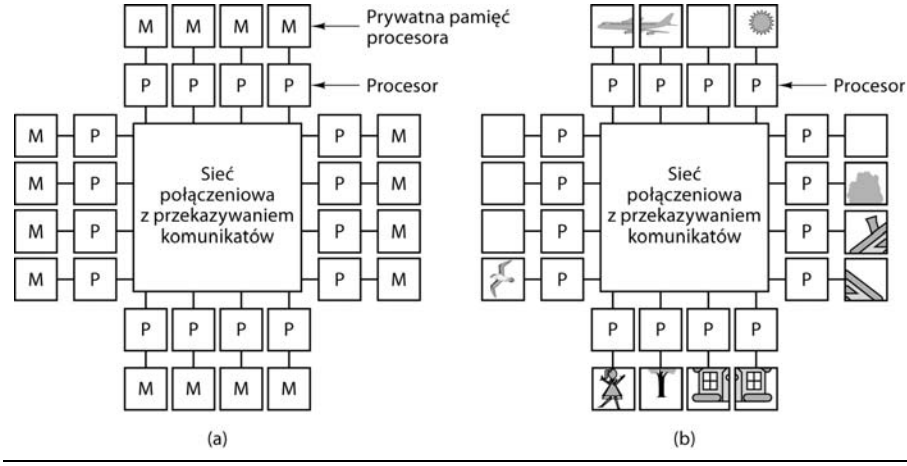
Ponieważ wszystkie procesory wieloprocesora postrzegają wspólną pamięć w identyczny sposób, ich praca sterowana jest przez pojedynczą kopię systemu operacyjnego, pojedyncza jest też tablica stron i tablica procesów. Gdy któryś z procesów zostaje zablokowany — w oczekiwaniu na wykonanie żądanej operacji — system operacyjny zapisuje w swej tablicy stan tego procesu i wybiera do wykonywania inny gotowy proces. Cecha ta odróżnia wieloprocesory od wielokomputerów, w których każdy procesor sterowany jest własną kopią systemu operacyjnego.

[12] Mowa o fizycznej przestrzeni adresowej, czyli o fizycznych adresach pamięci operacyjnej — *przyp. tłum.*

Wieloprocesor, jak każdy komputer, musi posiadać urządzenia wejścia-wyjścia — dyski, karty sieciowe itp.; w niektórych rozwiązaniach dostęp do tych urządzeń mają tylko niektóre procesory, wyposażone w tym celu w specjalne funkcje, w innych możliwy jest równouprawniony dostęp wszystkich procesorów do każdego z urządzeń. System, w którym każdy z procesorów ma równy dostęp do wszystkich modułów pamięci i wszystkich urządzeń wejścia-wyjścia oraz postrzegany jest przez system operacyjny na równi z innymi procesorami, nazywamy **wieloprocesorem symetrycznym** (**SMP**, **Symmetric MultiProcessor**).

Wielokomputery

System równoległy, w którym każdy procesor wyposażony jest we własną pamięć operacyjną, niedostępną dla innych procesorów, nazywamy **wielokomputerem** lub (rzadziej) **systemem z pamięcią rozproszoną** (*distributed memory system*); ta właśnie cecha jest podstawowym kryterium odróżniającym wielokomputery od wieloprocesorów. Każdy procesor wielokomputera, wykonując rozkazy LOAD i STORE, operować może tylko na swej prywatnej pamięci, pamięci innych procesorów są dla tych rozkazów po prostu niedostępne. Tym samym każdy procesor dysponuje własną fizyczną przestrzenią adresową. Schemat przykładowego wielokomputera przedstawiono na rysunku 8.16.



RYSUNEK 8.16. Przykładowy wielokomputer: (a) 16 procesorów składowych, z których każdy posiada prywatną pamięć; (b) bitmapa z rysunku 8.15 podzielona między 16 pamięci

Ponieważ ze względu na rozłączność pamięci procesy realizowane przez różne procesory nie mogą już komunikować się ze sobą za pośrednictwem rozkazów LOAD i STORE, konieczne jest zaimplementowanie innych mechanizmów komunikacji między nimi — rolę tę spełnia sieć połączeniowa, za pomocą której procesory mogą wymieniać komunikaty między sobą (*message-passing interconnection network*). Najbardziej znanymi przykładami wielokomputerów są BlueGene/L firmy IBM, Red Storm i klaster wyszukiwarki Google.

Nieobecność współdzielonej pamięci ma istotne implikacje pod względem sposobu tworzenia oprogramowania dla wielokomputerów. Jak widać na rysunku 8.16 (b), analizowana bitmapa jest teraz podzielona między 16 niezależnych pamięci i każdy procesor ma bezpośredni dostęp wyłącznie do „swojego" kawałka tej bitmapy. Gdy odkryje on, że analizowany przez niego obiekt wykracza poza granice owego kawałka, nie może tak po prostu (za pomocą operacji `LOAD`) odczytywać brakującej części — konieczne jest zapewnienie zupełnie innej metody dostępu.

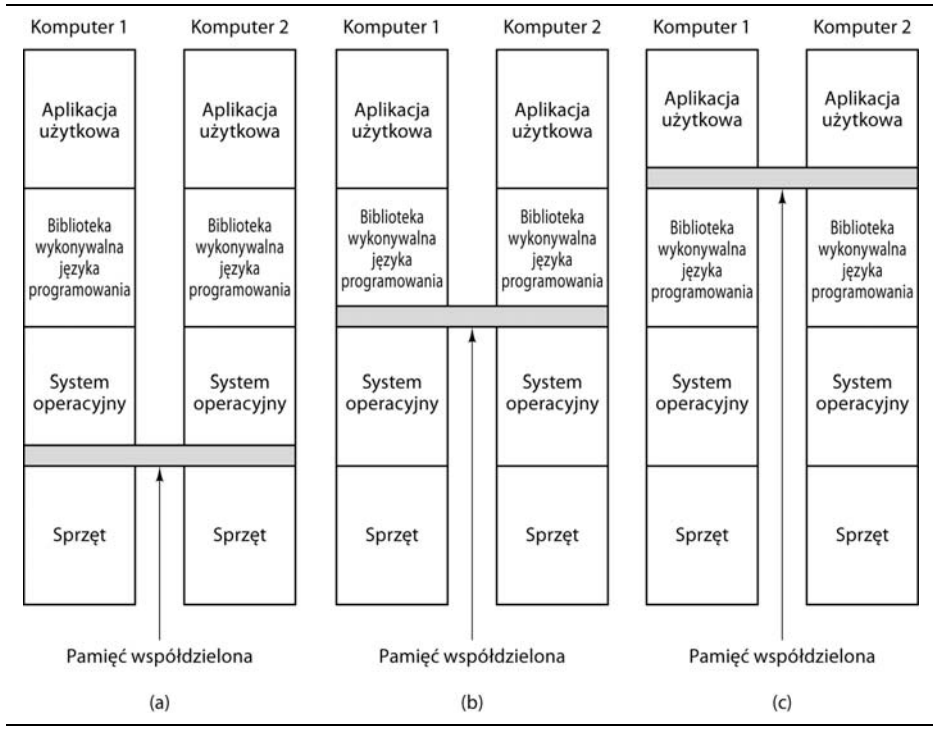
W szczególności procesor ten może zidentyfikować partnera, do którego należy pamięć zawierająca brakującą cześć obiektu, i wysłać do tego partnera komunikat z żądaniem udostępnienia kopii zawartości jego pamięci. Na czas oczekiwania na tę kopię procesor wchodzi w stan zawieszenia, a po jej otrzymaniu może normalnie kontynuować swą pracę.

W ten oto sposób, zamiast elementarnych operacji sprzętowych w rodzaju `LOAD` i `STORE` (jak w przypadku wieloprocesorów), konieczne jest zastosowanie programowych operacji elementarnych w rodzaju `send` (wyślij) i `receive` (odbierz). Komplikuje to tworzenie oprogramowania, a także czyni bardziej istotnym zagadnienie optymalnego podziału danych między procesory. W przypadku wieloprocesora podział ten mógł zostać dokonany w zasadzie dowolnie, bo i tak każdy procesor miał dostęp do całej pamięci, a jedyną konsekwencją różnych podziałów mogłaby być co najwyżej różnica w efektywności wykonywania programu. W przypadku wielokomputera konkretny podział danych związany jest ściśle ze strukturą przetwarzającego te dane oprogramowania — to kolejny dowód na to, że programowanie wielokomputerów jest znacznie trudniejsze od programowania wieloprocesorów.

Dlaczego więc w ogóle rozważa się budowę wielokomputerów, skoro wieloprocesory są dla programistów wygodniejsze? Otóż duże zespoły połączonych procesorów łatwiej jest realizować w postaci wielokomputerów niż w postaci wieloprocesorów — wielokomputer jest wówczas prostszy konstrukcyjnie i tańszy od wieloprocesora złożonego z tej samej liczby procesorów. Zapewnienie efektywnego współdzielenia pojedynczej pamięci przez **kilkaset** procesorów jest już nie lada problemem, podczas gdy wielokomputery złożone z **dziesięciu tysięcy** procesorów są czymś naturalnym. W dalszej części rozdziału przedstawimy wielokomputer, którego liczba procesorów sięga 50 000.

Projektanci komputerów równoległych stają więc przed trudnym wyborem, między wieloprocesorami — trudnymi w konstruowaniu, lecz łatwymi w programowaniu — i wielokomputerami, cechującymi się zależnością wręcz odwrotną. Optymalnych rozwiązań poszukuje się zwykle między skrajnościami i nie inaczej jest w tym przypadku: analiza sposobów, w jakie procesory mogą współdzielić pamięć między sobą i charakterystycznych dla każdego sposobu zalet i kłopotów doprowadziła do powstania rozwiązań hybrydowych, wykorzystujących zalety obydwu kategorii — wieloprocesorów i wielokomputerów. Bardzo istotnym kryterium oceny tych rozwiązań jest ich **skalowalność**, czyli zapewnienie coraz wydajniejszej pracy w miarę zwiększania liczby procesorów składowych.

Jedno z takich rozwiązań hybrydowych zasadza się koncepcyjnie na oczywistym fakcie — odzwierciedlonym zresztą w tytule niniejszej książki — że komputery nie są monolitami, lecz maszynami o logicznej strukturze wielopoziomowej (wielowarstwowej). Koncepcja ta otwiera drogę do implementowania pamięci współdzielonej na sposób warstwowy, jak zilustrowano to na rysunku 8.17. W części (a) widzimy typową architekturę wieloprocesora, czyli współdzielenie pamięci na poziomie sprzętu; wszystkie istotne składniki architektury — m.in. system operacyjny wraz ze swymi tablicami, w tym tablicą przydziału pamięci — występują w jednym egzemplarzu.



RYSUNEK 8.17. Rozmaite sposoby warstwowego symulowania pamięci współdzielonej: (a) czysto sprzętowy; (b) na poziomie systemu operacyjnego; (c) w zakresie aplikacji użytkownika

Proces potrzebujący nowego obszaru pamięci zwraca się z żądaniem jej przydziału do systemu operacyjnego, ten zaś po dokonaniu przydziału uaktualnia wspomnianą tablicę alokacji. Z perspektywy systemu operacyjnego w pojedynczą pamięć fizyczną mapowane są wirtualne przestrzenie adresowe poszczególnych procesów. Jak zobaczymy w dalszym ciągu rozdziału, sprzętowe współdzielenie pamięci może być zrealizowane na wiele sposobów.

Alternatywne podejście polega na wykorzystaniu sprzętu wielokomputera do zasymulowania wirtualnej pamięci współdzielonej przez **system operacyjny**. W rozwiązaniu tym, zwanym **rozproszoną pamięcią współdzieloną** (**DSM**, **Distributed Shared Memory**) (Li i Hudak, 1989), każdy z procesorów posiada odrębną wirtualną przestrzeń adresową, a strony tej pamięci mogą być dowolnie mapowane w ramki każdej z prywatnych pamięci procesorów. Gdy dany procesor odwołuje się do strony mapowanej w „nie swoją" pamięć fizyczną (wykonując na przykład rozkaz `LOAD`), system operacyjny może **w sposób całkowicie przezroczysty dla programu** udostępnić kopię stosownej ramki z pamięci odnośnego partnera. Przypomina to do złudzenia mechanizm stronicowania na żądanie, z tą jednak istotną różnicą, że (z perspektywy danego procesora) strony cieniowe jego stron wirtualnych znajdują się nie w pamięci dyskowej, lecz w fizycznej (prywatnej) pamięci operacyjnej któregoś z jego partnerów. Z punktu widzenia programu użytkownika daje to złudzenie dużej pamięci współdzielonej. Opisaną koncepcję zilustrowano w części (b) rysunku 8.17.

Symulację współdzielenia pamięci można także przeprowadzić na poziomie **aplikacji użytkowej**, a dokładnie — na poziomie biblioteki wykonywalnej języka, w którym aplikacja ta została stworzona. Niektóre języki programowania oferują pewne abstrakcje pamięci współdzielonej, implementowane wewnętrznie przez własne biblioteki wykonywalne. Przykładowy przedstawiciel tej kategorii — język Linda — oferuje taką abstrakcję w postaci **krotek** (*tuples*), czyli rekordów stanowiących kolekcje pól. Procesy mogą odczytywać krotki ze współdzielonej przestrzeni krotek i zapisywać je w tej przestrzeni. Ponieważ zarządzanie przestrzenią krotek wykonywane jest całkowicie przez sam język, żadne wsparcie w tym zakresie ze strony sprzętu ani systemu operacyjnego nie jest konieczne.

Innym przykładem języka symulującego we własnym zakresie współdzieloną pamięć jest język Orca; elementem jego współdzielonej przestrzeni są nie krotki, ale generyczne **obiekty**. Poszczególne procesy mogą wywoływać metody tych obiektów; jeśli wykonanie wywołanej metody zmieni zawartość obiektu, to środowisko języka odpowiedzialne jest za to, by uaktualnione zostały wszystkie kopie tego obiektu we wszystkich procesorach. Obiekty te są encjami czysto programowymi, więc ponownie ich realizacja odbywa się bez wsparcia ze strony sprzętu i systemu operacyjnego. Do języków Linda i Orca powrócimy jeszcze w dalszej części rozdziału.

Taksonomia komputerów równoległych

Powróćmy teraz na chwilę do tematu zasadniczego, czyli rozumianej w sposób generalny architektury komputerów równoległych. W ciągu minionych lat zaproponowano i zrealizowano wiele rozwiązań tego typu, naturalnym jest więc dążenie do jakiegoś ich usystematyzowania. Poczyniono wiele prób tego rodzaju, z ograniczonym powodzeniem (Flynn, 1972) i (Treleaven, 1995), lecz królestwo komputerów równoległych wciąż czeka na swojego Karola Linneusza[13], z konieczności więc zaproponować możemy tylko przybliżoną — właściwie jedyną będącą w praktycznym użyciu — klasyfikację zaproponowaną przez Flynna. Przedstawiamy ją w tabeli 8.3.

TABELA 8.3.
Taksonomia komputerów równoległych według Flynna

Liczba strumieni rozkazów	Liczba strumieni danych	Nazwa	Przykłady
1	1	SISD	Klasyczna architektura von Neumanna.
1	Wiele	SIMD	Superkomputery wektorowe, procesory macierzowe.
Wiele	1	MISD	Zasadniczo brak.
Wiele	Wiele	MIMD	Wieloprocesory, wielokomputery.

Podstawą klasyfikacji dokonanej są dwie koncepcje — strumieni rozkazów i strumieni danych. Strumień rozkazów jest odpowiednikiem licznika rozkazów — system złożony z *n* procesorów posiada *n* liczników rozkazów, a więc *n* strumieni rozkazów.

Strumień danych jest zbiorem operandów; system rejestrujący temperaturę mierzoną przez *n* czujników posiada *n* strumieni danych.

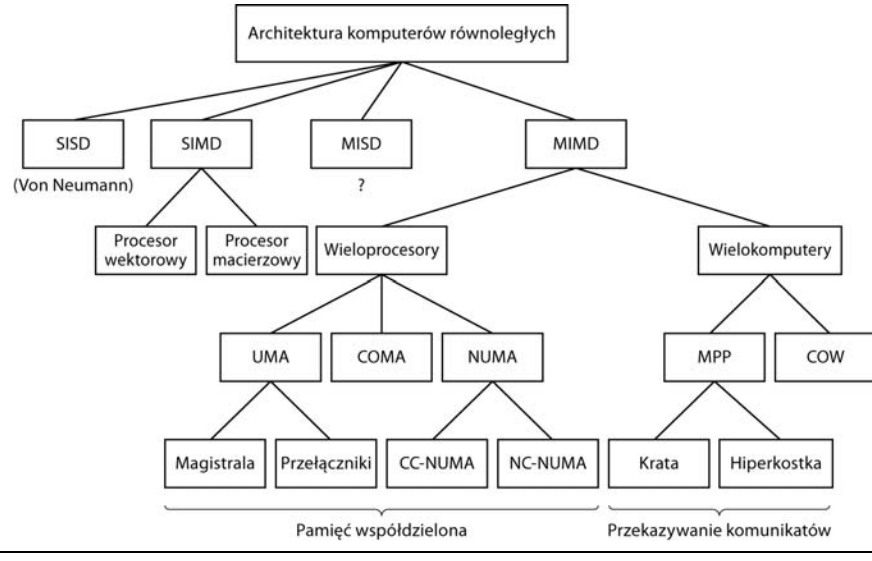
[13] Karol Linneusz (1707 – 1778), szwedzki biolog, który zaproponował stosowany do dziś system klasyfikacji zwierząt i roślin w postaci podziału na królestwa, gromady, klasy, rzędy, rodziny, typy i gatunki.

Ponieważ strumienie danych i strumienie rozkazów są w dużym stopniu niezależne, więc można na ich podstawie wyodrębnić cztery typy architektur, zgodnie z tabelą. Najprostsza architektura — SISD — to klasyczny komputer von Neumanna, z jednym strumieniem rozkazów i jednym strumieniem danych, wykonujący w danej chwili tylko jedną operację. Komputery typu SIMD posiadają jedną jednostkę sterującą i wiele jednostek arytmetyczno-logicznych (ALU), dzięki czemu w danej chwili pojedynczy rozkaz może być wykonywany na wielu zestawach danych równocześnie. Prototypem komputera SIMD była maszyna ILLIAC IV, przedstawiona na rysunku 2.6. Typowe komputery tej kategorii spotyka się coraz rzadziej, lecz wiele konwencjonalnych komputerów posiada w swym repertuarze rozkazy typu SIMD do przetwarzania danych multimedialnych — czego przykładem mogą być rozkazy SSE procesora Pentium. Jest jednak dziedzina, w której technologia SIMD znaleźć może zastosowanie w przyszłości — mowa o tzw. procesorach strumieniowych, projektowanych specjalnie na potrzeby niezwykle wydajnego renderowania grafiki (Kapasi i in., 2003).

Architektura MISD, zgodnie z którą wiele strumieni rozkazów operuje jednocześnie na tym samym zestawie danych, jest kategorią cokolwiek dziwną i właściwie nie sposób podać choćby jednego przykładu komputera pracującego w ten właśnie sposób. Niektórzy skłonni są uważać komputery potokowe za przedstawicieli tej właśnie kategorii.

Wreszcie architektura MIMD to zestaw niezależnych procesorów, współpracujących ze sobą w ramach większego systemu, a więc przede wszystkim wieloprocesory oraz wielokomputery i w ogóle większość komputerów równoległych.

Na tym kończy się taksonomia Flynna, lecz my uczyniliśmy ją bardziej szczegółową, zgodnie z rysunkiem 8.18. I tak komputery typu SIMD podzieliliśmy na dwie podgrupy. W pierwszej z nich plasują się superkomputery numeryczne i inne maszyny operujące na wektorach, wykonujące tę samą operację jednocześnie na wszystkich elementach danego wektora; druga podgrupa obejmuje komputery równoległe, w której pojedyncza jednostka sterując dokonuje rozdziału między wiele niezależnych jednostek ALU, jak w przypadku komputera ILLIAC IV.



RYSUNEK 8.18. Rozszerzona taksonomia komputerów równoległych

Kategorię MIMD dzieli się na wieloprocesory (czyli komputery z pamięcią współdzieloną) i wielokomputery (czyli komputery z procesorami przekazującymi sobie komunikaty). Wśród wieloprocesorów wyróżniliśmy trzy kategorie, różniące się sposobem realizacji współdzielenia pamięci, w dużych systemach zwykle podzielonej na wiele modułów. W architekturze zwanej **UMA** (**Uniform Memory Access**, jednolity dostęp do pamięci) czas dostępu jest jednakowy dla każdej pary „procesor-moduł pamięci"; innymi słowy, każdy procesor może odczytywać dowolne słowo z pamięci z takim samym opóźnieniem (jeśli technicznie nie jest to możliwe, szybsze odwołania zostają sztucznie spowolnione do prędkości najwolniejszych, przez co programiści nie muszą się przejmować tą różnicą). „Jednolitość" czasu wszystkich odwołań czyni wydajność systemu wysoce przewidywalną — a to bardzo ważny czynnik w tworzeniu efektywnego kodu.

Przeciwieństwem architektury z „wyrównanym" czasem dostępu jest architektura **NUMA** (**NonUniform Memory Access**, niejednolity dostęp do pamięci), w ramach której każdy procesor uzyskuje najszybciej dostęp do tych modułów pamięci, które znajdują się najbliżej niego — dostęp do odległych modułów wiąże się z większymi opóźnieniami. Rodzi to określone wymagania pod względem umieszczania określonych fragmentów kodu i danych w określonych modułach, jeżeli osiągnięta ma być maksymalna wydajność systemu. Architektura **COMA** (**Cache Only Memory Access**) także charakteryzuje się niejednolitym czasem dostępu, lecz w innym sensie. W dalszym ciągu rozdziału zajmiemy się bardziej szczegółowo każdą z trzech wymienionych kategorii.

Procesory wielokomputerów, w przeciwieństwie do wieloprocesorów, nie mają możliwości czytania i pisania w pamięci innych procesorów za pomocą rozkazów `LOAD` i `STORE`, lecz system operacyjny lub środowisko uruchomieniowe języka programowania może im stworzyć iluzję takiej możliwości, symulując pamięć współdzieloną na przykład za pomocą sprzętowych mechanizmów pamięci wirtualnej. Owa iluzoryczność jest właśnie istotnym czynnikiem różniącym wielokomputery od wieloprocesorów. Fakt braku współdzielonej pamięci w architekturze wielokomputerów powoduje, iż często architekturę tę określa się akronimem **NORMA** (**NO Remote Memory Access**, „brak dostępu do odległej pamięci").

Wielokomputery można podzielić z grubsza na dwie kategorie. Pierwszą z nich stanowią **komputery masywnie równoległe** (**MPP**, **Massively Parallel Processors**), będące superkomputerami składającymi się z wielu procesorów ściśle połączonych poprzez szybkie, dedykowane, specyficzne sieci. Przykładem komercyjnego komputera tej kategorii jest IBM SP/3.

Druga kategoria wielokomputerów to zespoły komputerów PC lub stacji roboczych, montowanych zwykle na stojakach (*rack-mounted*) i łączonych przy użyciu komercyjnych technologii „z półki" (*off-the-shelf*). Logicznie nie różnią się one niczym od przedstawicieli poprzedniej kategorii, są jednak wielokrotnie tańsze od superkomputerów kosztujących miliony dolarów i składane oraz wykorzystywane w odmienny sposób. Opatrywane bywają różnymi akronimami, między innymi **NOW** (**Network of Workstations**, sieć stacji roboczych), **COW** (**Clusteretwork of Workstations**, klaster stacji roboczych) lub nazywane są po prostu **klastrami** (*clusters*).

8.3.2. Semantyka współdzielenia pamięci

Mimo iż z logicznego punktu widzenia wszystkie procesory wieloprocesora postrzegają współdzieloną pamięć jako monolityczną przestrzeń adresową (fizyczną), to najczęściej pamięć ta podzielona jest konstrukcyjnie na wiele modułów, a moduły te są w skomplikowany sposób połączone z procesorami, jak wyjaśnialiśmy to w punkcie 8.1.2. Poszczególne procesory działają autonomicznie i w sytuacji, gdy kilka z nich dokonuje równoczesnego zapisu lub odczytu do (z) tego samego słowa pamięci, uzyskiwane w wyniku tych operacji wyniki podatne są na problem wyścigu opisywany w rozdziale 6.; chaos ten potęgowany jest dodatkowo faktem, że w celu zwiększenia efektywności funkcjonowania takiej pamięci jest ona zwykle wspomagana przez wiele pamięci podręcznych, co powoduje, że w danej chwili określony obiekt może być w niej przechowywany w kilku niespójnych ze sobą wersjach. Żywiołowe utrzymywanie tego stanu wyklucza jakąkolwiek praktyczną użyteczność pamięci współdzielonej i oczywiste jest, że jej wykorzystywanie musi zostać podporządkowane pewnym zasadom, określającym **semantykę współdzielenia**, czyli zachowanie się pamięci poddanej określonym sekwencjom operacji odczytu i zapisu.

Semantykę współdzielenia potraktować można jako swoisty kontrakt między sprzętową realizacją pamięci a korzystającym z niej oprogramowaniem (Adve i Hill, 1990): jeżeli mianowicie oprogramowanie funkcjonować będzie według ustalonych reguł, zwanych **modelami spójności** (*consistency models*), może oczekiwać od pamięci dostarczenia (w operacjach odczytu) wartości określonych przez te reguły. Zaproponowano i zaimplementowano wiele modeli spójności, w dalszym ciągu przedyskutujemy najważniejsze z nich.

Załóżmy teraz, że procesor o numerze 0 (oznaczmy go `CPU 0`) wpisuje wartość `1` do któregoś słowa w pamięci, chwilę później procesor `CPU 1` wpisuje do tego samego słowa wartość `2`. Następnie procesor `CPU 2` dokonuje odczytu tego słowa i w wyniku tego odczytu otrzymuje wartość `1`. Czy ma to oznaczać, że komputer się po prostu zepsuł i właściciel powinien oddać go do naprawy? I tak, i nie — prawidłowa odpowiedź zależy od tego, zgodnie z jakim modelem spójności zaimplementowane zostało współdzielenie pamięci.

Spójność bezwzględna (ścisła)

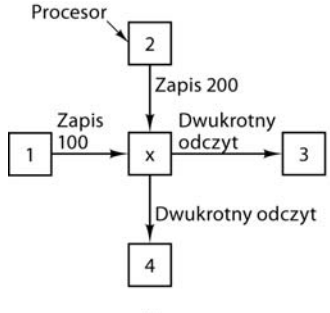
Model **ścisłej spójności** (*strict consistency*) jest modelem pojęciowo najprostszym. Zgodnie z nim odczyt zawartości dowolnego słowa pamięci daje w wyniku wartość, która została zapisana w tym słowie najpóźniej. Mimo prostoty koncepcyjnej (i wynikającej stąd atrakcyjności dla programistów) model ten nie daje się pogodzić z wymogami efektywności, wymagałby bowiem pamięci w postaci pojedynczego modułu, funkcjonującej według zasady „pierwszy przybył, pierwszy obsłużony", co praktycznie wyklucza cache'owanie i w ogóle jakąkolwiek replikację danych. Taka pamięć szybko stałaby się wąskim gardłem, więc model ten można rozważać wyłącznie w kategoriach teoretycznej możliwości.

Spójność sekwencyjna

W modelu **spójności sekwencyjnej** (*sequential consistency*), zaproponowanym w pracy (Lamport, 1979), inicjowane przez różne procesory operacje odczytów i zapisów układają

się ostatecznie w kolejności trudnej do przewidzenia a priori, jednakże kolejność ta będzie **identyczna z perspektywy każdego procesora**.

Aby lepiej zrozumieć tę zasadę, spójrzmy na przykład przedstawiony na rysunku 8.19. Załóżmy mianowicie, że procesor CPU 1 wpisuje wartość 100 do słowa pamięci, które symbolicznie oznaczyliśmy przez *x*. Nanosekundę później procesor CPU 2 wpisuje do tego samego słowa wartość 200. W kolejnej nanosekundzie (w której operacja zapisu zainicjowana przez CPU 2 może nie być jeszcze zakończona!) do głosu dochodzą dwa procesory — CPU 3 i CPU 4; każdy z nich wykonuje błyskawicznie **dwie** operacje odczytu ze słowa *x*, jak przedstawiono to na diagramie widocznym w części (a) rysunku. Mamy więc **sześć** operacji (dwa zapisy i cztery odczyty), które mogą ułożyć się potencjalnie w różną kolejność; trzy z możliwych uporządkowań przedstawiamy w częściach (b) – (d). Zgodnie z uporządkowaniem (b) procesor CPU 3 dwukrotnie otrzymuje wartość 200 — co oznaczymy jako (200, 200), podobnie CPU 4 otrzymuje wartości (200, 200). W uporządkowaniu (c) wartości otrzymane przez procesory CPU 3 i CPU 4 równe są (odpowiednio) (100, 200) i (200, 200), a w uporządkowaniu (d) wyniosą one odpowiednio (100, 100) i (200, 200). Wszystkie trzy uporządkowania są poprawne z punktu widzenia modelu spójności sekwencyjnej — jak też wiele innych uporządkowań, których na rysunku nie uwzględniliśmy.



W100	W100	W200
W200	R3 = 100	R4 = 200
R3 = 200	W200	W100
R3 = 200	R4 = 200	R3 = 100
R4 = 200	R3 = 200	R4 = 100
R4 = 200	R4 = 200	R3 = 100
(b)	(c)	(d)

RYSUNEK 8.19. Model spójności sekwencyjnej: (a) dwa procesory zapisujące dane w słowie pamięci i dwa dokonujące odczytu zawartości tego słowa; (b) – (d) trzy możliwe sekwencje przeplatania się dwóch operacji zapisu z czterema operacjami odczytu

Zauważmy jednak, że — zgodnie z regułami modelu spójności sekwencyjnej — pewne uporządkowania są absolutnie wykluczone. W szczególności niemożliwe jest, by procesory CPU 3 i CPU 4 otrzymały wartości (odpowiednio) (100, 200) i (200, 100). Gdyby bowiem tak się stało, oznaczałoby to, że z punktu widzenia procesora CPU 3 zapis wartości 200 przez CPU 2 zakończył się później niż zapis wartości 100 przez CPU 1, zaś z punktu widzenia procesora CPU 4 zdarzenia te wystąpiły w kolejności odwrotnej. Zgodnie z wymogami modelu spójności sekwencyjnej kolejność ta musi być identyczna.

Mimo iż reguły modelu spójności sekwencyjnej są słabsze niż reguły modelu spójności bezwzględnej, jest to model bardzo użyteczny. Nie precyzując konkretnego uporządkowania operacji, zapewnia on jednak, iż będzie ono identyczne z punktu widzenia wszystkich procesorów. Mimo iż spełnienie tego warunku może wydawać się oczywiste, istnieją modele spójności, które tego nie gwarantują.

Spójność procesorowa

Bardziej swobodny, lecz łatwiejszy do zaimplementowania w dużych systemach model **spójności procesorowej** (*processor consistency*) (Goodman, 1989) opiera się na dwóch następujących regułach:

(1) Zapisy dokonywane przez pamięci przez dany procesor widoczne są dla pozostałych procesorów w kolejności identycznej z tą, w jakiej zostały zainicjowane.
(2) Dla każdego słowa w pamięci, wszystkie procesory widzą tę samą kolejność wykonywanych nim operacji zapisu.

Obydwie te reguły są jednakowo ważne. Zgodnie z pierwszą z nich, jeśli procesor `CPU 1` inicjuje zapis (do pewnego słowa *x* w pamięci) wartości (kolejno) `1A`, `1B` i `1C`, to wszystkie inne procesory obserwować będą te trzy zapisy w takiej właśnie kolejności. Innymi słowy, dowolny procesor, nieustannie odczytujący w szybkiej pętli zawartość słowa *x*, nigdy nie uzyska np. wartości `1A` po wartości `1B`, bo z jego (i innych procesorów) punktu widzenia wartość `1A` zapisana została wcześniej niż wartość `1B`. Druga reguła konieczna jest dla zagwarantowania, że każde słowo w pamięci ma *jednoznacznie* określoną wartość, niezależnie od tego, jak wymyślna byłaby sekwencja operacji zapisu wykonywana na tym słowie.

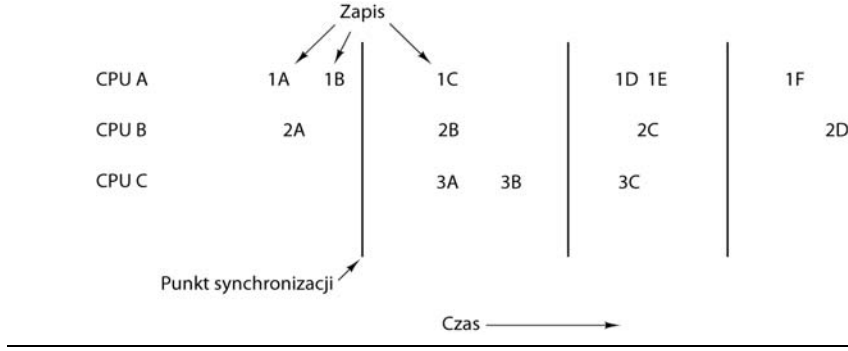
Nawet przy opisanym wymaganiu projektant pamięci ma pewne pole manewru, bowiem reguły modelu *nie* określają wzajemnej kolejności operacji inicjowanych przez *różne procesory*. Jeśli równocześnie z zapisywaniem przez `CPU 1` wartości `1A`, `1B` i `1C` w słowie *x* procesor `CPU 2` zapisuje w tym samym słowie wartości `2A`, `2B` i `2C`, to zagwarantowana jest taka właśnie kolejność w ramach *każdego z tych dwóch ciągów*, lecz *przeplatanie* się tych ciągów nie jest już jednoznacznie określone, czyli może przyjąć na przykład postać `1A`, `1B`, `2A`, `2B`, `1C`, `2C` lub `2A`, `1A`, `2B`, `2C`, `1B`, `1C`, co więcej — postaci te mogą być różne dla różnych procesorów. Można więc powiedzieć, że spójność procesorowa stanowi wynik zawężenia spójności sekwencyjnej do poszczególnych procesorów inicjujących operacje zapisu.

Należy w tym miejscu zaznaczyć, że niektórzy autorzy odmiennie definiują spójność procesorową, na przykład nie wymagając spełnienia drugiej z wymienionych reguł.

Spójność słaba

Model **słabej spójności** (*weak consistency*) (Dubois i in., 1986) nie gwarantuje nawet identyczności sekwencji wynikającej ze spójności procesorowej, tak więc operacje `1A`, `1B` i `1C` inicjowane kolejno przez procesor `CPU 1` mogą być widoczne dla jednego procesora w kolejności (na przykład) `1B`, `1A` i `1C`, zaś dla innego (na przykład) `1C`, `1A` i `1B`. Nie oznacza to bynajmniej kompletnego chaosu, przed tym bowiem chronią **punkty synchronizacji** stanowiące część modelu. Gdy uruchomiona zostaje synchronizacja, wstrzymane zostaje inicjowanie nowych operacji zapisu, aż do kompletnego zrealizowania wszystkich operacji zapisu aktualnie trwających. W efekcie synchronizacja ta doprowadza do „opróżnienia potoku" i w konsekwencji wprowadza pamięć w jednoznacznie określony stan, charakteryzujący się brakiem „wiszących" operacji, które mogłyby go zmienić. **Same operacje synchronizacji podlegają spójności sekwencyjnej**, a więc gdy inicjowane są przez różne procesory, poszczególne inicjowania widziane są przez wszystkie procesory w tej samej kolejności.

W modelu słabej spójności czas podzielony jest na dobrze określone odcinki, rozgraniczane (spójnymi sekwencyjnie) punktami synchronizacji, jak schematycznie zilustrowano to na rysunku 8.20. Operacje `1A` i `1B` mogą być widziane w różnej kolejności przez różne procesory, żaden procesor nie zobaczy ich jednak wcześniej niż operacji `1C`, od której oddzielone są punktem synchronizacji: z punktu widzenia każdego z procesorów operacje `1A`, `1B` i `2A` wystąpić muszą przez operacjami `1C`, `2B`, `3A` i `3B`, bowiem żadna z operacji tej drugiej grupy nie może rozpocząć się przed kompletnym zakończeniem wszystkich operacji z grupy pierwszej. Jak więc widać, w opisywanym modelu oprogramowanie może w ograniczonym zakresie wymuszać kolejność w odniesieniu do poszczególnych zbiorów (nie ciągów!) operacji, przez okresowe wykonywanie operacji synchronizujących. Każda z tych operacji wymaga oczywiście określonego oczekiwania na zakończenie trwających operacji zapisu.



RYSUNEK 8.20. Punkty synchronizacji wyznaczające podział czasu w modelu słabej spójności

Spójność zwalniania

Podstawowym problemem modelu słabej spójności jest pogorszenie efektywności związane z koniecznością oczekiwania (w punktach synchronizacji) na zakończenie trwających operacji zapisu. Nie ma tej wady model **spójności zwalniania** (*release consistency*) (Giarachorloo i in., 1990) wykorzystujący mechanizm podobny do sekcji krytycznych (p. rozdział 6.). Nowy proces, pragnący wejść do sekcji krytycznej, musi oczekiwać na zakończenie trwających operacji zapisu, **nie musi tego jednak robić proces opuszczający sekcję krytyczną**.

Sekcja krytyczna związana jest z określonym obszarem współdzielonych danych (na przykład z konkretnym słowem w pamięci lub jej całym modułem) i fizycznie jest współdzieloną zmienną. W celu uzyskania dostępu do chronionych danych proces musi wykonać na tej zmiennej operację `acquire`, a po jej wykonaniu operować może na tych danych w dowolny sposób; po wykonaniu wszystkich zamierzonych operacji powinien wykonać na wspomnianej zmiennej operację `release`. Operacja ta **nie** wstrzymuje procesu w oczekiwaniu na zakończenie rozpoczętych przezeń operacji zapisu i proces ten może natychmiast kontynuować pracę, **sama jednak oczekuje na ich zakończenie**.

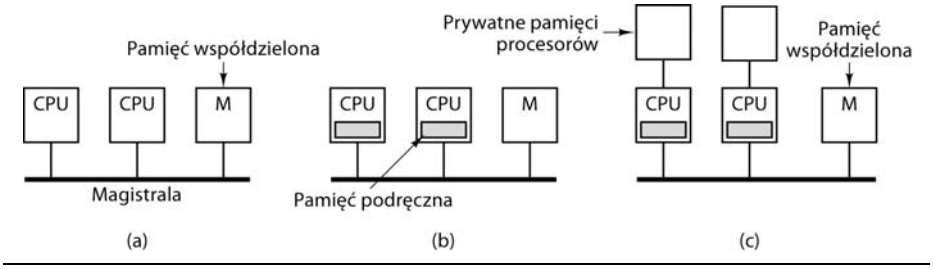
Tak więc mimo że proces użytkownika nie został wstrzymany, obszar wspólnych danych nadal jest chroniony do momentu, gdy zakończą się wszystkie dotyczące go operacje zapisu i znajdzie się on w jednoznacznym, stabilnym stanie. Gdy w międzyczasie inny proces wykona operację `acquire` na zmiennej chroniącej te dane, będzie

musiał poczekać, aż zakończy się wykonywanie operacji `release` dla tej zmiennej. Operacje `acquire` wykonywane na zmiennych chroniących dane już ustabilizowane nie powodują żadnego oczekiwania i to właśnie jest podstawową zaletą tego modelu w porównaniu z modelem słabej spójności. Wynikająca stąd większa efektywność (w porównaniu ze słabą spójnością) okupiona jest jednak nieco bardziej skomplikowaną implementacją.

Opisane modele spójności nie wyczerpują bynajmniej zagadnienia, które samo w sobie wciąż jest przedmiotem intensywnych badań i eksperymentów. Zainteresowanych tym tematem Czytelników odsyłamy do prac (Cain i Lipasti, 2004) i (Hammond i in., 2004).

8.3.3. Architektura UMA wieloprocesora symetrycznego

Architektura najprostszych wieloprocesorów bazuje na pojedynczej magistrali, do której przyłączone są dwa lub więcej procesorów (CPU) oraz kilka modułów pamięci, jak przedstawiono to na rysunku 8.21 (a). Gdy któryś z procesorów zamierza dokonać odczytu słowa z pamięci, sprawdza najpierw, czy magistrala nie jest zajęta przez inny procesor i jeżeli faktycznie nie jest zajęta, wysyła na tę magistralę adres słowa, odpowiednio ustawia stan sygnałów sterujących i oczekuje, aż zapisana w słowie wartość pojawi się na magistrali.



RYSUNEK 8.21. Trzy typy wieloprocesorów bazujących na magistrali: (a) bez cache'owania; (b) z cache'owaniem; (c) z cache'owaniem i prywatnymi pamięciami procesorów

Jeżeli jednak okaże się, że magistrala jest zajęta, procesor musi poczekać, aż stanie się ona wolna. Pojawia się w związku z tym pierwszy problem projektowy, jakim jest rywalizacja procesorów o dostęp do magistrali. Nie sprawia ona szczególnych kłopotów przy dwóch lub trzech procesorach, gdy jednak liczba procesorów sięgnie 32 lub 64, staje się ona koszmarem: system ograniczany jest silnie przez przepustowość magistrali, która staje się niesamowicie wąskim gardłem, a większość procesorów spędza czas w stanie bezczynności.

Problem ten można znacznie złagodzić, dodając do każdego procesora pamięć podręczną (cache), co widać w części (b) rysunku 8.21; pamięć ta może być zintegrowana z procesorem wewnątrz układu lub znajdować się na zewnątrz procesora. Ponieważ w tych warunkach znakomita większość odczytów dokonywana jest przez procesory z ich pamięci cache, nie z pamięci współdzielonej, obciążenie magistrali zostaje znacznie zredukowane i może ona efektywnie obsługiwać więcej procesorów.

W części (c) rysunku przedstawiono rozwiązanie bardziej zaawansowane, zgodnie z którym każdy procesor posiada nie tylko własną pamięć cache, lecz także prywatną pamięć lokalną, z którą kontaktuje się za pośrednictwem dedykowanej (prywatnej) magistrali. Aby konfiguracja taka została optymalnie wykorzystana, cały kod programów, wszystkie ich łańcuchy, stałe i inne niemodyfikowalne elementy oraz stosy i zmienne lokalne powinny być umieszczone w lokalnych pamięciach procesorów; w pamięci współdzielonej powinny znajdować się tylko modyfikowalne zmienne współdzielone. Odpowiedzialność za tę organizację spoczywa na kompilatorach i choć stają się one z tego powodu bardziej skomplikowane, to jednak dzięki właściwemu wykorzystaniu prywatnych pamięci procesorów obciążenie magistrali zredukowane zostaje w jeszcze większym stopniu.

Cache'owanie z podsłuchiwaniem

Mimo iż cache'owanie w dużym stopniu łagodzi problem przeciążenia magistrali, to jednocześnie staje się źródłem innego fundamentalnego problemu. Załóżmy, że pamięć współdzielona działa zgodnie z semantyką spójności sekwencyjnej (patrz 8.3.2): co stanie się, gdy procesory `CPU 1` i `CPU 2` zechcą odczytać słowo mapowane w ten sam wiersz pamięci cache? Oczywiście, jeżeli nie zajdą jakieś szczególne warunki, powinny one otrzymać tę samą wartość. Załóżmy jednak, że `CPU 1` modyfikuje wspomniane słowo (za pośrednictwem swego cache), a chwilę później `CPU 2` odczytuje jego zawartość, oczywiście ze swego prywatnego cache. Wartość uzyskana w wyniku tego odczytu będzie oczywiście nieaktualna.

Opisany problem znany jest powszechnie pod nazwą **spójności cache** (*cache consistency* lub *cache coherence*) i jest problemem bardzo poważnym. Bez zadowalającego rozwiązania staje się on czynnikiem dyskwalifikującym pomysł wykorzystania pamięci cache, a liczba procesorów przyłączonych do wspólnej magistrali ograniczona zostaje do dwóch lub trzech. Na szczęście w ciągu minionych lat zaproponowano wiele sposobów jego rozwiązania, między innymi w pracach (Goodman, 1983) i (Papamarcos i Patel, 1984). Mimo iż poszczególne rozwiązania, określane wspólną nazwą **protokołów spójności cache** (*cache coherence protocols*), różnią się wieloma szczegółami, wszystkie zapobiegają nieakceptowalnej sytuacji, jaką jest obecność dwóch lub więcej różnych wersji tego samego wiersza cache w różnych pamięciach podręcznych.

Opisanej spójności nie da się zapewnić bez konsekwentnego śledzenia adresów i wartości pojawiających się na magistrali, w związku z czym wyposażono każdą z pamięci podręcznych w specjalny kontroler, śledzący żądania pojawiające się na magistrali ze strony innych cache i ich procesorów i wykonujący stosowne akcje w określonych sytuacjach. Ponieważ magistrala jest nieustannie „podsłuchiwana" przez wspomniane kontrolery, opisany system cache'owania nazywany jest **cache'owaniem podsłuchującym** (*snooping cache*) lub (rzadziej) **wścibskim cache'owaniem** (*snoopy cache*). Zbiór reguł, implementowanych przez pamięci podręczne, procesory i pamięci w celu zapobieżenia rozbieżnościom wierszy w poszczególnych pamięciach podręcznych składa się na wspomniany protokół spójności cache. Jednostki (porcje), w których wymieniana jest zawartość pamięci operacyjnej i pamięci cache, nazywane są **linijkami** lub **wierszami cache** (*cache lines*) i mają zazwyczaj rozmiar 32 lub 64 bajtów.

Najprostszym protokołem spójności cache jest protokół **zapisu jednoczesnego** (*write through*), którego zasady wyjaśnić można, rozróżniając cztery sytuacje wyszczególnione w tabeli 8.4. Gdy procesor próbuje odczytać słowo, którego nie ma w pamięci podręcznej (nazywa się to **chybieniem odczytu** — *read miss*), kontroler cache ładuje to słowo z pamięci współdzielonej (której zawartość jest zawsze aktualna) do odpowiedniego wiersza pamięci podręcznej. Kolejne próby odczytu tego samego słowa przebiegać już będą w trybie **trafień odczytu** (*read hit*).

TABELA 8.4.
Funkcjonowanie protokołu zapisu jednoczesnego (write through); puste rubryki oznaczają brak jakiejkolwiek akcji

Sytuacja	Żądanie lokalne[14]	Żądanie zdalne
Chybienie odczytu	Pobranie danych z pamięci współdzielonej.	
Trafienie odczytu	Użycie danych z pamięci podręcznej.	
Chybienie zapisu	Aktualizacja danych w pamięci współdzielonej.	
Trafienie zapisu	Aktualizacja danych w pamięci współdzielonej i pamięci podręcznej.	Unieważnienie wiersza w pamięci podręcznej.

Gdy w pamięci podręcznej nie ma wiersza zawierającego słowo, które procesor chce **zmodyfikować** (nazywa się to **chybieniem zapisu** — *write miss*), modyfikowane jest jedynie słowo w pamięci współdzielonej, lecz **odpowiadający mu wiersz nie jest sprowadzany do pamięci podręcznej**. Gdy natomiast procesor modyfikuje słowo reprezentowane w pamięci podręcznej (czyli w sytuacji **trafienia zapisu** — *write hit*), uaktualniane jest zarówno słowo w pamięci współdzielonej, jak i odpowiadający mu wiersz w pamięci podręcznej. Innymi słowy, zapis odbywa się „przez bufor", po angielsku *write through buffer*, stąd nazwa protokołu.

Przeanalizujmy teraz poszczególne sytuacje z punktu widzenia kontrolera podsłuchującego: przez `cache1` oznaczmy pamięć podręczną, w kontekście której zachodzą poszczególne sytuacje, zaś przez `cache2` pamięć podręczną, która poprzez podsłuchiwanie magistrali zmuszona jest dostosowywać swój stan do akcji wykonywanych w kontekście `cache1`. Gdy `cache1` doświadcza sytuacji chybienia odczytu, pobiera żądany wiersz z pamięci współdzielonej; `cache2` widzi to, lecz nie podejmuje w związku z tym żadnej akcji. W sytuacji, gdy `cache1` zawiera już żądany wiersz (trafienie odczytu), w ogóle nie odwołuje się do magistrali i `cache2` nie jest nawet świadoma całej sprawy.

Operacje zapisu są bardziej interesujące. Gdy procesor `CPU 1` inicjuje operację zapisu, `cache1` wysyła na magistralę żądanie zapisu niezależnie od tego, czy mamy do czynienia z trafieniem, czy z chybieniem. `Cache2` rejestruje to żądanie i sprawdza, czy jest w posiadaniu związanego z tym żądaniem wiersza. Jeśli nie, to z jego punktu widzenia jest to zdalne chybienie lub trafienie zapisu, czyli sytuacja niemająca związku z jego (`cache2`) własną zawartością (drobna uwaga do ostatniego wiersza tabeli 8.4: określenie „chybienie zapisu" odnosi się do **podsłuchującej** pamięci podręcznej, czyli `cache2`; nie ma znaczenia, czy wiersz związany z żądaniem zapisu znajduje się w `cache1`, czy też nie).

[14] **Żądanie lokalne** jest, z punktu widzenia danej pamięci podręcznej, żądaniem generowanym przez jej własny procesor; żądania zdalne są, z punktu widzenia tej pamięci, żądaniami generowanymi przez inne procesory — *przyp. tłum.*

Załóżmy teraz, że CPU 1 (czyli procesor posiadający cache1) dokonuje modyfikacji słowa, które **jest** reprezentowane w cache2. Gdyby w tym momencie cache2 nie wykonała żadnej czynności, jej zawartość stałaby się nieaktualna (z powodu nieaktualności wiersza zawierającego modyfikowane słowo); by tego uniknąć, wykonuje ona najprostszą rzecz, jaką w tej sytuacji można wykonać — **pozbywa się** rzeczonego wiersza (nazywa się to **unieważnieniem wiersza** — *invalidate cache line*). Ponieważ wszystkie pamięci podręczne obserwują wszystkie żądania pojawiające się na magistrali, ostatecznym efektem operacji zapisu inicjowanej przez dany procesor jest (ewentualne) uaktualnienie odpowiedniego wiersza w jego własnym cache, (ewentualne) unieważnienie zawierających modyfikowane słowo wierszy w innych cache i uaktualnienie modyfikowanego słowa w pamięci współdzielonej. W ten oto sposób zmodyfikowana wartość znajduje się w pamięci współdzielonej i co najwyżej w pamięci podręcznej procesora modyfikującego — groźba niespójności cache zostaje więc zażegnana.

Oczywiście procesor CPU 2 (ten zawierający cache2) może natychmiast zażądać odczytu słowa, które dopiero co zmodyfikował procesor CPU 1. Ponieważ słowo to nie jest reprezentowane w cache2, zawierający go wiersz zostanie pobrany z **uaktualnionej już** pamięci współdzielonej. To samo mogą zrobić inne procesory; gdy któryś z nich zmodyfikuje odczytane słowo, zawierające je wiersze zostaną unieważnione w pamięciach podręcznych innych procesorów i zawartość wszystkich pamięci podręcznych pozostanie spójna.

Opisany protokół można zmodyfikować w ten sposób, by zamiast unieważniania zmodyfikowanych wierszy następowało ich uaktualnianie. Koncepcyjnie uaktualnienie wiersza równoważne jest jego unieważnieniu i sprowadzeniu z pamięci, choć w praktyce może być wykonane znacznie szybciej. W związku z tym konstruktor protokołu spójności cache musi dokonać wyboru między dwiema strategiami zachowywania spójności: **strategią aktualizacji** (*update strategy*) i **strategią unieważniania** (*invalidate strategy*). Pierwsza z nich wiąże się z przesyłaniem dodatkowych komunikatów, lecz zapobiega występowaniu wielu chybień odczytu, nieuchronnych przy strategii unieważniania.

Innym pomysłem modyfikacji opisanego algorytmu jest uaktualnianie zawartości podsłuchujących pamięci podręcznych w przypadku chybienia zapisu (czyli w przypadku nieobecności w nich słowa modyfikowanego przez inne procesory). Modyfikacja ta nie wpływa w żadnej sposób na poprawność protokołu, może jednak w pewnych warunkach poprawić jego wydajność. Stopień tej poprawy uzależniony jest od **lokalności odwołań** operacji zapisu, czyli od prawdopodobieństwa, że modyfikowane właśnie słowo niebawem modyfikowane będzie ponownie. Jeśli prawdopodobieństwo to jest duże, istnieją uzasadnione przesłanki do zastosowania opisanej modyfikacji, nazywanej **alokacją przy zapisie** (*write-allocate policy*). Jeśli jest ono małe, korzystniejsze będzie unieważnianie odnośnych wierszy; nawet bowiem jeśli zmodyfikowane słowo będzie niebawem **odczytywane**, to chybienie odczytu będzie tylko jednokrotne.

Jak wszystkie proste rozwiązania, tak i to pozostawia wiele do życzenia pod względem efektywności. Każda operacja zapisu wiąże się z wysyłaniem żądań na magistralę i dostępem do pamięci współdzielonej, przez co już przy umiarkowanej liczbie procesorów magistrala ta staje się zbyt wąskim gardłem. Konieczne staje się więc zastosowanie innych protokołów, przede wszystkim **rezygnujących z uaktualniania współdzielonej pamięci przy każdej operacji zapisu**; w zamian uaktualniany jest tylko odpowiedni wiersz w pamięci cache z jednoczesnym ustawieniem specjalnego bitu informującego, że zawartość reprezentowanego przez ten wiersz obszaru w pamięci współdzielonej jest nieaktualna (bit ten nazywa się w skrócie **bitem naruszenia**

— *dirty bit*). Taki zmodyfikowany wiersz prędzej czy później i tak zostanie zapisany do pamięci współdzielonej, lecz prawdopodobnie po wykonaniu wielu operacji zapisu. Protokoły tego typu nazywane są **protokołami zapisu opóźnionego** (*write back protocols*).

Protokół MESI

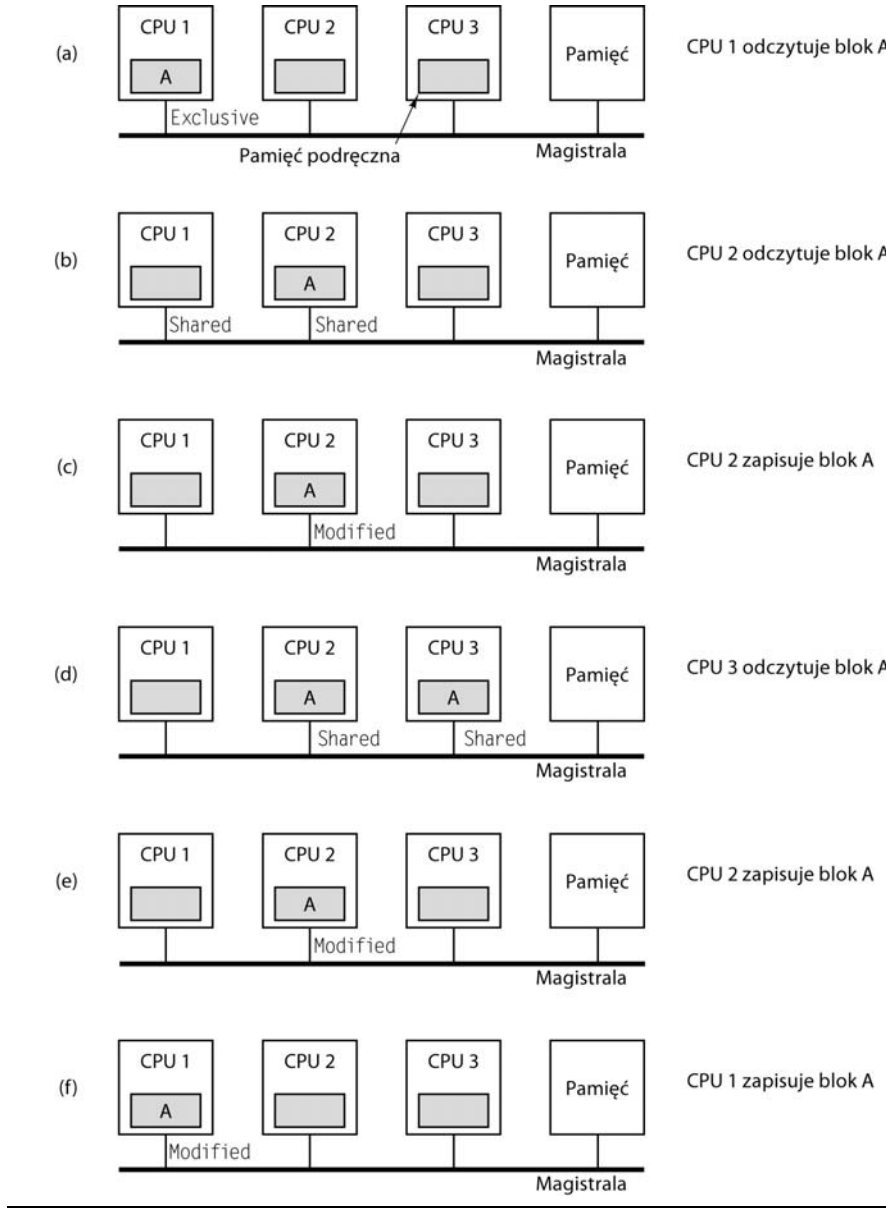
Jednym z popularnych protokołów zapisu opóźnionego jest protokół **MESI**, wywodzący swą nazwę z pierwszych liter oznaczeń stanów, w jakich znajdować się może pracująca w zgodzie z nim pamięć podręczna (Papamarcos i Patel, 1984). Protokół ten stanowi unowocześnioną wersję **protokołu zapisu jednokrotnego** (*write-once protocol*) (Goodman, 1983). Protokół MESI stosowany jest między innymi w procesorze Pentium do podsłuchiwania magistrali. Zgodnie z nim, każdy linijka wiersz danej pamięci podręcznej może znajdować się w jednym z czterech następujących stanów[15]:

- `Invalid` (nieważny) — wiersz nie zawiera poprawnych danych.
- `Shared` (współdzielony) — wiele pamięci podręcznych zawiera ten wiersz; zawartość pamięci współdzielonej jest aktualna.
- `Exclusive` (wyłączny) — żadna inna pamięć podręczna nie zawiera tego wiersza; zawartość pamięci współdzielonej jest aktualna.
- `Modified` (zmodyfikowany) — wiersz jest aktualny, lecz nieaktualny jest odpowiadający mu obszar w pamięci współdzielonej. Nie istnieją kopie wiersza w innych pamięciach podręcznych.

Bezpośrednio po zainicjowaniu pracy systemu wszystkie wiersze wszystkich pamięci podręcznych oznaczane są jako nieważne (`Invalid`). Gdy po raz pierwszy któryś z procesorów dokonuje odczytu słowa z pamięci współdzielonej (na rysunku 8.22 (a) jest to procesor `CPU 1`), musi to zrobić wprost z pamięci za pośrednictwem magistrali, ładując do odpowiedniego wiersza obszar pamięci zawierający żądane słowo; wiersz ten oznaczony zostaje jako `Exclusive`, bowiem jest on jedynym wierszem (wśród wszystkich pamięci podręcznych) reprezentującym odnośny adres. Kolejne odczyty tego samego słowa nie będą już wymagały kontaktu z magistralą. Gdy teraz inny procesor (`CPU 2`) dokona odczytu tego samego słowa, również musi uczynić to za pośrednictwem magistrali (bo jego pamięć podręczna jest pusta); zauważy to podsłuchujący magistralę procesor `CPU 1` i zasygnalizuje, że posiada już w swej pamięci podręcznej odnośny wiersz, wskutek czego obydwa wiersze — ten w pamięci podręcznej `CPU 1` i ten nowo odczytany w pamięci podręcznej `CPU 2` — oznaczone zostaną jako `Shared`, jak na rysunku 8.22 (b).

Zobaczmy teraz, co się stanie, gdy `CPU 2` zmodyfikuje wiersz (w swojej pamięci podręcznej) znajdujący się w stanie `Shared`. Wiersz ten oznaczony zostaje jako `Modified`, jednocześnie na magistralę wysyłany jest specjalny sygnał nakazujący unieważnienie wszystkich kopii tego wiersza w innych pamięciach podręcznych, czyli oznakowanie odnośnych wierszy jako `Invalid`. Sytuacji tej odpowiada cześć (c) rysunku. Modyfikacja wiersza *nie* jest odzwierciedlana w pamięci współdzielonej. Zauważmy, że gdyby modyfikowany wiersz znajdował się w stanie `Exclusive`, nie `Shared`, informowanie innych pamięci podręcznych o jego zmodyfikowaniu nie byłoby potrzebne.

[15] Zachowaliśmy oryginalne nazwy stanów — *przyp. tłum.*



RYSUNEK 8.22. Współdziałanie pamięci podręcznych wieloprocesora zgodnie z protokołem MESI

Nasz wieloprocesor wciąż pracuje i w pewnej chwili procesor CPU 3 dokonuje odczytu obszaru (w pamięci współdzielonej) odpowiadającego wierszowi zmodyfikowanemu przez CPU 2. Sprawa wygląda interesująco, bo zawartość tego obszaru w pamięci współdzielonej **nie jest aktualna**. Procesor CPU 2, który (jak wszystkie procesory) śledzi poczynania CPU 3, wysyła mu w związku z tym sygnał nakazujący oczekiwanie,

po czym na podstawie własnej pamięci podręcznej dokonuje uaktualnienia rzeczonego obszaru w pamięci współdzielonej. Gdy uaktualnianie to się zakończy, CPU 3 może pobrać żądany wiersz z pamięci współdzielonej, jak gdyby nic specjalnego się nie stało. Wiersz ten obecny jest teraz w pamięciach podręcznych obydwu procesorów — CPU 2 i CPU 3 — więc w każdej z tych pamięci oznaczony zostaje jako Shared, jak w części (d) rysunku. Po chwili CPU 2 ponownie modyfikuje ten sam wiersz, wskutek czego zostaje on unieważniony w pamięci podręcznej CPU 3, zaś w pamięci podręcznej CPU 2 oznaczony zostaje jako Modified, co odzwierciedlone zostało w części (e) rysunku.

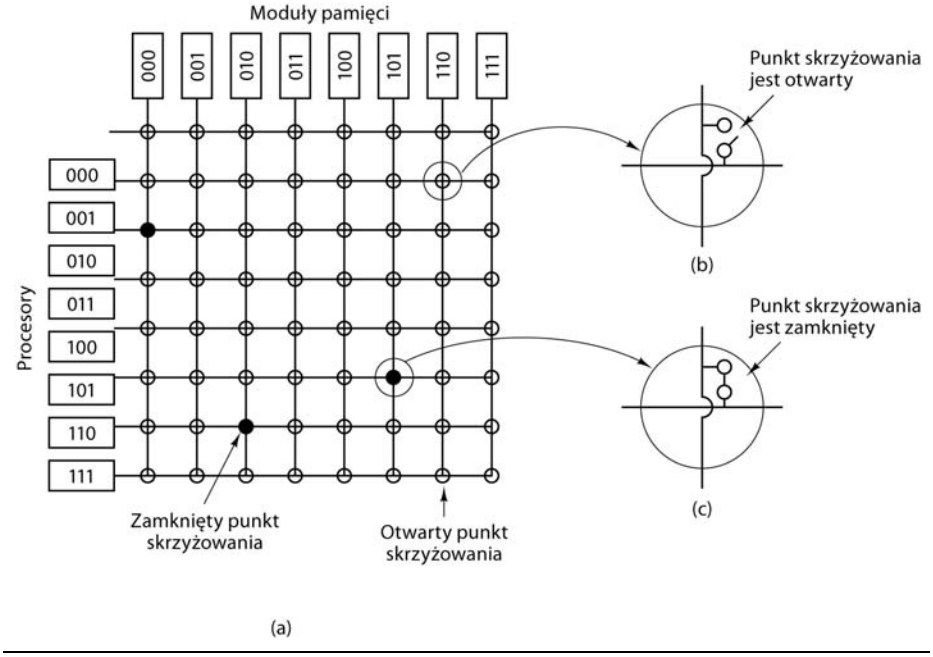
W końcu procesor CPU 1 zamierza zmodyfikować obszar pamięci współdzielonej, odpowiadający wspomnianemu wierszowi — obszar, jak wiadomo, **z nieaktualną zawartością**. Obserwujący to CPU 2 wysyła swemu partnerowi CPU 1 sygnał nakazujący oczekiwanie i zawartość wspomnianego obszaru uaktualnia; jednocześnie wiedząc, że odnośny wiersz zostanie zmodyfikowany, unieważnia go w swej pamięci podręcznej. Zauważmy, że CPU 1 **nie posiada w swej pamięci podręcznej wiersza reprezentującego modyfikowany obszar** i to, co się teraz stanie, zależne jest od tego, czy protokół implementuje alokację przy zapisie (patrz poprzedni podpunkt), czy nie. Jeśli jej nie implementuje, zapis zostanie dokonany wprost do pamięci współdzielonej, w przeciwnym razie zmodyfikowana zawartość wpisana zostanie do stosownego wiersza w pamięci podręcznej CPU 1, a wiersz ten oznaczony zostanie jako Modified, jak w części (f) rysunku; **odnośny obszar pamięci współdzielonej nie zostanie zmodyfikowany**.

Wieloprocesory UMA z przełącznicami krzyżowymi

Nawet przy wykorzystaniu wszelkich możliwości optymalizacyjnych użycie tylko jednej magistrali ogranicza w praktyce liczbę procesorów wieloprocesora UMA do 16 lub 32. W układach bardziej złożonych konieczne jest użycie bardziej zaawansowanych połączeń. Najprostszy obwód, łączący *n* procesorów z *k* pamięciami nazywany jest **przełącznicą krzyżową** (*crossbar switch*) i pokazany na rysunku 8.23. Przełącznice takie wykorzystywane były przez dziesięciolecia w centralach telefonicznych w celu wykonywania dowolnych przełączeń między liniami połączeń przychodzących a liniami dla połączeń wychodzących.

Każde skrzyżowanie linii poziomej (połączenie przychodzące) z pionową (połączenie wychodzące) nazywane jest **punktem skrzyżowania** (*crosspoint*). Fizycznie punkt skrzyżowania ma postać małego przełącznika, który może być elektrycznie „zamykany" i „otwierany" zależnie od tego, czy wspomniane linie mają być ze sobą połączone, czy też nie. Na rysunku 8.23 (a) widzimy trzy zamknięte punkty skrzyżowań, tworzące połączenia między następującymi parami „procesor-pamięć": (001, 000), (101, 101) i (110, 010). Możliwych jest wiele innych kombinacji — ich liczba równa jest liczbie wszystkich możliwych ustawień ośmiu wież na szachownicy w taki sposób, by żadna ich para nie szachowała się nawzajem.

Jedną z najwspanialszych własności przełącznic krzyżowych jest to, że są one **sieciami nieblokującymi** (*nonblocking networks*): zajętość linii lub punktu skrzyżowania nie stanowi przeszkody w połączeniu dowolnego procesora z dowolnym modułem pamięci, oczywiście pod warunkiem, że moduł ten jest dostępny. Nie jest ponadto potrzebne wstępne planowanie połączeń: nawet jeżeli siedem procesorów zostało już połączonych z siedmioma modułami pamięci, bez przeszkód można przyłączyć pozostały procesor do pozostałego modułu pamięci. W dalszym ciągu rozdziału przedstawimy sieci połączeniowe, które nie mają tych własności.

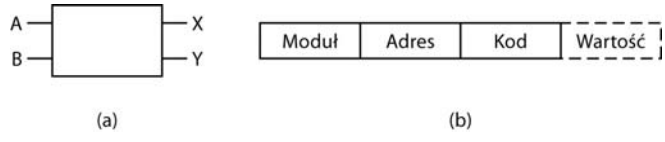


RYSUNEK 8.23. (a) Przełącznica krzyżowa 8 × 8; (b) Otwarty punkt skrzyżowania; (c) Zamknięty punkt skrzyżowania

Wadą przełącznic krzyżowych jest **kombinatoryczna** zależność punktów skrzyżowań od liczby procesorów i liczby pamięci — przy *n* procesorach i *k* pamięciach dodanie nowej pamięci wymaga dodania *n* nowych punktów skrzyżowań, a dodanie nowego procesora — *k* nowych punktów skrzyżowań. Przy umiarkowanej liczbie procesorów i pamięci przełącznice krzyżowe spisują się bardzo dobrze: w dalszym ciągu rozdziału przedstawimy przykładowy projekt bazujący na nich — Sun Fire E25K. Przełącznik z 1000 procesorów i 1000 pamięci wymagałby jednak miliona punktów skrzyżowań — konstruowanie tak dużych przełącznic nie jest opłacalne, konieczne są całkowicie inne rozwiązania.

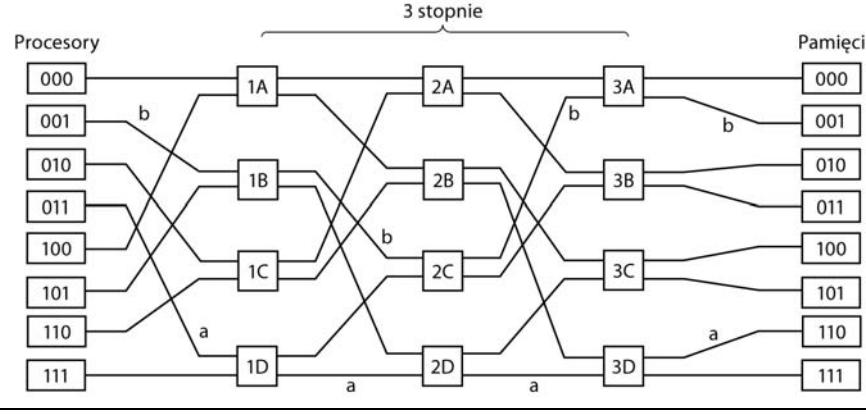
Wieloprocesory UMA wykorzystujące wielostopniowe sieci przełączające

„Całkowicie inne rozwiązania" mogą w tym przypadku wykorzystywać prosty przełącznik 2 × 2 pokazany na rysunku 8.24 (a), posiadający dwa wejścia i dwa wyjścia. Komunikat z dowolnego wejścia może być skierowany na dowolne wyjście. Sam komunikat składa się z czterech części, widocznych w części (b) rysunku. Pole `Moduł` identyfikuje wykorzystywany moduł pamięci, a pole `Adres` — konkretny adres wewnątrz tego modułu. W polu `Kod` znajduje się kod operacji, na przykład `READ` lub `WRITE`. Opcjonalne pole `Wartość` może zawierać operand, na przykład 32-bitowe słowo do zapisania przez rozkaz `WRITE`. O tym, na które z wyjść — `X` czy `Y` — skierowany zostanie komunikat, decyduje zawartość pola `Moduł`.



RYSUNEK 8.24. (a) Przełącznik 2 × 2; (b) Format komunikatu

Takie przełączniki 2 × 2 mogą być łączone w duże, **wielostopniowe sieci przełączające** (*multistage switching networks*), których szczególnymi przypadkami są **sieci omega** — tanie i obywające się bez żadnego dodatkowego wyposażenia. Na rysunku 8.25 widoczny jest przykład sieci omega łączącej osiem procesorów z ośmioma modułami pamięci, przy użyciu 12 przełączników. Generalnie rzecz biorąc, sieć omega łącząca n procesorów z n modułami pamięci składa się z $log_2 n$ stopni, z których każdy realizowany jest za pomocą $\frac{n}{2}$ przełączników. Całkowita liczba przełączników wynosi więc $\frac{n}{2} * log_2 n$, co jest znacznym postępem (zwłaszcza w przypadku dużych n) w porównaniu z n^2 punktami skrzyżowań, koniecznymi przy połączeniu n procesorów z n modułami pamięci za pomocą przełącznic krzyżowych.



RYSUNEK 8.25. Przykładowa sieć omega

Sposób „odrutowania” sieci omega nazywany jest często **tasowaniem zupełnym** (*perfect shuffle*), bowiem przemieszanie sygnałów na każdym stopniu przypomina jako żywo tasowanie talii kart przez jej „przecięcie” na pół i następnie naprzemienne złożenie „karta po karcie”. Aby zrozumieć działanie sieci omega, prześledźmy szczegółowo wędrówkę komunikatu związanego z odczytywaniem przez procesor `011` słowa z modułu pamięci `110`. Procesor przesyła w związku z tym do przełącznika `1D` komunikat zawierający w polu `Moduł` wartość `110`. Przełącznik pobiera z tego pola pierwszy (od lewej strony) bit i wykorzystuje go w charakterze „informacji marszrutowania”: wartość `0` oznacza wybór górnego wyjścia, wartość `1` — wybór wyjścia dolnego. Rzeczony bit ma wartość `1`, komunikat trafia więc do przełącznika `2D`.

Przełącznik 2D, znajdujący się na drugim stopniu sieci, w analogiczny sposób wykorzystuje drugi (od lewej) bit — ma on wartość 1, więc komunikat trafia do przełącznika 3D. Ten z kolei dokonuje testowania trzeciego bitu i po stwierdzeniu, że ten ma wartość 0, kieruje komunikat do docelowego modułu pamięci 110. Opisaną ścieżkę, którą przebywa komunikat od procesora do modułu pamięci, oznaczyliśmy na rysunku literą a.

W miarę jak komunikat przemieszcza się przez kolejne stopnie sieci, kolejne bity wartości zapisanej w polu Moduł stają się niepotrzebne i można by je było wykorzystać do bieżącego kodowania trasy powrotnej, dla odpowiedzi na komunikat[16]. Trasa powrotna odpowiadająca ścieżce a wiedzie kolejno przez dolne (1) wejście przełącznika 3D, dolne (1) wejście przełącznika 2D i górne (0) wejście przełącznika 1D. Nieprzypadkowo kodujące ją bity, czytane w odwrotnej kolejności (i w takiej zapisywane sukcesywnie do pola Moduł), dają numer procesora wysyłającego pierwotny komunikat (011).

Załóżmy, że równocześnie z opisanym wysyłaniem przez procesor 011 komunikatu do modułu pamięci 110 inny procesor — 001 — zamierza zapisać słowo w module pamięci 001. Związany z tym zamiarem komunikat przepływa kolejno przez przełączniki 1B, 2C i 3A — ścieżkę tę oznaczyliśmy literą b. Gdy dotrze on do docelowego modułu pamięci, w jego polu Moduł znajdować się będzie (odwrócony) kod trasy powrotnej 001, czyli numer procesora źródłowego. Ponieważ ścieżki a i b nie zawierają wspólnych przełączników i dla każdej z nich inny jest procesor źródłowy oraz docelowy moduł pamięci, mogą one niezależnie przesyłać swe komunikaty w tym samym czasie.

Równoległość taka nie zawsze jest jednak możliwa, co stanie się oczywiste, gdy równocześnie z przepływem komunikatu przez ścieżkę b procesor 000 zechce wysłać komunikat do modułu pamięci 000. Stosowny komunikat, po bezproblemowym przejściu przez przełączniki 1A i 2A, dotrze do przełącznika 3A i w tym momencie pojawi się problem, bowiem przełącznik 3A wykorzystywany jest przez ścieżkę b. Komunikat inicjowany przez procesor 000 będzie musiał poczekać na zwolnienie przełącznika 3A. Jak więc widać, w przeciwieństwie do przełącznic krzyżowych, sieci omega są **sieciami blokującymi** — nie każdy zbiór komunikatów może być przetwarzany w sposób całkowicie zrównoleglony, konflikty mogą pojawiać się zarówno wskutek wzajemnych „krzyżowań" ścieżek, jak i tras powrotnych na przełącznikach.

Jest wysoce pożądanym rozproszenie odwołań do pamięci jak najbardziej równomiernie po poszczególnych modułach pamięci. Jednym ze sposobów tego rozpraszania jest wykorzystywanie wybranych bitów adresu jako numeru modułu. Załóżmy, że w komputerze z pamięcią bajtową większość odwołań dotyczy słów 32-bitowych. Dwa najmłodsze bity adresu są wówczas równe 00, lecz trzy następne mogłyby być traktowane jako numer jednego z ośmiu modułów pamięci. W ten oto sposób kolejno adresowane słowa ulokowane byłby w kolejnych modułach — taki sposób rozproszenia adresów między moduły nazywa się **przeplotem** (*interleave*). Ponieważ większość ciągów odwołań do pamięci to odwołania do kolejnych adresów[17], „przeplatane" pamięci są szczególnie wydajne z punktu widzenia zrównoleglonego przetwarzania.

[16] Na trasie powrotnej wejścia i wyjścia przełączników zamienione zostają rolami w stosunku do ścieżki oryginalnej — *przyp. tłum.*

[17] Wynika to wprost z naturalnej kolejności przetwarzania podstawowych struktur danych — tablic, rekordów, plików mapowanych — oraz naturalnej kolejności przepływu sterowania w programie (w blokach liniowych, między skokami) — *przyp. tłum.*

Możliwe jest także takie projektowanie sieci przełączających, by między każdą parą „procesor-pamięć" istniało kilka ścieżek (w celu bardziej równomiernego rozłożenia obciążeń) oraz by prawdopodobieństwo blokowania zredukowane było do minimum lub całkowicie wyeliminowane.

8.3.4. Wieloprocesory NUMA

Możliwości rozbudowy wieloprocesorów pracujących na bazie pojedynczej magistrali kończą się w okolicach kilkudziesięciu procesorów składowych, a zastosowanie kosztownego sprzętu w rodzaju przełącznic krzyżowych lub sieci przełączających niewiele jest w stanie poprawić w tym względzie. Aby można było mówić o **kilkuset** procesorach składowych, konieczna jest rezygnacja z pewnych założeń architektonicznych i pierwszym z takich założeń jest identyczny czas dostępu każdego procesora do wszystkich modułów pamięci. Zróżnicowanie czasu dostępu każdego z procesorów do różnych modułów pamięci prowadzi do koncepcji wieloprocesorów z **niejednolitym dostępem do pamięci**, w skrócie **NUMA** (**NonUniform Memory Access**). Podobnie jak ich kuzyni z kategorii UMA, wieloprocesory NUMA pracują w oparciu o pojedynczą fizyczną przestrzeń adresową, jednakże dla każdego procesora jego lokalne moduły pamięci dostępne są szybciej niż pozostałe, zdalne moduły. Dzięki temu programy napisane dla wieloprocesorów UMA mogą funkcjonować bez zmian na wieloprocesorach NUMA, lecz funkcjonować będą wolniej przy tej samej częstotliwości zegara.

Wieloprocesory NUMA posiadają trzy następujące własności, które potraktowane łącznie odróżniają je od wieloprocesorów innych kategorii:

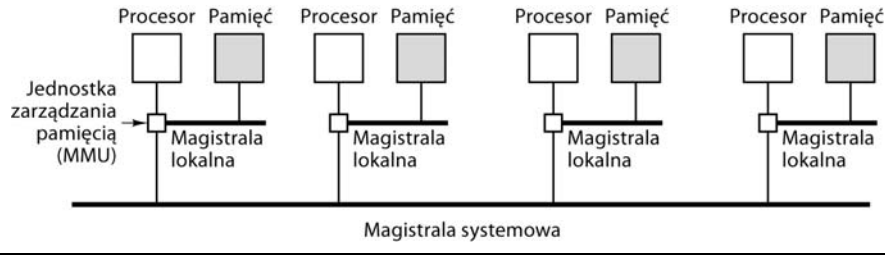
(1) Istnieje pojedyncza fizyczna przestrzeń adresowa widoczna dla wszystkich procesorów.
(2) Dostęp do zdalnych modułów pamięci realizowany jest za pomocą rozkazów LOAD i STORE.
(3) Dostęp każdego procesora do jego lokalnej pamięci jest szybszy niż dostęp do pamięci innych procesorów.

Gdy dostęp do zdalnej pamięci nie jest wspomagany cache'owaniem, architekturę wieloprocesora określa się skrótem **NC-NUMA** (**Non-cached NUMA**); analogicznie architektura wspomagana systemem spójnego cache'owania opatrywana jest skrótem **CC-NUMA** (**Coherent-cached NUMA**), zwykle przez ludzi od sprzętu, bowiem programiści zwykli nazywać ją **sprzętowym DSM** (**Hardware DSM**), ponieważ jest ona sprzętowym odpowiednikiem programowego symulowania współdzielonej pamięci w ramach DSM (patrz punkt 8.3.1)[18].

Jednym z pierwszym wieloprocesorów kategorii NC-NUMA był zbudowany na uniwersytecie Carnegie-Mellon komputer Cm* (choć tę jego nazwę ukuto dopiero po kilku latach eksploatacji), którego architekturę przedstawiono w uproszczonej formie

[18] O podobieństwach między stronicowaniem i cache'owaniem pisaliśmy w punkcie 6.1.10 — *przyp. tłum.*

na rysunku 8.26 (Swan i in., 1977). Komputer ten składa się z kolekcji procesorów LSI-11, z których każdy połączony jest za pośrednictwem prywatnej magistrali ze swą pamięcią lokalną (LSI-11 jest jednoukładową wersją jednostki centralnej minikomputera PDP-11 firmy DEC, popularnego w latach 70. ubiegłego wieku). Każdy procesor połączony jest także z magistralą systemową. Generowane przez procesor odwołanie do pamięci trafia do jego jednostki MMU, gdzie następuje sprawdzenie, czy odnosi się ono do jego lokalnej pamięci. Jeśli tak, odwołanie to realizowane jest poprzez lokalną magistralę; jeśli nie, jest ono kierowane przez magistralę systemową do procesora posiadającego odnośny moduł pamięci — w tym drugim przypadku realizacja odwołania trwa oczywiście dłużej niż w pierwszym. Wykonywanie na danym procesorze programu znajdującego się w zdalnym (z punktu widzenia tego procesora) module pamięci przebiega około 10 razy wolniej w porównaniu z sytuacją, gdy program ten znajduje się w pamięci lokalnej tego procesora.



RYSUNEK 8.26. Wieloprocesor kategorii NUMA bazujący na dwóch rodzajach magistral. Pierwszym komputerem wykonanym w tej architekturze był Cm*

W architekturze NC-NUMA spójność pamięci gwarantowana jest niejako samoczynnie, nie wykorzystuje on bowiem cache'owania, które mogłoby tę spójność zaburzyć. Każde słowo pamięci istnieje tylko w jednym egzemplarzu, bez kopii. Płaconą za to ceną jest silne uzależnienie wydajności komputera od rozłożenia poszczególnych stron między poszczególne moduły pamięci; w związku z tym wieloprocesory NC-NUMA wykorzystują zaawansowane oprogramowanie, przenoszące strony między modułami w celu otrzymania ich optymalnej konfiguracji.

Oprogramowanie to ma zazwyczaj formę demona zwanego **skanerem stron** (*page scanner*) i uruchamianego w odstępach kilkusekundowych. Jego zadaniem jest gromadzenie statystyki odwołań do stron i wykorzystywanie jej do optymalizacji ich rozłożenia między moduły. Jeśli okazuje się, że dana strona zamapowana jest nieoptymalnie (nie w ten moduł, co trzeba), skaner oznacza ją w tablicy stron jako nieobecną; najbliższe do niej odwołanie spowoduje wystąpienie wyjątku braku strony, a w ramach obsługi tego wyjątku strona zamapowana zostanie w „optymalny" moduł. Aby uniknąć zbyt częstego przenoszenia stron i związanego z tym migotania, przyjmuje się regułę, że strona, od momentu zamapowania w dany moduł, nie może zostać usunięta z tego modułu przed upływem określonego a priori interwału czasowego Δt. Zaproponowano wiele algorytmów optymalizacji rozłożenia stron między moduły pamięci, lecz jak pokazuje doświadczenie, żaden z nich nie jest wyraźnie najlepszy w każdych okolicznościach (LaRowe i Ellis, 1991).

Wieloprocesory NUMA ze spójnym cache'owaniem

Wieloprocesory NUMA pozbawione cache'owania niezbyt dobrze poddają się skalowaniu, bowiem w miarę wzrostu liczby procesorów efektywność odwołań do zdalnych modułów pamięci pogarsza się coraz bardziej. Wzbogacenie wieloprocesora o cache'owanie wymaga automatycznie jego wzbogacenia o mechanizmy zapewniające spójność takiego cache'owania. Jak pisaliśmy wcześniej w niniejszym rozdziale, spójność tę można osiągnąć przez podsłuchiwanie magistrali przez procesory — technicznie nieskomplikowane, lecz powyżej pewnej graniczy (liczby procesorów) absolutnie nieopłacalne i kwalifikujące się do zastąpienia lepszymi rozwiązaniami.

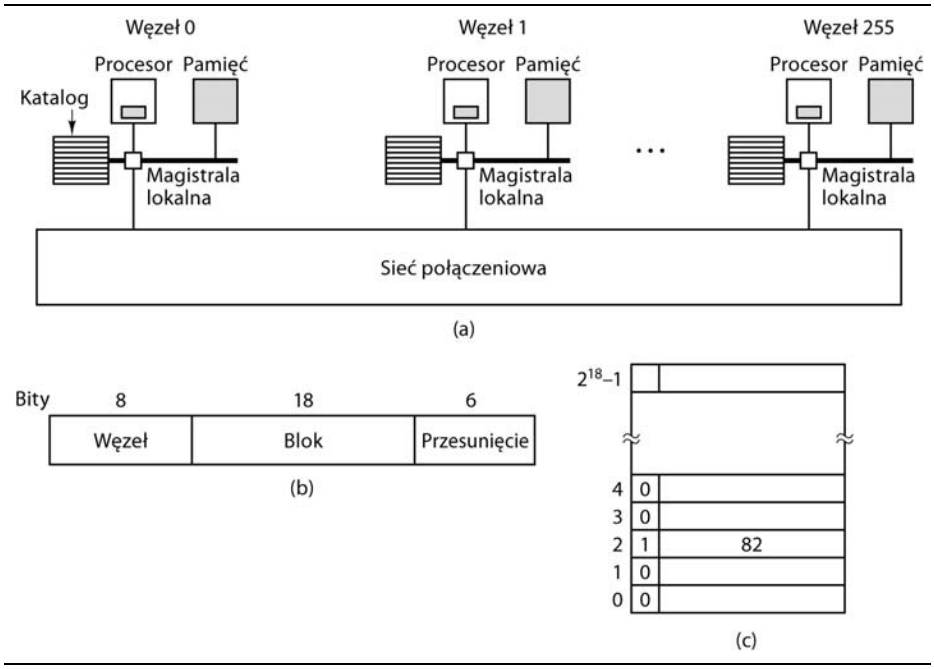
Najpopularniejszym obecnie podejściem do tworzenia dużych wieloprocesorów kategorii CC-NUMA jest koncepcja **wieloprocesorów katalogowych** (*directory-based multiprocessors*). Istotą tej koncepcji jest utrzymywanie bazy danych zawierającej informację o każdym wierszu pamięci podręcznej — jego statusie oraz odpowiadającym mu module i adresie wewnątrz tego modułu. Ponieważ baza ta wykorzystywana jest przy każdym odwołaniu dowolnego procesora do pamięci, jest zrozumiałe, że musi być ona zrealizowana przez ekstremalnie szybki sprzęt specjalnego przeznaczenia, zdolny do realizacji zapytań w ułamku cyklu zegarowego magistrali.

Aby nieco przybliżyć tę koncepcję, rozpatrzmy hipotetyczny wieloprocesor katalogowy z 256 węzłami, z których każdy jest procesorem połączonym za pomocą lokalnej magistrali z prywatną pamięcią RAM o pojemności 16 MB. Łączna pojemność pamięci wynosi więc 16 MB × 256 = 2^{32} bajtów, a jej adresy są statycznie i równomiernie podzielone między kolejne węzły: 0 – 16 MB w węźle 0, 16 MB – 32 MB w węźle 2 i ogólnie $16 * 2^n$ MB – $16 * 2^{n+1}$ MB w węźle n, n = 0, …, 255. Fizyczna przestrzeń adresowa podzielona jest między 2^{26} wierszy pamięci podręcznej o wielkości 64 bajtów każdy. Węzły połączone są ze sobą za pomocą specjalnej sieci, co widoczne jest na rysunku 8.27 (a); sieć ta może mieć topologię kraty (*grid*), supersześcianu (*hypercube*) lub inną. Każdy z węzłów utrzymuje ponadto informację o 2^{18} blokach (64-bajtowych) swej pamięci fizycznej. Załóżmy ponadto tymczasowo, że każdy blok reprezentowany jest w co najwyżej jednej pamięci podręcznej.

Aby zrozumieć, jak to wszystko funkcjonuje, załóżmy, że procesor w węźle `20` wykonuje rozkaz, którego efektem jest odczyt słowa spod fizycznego adresu `0x24000108` (wyliczonego przez MMU w procesie mapowania stron wirtualnych). MMU dzieli ten adres na numer węzła, numer bloku i przesunięcie wewnątrz bloku; w naszym przypadku wartości te równe są (dziesiętnie) 36, 4 i 8. Żądane słowo znajduje się więc w pamięci węzła 36 (nie 20), więc MMU węzła 20 wysyła stosowny komunikat do węzła 36 z zapytaniem, czy blok 4 jego pamięci reprezentowany jest w pamięci cache, a jeśli tak, to w którym węźle[19].

Gdy komunikat ten dociera do węzła 36 (za pośrednictwem sieci połączeniowej), jest kierowany do sprzętowych układów realizujących wspomnianą wcześniej bazę danych w celu sprawdzenia informacji na temat czwartego bloku — przy pomocy specjalnego indeksu odczytywana jest czwarta pozycja spośród 2^{18} pozycji katalogu. Jak widać na rysunku 8.27 (c), czwarty blok **nie** jest reprezentowany w pamięci cache w żadnym węźle, wobec czego jest odczytywany z lokalnej pamięci węzła 36 i przesyłany do węzła 20; co więcej, w czwartą pozycję katalogu w węźle 36 wpisywana jest liczba 20 w celu zaznaczenia, że czwarty blok tego węzła reprezentowany jest w pamięci podręcznej w węźle 20.

[19] Przed chwilą założyliśmy, że może być reprezentowana w co najwyżej jednym węźle — *przyp. tłum.*



RYSUNEK 8.27. Wieloprocesor katalogowy CC-NUMA: (a) wieloprocesor o 256 węzłach; (b) sposób podziału 32-bitowego adresu; (c) katalog w węźle 36.

Prześledźmy teraz losy innego żądania, które MMU węzła 20 kieruje do bloku 2 węzła 36. Jak widzimy na rysunku 8.27 (c), blok ten reprezentowany jest w pamięci podręcznej węzła 82. Oprogramowanie zarządzające wieloprocesorem powinno dokonać skopiowania stosownego wiersza pamięci podręcznej z węzła 82 do węzła 20, unieważnienia tego wiersza w węźle 82 i zmiany wartości 82 na 20 w pozycji 2 katalogu węzła 36 — to wszystko za pomocą stosownych komunikatów. Mimo iż nominalnie mamy do czynienia z „wieloprocesorem z pamięcią współdzieloną", to faktycznie jego funkcjonowanie wiąże się z (ukrytym) przesyłaniem sporej ilości komunikatów.

Spróbujmy teraz oszacować narzut, jaki do ogólnego rozmiaru pamięci fizycznej węzła (16 MB) wnosi obecność w nim katalogu. 2^{18} pozycji 9-bitowych w stosunku do 16 MB to $\frac{2^{18} * 9}{16 * 2^{20} * 8} \approx 1{,}76\,\%$ co generalnie jest do przyjęcia (należy tylko pamiętać, że ta dodatkowa pamięć jest znacznie szybsza, a więc znacznie droższa od podstawowej pamięci 16 MB). Przy 32-bitowych wierszach pamięci cache mielibyśmy 2^{19} pozycji i narzut ten zwiększyłby się dwukrotnie, natomiast przy wierszach 128-bitowych byłby (odwrotnie) o połowę mniejszy.

Oczywistym ograniczeniem opisywanego modelu jest ograniczenie reprezentowania każdego z bloków pamięci w pamięci cache do co najwyżej jednego węzła (czyli po prostu — w co najwyżej jednej pamięci podręcznej). Gdyby dopuścić reprezentowanie jednego bloku w kilku pamięciach podręcznych, musielibyśmy dysponować jakimś sposobem ewidencjonowania tych pamięci, by w przypadku zmodyfikowania stosownego wiersza w jednej z nich móc unieważnić ten wiersz we wszystkich pozostałych.

Jednym ze sposobów tego ewidencjonowania jest umożliwienie zapisywania k pozycji ($k > 1$) w każdej pozycji katalogu, co pozwala reprezentowanie każdego bloku w co najwyżej k pamięciach podręcznych. Inny sposób polega na zastąpieniu numerów węzłów prostą bitmapą, w której każdemu węzłowi odpowiada pojedynczy bit. Jednakże 256 bitów (plus dodatkowe bity pomocnicze) dla każdej z 2^{18} pozycji oznacza narzut niebagatelny, bo przekraczający 50%. Rozwiązaniem mniej pamięciożernym jest łączenie wszystkich informacji o danym bloku z listę wiązaną i umieszczanie nagłówków takich list w pozycjach katalogu. Oznacza to jednak dodatkowe zapotrzebowanie na pamięć na wskaźnikach (w elementach list) i dodatkowy czas konieczny do przeszukiwania list. Każde z trzech wymienionych rozwiązań posiada swe zalety i wady, wszystkie natomiast znalazły zastosowanie w rzeczywistych, działających systemach.

Inne usprawnienie w projekcie katalogów polega na utrzymywaniu informacji o tym, które linie cache są „czyste" (*clean*), czyli odpowiadają **aktualnym** blokom w pamięci RAM, a które „naruszone" (*dirty*), czyli odpowiadające **nieuaktualnionym** blokom w pamięci RAM. Żądanie odnoszące się do „czystego" wiersza może być zaspokojone wprost z pamięci RAM, natomiast żądanie odnoszące się do wiersza „naruszonego" wymaga wpierw odnalezienia węzła, którego pamięć podręczna zawiera ten wiersz, gdyż tylko tam (a nie w pamięci RAM) znajduje się aktualna informacja. Rozwiązanie to nie przynosi jednak zauważalnych korzyści w sytuacji, gdy dopuszczalna jest co najwyżej jedyna kopia bloku pamięci RAM w pamięci cache (jak na rysunku 8.27) ponieważ każde nowe żądanie dostępu do określonego bloku wiąże się z wysłaniem komunikatu unieważniającego jego kopię w pamięci cache.

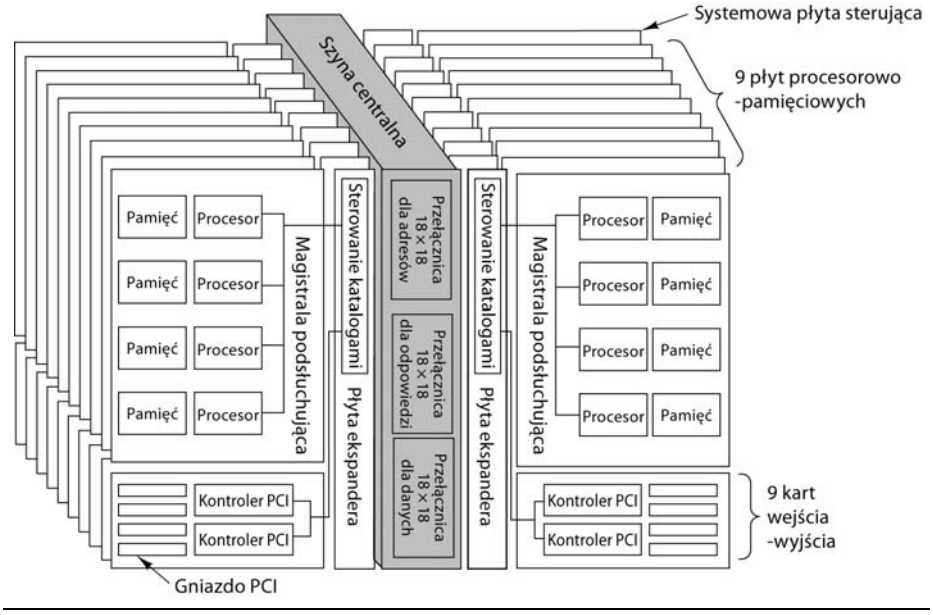
Oczywiście zmodyfikowanie (w opisanym wariancie) wiersza w pamięci cache wymaga przesłania informacji o tym do macierzystego węzła, czyli węzła, którego pamięć RAM zawiera blok reprezentowany przez ten wiersz. Jeżeli możliwe jest istnienie wielu kopii tego bloku, to dodatkowo unieważnić trzeba wszystkie kopie oprócz zmodyfikowanej. Wiąże się to z ryzykiem wystąpienia zjawiska wyścigu (patrz punkt 6.3.2) i konieczne jest zastosowanie specjalnego protokołu, by ryzyko to wyeliminować. I tak na przykład, w celu zmodyfikowania współdzielonego (*shared*) wiersza cache odpowiedni węzeł mógłby wysłać żądanie wyłącznego (*exclusive*) dostępu do tego wiersza **przed** jego zmodyfikowaniem, a to automatycznie mogłoby powodować jego unieważnienie w innych węzłach, jeszcze przed przyznaniem wyłącznego dostępu. Dyskusja na temat innych sposobów optymalizowania wydajności wieloprocesorów CC-NUMA znajduje się w pracy (Stenstrom i in., 1997).

Wieloprocesor NUMA Sun Fire E25K

Przykładami wieloprocesorów NUMA z pamięcią współdzieloną są komputery rodziny Sun Fire firmy Sun Microsystems. W rodzinie tej obecnych jest kilka różnych modeli, my skoncentrujemy się na modelu E25K, składającym się z 72 procesorów UltraSPARC IV. Każdy z tych procesorów jest de facto *dwuprocesorem*, czyli parą procesorów UltraSPARC III Cu współdzielących pojedynczą pamięć operacyjną i pojedynczą pamięć podręczną. Model E15K jest podobny, złożony jednak z pojedynczych procesorów UltraSPARC III. Istnieją jeszcze prostsze modele tej rodziny, nas jednak interesuje głównie praca i współpraca poszczególnych procesorów.

System E25K składa się z 4, 9 lub 18 płyt głównych typu *Uniboard*, z których każda zawiera kartę procesorowo-pamięciową, kartę wejścia-wyjścia z czterema gniazdami PCI i płytę ekspandera spajającą te dwie karty oraz łączącą je z szyną centralną

(*Centerplane*). Szyna ta mechanicznie utrzymuje wszystkie płyty oraz realizuje logikę przełączania. Każda karta procesorowo-pamięciowa E25K zawiera cztery dwuprocesory i cztery moduły pamięci po 8 GB każdy, czyli osiem procesorów i 32 GB pamięci RAM (E15K zawiera cztery procesory 32 GB pamięci RAM). Pełna konfiguracja E25K (z 18 płytami *Uniboard*) obejmuje więc 144 procesory, 576 GB pamięci RAM i 72 gniazda PCI — co obrazowo pokazano na rysunku 8.28. Co ciekawe, liczba 18 płyt głównych podyktowana została względami nie projektowymi, a praktycznymi: przy większej liczbie płyt komputera nie dałoby się wnieść (bez demontowania) przez otwór drzwiowy. W przeciwieństwie do programistów myślących kategoriami „zer i jedynek", inżynierowie od sprzętu, jak widać, muszą także troszczyć się o procedury wnoszenia komputera do budynku.



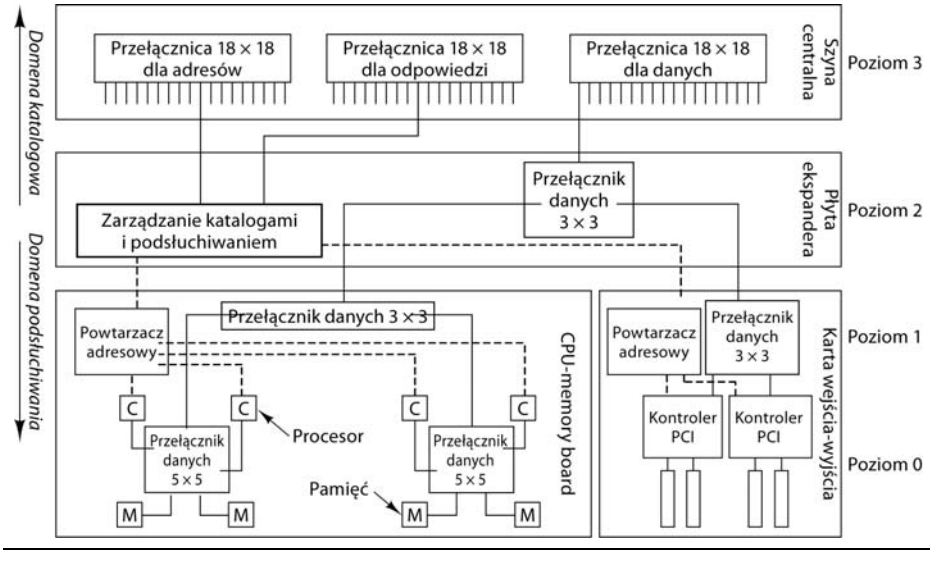
RYSUNEK 8.28. Wieloprocesor E25K firmy Sun Microsystems

Szyna centralna (*Centerplane*) składa się z trzech przełącznic krzyżowych 18 × 18 łączących ze sobą 18 płyt głównych. Jedna z tych przełącznic dedykowana jest liniom adresowym, druga związana jest z przesyłaniem odpowiedzi, trzecia natomiast odpowiedzialna jest za transfer danych. Oprócz 18 ekspanderów do szyny centralnej przyłączona jest także systemowa płyta sterująca. Zawiera ona tylko jeden procesor i stanowi interfejs do napędów CD-ROM, taśm, linii transmisji szeregowej i innych urządzeń zewnętrznych niezbędnych do uruchamiania, utrzymywania i kontrolowania pracy systemu.

Sercem każdego wieloprocesora jest jego podsystem zarządzania pamięcią. Jak połączyć 144 procesory z modułami pamięci rozproszonej? Rozwiązania oczywiste — duża, współdzielona magistrala podsłuchująca lub przełącznica krzyżowa 144 × 72 — nie są dobrymi pomysłami: pojedyncza magistrala to zbyt wąskie gardło na tyle procesorów, zaś tak duża przełącznica byłaby zbyt kosztowna i zbyt trudna w konstrukcji. Wieloprocesory takie jak E25K wymagają projektów bardziej wyrafinowanych.

Na poziomie płyty głównej lokalne procesory prowadzą podsłuchiwanie wszystkich odwołań do bloków, które aktualnie są na tej płycie cache'owane. Gdy procesor żąda dostępu do pamięci, dokonuje najpierw konwersji adresu wirtualnego na adres fizyczny (adresy fizyczne są 43-bitowe, jednakże ze względu na ograniczenie liczby płyt głównych do 18 — z powodów wcześniej opisanych — ich zakres ograniczony jest do 576 GB). Jeśli żądany blok znajduje się w pamięci podręcznej procesora, jest stamtąd pobierany. W przeciwnym razie układy podsłuchujące sprawdzają, czy jest on obecny w którejkolwiek pamięci podręcznej na płycie głównej. Jeśli tak, zostaje z tej pamięci pobrany; jeśli nie, żądanie przekazywane jest do (opisywanej poniżej) krzyżowej przełącznicy adresowej 18 × 18. Układy podsłuchujące zdolne są generować po jednym sygnale w każdym cyklu zegarowym; zegar systemowy pracuje z częstotliwością 150 MHz, więc w ciągu sekundy w obrębie jednej płyty sygnałów takich wystąpić może 150 milionów, zaś w obrębie całego systemu — 18 × 150 milionów = 2,7 miliarda.

Mimo iż pod względem logicznym układy podsłuchujące tworzą magistralę — jak wynika to z rysunku 8.28 — pod względem fizycznym są one zorganizowane w strukturę drzewiastą, zdolną do transmitowania sygnałów w obydwu kierunkach swej hierarchii. Adres generowany przez procesor lub magistralę PCI kierowany jest do powtarzacza adresowego (*address repeater*) za pośrednictwem połączenia dwupunktowego, jak widać to na rysunku 8.29. Dwa powtarzacze adresowe połączone są ze sobą na płycie ekspandera, skąd sygnały rozchodzą się w głąb drzewa w celu poszukiwania odpowiedniego układu. W ten sposób uniknięto konieczności konstruowania magistrali integrującej trzy płyty.



RYSUNEK 8.29. Czteropoziomowa struktura połączeń wewnętrznych wieloprocesora E25K. Liniami przerywanymi oznaczono ścieżki adresowe, liniami ciągłymi — ścieżki danych

Przepływ danych zrealizowany jest w postaci czteropoziomowego połączenia widocznego na rysunku 8.29; tak duży stopień komplikacji podyktowany został względami wydajności wieloprocesora. Na poziomie 0 pary dwuprocesorów łączone

są z parami modułów pamięci za pomocą małych przełącznic krzyżowych, zapewniających jednocześnie połączenie z poziomem 1; przełącznice te są specjalizowanymi układami ASIC. We wszystkich tych przełącznicach dostępne są wszystkie wejścia, zarówno wierszowe, jak i kolumnowe, nie wszystkie jednak ich kombinacje są wykorzystywane (bo nie wszystkie mają sens). Wszystkie układy przełączające na płycie zbudowane są z przełącznic 3×3.

Jak wcześniej pisaliśmy, każda płyta główna składa się z trzech komponentów: karty procesorowo-pamięciowej, karty wejścia-wyjścia i spajającej je płyty ekspandera. Znajdująca się na płycie ekspandera przełącznica 3×3 należy już do poziomu drugiego i służy do połączenia pamięci operacyjnej z portami wejścia wyjścia (do których dostęp w procesorach UltraSPARC zorganizowany jest na zasadzie mapowania w adresy wirtualne). Wszystkie transfery danych, zarówno do (z) pamięci, jak i do (z) portów wejścia-wyjścia przechodzą przez układy przełączające na poziomie 2. Transfer danych między płytami głównymi odbywa się za pośrednictwem krzyżowej przełącznicy danych 18×18 na poziomie 3. Dane transferowane są w porcjach 32-bitowych w jednym cyklu zegara, tak więc przesłanie standardowej jednostki 64-bitowej wymaga dwóch cykli.

Po zapoznaniu się z organizacją komponentów wieloprocesora zobaczmy teraz, jak funkcjonuje jego pamięć współdzielona. Na najniższym poziomie organizacji 576 GB przestrzeń adresowa podzielona jest na 2^{29} bloków 64-bajtowych, stanowiących niepodzielną (atomową) jednostkę transferu. Każdemu z tych bloków przypisana jest macierzysta (*home*) płyta główna, na której jest on fizycznie obecny, jeśli nie jest reprezentowany w żadnej pamięci podręcznej. W danej chwili większość bloków obecnych jest na swych płytach macierzystych, jednakże każdy z procesorów zażądać może dostępu do dowolnego bloku pamięci na dowolnej płycie; procesor ten doprowadza wówczas do pojawienia się odnośnego bloku w swej własnej pamięci podręcznej (o ile blok ten już się w niej nie znajduje) i stamtąd pobiera jego kopię. Składowe procesory każdego dwuprocesora wspólnie wykorzystują jedną pamięć podręczną i współdzielą dostęp do wszystkich bloków stanowiących jej zawartość.

Każdy blok pamięci oraz każdy wiersz pamięci podręcznej każdego procesora znajdować się mogą w jednym z trzech następujących stanów:

(1) Dostęp na wyłączność (do zapisu).
(2) Dostęp współdzielony (do odczytu).
(3) Brak informacji.

Gdy procesor zamierza odczytać słowo z pamięci lub w je niej zapisać, rozpoczyna od sprawdzenia, czy w jego własnej pamięci podręcznej znajduje się blok zawierający to słowo. Jeśli wynik tego sprawdzenia jest negatywny, procesor wysyła komunikat rozgłaszający (*broadcast*) w obrębie swej płyty głównej w celu sprawdzenia, czy żądany blok znajduje się w którejś z pamięci podręcznych na tej płycie. Jeśli znajduje się i jeśli zawierająca go linia jest w stanie dostępu na wyłączność, blok zostaje przesłany do procesora, zaś linia zostaje unieważniona.

Jeśli układy podsłuchujące nie znajdą na płycie żądanej linii, bądź znajdą ją, lecz będzie się ona znajdować w stanie dostępu współdzielonego, do macierzystej płyty bloku wysłane zostaje żądanie określenia statusu bloku; status ten kodowany jest w bitach ECC bloku, jest więc natychmiast dostępny. Jeśli blok jest współdzielony przez jedną lub więcej płyt głównych bądź nie jest współdzielony w ogóle, jest on transmitowany ze swej płyty macierzystej, poprzez przełącznicę danych, do pamięci podręcznej procesora żądającego dostępu, w dwóch cyklach zegara.

Jeśli procesor żąda dostępu do bloku tylko w celu jego odczytu, w odpowiedniej pozycji katalogu na płycie macierzystej bloku odnotowany zostaje fakt utworzenia kolejnej kopii bloku w jednej z pamięci podręcznych. Jeśli jednak procesor zamierza wykonać na bloku operację zapisu, do wszystkich płyt głównych zawierających kopie tego bloku (w pamięciach podręcznych) wysyłany jest komunikat z żądaniem unieważnienia tych kopii. W efekcie jedyną kopią bloku pozostaje kopia w pamięci podręcznej na płycie, z której procesor wysłał żądanie dostępu.

Rozpatrzmy teraz przypadek, gdy żądany blok znajduje się w stanie dostępu wyłącznego i jego jedyna kopia znajduje się zarówno poza płytą macierzystą, jak i poza płytą zawierającą procesor żądający dostępu. Gdy płyta macierzysta otrzyma zapytanie o status bloku, odsyła pytającemu procesorowi komunikat określający położenie wspomnianej kopii; procesor ten wysyła następnie żądanie dostępu już do właściwej płyty, w efekcie czego na jego własnej płycie tworzona jest kolejna kopia bloku. Gdy dostęp ten związany jest z odczytem, obydwie kopie bloku oznakowane zostają jako współdzielone, podobnie jak sam blok na swej płycie macierzystej; gdy dostęp ten dotyczy zapisu, poprzednia kopia bloku zostaje unieważniona i jedyną kopią bloku pozostaje kopia na płycie procesora żądającego dostępu (oznakowana dla dostępu na wyłączność).

Ponieważ w pamięci na każdej płycie głównej może znajdować się 2^{29} bloków pamięci, utrzymywanie kompletnej informacji o nich wymagałoby tyleż pozycji w katalogu. Katalogi (przeszukiwane w sposób asocjacyjny) są jednak zwykle znacznie mniejsze i dla niektórych bloków nie ma w nich żadnej informacji. Uzyskanie tej informacji wymaga więc wysłania (przez płytę główną odnośnego bloku) komunikatu rozgłaszającego do pozostałych 17 płyt głównych; zadanie odebrania odpowiedzi (i przy okazji uaktualnienia katalogu) przejmuje na siebie przełącznica krzyżowa odpowiedzi. Przez oddzielenie jej od pozostałych przełącznic — dla adresów i danych — przepustowość systemu zostaje znacznie zwiększona.

Dzięki rozproszeniu obciążenia na wiele urządzeń zlokalizowanych na różnych płytach i kartach, komputer Sun Fire E25K zdolny jest osiągać wyjątkowo dużą wydajność. Oprócz wspomnianych wcześniej 2,7 miliardów sygnałów podsłuchujących (w ciągu sekundy), centralna szyna zdolna jest realizować do dziewięciu równoległych transferów, między dziewięcioma parami połączonych płyt głównych. Jako że przełącznica danych jest 32-bitowa, w ciągu jednego cyklu zegara przez szynę centralną może przepływać $32 \times 9 = 288$ bajtów danych; przy częstotliwości zegara 150 MHz daje to zagregowane pasmo transferu (dla zdalnych dostępów) na poziomie 150 MHz $\times$ 288 = ponad 40 GB na sekundę. Jeśli oprogramowanie napisane jest w ten sposób, że większość dostępów do pamięci ma charakter lokalny, efektywny transfer systemowy może być jeszcze większy.

Czytelnikom, na których komputer Sun Fire E25K zrobił oszałamiające wrażenie, możemy polecić prace (Charlesworth, 2002) i (Charlesworth, 2001) zawierające mnóstwo technicznych informacji na jego temat.

8.3.5. Wieloprocesory COMA

Słabą stroną wieloprocesorów NUMA jest fakt, że odwołania do zdalnych modułów pamięci realizowane są znacznie wolniej niż odwołania do modułów lokalnych. W przypadku wieloprocesorów CC-NUMA różnica ta jest w pewnym stopniu maskowana

dzięki cache'owaniu, niemniej jednak w przypadku dużego rozmiaru zdalnie transmitowanych danych chybienia w pamięciach podręcznych stają się częste i wydajność całego systemu gwałtownie się pogarsza.

Mamy więc sytuację następującą. Wieloprocesory UMA są wydajne, lecz ograniczone w skali i rozbudowie oraz niesamowicie drogie. Wieloprocesory NC-NUMA skalują się znacznie lepiej, lecz wymagają skomplikowanych zabiegów optymalizujących rozmieszczenie stron, z wątpliwymi nieraz rezultatami. Z kolei wieloprocesory kategorii CC-NUMA, jak Sun Fire E25K, mogą doświadczać znaczącego pogorszenia wydajności, jeśli wiele procesorów sięga jednocześnie do odległych danych. I tak źle, i tak niedobrze.

Jednym z pomysłów zmierzających do przełamania tego fatum jest wykorzystanie głównej pamięci każdego procesora jako dużej pamięci podręcznej. Istotną cechą opartych na tym pomyśle wieloprocesorów, oznaczanych akronimem **COMA** (**Cache Only Memory Access**, dostęp wyłącznie przez pamięć cache), jest rezygnacja z przypisywania poszczególnych fragmentów (stron) fizycznej przestrzeni adresowej do określonych procesorów macierzystych, jak w przypadku maszyn NUMA; tak naprawdę strony te w ogóle tracą swe znaczenie.

W zamian fizyczna przestrzeń adresowa podzielona zostaje na linie cache, które mogą na żądanie migrować po systemie. Bloki odpowiadające liniom cache nie mają macierzystych procesorów, podobnie jak nomadowie w niektórych krajach Trzeciego Świata mają swe domy tam, gdzie się aktualnie znajdują. Pamięć zawierająca potrzebny wiersz (blok) nosi nazwę **pamięci oddziaływania** (*attraction memory*). Wykorzystanie pamięci głównej jako wielkiego cache znacząco zwiększa prawdopodobieństwo trafienia, a w konsekwencji wydajność.

Niestety, nie ma nic za darmo. Systemy COMA niosą ze sobą dwa nowe problemy:

- W jaki sposób lokalizować wiersze cache?
- Jeżeli wiersz usunięty zostanie z pamięci cache, co stanie się, gdy będzie on ostatnią (jedyną) kopią bloku?

Pierwszy problem wynika z samej specyfiki stronicowania: gdy wyliczony zostaje adres fizyczny i odpowiadający mu wiersz nie znajduje się w **prawdziwej** pamięci podręcznej, to nie ma prostego sposobu na to, by stwierdzić, czy w ogóle znajduje się on w pamięci głównej i jeżeli tak, to w której. Sprzęt stronicujący nic tu nie pomoże, ponieważ z jego perspektywy każda strona składa się z ciągłej sekwencji wierszy, które teraz sekwencję tworzyć przestały i rozproszone zostały niezależnie po całym systemie. Co więcej, wiedząc, że danego wiersza nie ma w pamięci głównej, nie sposób powiedzieć, gdzie może się on znajdować. Nie istnieje bowiem procesor macierzysty, który można by o to zapytać.

Konieczne są więc nowe mechanizmy, na przykład wprowadzenie znaczników wierszy i pamięć asocjacyjna efektywnie porównująca znacznik żądanego wiersza ze znacznikami wierszy znajdujących się w danej pamięci głównej. To rozwiązanie wymaga dodatkowego sprzętu.

Całkowicie odmienne rozwiązanie nawiązuje do tradycyjnego stronicowania i polega na możliwości przechowywania w pamięci fizycznej nie całych stron, a jedynie ich wybranych wierszy; odróżnianie wierszy obecnych od nieobecnych odbywa się za pomocą specjalnej bitmapy, osobnej dla każdej strony. Zgodnie z tym rozwiązaniem zwanym **prostym COMA** (*simple COMA*), jeżeli wiersz znajduje się w pamięci, musi

on zajmować właściwą pozycję wewnątrz swej strony, jeżeli nie ma żądanego wiersza, to każda próba dostępu do niego spowoduje wyjątek i oprogramowanie będzie mogło wiersz ten zlokalizować.

Prowadzi to natychmiast do problemu znajdowania wierszy, które są faktycznie zdalne. Jednym z pomysłów na jego rozwiązanie jest przypisanie każdej stronie procesora macierzystego w tym sensie, że w katalogu związanym z tym procesorem przechowywana będzie informacja o stronie (a nie sama strona). Odnośną stronę można zlokalizować, wysyłając zapytanie do wspomnianego procesora macierzystego. Innym pomysłem jest zorganizowanie pamięci w strukturę drzewiastą i poszukiwanie linii przez wędrówkę komunikatu od liści w kierunku korzenia.

Drugi z sygnalizowanych na wstępie problemów wiąże się z „wymiataniem" wierszy z pamięci podręcznej w związku z wprowadzaniem nowych, podobnie jak w pamięciach wieloprocesorów NUMA. Jeśli w wyniku tego procesu usunięty zostanie wiersz zawierający **jedyną** kopię bloku, pojawią się problemy.

Problemów tych można uniknąć, sprawdzając (w katalogu) przed usunięciem wiersza, czy gdziekolwiek istnieje jeszcze jego kopia. Inny pomysł polega na wyróżnieniu dokładnie jednego wiersza jako głównej kopii bloku i kategorycznym zabronieniu jego usuwania; uwalnia to od konieczności przeszukiwania katalogu. Tak czy inaczej, maszyny COMA wydają się wielce obiecujące, jeśli chodzi o wydajność (lepszą niż w przypadku CC-NUMA), jednakże jak na razie zbudowano niewiele takich maszyn i doświadczenia w tej dziedzinie są w związku z tym ograniczone. Dwiema pierwszymi konstrukcjami tej kategorii były KSR-1 (Burkhardt i in., 1992) i Data Diffusion Machine (Hagersten i in., 1992). Znacznie nowszym projektem jest SDA ARC (Eschmann i in., 2002).

8.4. WIELOKOMPUTERY PRZEKAZUJĄCE KOMUNIKATY

Zgodnie z klasyfikacją przedstawioną na rysunku 8.18 komputery typu MIMD dzielą się na dwie zasadnicze kategorie: wieloprocesory i wielokomputery. Jak widzieliśmy, wieloprocesory jawią się systemom operacyjnym jako maszyny ze współdzieloną pamięcią, która dostępna jest dla wszystkich procesorów za pośrednictwem elementarnych rozkazów sprzętowych w rodzaju `LOAD` i `STORE`. Dostęp poszczególnych procesorów do wspólnej pamięci może być zorganizowany na różne sposoby: na bazie magistral podsłuchujących, przełącznic krzyżowych, wielostopniowych sieci przełączających i według rozmaitych schematów bazujących na katalogach. Niezależnie od konkretnej konstrukcji wieloprocesora, pisane dla niego oprogramowanie wykorzystywać może fakt swobodnego dostępu każdego procesora do dowolnego adresu pamięci, bez względu na implementację tego dostępu czy topologię, według której zorganizowane są moduły pamięci. Czyni to wieloprocesory atrakcyjnymi dla programistów.

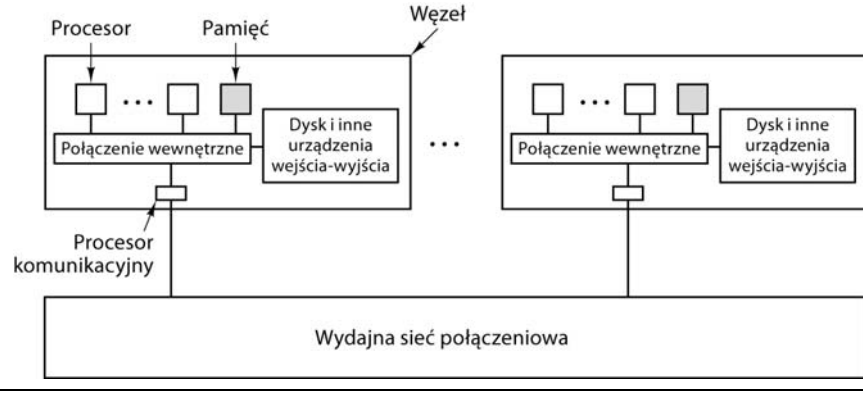
Z drugiej jednak strony, wieloprocesory to komputery z niewielką stosunkowo liczbą procesorów, z trudnością poddające się skalowaniu przy wzroście ich liczby powyżej pewnej (niewielkiej) granicy. Zwróćmy uwagę na to, jak dużo sprzętu upakować musiała firma Sun Microsystems w komputerze Sun Fire E25K, a przecież zawiera on tylko 144 procesory (czy 72 dwuprocesory, jak kto woli). To właśnie

ograniczenie jest decydującym czynnikiem popularności wielokomputerów — w dalszym ciągu rozdziału opiszemy przykład wielokomputera łączącego 65 536 procesorów. Dużo jeszcze czasu będzie musiało upłynąć, nim ktoś zdecyduje się na stworzenie komercyjnego wieloprocesora o tej skali, w każdym razie będące wówczas w użyciu wielokomputery z pewnością zawierać będą miliony procesorów.

Swobodny dostęp do całej pamięci wieloprocesora ma też swoją cenę w postaci olbrzymiego obciążenia modułów pamięci, magistral i katalogów; równoczesny dostęp wielu procesorów do tych samych zmiennych może efektywność pracy całego systemu w poważnym stopniu obniżyć.

Już w poprzednim punkcie podaliśmy praktyczny przejaw zasady „coś za coś”: by zwiększyć granicę skalowalności wieloprocesora, należało zrezygnować z jednolitego zarządzania całą pamięcią współdzieloną, dopuszczając, by czas dostępu procesorów do zdalnych modułów pamięci mógł być znacząco większy niż czas dostępu do modułów lokalnych. Stosując konsekwentnie tę zasadę i rezygnując w ogóle z pamięci współdzielonej, dochodzimy do koncepcji wielokomputera, którego każdy procesor ma bezpośredni dostęp jedynie do swej własnej pamięci, niedostępnej (w sposób bezpośredni) dla innych procesorów. Zmienia to w sposób zasadniczy sposób tworzenia oprogramowania dla wielokomputerów, bowiem procesor w celu odczytania lub zapisania słowa w zdalnej pamięci posługiwać się musi — zamiast elementarnymi rozkazami `LOAD` i `STORE` — prymitywami systemu operacyjnego w rodzaju `send` i `receive`.

Każdy węzeł wielokomputera składa się zwykle z jednego lub kilku procesorów, pewnej ilości pamięci RAM dostępnej dla tych procesorów (i tylko dla nich), dysku i ewentualnie innych urządzeń wejścia-wyjścia oraz procesora komunikacyjnego. Procesory komunikacyjne wszystkich węzłów połączone są za pomocą szybkich sieci, omawianych w punkcie 8.1.2 i wykorzystujących rozmaite topologie, schematy przełączania i algorytmy routingu. Tym, co dla wszystkich wielokomputerów wspólne, jest sposób nawiązywania połączeń między węzłami: gdy wykonywany przez któryś z procesorów węzła program wykonuje systemową funkcję `send`, zauważa to procesor komunikacyjny i wysyła do docelowego komputera (zwykle po uzyskaniu stosownych uprawnień) blok zawierający odpowiedni zestaw danych. Schemat modelowego wielokomputera przedstawiono poglądowo na rysunku 8.30.



RYSUNEK 8.30. Struktura modelowego wielokomputera

8.4.1. Sieci połączeniowe

Jak widać na rysunku 8.30, poszczególne węzły wielokomputera spajane są za pomocą szybkiej sieci połączeniowej, której przyjrzymy się teraz nieco dokładniej. Interesujące jest to, że sieć ta jest elementem upodabniającym do siebie wieloprocesory i wielokomputery, bowiem w dużych wieloprocesorach poszczególne procesory też muszą być efektywnie łączone z rozproszonymi modułami pamięci. Tym samym treść niniejszego punktu, choć znajdującego się w podrozdziale traktującym o wielokomputerach, w dużej części stosuje się także do wieloprocesorów.

Podstawowym powodem, dla którego sieci połączeniowe wieloprocesorów i sieci połączeniowe wielokomputerów są tak do siebie podobne, jest oparcie ich obydwu na przesyłaniu komunikatów. Nawet w „samodzielnym" komputerze, gdy jego jedyny procesor chce odczytać słowo z pamięci (lub zapisać je w niej), także musi odpowiednio ustawić stan linii na magistrali i oczekiwać na odpowiedź, czyli dostarczenie zawartości. W dużych wieloprocesorach komunikacja między procesorami a zdalnymi modułami pamięci odbywa się za pomocą ustalonych komunikatów, zwanych **pakietami**, niosących zarówno treść żądania, jak i zawartość stanowiącą treść odpowiedzi.

Topologie

Pod pojęciem topologii sieci rozumiemy sposób połączenia jej węzłów, na przykład w pierścień lub kratę. Topologia konkretnej sieci może być wyrażona w postaci grafu, którego wierzchołki reprezentują węzły sieci, a krawędzie — połączenia między tymi węzłami. W matematyce liczba krawędzi spotykających się w danym wierzchołku („incydentnych z wierzchołkiem") nazywa się **stopniem** (*degree*) tego wierzchołka; analogicznym pojęciem odnoszącym się do węzła sieci jest jego **obciążenie wyjściowe** (*fanout*) równe liczbie wychodzących z tego węzła połączeń. Im większa jest ta wartość, tym bardziej sieć odporna jest na awarie, czyli zdolna jest funkcjonować nawet w przypadku uszkodzenia pewnych połączeń. Gdy mianowicie z każdego węzła sieci wychodzi k połączeń, to jeżeli jej okablowanie wykonane jest prawidłowo, może ona funkcjonować w pełni poprawnie nawet wówczas, gdy tylko jedno spośród tych k połączeń działać będzie prawidłowo.

Inną własnością sieci połączeniowej (lub reprezentującego ją grafu) jest jej **średnica** (*diameter*). Jeżeli będziemy mierzyć odległość między dwoma węzłami liczbą krawędzi, przez które należy koniecznie przejść, by dostać się z jednego węzła do drugiego, średnicą grafu jest odległość między parą węzłów najbardziej odległych od siebie. Średnica sieci połączeniowej ma ścisły związek z jej opóźnieniem w przypadku pesymistycznym: przebycie przez pakiet każdego połączenia wymaga pewnego czasu, a więc łączny czas wędrówki pakietu jest tym większy, im więcej krawędzi musi on przebyć na swej drodze. Sieci o mniejszej średnicy są więc sieciami szybszymi. Wielkością pokrewną średnicy sieci jest **średnia odległość** między parą jej węzłów, pozostająca w ścisłym związku ze średnim czasem transmisji pakietu.

Innym ważnym parametrem sieci jest jej **szerokość połowienia** (*bisection width*), równa minimalnej liczbie krawędzi, które należy usunąć, by sieć „rozpadła się" na dwie *równe*, odrębne sieci. Dodając do siebie maksymalną przepustowość tych usuniętych krawędzi, otrzymujemy wartość tzw. **przepustowości przepołowienia** (*bisection bandwidth*), będącej miernikiem pojemności sieci, czyli określającej ilość danych, jakie sieć ta jest w stanie przetransmitować w ciągu sekundy. Jeżeli na przykład

przepustowość ta równa jest 800 b/s, to jest to jednocześnie wartość rzeczywistej przepustowości sieci w skrajnie niekorzystnych warunkach. Wielu projektantów skłonnych jest uważać przepustowość przepołowienia za jedną z najważniejszych metryk sieci połączeniowych, stąd maksymalizacja tej metryki jest zwykle jednym z podstawowych celów projektowych.

Kolejną cechą sieci połączeniowych jest ich **wymiarowość** (*dimensionality*). Na potrzeby niniejszego rozdziału zdefiniujemy ją jako liczbę możliwych wyborów[20] na drodze od ustalonego węzła początkowego do ustalonego węzła końcowego. Jeśli droga ta jest jedyna, żadnego wyboru nie ma i sieć jest wówczas zerowymiarowa. Jeśli możliwy jest jeden wybór binarny, na przykład kierunek wschodni albo zachodni, sieć jest jednowymiarowa; jeśli możliwe są dwa takie wybory, na przykład jeden w kierunku poziomym (wschód albo zachód) i jeden w pionowym (północ albo południe), wymiarowość sieci równa jest 2.

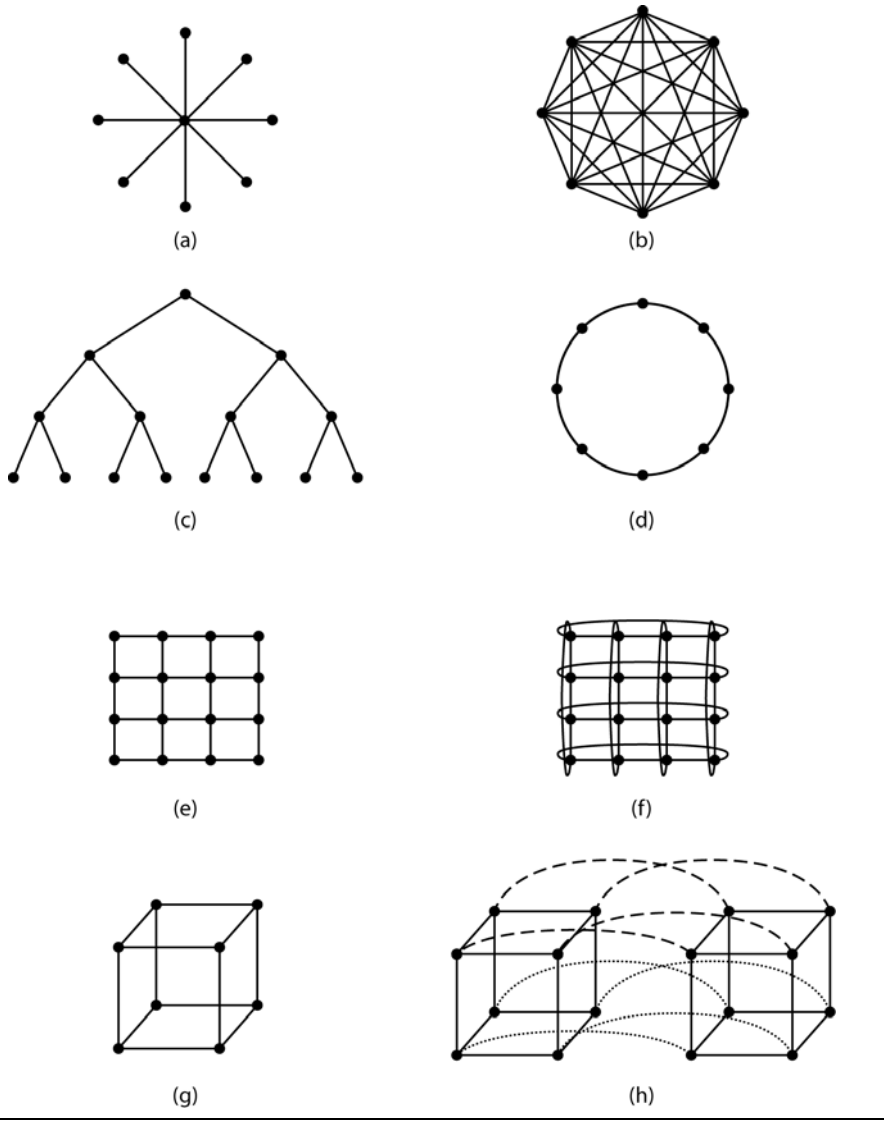
Kilka przykładowych, najpopularniejszych i najczęściej stosowanych topologii sieci pokazano na rysunku 8.31; dla uproszczenia pokazaliśmy jedynie grafy reprezentujące sieci o konkretnych topologiach, abstrahując od struktury (procesorów, pamięci, dysków itp.) poszczególnych węzłów. W części (a) rysunku widzimy zerowymiarową topologię **gwiazdy** (*star*), w ramach której procesory i pamięci połączone są za pośrednictwem centralnego przełącznika. Przełącznik ten jest jednocześnie wąskim gardłem konfiguracji, a prostota koncepcji idzie w parze z bardzo słabą odpornością na awarie — uszkodzenie centralnego przełącznika unieruchamia automatycznie całą sieć.

Inną zerowymiarową topologią jest widoczne w części (b) **pełne połączenie** (*full interconnection*). Każda para węzłów jest bezpośrednio ze sobą połączona, co maksymalizuje przepustowość przepołowienia, minimalizuje średnicę i czyni sieć wyjątkowo odporną na awarie (nawet uszkodzenie sześciu z każdych siedmiu połączeń wychodzących z węzła nie przeszkadza w dalszym poprawnym funkcjonowaniu sieci. Wadą tej konfiguracji jest jej koszt wynikający z *kombinatorycznej* (względem liczby węzłów) liczby połączeń: w przypadku k węzłów liczba połączeń równa jest $\frac{k(k-1)}{2}$.

W części (c) widoczna jest topologia **drzewa** (*tree*). Przepustowość przepołowienia równa jest dla struktury drzewiastej przepustowości pojedynczego łącza. Ponieważ obciążenie łączy zwiększa się wraz ze zbliżaniem się do korzenia drzewa, położone w pobliżu korzenia węzły stają się wąskim gardłem dla przepływu danych. Problem ten rozwiązuje się najczęściej przez umieszczanie szybszych łączy w obszarze zbliżonym do korzenia, tak by w miarę oddalania się od korzenia szybkość ta malała w tempie wykładniczym. Jeżeli więc na przykład łącza na poziomie liści mają przepustowość b, to łącza na poziomie bezpośrednio wyższym mają tę przepustowość dwukrotnie większą ($2b$), łącza na kolejnym poziomie — czterokrotnie większą ($4b$) itp. Koncepcja ta, zwana modelem **grubego drzewa** (*fat tree*), wykorzystana została w konstrukcji kilku komercyjnych wielokomputerów, miedzy innymi w (nieistniejącym już) modelu CM serii Thinking Machines.

Widoczny w części (d) **pierścień** (*ring*) jest topologią jednowymiarową, ponieważ pakiet może poruszać się według jednej z dwóch orientacji: „zegarowej" i „antyzegarowej". W części (e) widzimy topologię dwuwymiarową, zwaną **siatką** (*grid*) lub **kratą** (*mesh*) i stosowaną w wielu rozwiązaniach komercyjnych. Topologia ta jest wysoce regularna, łatwo poddaje się skalowaniu, w wyniku którego jej średnica rośnie stosunkowo wolno, bo proporcjonalnie do pierwiastka kwadratowego z liczby

[20] Pod pojęciem „wyboru" autor rozumie tu wybór spośród **dwóch** możliwości — *przyp. tłum.*



RYSUNEK 8.31. Rozmaite topologie sieci; kropki oznaczają przełączniki — procesory i pamięci nie zostały uwzględnione: (a) gwiazda; (b) połączenie kompletne; (c) drzewo; (d) pierścień; (e) siatka (krata); (f) podwójny torus; (g) sześcian (kostka); (h) czterowymiarowy hipersześcian (hiperkostka)

węzłów. Odmianą siatki jest **podwójny torus** (*double torus*) widoczny w części (f) — powstaje ona wskutek bezpośredniego połączenia skrajnych węzłów siatki, co nie tylko zmniejsza średnicę, lecz także czyni sieć bardziej odporną na awarie.

Inną popularną topologią (nie pokazaną na rysunku) jest **torus trójwymiarowy**. Powstaje on na bazie trójwymiarowej kraty, której węzły umieszczone są w punktach o współrzędnych (i, j, k), gdzie każda z wartości i, j oraz k przebiega wartości całkowite, od 1 aż do ustalonej granicy (odpowiednio) l, m oraz n. Każdy węzeł posiada sześciu

najbliższych sąsiadów — po dwóch w każdym z trzech kierunków — z którymi jest bezpośrednio połączony. Wspomniany torus otrzymujemy, łącząc ze sobą pary skrajnych węzłów, podobnie jak w przypadku torusa dwuwymiarowego.

Sześcian (*cube*), zwany także **kostką**, widoczny w części (f), jest regularną topologią trójwymiarową; na rysunku pokazano sześcian $2 \times 2 \times 2$, lecz w ogólnym przypadku może on łączyć węzły według schematu $k \times k \times k$. Przez połączenie odpowiadających sobie węzłów dwóch sześcianów trójwymiarowych otrzymujemy czterowymiarową topologię widoczną w części (h), zwaną **hipersześcianem** lub **hiperkostką** (*hypercube*). Łącząc ze sobą w analogiczny sposób dwie hiperkostki o wymiarze n otrzymujemy hiperkostkę o rozmiarze n+1. Średnica sieci o topologii hiperkostki jest liniową funkcją jej wymiaru, a więc — co łatwo wydedukować — logarytmiczną funkcją liczby węzłów. 10-wymiarowa hiperkostka złożona jest z 1024 węzłów, lecz jej średnica równa jest (tylko) 10; gdyby węzły te połączyć w kratę 32×32, średnica takiej sieci wynosiłaby ponad sześciokrotnie więcej (62). Niestety, wraz ze wzrostem wymiaru hiperkostki rośnie także stopień jej węzłów, a więc liczba połączeń, co powoduje, iż jest ona droższa w realizacji od innych topologii łączących tę samą liczbę węzłów[21]. Mimo to topologia hiperkostki, ze względu na swe zalety, wykorzystywana jest powszechnie w komercyjnych zastosowaniach wymagających dużej wydajności.

Wielokomputery posiadają zróżnicowane struktury i rozmiary, trudno więc byłoby podać jakąś rozsądną ich taksonomię. Wszystkie one mogą być jednak podzielone na dwie zasadnicze grupy: masywnie zrównoleglone procesory oraz klastry.

8.4.2. MPP — masywnie zrównoleglone procesory

Masywnie zrównoleglone procesory (**MPP**, **Massive Paralell Processors**) mają postać superkomputerów kosztujących miliony dolarów i wykorzystywanych w badaniach naukowych, inżynierii i przemyśle do wykonywania szczególnie złożonych obliczeń, do zarządzania intensywnymi strumieniami transakcji oraz w hurtowniach danych do superefektywnego zarządzania olbrzymimi bazami danych. Początkowo komputery MPP wykorzystywane były jedynie w badaniach naukowych, z biegiem czasu jednak coraz więcej z nich używanych jest w zastosowaniach komercyjnych. W pewnym sensie przypominają one potężne komputery *mainframe* z lat 60. ubiegłego stulecia[22] — mają jednak znacznie subtelniejszą strukturę wewnętrzną i porównanie to przypomina (słuszną skądinąd) konkluzję paleontologów, iż widoczne właśnie stado wróbli to potomkowie potwora *Tyrannosaurus Rex*. Mówiąc obrazowo, komputery MPP są maszynami SIMD — superkomputerami wektorowymi i procesorami macierzowymi — przesuniętymi na sam szczyt cyfrowego łańcucha pokarmowego…

Większość tych maszyn wykorzystuje standardowe procesory, najczęściej Pentium, UltraSPARC i PowerPC. Tym, co odróżnia MPP od innych kategorii komputerów, jest wyjątkowo szybka, specjalizowana sieć zaprojektowana w celu przesyłania komunikatów z minimalnym opóźnieniem i ogromną przepustowością. Obydwie te cechy są niezwykle istotne, bowiem większość komunikatów to małe pakiety (znacznie

[21] Koszt hiperkostki łączącej n węzłów, wyrażający się liczbą krawędzi, jest proporcjonalny do $n * \log_2 n$ — *przyp. tłum.*

[22] O wydajności około **miliard** razy mniejszej — *przyp. tłum.*

poniżej 256 bajtów), jednak największy wkład do ogólnego obciążenia wnoszą duże komunikaty (przekraczające 8 KB). Cały wyrafinowany sprzęt MPP zarządzany jest oczywiście olbrzymim oprogramowaniem o specyficznej konstrukcji.

Kolejną cechą charakteryzującą MPP są ich kolosalne możliwości w zakresie operacji wejścia-wyjścia. Problemy kwalifikujące się do rozwiązywania przez te komputery wiążą się z ultraszybkim przemieszczaniem (między pamięcią i dyskami) olbrzymich ilości danych, liczonych w terabajtach.

Specyfiką maszyn MPP jest także ich zdolność do tolerowania defektów (*fault tolerance*). Przy kilku tysiącach procesorów prawdopodobieństwo wystąpienia kilku awarii w ciągu tygodnie jest całkiem realne, lecz utrata w związku z tym rezultatów 18-godzinnych obliczeń — absolutnie niedopuszczalna. MPP są wiec wyposażone w sprzętowe i programowe środki monitorowania pracy systemu, wykrywania ewentualnych usterek i sprawnego powrotu do normalnej pracy w przypadku wystąpienia tych ostatnich.

Mimo iż omówienie generalnych zasad projektowania komputerów MPP z pewnością byłoby pożyteczne i pouczające, nie sposób go tu dokonać z tej oczywistej przyczyny, iż zasad takich po prostu nie ma. O MPP można powiedzieć w sensie ogólnym jedynie to, iż stanowią one wynik połączenia mniej lub bardziej standardowych węzłów za pośrednictwem bardzo szybkich sieci, które omawialiśmy wcześniej w niniejszym rozdziale. W zamian więc przyjrzymy się dwóm przykładom komputerów MPP funkcjonujących w praktyce: BlueGene/L i Red Storm.

BlueGene

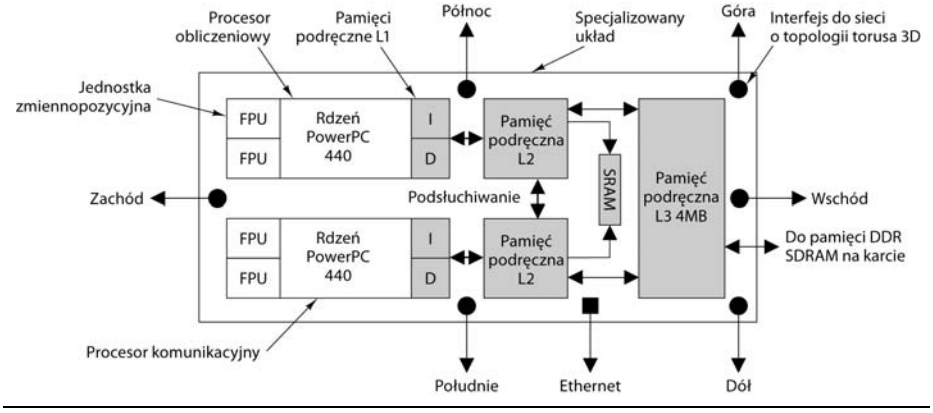
W roku 1999 firma IBM rozpoczęła realizację projektu masywnie zrównoleglonego komputera przeznaczonego do rozwiązywania problemów wymagających gigantycznej mocy obliczeniowej, głównie problemów z zakresu nauk biologicznych. Przykładowo, biologowie przekonani są, że trójwymiarowa struktura białka determinuje jego funkcje życiowe, lecz „obliczenie" konstrukcji małej nawet struktury białkowej na podstawie znanych praw fizyki zajęłoby dostępnym wtedy superkomputerom wiele lat! W organizmie ludzkim obecnych jest ponad pół miliona różnych struktur białkowych, wiele z nich to duże struktury, a ich nieprawidłowe pofałdowanie (*misfolding*) uważane jest za przyczynę wielu chorób, m.in. mukowiscydozy. Stało się oczywiste, że określenie trójwymiarowej struktury wszystkich ludzkich białek wymagałoby użycia wszystkich znajdujących się na świecie komputerów, po uprzednim zwiększeniu ich mocy obliczeniowej o kilka rzędów wielkości. Modelowanie struktur białkowych to tylko jedno z zastosowań, z myślą o których skonstruowany został BlueGene; równie złożone problemy obliczeniowe wywodzą się z dziedziny dynamiki molekularnej, modelowania klimatu, astrofizyki, a nawet symulacji finansowych.

Zgodnie z rozeznaniem firmy IBM zapotrzebowanie rynkowe na masywne superkomputery było wystarczająco duże, by opłaciło się zainwestować 100 milionów dolarów w projekt BlueGene. W listopadzie roku 2001 Livermore National Laboratory, prowadzone przez Departament Energii USA, stało się pierwszym partnerem i pierwszym klientem pierwszej wersji komputera rodziny BlueGene, nazwanego **BlueGene/L**.

Cele, jakie postawiono sobie u podstaw projektu BlueGene, wykraczały daleko poza li tylko zbudowanie najszybszego na świecie komputera MPP. Miał on być jednocześnie komputerem najbardziej na świecie wydajnym pod względem jednostki

wykonanych obliczeń (teraflopu[23], w skrócie TF) uzyskanej (odpowiednio) za cenę 1 dolara (TF/USD), przy zużyciu jednostkowej energii (TF/dżul[24]) i przypadającej na jednostkę gabarytu urządzenia (TF/m^3). Z tej racji firma IBM ostatecznie zarzuciła filozofię stosowaną w odniesieniu do poprzednio projektowanych MPP, sprowadzającą się do poświęcania na ołtarzu wydajności wszystkiego, co tylko można kupić za pieniądze. W zamian postanowiono zbudować system oparty na energooszczędnych, średnio szybkich, specjalizowanych komponentach upakowanych z dużą gęstością. Pierwszy taki układ ujrzał światło dzienne z czerwcu 2003 roku, a pierwsza ćwiartka systemu BlueGene/L, złożona z 16 384 węzłów obliczeniowych, była w pełni gotowa w listopadzie tegoż roku, kiedy to uzyskała certyfikat najszybszego na świecie superkomputera, uzyskując wydajność około 71 TF/s. Pobierana w związku z tym energia — niecałe 400 kilowatów — plasuje ten komputer w kategorii najbardziej energooszczędnych komputerów na świecie, przy zużyciu energii (jak łatwo policzyć) 177,5 megaflopa[25] na dżul. Rozbudowa komputera do pełnej konfiguracji, obejmującej 65 536 węzłów obliczeniowych, ma zostać ostatecznie ukończona latem 2005 roku[26].

Sercem systemu BlueGene/L jest specjalizowany układ scalony o strukturze logicznej przedstawionej na rysunku 8.32. Składa się on z dwóch rdzeni PowerPC 440 pracujących z częstotliwością 700 MHz. PowerPC jest superskalarnym, dwupotokowym procesorem stosowanym powszechnie w systemach wbudowanych. Każdy rdzeń posiada parę dwupotokowych jednostek zmiennopozycyjnych, które łącznie wykonywać mogą cztery rozkazy zmiennopozycyjne w jednym cyklu zegara, jak również inne rozkazy z kategorii SIMD, przydatne do obliczeń na wektorach. Mimo niezłej wydajności, trudno jednak zaliczyć ów układ do wiodących wieloprocesorów.



RYSUNEK 8.32. Specjalizowany węzeł obliczeniowy komputera BlueGene/L

[23] 1 teraflop to 10^{12} (bilion) operacji zmiennopozycyjnych — *przyp. tłum.*

[24] Równoważną jednostką jest wydajność odnoszona do pobieranej mocy, czyli $\frac{\text{TF/s}}{\text{wat}}$ — *przyp. tłum.*

[25] Megaflop to milion operacji zmiennopozycyjnych — *przyp. tłum.*

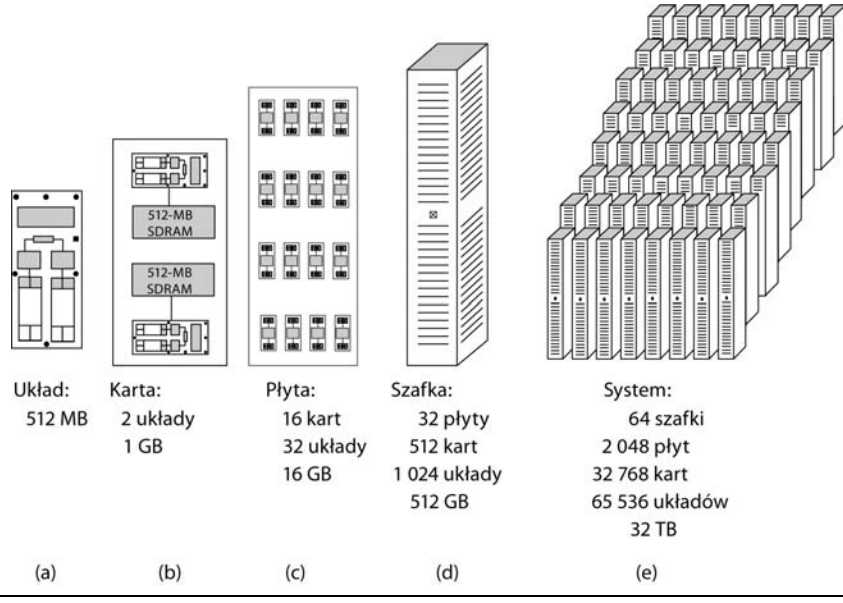
[26] W internetowym rankingu najszybszych komputerów z listopada 2005 roku BlueGene/L notowany jest (oczywiście na pierwszym miejscu) z osiągiem 280,6 TF/s — *przyp. tłum.*

Obydwa rdzenie układu mają identyczną strukturę, przeznaczone są jednak do odmiennych zadań i programowane są odmiennie. Jeden rdzeń przeznaczony jest do celów obliczeniowych, zadaniem drugiego jest zarządzanie komunikacją z pozostałymi 65 535 układami.

Układ wyposażony jest w trzypoziomową pamięć podręczną. Poziom pierwszy składa się z pamięci L1 w każdym rdzeniu, po 32 KB dla rozkazów i danych każda. Nie ma żadnej synchronizacji między tymi pamięciami, bo też i standardowy rdzeń PowerPC 440 takowej nie zapewnia, a firma IBM chciała wykorzystać standardowe układy bez ich modyfikowania. Pamięć podręczna drugiego poziomu (L2) składa się z dwóch standardowych modułów po 2 KB; moduły te służą raczej jako bufory pobierania wstępnego (*prefetch buffers*) niż jako rzeczywiste pamięci podręczne. Podsłuchują się one nawzajem i uzgadniają swą zawartość. 4-megabajtowa pamięć podręczna na poziomie trzecim (L3) spełnia rolę zasilacza dla pamięci poziomu drugiego. Odwołanie do pamięci, które okazuje się chybione na poziomie L1, lecz trafione na poziomie L2, zajmuje ok. 11 cykli zegara. Chybienie na poziomie L2, lecz trafienie na poziomie L3 to ok. 28 cykli. Odwołanie do pamięci SDRAM w związku z chybieniem na poziomie L3 wymaga natomiast ok. 75 cykli.

Moduły pamięci podręcznej L2 połączone są z małą pamięcią SRAM. Pamięć ta połączona jest z nóżkami JTAG układu, zapewniając w ten sposób obsługę bootowania i testowania, komunikację z głównym hostem, utrzymywanie stosu systemowego i dostarczanie mechanizmów synchronizacyjnych w postaci semaforów i barier.

Na bezpośrednio wyższym poziomie firma IBM zaprojektowała kartę rozszerzającą, zawierającą dwa układy widoczne na rysunku 8.32, z których każdy połączony jest z modułem pamięci SDRAM o pojemności 512 MB. Całkowita pojemność pamięci na karcie wynosi więc 1 GB, a w przyszłej wersji kart może sięgać nawet 4 GB. W części (a) rysunku 8.33 widzimy miniaturę układu z rysunku 8.32, a w części (b) wspomnianą kartę.



RYSUNEK 8.33. Pamięć BlueGene/L: (a) układ; (b) karta rozszerzająca; (c) płyta; (d) szafka; (e) system

Karty montowane są na płytach, po 16 na jednej, jak widać w części (c) rysunku. Na każdej płycie znajduje się więc 16 GB pamięci operacyjnej i 32 procesory.

Na następnym poziomie konstrukcyjnym płyty montowane są w szafkach (część (d) rysunku) o wymiarach 60 × 90 cm, po 16 w górnej i dolnej części szafki. Obydwie grupy — dolna i górna — rozdzielone są specjalnym przełącznikiem, który umożliwia wymontowanie jednej grupy w celach serwisowych i przekazanie pełnionych przez nią funkcji specjalnej grupie rezerwowej.

Cały system składa się z 64 takich szafek, zawierających łącznie 65 536 węzłów obliczeniowych, czyli 131 072 dwupotokowe jednostki stałopozycyjne i 262 144 dwupotokowe jednostki zmiennopozycyjne. Dzięki temu komputer zdolny jest w jednym cyklu zegara inicjować do 768 432 rozkazów; ponieważ jednak w każdym węźle obliczeniowym jedna jednostka stałopozycyjna powiązana jest z jednostką zmiennopozycyjną, wartość ta jest nieco mniejsza i wynosi 655 360 rozkazów w jednym cyklu zegarowym, czyli $4{,}6 \times 10^{14}$ rozkazów na sekundę, co zostawia daleko w tyle wszystkie wyprodukowane wcześniej komputery.

BlueGene/L jest wielokomputerem w tym sensie, że każdy z procesorów ma dostęp jedynie do swej lokalnej pamięci 512 MB na karcie i żadna para procesorów nie współdzieli tego samego modułu pamięci. Procesory nie są wyposażone w żadne mechanizmy w rodzaju segmentacji czy stronicowania, nie ma bowiem lokalnych dysków, na bazie których mechanizmy te można byłoby realizować. W zamian istnieją w systemie 1024 węzły wejścia-wyjścia, zapewniające dostęp do dysków i innych urządzeń peryferyjnych.

Mimo ogromnej skali systemu, projektanci postanowili uczynić go systemem prostym, z nieznacznym udziałem najnowszych technologii, związanych głównie z dużym stopniem upakowania komponentów. Przyczyna takiej decyzji nie jest bynajmniej tajemnicą: BlueGene ma być przede wszystkim komputerem niezawodnym i wysoce dostępnym, w związku z czym szczególnie starannie zaprojektowano jego zasilanie, chłodzenie i okablowanie — tak, by średni czas międzyawaryjny dało się utrzymać na poziomie 10 dni.

Węzły obliczeniowe komputera połączone są za pomocą skalowalnej, superszybkiej sieci o topologii trójwymiarowego torusa o rozmiarach 64 × 32 × 32. W konsekwencji każdy procesor połączony jest logicznie z sześcioma innymi, po dwa w każdym z trzech wzajemnie prostopadłych kierunków, odpowiednio wschód-zachód, północ-południe oraz góra-dół, jak na rysunku 8.32. Pod względem fizycznym każda szafka z 1024 węzłami stanowi torus 8 × 8 × 16; każda para sąsiadujących szafek tworzy torus 8 × 8 × 32, zaś cztery takie pary w jednym rzędzie składają się na torus 8 × 32 × 32. Ostatecznie osiem takich rzędów to właśnie torus o rozmiarach 64 × 32 × 32.

Wszystkie połączenia w tej sieci są łączami dwupunktowymi o przepustowości 1,4 Gb/s. Ponieważ każdy z 53 536 węzłów posiada trzy łącza do węzłów o „wyższej numeracji", każde w jednym z trzech prostopadłych kierunków, całkowita przepustowość sieci wynosi 275 terabitów na sekundę. Przy założeniu, że treść niniejszej książki zawiera informację na poziomie ok. 300 milionów bitów, BlueGene/L zdolny jest do przetransmitowania w ciągu sekundy — uwaga — 900 000 kopii tej książki (czy i do czego mogłoby się to w praktyce przydać — tę interesującą kwestię pozostawiamy jako ćwiczenie dla Czytelników).

Komunikacja w obrębie trójwymiarowego torusa ma formę **wirtualnego przecinania** (*virtual cut-through*). Jest to technika pokrewna przekazywaniu pakietów z zapamiętywaniem (*store and forward*), z tą różnicą, że nadchodzący do węzła pakiet nie jest w węźle „kumulowany" przed wysłaniem, lecz każdy jego bajt przekazywany jest

natychmiast do następnego węzła. Możliwe jest zarówno statyczne, jak i adaptacyjne trasowanie pakietów. Implementacja wirtualnego przecinania wymaga niewielkiego, dodatkowego sprzętu specjalnego przeznaczenia.

Oprócz trójwymiarowego torusa, przeznaczonego do transportu danych, BlueGene/L wyposażony jest jeszcze w cztery sieci komunikacyjne. Pierwsza z nich jest siecią kombinacyjną zorganizowaną w strukturę drzewiastą. Wiele operacji wykonywanych przez komputery silnie zrównoleglone — jak BlueGene/L — wymaga udziału wszystkich węzłów; przykładem takiej operacji jest obliczanie wartości minimalnej ze zbioru 65 536 wartości, zapamiętanych po jednej w każdym węźle. Gdy wszystkie węzły zorganizowane są w drzewo binarne, operację tę można wykonać szczególnie efektywnie: wartość minimalna z dwóch węzłów przekazywana jest do ich węzła macierzystego, gdzie jest porównywana z wartością tego ostatniego. Mniejsza z tych dwóch wartości przesyłana jest poziom wyżej, wraz z analogicznie obliczoną wartością węzła-brata. Całość kończy się w korzeniu drzewa, a natężenie ruchu w sieci — to ważne — maleje wraz z przybliżaniem się do korzenia. To znacznie mniej obciążające w porównaniu z „uderzeniowym" wysłaniem 65 536 komunikatów jednocześnie ze wszystkich węzłów.

Druga z dodatkowych sieci przeznaczona jest do obsługi globalnych przerwań i barier. Bariera jest rodzajem mechanizmu synchronizującego: niektóre algorytmy równoległe wykonywane są w kolejnych fazach, przy czym żaden z węzłów nie może rozpocząć realizacji kolejnej fazy wcześniej, niż zakończy się realizacja bieżącej fazy we wszystkich innych węzłach. Opisywana sieć umożliwia definiowanie wspomnianych faz przez oprogramowanie, udostępniając środki do zawieszania działania każdego węzła w momencie zakończenia przez niego realizacji bieżącej fazy i zwalniania wszystkich węzłów w momencie, gdy wszystkie one fazę tę zakończą. Mechanizm ten wykorzystywany jest także w obsłudze przerwań.

Dwie pozostałe sieci wykorzystują gigabitowy Ethernet. Jedna z nich zapewnia połączenie węzłów wejścia-wyjścia z serwerami plików — zewnętrznymi w stosunku do BlueGene/L — i pośrednio z internetem. Druga wykorzystywana jest na potrzeby testowania i debugowania systemu.

Każdy z węzłów obliczeniowych i komunikacyjnych realizuje mały i prosty program jądra w ramach pojedynczego procesu i w trybie jednego użytkownika. Proces ten podzielony jest na dwa wątki, z których jeden realizowany jest przez wszystkie procesory węzła. Prostota ta zapewnić ma dużą wydajność i dużą niezawodność.

Węzły wejścia-wyjścia pracują pod kontrolą systemu Linux i każdy z nich realizować może jednocześnie wiele procesów.

Dla zwiększenia niezawodności każda z aplikacji realizowanych na komputerze może okresowo tworzyć punkty kontrolne, wywołując przeznaczoną do tego funkcję biblioteczną. W przypadku awarii systemu, po wznowieniu jego pracy wykonywanie aplikacji rozpocznie się od jej punktu kontrolnego, nie od początku.

Więcej informacji na temat komputera BlueGene/L znaleźć można w pracach (Adiga i in., 2002), (Almasi i in., 2003a i 2003b) oraz (Blumrich i in., 2005).

Red Storm

Zmieniamy kolor, z błękitnego na czerwony: drugi z prezentowanych przez nas komputerów MPP, zwany „Czerwoną burzą", w oryginale *Red Storm* (znany również pod roboczą nazwą *Thor's Hammer* — „Młot Thora" — od słynnego piaskowca w Bryce Canyon), zbudowany został w laboratorium Sandia National Laboratory należącym do

firmy Lockheed Martin. Służy on do przetwarzania jawnych i utajnionych (*classified*) informacji na rzecz Departamentu Energii USA; w zakres kategorii utajnionych wchodzi między innymi projektowanie i symulacje broni nuklearnej, a więc dziedziny wymagające szczególnie intensywnych obliczeń.

Laboratorium Sandia, które uczestniczy w tym biznesie od dawna, początkowo wykorzystywało superkomputery wektorowe, lecz z biegiem lat realia technologiczne oraz względy ekonomiczne coraz wyraźniej przemawiać zaczęły na rzecz komputerów MPP. Uruchomiony na początku 2002 roku pierwszy z komputerów tej kategorii — zwany ASCII Red — okazał się komputerem cokolwiek nieudanym: mimo 9460 węzłów, prawie 1,2 TB pamięci RAM i 12,5 TB przestrzeni dyskowej, zdolny był osiągnąć wydajność „zaledwie" 3 TF/s.

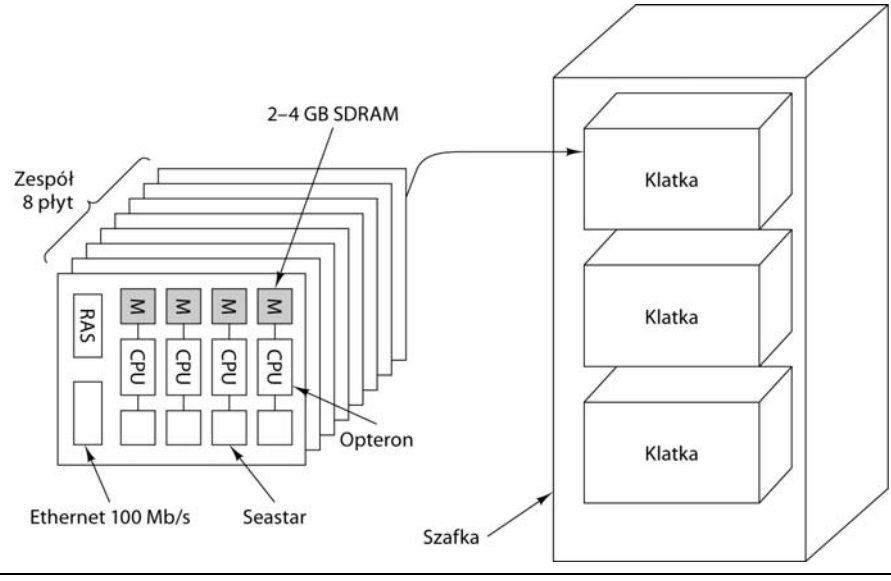
Latem tegoż roku laboratorium zleciło więc firmie CRAY Research — znanemu dostawcy superkomputerów — wykonanie następcy i gotowy produkt — Red Storm — dostarczony został dwa lata później, w sierpniu 2004 roku. Tak krótki okres projektowania i realizacji jest w obszarze superkomputerów ewenementem, a możliwy był głównie dlatego, że firma CRAY postanowiła skorzystać niemal w całości z gotowych komponentów „ z półki", wyjąwszy specjalizowanie układu realizujące routing między węzłami.

Węzły komputera Red Storm zbudowane są na bazie procesorów Opteron firmy AMD, które posiadają kilka wspaniałych cech uzasadniających w pełni ów wybór. Po pierwsze, procesory te mogą pracować w trzech trybach operacyjnych. Tryb dziedzictwa (*legacy mode*) to tryb 16- i 32-bitowego adresowania typowy dla procesorów serii 80x86 i Pentium — binarne programy przeznaczone dla tych procesorów mogą być w tym trybie wykonywane bez jakichkolwiek zmian. W trybie zgodności (*compatibility mode*) system operacyjny pracuje w trybie 64-bitowym i może adresować 2^{48} bajtów pamięci wirtualnej; programy użytkowe pracują jednak w trybie 32-bitowym. Wreszcie, tryb pełny (*long mode*) oznacza nieskrępowane wykorzystywanie możliwości procesora: pod kontrolą 64-bitowego systemu operacyjnego mogą być realizowane programy zarówno 64-bitowe, jak i 32-bitowe, co czyni łatwiejszą wymianę procesorów na nowsze w działających systemach.

Drugą pozytywną cechą procesorów Opteron jest ich zoptymalizowanie pod kątem szybkości transmisji między procesorem a pamięcią. Jak dotąd, moduły pamięci, mimo iż coraz szybsze, nie były w stanie dotrzymywać kroku coraz to szybszym procesorom, czego najważniejszą konsekwencją były dość czasochłonne odwołania do pamięci w przypadku chybienia pamięci cache na poziomie L2. Firma AMD poradziła sobie z tym problemem poprzez zintegrowanie kontrolera pamięci z procesorem, dzięki czemu kontroler ten może pracować z szybkością wyznaczaną przez zegar procesora, a nie zegar magistrali pamięciowej. Kontroler ten może obsługiwać do ośmiu modułów DIMM o pojemności do 4GB każdy, co daje maksymalnie 32 GB pamięci fizycznej na każdy procesor; w komputerze Red Storm każdy procesor dysponuje jednak pamięcią znacznie mniejszą, w granicach 2 – 4 GB, co jednak wobec konsekwentnie zmniejszających się cen pamięci na pewno nie jest ostateczną granicą. Nie jest także wykluczona wymiana procesorów Opteron na wersję z dwoma rdzeniami.

Z każdym procesorem Opteron połączony jest specjalny procesor sieciowy **Seastar** firmy IBM. Procesory Seastar są najbardziej krytyczną częścią systemu, obsługują bowiem niemal całą transmisję danych; gdyby nie superszybka łączność, jaką są w stanie zapewnić, system z łatwością mógłby się załamać wobec intensywnych strumieni danych.

Mimo iż procesory Opteron są ogólnie dostępnym produktem „z półki”, system ich upakowania jest już wynalazkiem firmy CRAY. Na pierwszym poziomie integracji cztery procesory umieszczane są na wspólnej płycie, zawierającej ponadto cztery procesory sieciowe Seastar, 4 moduły pamięci RAM o pojemności 2 – 4 GB, procesor RAS (*Reliability*, *Availability*, *Service* — niezawodność, dostępność i serwis) i układ Ethernetu 100 Mb/s, jak przedstawiono to na rysunku 8.34.



RYSUNEK 8.34. Sposób upakowania komponentów komputera Red Storm

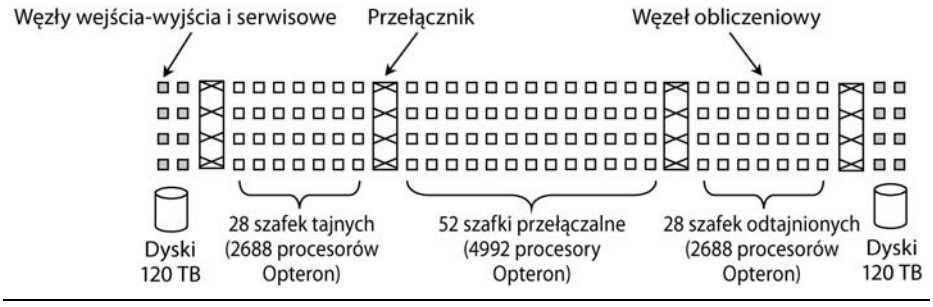
Zespół ośmiu takich płyt, osadzonych w gniazdach płyty głównej, zamykany jest w tzw. klatce (*cage*), zaś trzy klatki montowane są w szafce (*cabinet*), zawierającej ponadto niezbędne zasilacze i system chłodzenia. W każdej szafce znajduje się więc 96 procesorów. Cały system składa się ze 108 takich szafek, co daje łącznie 10 368 procesorów i 10 TB pamięci RAM. Każdy procesor ma dostęp wyłącznie do własnej pamięci — nie ma pamięci współdzielonej. Teoretyczna moc obliczeniowa systemu wynosi 41 TF/s.

Łączność między procesorami Opteron zapewniają specjalizowane routery Seastar, po jednym dla każdego procesora. Routery te połączone są w trójwymiarowy torus o rozmiarach 27 × 16 × 24 — w każdym jego węźle znajduje się jeden router. Każdy router połączony jest z sześcioma sąsiadami (po dwóch w każdym z trzech prostopadłych kierunków) i ze „swoim” procesorem Opteron za pośrednictwem dwukierunkowych łączy o przepustowości 24 Gb/s. Przesłanie danych między dwoma sąsiednimi węzłami trwa 2 mikrosekundy, zaś rozesłanie danych między wszystkie węzły systemu wymaga 5 mikrosekund. 100-megabitowa sieć Ethernet wykorzystywana jest na potrzeby serwisu i konserwacji systemu.

Oprócz wymienionych 108 szafek „obliczeniowych”, system zawiera także 16 szafek z procesorami wejścia-wyjścia, po 32 procesory — Opteron — w każdej szafce. Owe 512 procesorów podzielone zostało po połowie dla celów realizacji wejścia-wyjścia i dla celów serwisowych. Pozostałą cześć systemu zajmują dyski, zorganizowane

w macierze RAID 3 i RAID 5, z dyskiem parzystości (*parity drive*) i dyskiem zapasowym (*hot spare*) każda. Całkowita pojemność dysków wynosi 240 TB, a ich łączna przepustowość stała kształtuje się w granicach 50 GB/s.

System podzielony jest na sekcję tajną (*classified*) i odtajnioną (*unclassified*) za pomocą mechanicznych przełączników umożliwiających złączanie i rozłączanie tych sekcji. Jak pokazano na rysunku 8.35, każda z sekcji zawiera po 2688 procesorów obliczeniowych, pozostałe 4992 tworzą sekcję przełączalną, która może być dołączana do sekcji tajnej lub odtajnionej. Każdy z procesorów w sekcji tajnej posiada 4 GB pamięci RAM, pozostałe procesory posiadają jej po 2 GB. Węzły wejścia-wyjścia i serwisowe podzielone są po równo między obydwie sekcje.



RYSUNEK 8.35. Podział systemu Red Storm na sekcję tajną i odtajnioną

Cały system Red Storm umieszczony jest w specjalnie zbudowanej dla niego hali o powierzchni 2000 m^2; powierzchnię tę obliczono z myślą o przyszłej rozbudowie komputera, docelowo do 30 000 procesorów. Obecnie wszystkie procesory komputera pochłaniają 1,6 megawatów mocy, kolejny megawat pochłaniany jest przez dyski. Doliczając do tego moc zużywaną przez układy chłodzenia otrzymujemy łączny pobór mocy ok. 3,5 megawata.

Łączny koszt sprzętu i oprogramowania komputera Red Storm wyniósł 90 mln USD; budynek wraz z systemem chłodzenia to dodatkowo ponad 9 mln, co łącznie daje około 100 mln. Część tych kosztów to jednorazowe koszty projektowania, niemniej jednak każdy, kto chciałby posiadać dokładną kopię komputera Red Storm w obecnej wersji, i tak powinien posiadać w zanadrzu coś ok. 60 mln USD. Dla mniej zamożnych (lub bardziej oszczędnych?) klientów rządowych i biznesowych firma CRAY przygotowuje skromniejszą wersję systemu o nazwie X3T.

Na każdym z węzłów obliczeniowych wykonywany jest prosty program jądra o nazwie **catamount**, natomiast każdy z węzłów wejścia-wyjścia i węzłów serwisowych sterowany jest systemem Linux wzbogaconym o obsługę MPI (opisywaną w dalszej części rozdziału). Węzły RAS realizują natomiast okrojoną wersję Linuksa. Duża część oprogramowania — alokatory procesorów, planiści, biblioteki MPI, biblioteki matematyczne i programy użytkowe — przeniesiona została z poprzedniego komputera Red ASCI.

W systemach tak dużych jak Red Storm kluczowym zagadnieniem jest osiągnięcie dużej niezawodności. Niezawodność zapewnić też mają procesory RAS (jeden taki procesor znajduje się na każdej płycie), zaprojektowane specjalnie na potrzeby utrzymywania pracy systemu, jego konserwacji i serwisu. Oczekuje się, że średni czas

międzyawaryjny (MTBF, *Mean Time Between Failures*) kształtować się będzie w granicach 50 godzin. Sprzęt komputera Red ASCI cechował się średnim czasem międzyawaryjnym rzędu 900 godzin, jednakże liczne błędy w oprogramowaniu obniżały tę wielkość do ok. 40 godzin. Tak się bowiem niedobrze składa, że mimo coraz bardziej niezawodnego sprzętu, najsłabszym elementem systemów komputerowych pozostaje ich oprogramowanie.

Więcej informacji na temat komputera Red Storm znaleźć można w pracy (Brightwell i in., 2005).

Porównanie komputerów BlueGene/L i Red Storm

Obydwa omawiane komputery są podobne pod wieloma względami i kompletnie różnią się od siebie pod innymi. Interesujące będzie więc zestawienie ich wybranych cech, co uczyniliśmy w tabeli 8.5.

TABELA 8.5.
Porównanie BlueGene/L i Red Storm

Cecha	BlueGene/L	Red Storm
Procesory	32-bitowy PowerPC	64-bitowy Opteron
Częstotliwość zegara	700 MHz	2 GHz
Liczba procesorów obliczeniowych	65 536	10 368
Liczba procesorów na płycie	32	4
Liczba procesorów w szafce	1024	96
Liczba szafek obliczeniowych	64	108
Moc obliczeniowa	71 TF/s	41 TF/s
Pamięć jednego procesora	512 MB	2 – 4 GB
Całkowita pamięć	32 TB	10 TB
Router	PowerPC	Seastar
Liczba routerów	65 536	10 368
Sieć połączeniowa	Trójwymiarowy torus 64 × 32 × 32	Trójwymiarowy torus 27 × 16 × 24
Inne sieci	Ethernet gigabitowy	Fast Ethernet
Partycjonowalny	Nie	Tak
System operacyjny procesorów obliczeniowych	Specyficzny	Specyficzny
System operacyjny węzłów wejścia-wyjścia	Linux	Linux
Producent	IBM	CRAY Research
Cena	Bardzo wysoka	Bardzo wysoka

Obydwa komputery zostały zbudowane mniej więcej w tym samym czasie, zatem różnice między nimi nie mają uwarunkowań technologicznych, a wynikać mogą jedynie z różnych wizji projektantów i (w pewnym stopniu) z różnic między samymi firmami IBM i CRAY Research. BlueGene/L zaprojektowany został od podstaw jako komputer komercyjny, który firma IBM spodziewa się sprzedać w wielu egzemplarzach firmom zajmującym się badaniami biotechnologicznymi, farmaceutycznymi itp. Red Storm powstał natomiast w jednym egzemplarzu na konkretne zamówienie, choć — jak wcześniej pisaliśmy — firma CRAY planuje seryjną produkcję jego okrojonej, tańszej wersji.

Wizja projektantów z IBM byłą dość wyrazista: wykorzystanie istniejących rdzeni w celu stworzenia specjalizowanego układu, który można będzie tanio produkować w sposób masowy i który, sam nie będąc specjalnie szybkim, może zostać wykorzystany jako „masowy" budulec dla potężnego komputera, w wyniku połączenia wielu tysięcy jego egzemplarzy siecią o umiarkowanej prędkości. Wizja projektantów z CRAY Research jawi się jako równie klarowna, lecz całkowicie odmienna, zakładała bowiem wykorzystanie szybkiego, uniwersalnego, 64-bitowego układu „z półki", który wzbogacony o dużą ilość pamięci i specjalizowany router stał się wydajnym węzłem obliczeniowym — o wiele wydajniejszym niż węzeł BlueGene/L, dzięki czemu można było zadowolić się mniejszą liczbą węzłów i połączyć je ze sobą szybszą siecią.

Bezpośrednią konsekwencją opisanych różnic w wizji projektu okazały się różnice w sposobie pakowania samych węzłów. Ponieważ w węźle BlueGene/L procesor obliczeniowy zintegrowany jest z routerem w jednym układzie, można było osiągnąć bardzo duże upakowanie węzłów — 1024 na szafkę. Węzeł komputera Red Storm zbudowany został natomiast na bazie oryginalnego, niemodyfikowanego procesora „z półki", więc zarówno pamięć, jak i router trzeba było dołączyć na zewnątrz tego procesora; w rezultacie w jednej szafce komputera mieści się tylko 96 procesorów (węzłów), a sam komputer zajmuje większą powierzchnię oraz zużywa ponad ośmiokrotnie więcej energii niż BlueGene/L.

W egzotycznym świecie obliczeniowym dużych laboratoriów kryterium ostatecznym jest wydajność. BlueGene/L wygrał co prawda pierwsze starcie z Red Stormem (71 TF/s kontra 41 TF/s), lecz — w przeciwieństwie do BlueGene/L — Red Storm można w razie potrzeby rozbudować o kolejne 10 368 Opteronów (na przykład wymieniając obecne na nowsze wersje dwurdzeniowe), co (teoretycznie) zwiększyć może wydajność do 82 TF/s. W odpowiedzi IBM może zwiększyć częstotliwość zegara — bo 700 MHz to przecież nie szczyt możliwości obecnych procesorów — i tak można wyliczać w nieskończoność. Ważne jest w tym wszystkim to, że superkomputery dalekie są jeszcze dziś od granicy swych fizycznych możliwości i zapewne jeszcze przez wiele lat wciąż będziemy zaskakiwani kolejnymi rewelacjami (głównie informacjami o nowych rekordach) w tej dziedzinie.

8.4.3. Obliczenia klastrowe

Innym sposobem tworzenia wielokomputerów jest łączenie pojedynczych komputerów w **klastry** (Anderson i in., 1995) oraz (Martin i in., 1997). Taki klaster składa się zazwyczaj z setek lub tysięcy stacji roboczych połączonych za pomocą komercyjnych komponentów sieciowych. Różnica między komputerami MPP a klastrami podobna jest do różnicy między komputerami *mainframe* a komputerami PC: obydwa posiadają procesor, pamięć, dysk (dyski), system operacyjny, lecz w przypadku komputerów *mainframe* komponenty te są szybsze (z wyjątkiem systemu operacyjnego), poza tym sposób korzystania z obydwu kategorii komputerów jest wyraźnie odmienny.

Historycznie rzecz biorąc, najważniejszą przesłanką przemawiającą na rzecz komputerów MPP była niesamowita szybkość wewnętrznych sieci łączących poszczególne węzły — szybkość, której nie sposób było zapewnić za pomocą sieci zewnętrznych, budowanych na bazie komponentów „z półki". Postęp technologiczny zlikwidował jednak tę lukę i klastry komputerów zdają się spychać komputery MPP do

zastosowań bardziej specyficznych, podobnie jak komputery PC praktycznie wyeliminowały komputery *mainframe* z tzw. codziennych zastosowań. W efekcie dzisiejsze komputery MPP to komputery wysokobudżetowe, których wydajność jest kryterium jedynym bez względu na cenę — jednak komputery raczej nie dla tych, którzy muszą się liczyć z pieniędzmi…

Wśród rozmaitych rodzajów klastrów wyróżnić można dwie ich grupy: klastry scentralizowane i zdecentralizowane. Klaster **scentralizowany** to zwykle zespoły stacji roboczych montowanych na dużym stojaku lub kilku stojakach w tym samym pomieszczeniu. Często są to identyczne komputery, o mniejszych niż zwykle gabarytach (w celu zaoszczędzenia miejsca i kabli), a ich jedynymi urządzeniami peryferyjnymi są karty sieciowe i (ewentualnie) dyski. Gordon Bell, projektant komputerów PDP-11 i VAX, nazywa je „**bezgłowymi stacjami roboczymi**" (*headless workstations*), ponieważ nie są one połączone z jakimiś nadrzędnymi serwerami, a słynący z lakonicznych skrótów myślowych Amerykanie ukuli — na swój sposób makabryczny — termin „bezgłowa krowa", od *headless cluster of workstations*, czyli *headless COW*.

Komputery tworzące klaster zdecentralizowany rozproszone są zwykle w obrębie budynku lub osiedla; większość z nich używana jest przez swych właścicieli tylko przez kilka godzin w ciągu doby, więc zamiast stać bezczynnie — szczególnie w nocy — mogłyby w tym czasie robić coś pożytecznego. Są one najczęściej połączone siecią LAN, wyposażone są w komplet urządzeń peryferyjnych (choć trzeba przyznać, że klaster złożony z 1024 myszy jest równie użyteczny co brak myszy w ogóle) a co ważniejsze, ich właściciele przejawiają do nich specyficzny stosunek emocjonalny i zapewne niechętnie pozwoliliby dotykać ich klawiatury np. astronomowi, który chciałby przeprowadzić symulację Wielkiego Wybuchu. Połączenie wielu komputerów w klaster to także możliwość migrowania zadań pomiędzy komputerami — w sytuacji, gdy właściciel komputera zamierza z niego skorzystać, realizowane właśnie przezeń „obce" zadanie przekazane zostaje innemu, bezczynnemu komputerowi w sieci. Daje się to efektywnie zrealizować za pomocą specjalistycznego oprogramowania, w odniesieniu do specjalnie napisanych aplikacji.

Liczba komputerów w takim klastrze bywa zróżnicowana, lecz zwykle nie jest duża — od kilkunastu do (najwyżej) kilkuset. Możliwe jest jednak tworzenie takich klastrów, w których liczba komputerów — będących zwykłymi komputerami „z półki" — może osiągać wartość olbrzymią. Przekładem takiego klastra jest wyszukiwarka Google i dlatego warto przyjrzeć się jej bardziej szczegółowo.

Google

Google to niezwykle popularne narzędzie do wyszukiwania rozmaitych informacji w internecie. Choć główną przyczyną jej popularności jest bardzo prosty interfejs i szybki czas odpowiedzi, to już samego jej projektu „prostym" nazwać zdecydowanie nie można. Z punktu widzenia projektantów wyszukiwarki Google jej zadaniem jest znajdowanie, indeksowanie i zapamiętywanie danych z całej sieci WWW (liczącej ponad 8 miliardów stron i ponad 1 miliard obrazków), a potem odnajdywanie żądanej frazy w tym ogromie informacji, w ciągu pół sekundy! Co więcej, wyszukiwarka ma być dostępna bez przerwy, 24 godziny na dobę i sprostać musi masywnemu strumieniowi zapytań napływających nieustannie z całego świata. To jeszcze nie koniec: musi ona działać niezawodnie, nawet w obliczu trzęsienia ziemi i innych kataklizmów, nie mówiąc już o awarii sieci energetycznej, uszkodzeniach linii telekomunikacyjnych, awariach sprzętu i błędach w oprogramowaniu. A żeby było jeszcze ciekawiej, to

wszystko ma zostać zrealizowane najtaniej, jak tylko można — i faktycznie zostało, o czym się za chwilę przekonamy. W każdym razie, można odważnie stwierdzić, że zbudowanie klonu wyszukiwarki Google zdecydowanie nie nadaje się jako ćwiczenie dla Czytelnika.

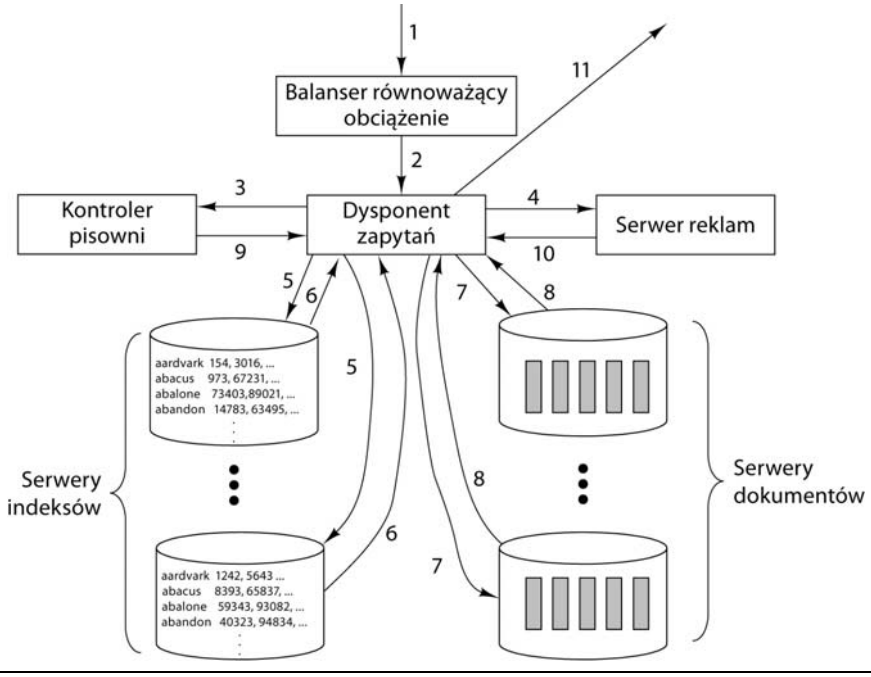
Jak więc to wszystko zostało zrobione? Przede wszystkim pracujące na rzecz Google komputery rozproszone są po licznych **centrach danych** (*data centers*) na całym świecie. Zapewnia to możliwość szybkiego backupu na wypadek, gdyby jedno z centrów pochłonięte zostało przez trzęsienie ziemi lub zmyte przez fale tsunami, lecz także powoduje automatyczne przekierowanie użytkownika wpisującego adres *www.google.com* do najbliższego mu geograficznie centrum, do którego przeglądarka wysyłać będzie formułowane żądania.

Każde ze wspomnianych centrów połączone jest z internetem za pomocą co najmniej jednego światłowodu OC-48 (o przepustowości 2,488 Gb/s) oraz za pomocą światłowodu OC-12 (622 Mb/s) z innym dostawcą telekomunikacyjnym na wypadek, gdyby łącze zasadnicze uległo uszkodzeniu. Przed konsekwencjami awarii sieci elektrycznej chronią zasilacze awaryjne (UPS) oraz agregaty prądotwórcze z silnikami Diesla. W rezultacie nawet potężny kataklizm zdolny jest jedynie pogorszyć efektywność Google, nie jest jednak w stanie sparaliżować go całkowicie.

Aby lepiej zrozumieć taki, a nie inny wybór projektantów architektury Google, przyjrzyjmy się dokładniej procesowi przetwarzania zapytania, które właśnie napłynęło do jednego z centrów danych. Na samym początku (krok 1 na rysunku 8.36) balanser równoważący obciążenie kieruje zapytanie do jednego z dysponentów (2), który rozdziela zapytanie na dwie kopie kierowane równolegle do kontrolera pisowni (3) i serwera reklam (4). W międzyczasie słowo będące treścią zapytania poszukiwane jest w serwerach indeksowych (5), zawierających informację o występowaniu niemal wszystkich możliwych słów w całej światowej sieci WWW. Każda pozycja, reprezentująca jedno słowo, jest listą wszystkich dokumentów (stron WWW, plików PDF, dokumentów MS Word, prezentacji PowerPointa itp.), w których słowo to wystąpiło, posortowanych w kolejności malejącej ważności. Ważność (ranking) strony w kontekście danego słowa ustalana jest według skomplikowanego (i nieopublikowanego) algorytmu, choć wiadomo, że dużą rolę odgrywa liczba odsyłaczy do tej strony w innych stronach WWW i rankingi tych ostatnich.

W celu osiągnięcia większej efektywności każdy indeks podzielony jest na porcje, zwane **szardami** (*shards*), które mogą być przeszukiwane równolegle. Podstawą podziału jest istotność (ranking) poszczególnych dokumentów, i tak szard nr 1 zawiera wszystkie słowa indeksu z wyszczególnieniem *n* najważniejszych dokumentów dla każdego z nich, szard nr 2 zawiera wszystkie słowa indeksu z wyszczególnieniem *n* kolejno mniej ważnych dokumentów, itd. W miarę jak światowa sieć będzie się nieuchronnie rozrastać, każdy z szardów prędzej czy później ulegnie kolejnemu podziałowi, w celu uzyskania jeszcze większego zrównoleglenia.

Serwery indeksowe zwracają następnie zbiór identyfikatorów dokumentów (6), który następnie przetwarzany jest zgodnie z ew. formułą boolowską zawartą w zapytaniu. Jeśli na przykład zapytanie będzie miało postać `+symfonia +pieśni +żałobnych`, to ze zbioru tego wybrane zostaną tylko te dokumenty, które zawierają frazę „symfonia pieśni żałobnych”. W kroku 7 dla każdego z wybranych dokumentów poszukiwany jest (w serwerach zawartości) tytuł, lokalizacja URL oraz mały fragment (fragmenty) tekstu sąsiadującego z wystąpieniem (wystąpieniami) poszukiwanej frazy. Serwery zawartości, obecne w każdym z centrów danych, zawierają kopie wszystkich zaindeksowanych stron — mniej więcej kilkaset terabajtów. Podobnie jak indeksy, także i dokumenty podzielone są na szardy dla lepszej efektywności. Mimo iż przetwarzanie



RYSUNEK 8.36. Schemat przetwarzania zapytania przez Google

zapytania rzadko kiedy wiąże się z przeszukiwaniem całej sieci WWW czy nawet jej kilku procent, to i tak przetworzenie w związku z nim 100 MB danych jest rzeczą normalną.

Gdy wynik (stanowiący odpowiedź na zapytanie) zwrócony zostanie do dysponenta zapytań (8), ten dokonuje posortowania informacji o poszczególnych dokumentach w kolejności malejącej ważności tych dokumentów. Jeśli kontroler pisowni stwierdzi, że treść zapytania może zawierać błędy, wyświetlane są sugestie co do (przypuszczalnie) poprawnej pisowni (9); na koniec do wyników zapytania dołączane są rozmaite reklamy (10) stosownie do „słów kluczowych" znalezionych w treści zapytania („hotel", „samolot" itp.) — oto jak darmowe Google zarabia na siebie. Ostatecznie całość formatowana jest do postaci strony w języku HTML (*HyperText Markup Language*) i odsyłana do przeglądarki użytkownika.

Znając już schemat działania Google, zajmijmy się teraz specyfiką jego architektury. Zazwyczaj wielkie firmy, stając w obliczu przetwarzania olbrzymich baz danych, intensywnych strumieni transakcji i wymagań dużej niezawodności, decydują się na zakup największego, najszybszego i najbardziej niezawodnego wyposażenia, jakie tylko można znaleźć na rynku. Google jest w tym względzie wyjątkiem: wykorzystuje ono tanie komputery, o umiarkowanej wydajności — ogromną liczbę takich komputerów, tworząc największy na świecie klaster „z półki".

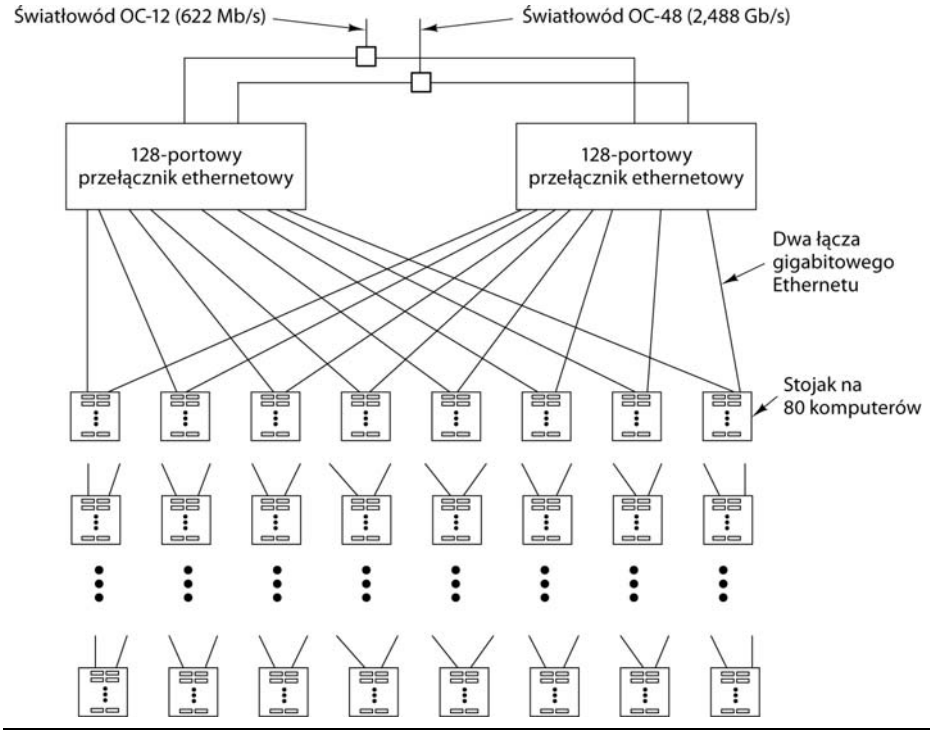
Decyzja taka potraktowana jest względami jak najkorzystniejszego stosunku ceny do wydajności. Popularne „pecety" są towarem tanim, czego nie można powiedzieć o wydajnych serwerach ani tym bardziej o wieloprocesorach. To prawda, lecz te ostatnie, zapewniając jedynie dwu- lub trzykrotnie większą wydajność kosztują 5 do 10 razy drożej, a więc wspomniany w wskaźnik jest dla nich 2 – 3 razy gorszy niż dla komputerów PC.

Oczywiście wysoką cenę płaci nabywca serwera nie tylko za większą wydajność, lecz także za większą niezawodność. Fakt, że komputery PC mogą psuć się (potencjalnie) znacznie częściej niż serwery z górnej półki, nie jest jednak w przypadku Google powodem do większego zmartwienia, bowiem oprogramowanie Google napisane zostało w taki sposób, by radzić sobie z zawodnym sprzętem niezależnie od jego klasy. Jak pokazuje praktyka, co roku psuje się ok. 2% komputerów Google, przy czym najczęstszą tego przyczyną są awarie dysków, a drugiej kolejności awarie zasilaczy i przekłamania w pamięciach RAM. Procesory psują się raczej rzadko, i tylko w jeden możliwy sposób — ulegają po prostu spaleniu. Tak naprawdę znacznie większym zmartwieniem niż zawodny sprzęt są błędy w oprogramowaniu. Podstawową reakcją na wystąpienie błędu jest po prostu ponowne uruchomienie komputera (elektroniczny odpowiednik lekarskiego zalecenia „wziąć dwie aspiryny i do łóżka").

Typowy googlowski pecet składa się z procesora Pentium 2 GHz, 512 MB pamięci RAM i dysku o pojemności 80 GB — na takim zestawie co mniej wymagający użytkownicy mogą sobie odczytywać pocztę elektroniczną lub dokonywać zakupów przez internet. Jedynym niecodziennym komponentem komputera jest (tani) chip ethernetowy. Takie właśnie komputery umieszczane są w płaskich obudowach typu 1u i układane po 40 z każdej strony szerokiego na ok. pół metra stojaka — na stojaku znajduje się więc 80 komputerów. Wszystkie one podłączone są do ethernetowego przełącznika znajdującego się wewnątrz stojaka. Przełączniki poszczególnych stojaków połączone są ze 128-portowymi przełącznikami prowadzącymi do światłowodów.

Schemat typowego centrum danych przedstawiony jest na rysunku 8.37. Sygnał nadchodzący przez światłowód OC-48 kierowany jest do dwóch przełączników 128-portowych, podobnie jak sygnał nadchodzący przez światłowód OC-12. Światłowody te używają specjalnych kart i nie zajmują żadnego z portów przełączników. Od każdego stojaka biegną cztery łącza ethernetowe, po dwa do każdego z przełączników — dzięki temu system będzie mógł działać mimo ewentualnej awarii któregoś przełącznika. Tak naprawdę, aby kompletnie wyeliminować któryś stojak, konieczna byłaby awaria bądź to wszystkich czterech wychodzących z niego łączy, bądź (odpowiednio dobranych) dwóch łączy i przełącznika. Mając jedną parę 128-portowych przełączników i cztery łącza wiodące od każdego stojaka, możemy podłączyć maksymalnie $\frac{128 * 2}{4} = 64$ stojaki; wobec 80 komputerów na stojaku daje to 5120 komputerów. Podane tu wartości są typowe, lecz niekoniecznie obowiązujące dla każdego centrum: przełączniki mogą mieć mniej albo więcej portów, a stojaki nie muszą być zapełniane maksymalną liczbą komputerów.

Przy tak dużym zagęszczeniu komputerów należy jeszcze odpowiednio rozwiązać problem odprowadzania wydzielanego ciepła. Każdy z komputerów zużywa ok. 120 W mocy, co daje prawie 10kW na stojak. Każdy stojak wymaga ok. 3 m^2 powierzchni, by personel techniczny mógł swobodnie montować i demontować komputery. Daje to zagęszczenie mocy $3000 \frac{W}{m^2}$; większość centrów danych projektowana jest z uwzględnieniem zagęszczenia $600 - 1200 \frac{W}{m^2}$, należy więc zachować znacznie większą odległość między stojakami niż ta wynikająca z wymienionych 3 m^2.



RYSUNEK 8.37. Schemat typowego centrum danych Google

W ciągu wielu lat eksploatacji Google nauczyło nas trzech rzeczy, jakie koniecznie należy zapamiętać w związku z używaniem dużych serwerów WWW:

(1) Komponenty muszą się psuć, więc uwzględnij to w swych planach.
(2) Zwielokrotniaj wszystko, co możliwe, w celu uzyskania większej przepustowości i dostępności.
(3) Optymalizuj wskaźnik stosunku ceny do wydajności.

Z pierwszej reguły wynika tyle, że musimy mieć oprogramowanie odporne na awarie. Nawet najlepszej jakości komponenty w końcu się kiedyś psują, a prawdopodobieństwo awarii układu złożonego z wielkiej liczby komponentów jest do tej liczby proporcjonalne. Oprogramowanie musi ten fakt uwzględniać i zapewnić pracę systemu niezależnie od liczby awarii w tygodniu.

Zgodnie z drugą regułą, zarówno sprzęt, jak i oprogramowanie powinny być zwielokrotniane. Wprowadzana w ten sposób redundancja nie tylko czyni system bardziej odpornym na awarie, lecz także zwiększa jego przepustowość. W przypadku Google komputery, dyski, kable, przełączniki itp. zwielokrotniane są w różnym stopniu. Ponadto, zawartość serwerów indeksowych i serwerów dokumentów podzielona jest na szardy i replikowana kilkakrotnie w każdym centrum danych. Same centra danych jako takie również są replikowane.

Trzecia reguła wynika z dwóch poprzednich. Jeśli system został prawidłowo zaprojektowany pod kątem radzenia sobie z awariami, to zakup kosztownych komponentów w rodzaju macierzy RAID z dyskami SCSI jest złym pomysłem. Wydanie

dziesięciokrotnie większej kwoty na sprzęt zapewniający dwukrotnie mniejszą częstotliwość awarii jest posunięciem mniej sensownym niż (dokonane za tę samą kwotę) 10-krotne powielenie taniego sprzętu po to, by móc radzić sobie z awariami tak często, jak tylko będą występować. Ponadto zwielokrotniony sprzęt zapewnia lepszą przepustowość.

Bardziej szczegółowe informacje na temat Google znaleźć można w pracach (Barroso i in., 2003) oraz (Ghemawat i in., 2003).

8.4.4. Oprogramowanie komunikacyjne dla wielokomputerów

Jedną z funkcji realizowanych przez oprogramowanie dla wielokomputerów (zazwyczaj w postaci funkcji bibliotecznych) jest komunikacja między procesami, niezbędna dla ich synchronizacji. W niniejszym punkcie poświęcimy temu oprogramowaniu kilka słów. W większości przypadków te same pakiety wspomnianego oprogramowania nadają się zarówno dla komputerów MPP, jak i klastrów, zatem aplikacje mogą być z łatwością przenoszone między tymi dwiema platformami.

W systemach z przekazywaniem komunikatów zwykle w danej chwili działa równolegle kilka (dwa lub więcej) procesów — na przykład jeden z procesów produkować może dane wykorzystywane przez inne procesy. Ponieważ procesy działają niezależnie od siebie, nie ma gwarancji, że w przypadku wyprodukowania przez pierwszy proces nowych danych pozostałe procesy będą przygotowane na ich przyjęcie.

Większość z systemów opartych na przekazywaniu komunikatów oferuje programiście dwa prymitywy: `send` i `receive`, powodujące (odpowiednio) wysłanie i odebranie komunikatu, choć możliwe są także inne rozwiązania semantyczne. Generalnie rzecz biorąc, istnieją trzy warianty komunikowania się procesów:

(1) Synchroniczne przekazywanie komunikatów.
(2) Buforowane przekazywanie komunikatów.
(3) Nieblokujące przekazywanie komunikatów.

W wariancie **synchronicznym**, gdy proces-nadawca wykonuje funkcję `send` w sytuacji, gdy proces-odbiorca nie wywołał jeszcze funkcji `receive`, nadawca zostaje zawieszony do momentu, aż odbiorca funkcje `receive` wywoła, w wyniku czego faktycznie przesłany (skopiowany) zostanie komunikat. Nadawca jest następnie odblokowywany („odwieszany”), co jest dla niego sygnałem, że wysłany przezeń komunikat został pomyślnie dostarczony. Ta metoda cechuje się prostą semantyką i nie wymaga buforowania, zaś jej podstawową wadą jest konieczność oczekiwania nadawcy na potwierdzenie odebrania wysłanego komunikatu.

W przypadku **buforowania** komunikatów wywołanie przez nadawcę funkcji `send` powoduje zapisanie wysyłanego komunikatu w specjalnym buforze (którym może być na przykład skrzynka pocztowa); proces-odbiorca, wywołując funkcję `receive`, dokona w rzeczywistości odczytu komunikatu z tegoż bufora. Proces wywołujący funkcję `send` **nie** zostaje zawieszony i może ponownie wysłać kolejny komunikat, bo jego **prywatny** bufor komunikatów pozostaje pusty; nie otrzymuje on jednak żadnej informacji o odbieraniu wysyłanych komunikatów, a nawet nie ma gwarancji, że

komunikaty te **w ogóle** zostaną odebrane — w najbardziej nawet niezawodnych systemach procesy ulegają awariom (zwykle z powodu błędów oprogramowania) i taki właśnie los spotkać mógł proces-odbiorcę, zanim ten zdążył wywołać funkcję `receive`.

Nieblokujące przekazywanie komunikatów bierze swą nazwę stąd, że proces-nadawca nie zostaje zablokowany po wywołaniu funkcji `send`. Komunikat pozostaje w prywatnym buforze nadawcy, o czym system operacyjny zostaje poinformowany, jednakże proces nie może wysłać kolejnego komunikatu, zanim system operacyjny nie zrobi użytku z tej informacji, czyli zanim nie opróżni prywatnego bufora procesu. Wynika stąd, że proces-nadawca musi w jakiś sposób sprawować kontrolę nad swym prywatnym buforem, bądź to za pomocą jego okresowego sprawdzania (*polling*), bądź też z wykorzystaniem mechanizmu przerwań — tak czy inaczej, nie przyczynia się to do prostszego tworzenia oprogramowania.

Zajmiemy się teraz pokrótce jednym z popularnych protokołów przekazywania komunikatów, dostępnym w większości wielokomputerów.

MPI — Message-Passing Interface

Przez kilka lat najpopularniejszym pakietem, realizującym protokoły komunikacyjne dla wielokomputerów, był **PVM**, **Paralell Virtual Machine**, czyli „wirtualna maszyna równoległa" (Geist i in., 1994) oraz (Sunderram i in., 1990). PVM został jednak z czasem wyparty przez pakiet **MPI**, **Message-Passing Interface**, „interfejs przekazywania komunikatów". MPI jest bardziej funkcjonalny i bardziej skomplikowany niż PVM: oferuje więcej opcji i więcej funkcji bibliotecznych z bardziej rozbudowanymi zestawami parametrów. Pierwsza wersja MPI, którą oznaczać będziemy MPI-1, została w 1997 roku rozbudowana do wersji, którą oznaczać będziemy MPI-2. Szczegółowe informacje na temat MPI znaleźć można w pracach (Gropp i in., 1994) oraz (Snir i in., 1996).

MPI-1, w przeciwieństwie do PVM, nie oferuje programiście żadnych funkcji związanych z tworzeniem procesów i zarządzaniem nimi: programista musi w tym celu skorzystać z lokalnych wywołań „swojego" systemu operacyjnego. Utworzone procesy zorganizowane zostają w statyczną, niezmienną grupę i właśnie takie grupy procesów stanowią dane, na których operuje MPI.

Filozofia MPI opiera się na czterech głównych koncepcjach: komunikatorach (*communicators*), strukturach danych komunikatów (*message data types*), operacjach komunikacyjnych (*communication operations*) i topologiach wirtualnych (*virtual topologies*). **Komunikatorem** nazywamy grupę procesów działających w określonym kontekście, którym fizycznie jest etykieta identyfikująca cokolwiek, na przykład fazę wykonywania. Zadaniem kontekstu jest różnicowanie niezwiązanych ze sobą komunikatów — wysyłanych i odbieranych — tak, by nawzajem ze sobą nie oddziaływały.

Pod względem fizycznym każdy komunikat jest strukturą danych określonego typu, na przykład ciągiem znaków (*characters*), liczba całkowitą — krótką (*short*), regularną (*regular*) lub długą (*long*), liczbą zmiennopozycyjną pojedynczej (*single*) lub podwójnej (*double*) precyzji; możliwe jest także definiowanie własnych typów komunikatów na bazie predefiniowanych typów podstawowych.

MPI oferuje programiście rozbudowany zestaw operacji; najbardziej bodaj podstawowa z nich służy do wysyłania komunikatów:

```
MPI_Send(bufor, licznik, typ_danych, przeznaczenie, znacznik, komunikator)
```

W wyniku powyższego wywołania zawarty w *buforze* komunikat o rozmiarze *count* elementów mający postać struktury danych typu *typ_danych* wysłany zostaje do swego *przeznaczenia*. Parametr *znacznik* umożliwia różnicowanie komunikatów — odbiorca może być bowiem zainteresowany otrzymywaniem wyłącznie komunikatów o określonym znaczniku. Parametr *komunikator* określa grupę procesów, do której komunikat jest kierowany; konkretny proces w tej grupie identyfikowany jest — za pomocą indeksu — przez parametr *przeznaczenie*.

Za pomocą analogicznej funkcji można odbierać komunikaty:

```
MPI_Recv(&bufor, licznik, typ_danych, źródło, znacznik, komunikator, &status)
```

Wywołujący tę funkcję proces sygnalizuje oczekiwanie na komunikat określonego *typu_danych*, z określonego *źródła*, identyfikowany przez określony *znacznik*.

MPI obsługuje cztery tryby wysyłania komunikatów. W trybie 1, synchronicznym, proces-nadawca nie może rozpocząć wysyłania komunikatu wcześniej, niż proces-odbiorca nie wywoła funkcji `MPI_Recv`. W trybie 2, buforowanym, wymóg ten nie obowiązuje. Tryb 3 jest trybem standardowym, a jego implementacja zależy od konkretnego środowiska, i może być implementacją synchroniczną albo asynchroniczną. W trybie 4, zwanym trybem gotowości (*ready*), wymaga się — podobnie jak w trybie synchronicznym — by odbiorca był gotów na odebranie komunikatu, lecz gotowości tej nie weryfikuje się. Każdy z tych czterech trybów może być zrealizowany w wariancie blokującym lub nieblokującym, w efekcie mamy więc osiem funkcji podstawowych (prymitywów). Odbieranie komunikatów może odbywać się w dwóch wariantach: blokującym lub nieblokującym.

MPI obsługuje różne formy komunikacji grupowej (kolektywnej), między innymi rozgłaszanie (*broadcasting*), rozpraszanie (*scattering*), gromadzenie (*gathering*), wymianę całkowitą (*total exchange*), agregację (*aggregation*) i bariery (*barrier*). „Kolektywność" komunikacji polega na tym, że biorą w niej udział wszystkie procesy z grupy i wszystkie one muszą używać kompatybilnych parametrów w wywołaniach funkcji — niespełnienie tego wymogu jest błędem. Przykładem komunikacji kolektywnej jest współdziałanie procesów zorganizowanych w strukturę drzewiastą, w której za pomocą komunikatów przesyłanych od liści w stronę korzenia następuje sumowanie wartości elementów bądź znajdowanie wartości maksymalnej.

Czwartą z podstawowych koncepcji MPI jest **topologia wirtualna**, zgodnie z którą wchodzące w skład grupy procesy mogą być logicznie zorganizowane w strukturę drzewa, pierścienia, kraty, torusa i innych jeszcze modeli. Organizacja taka umożliwia identyfikowanie (nazywanie) ścieżek przepływu komunikatów i ogólnie czyni komunikację bardziej przejrzystą.

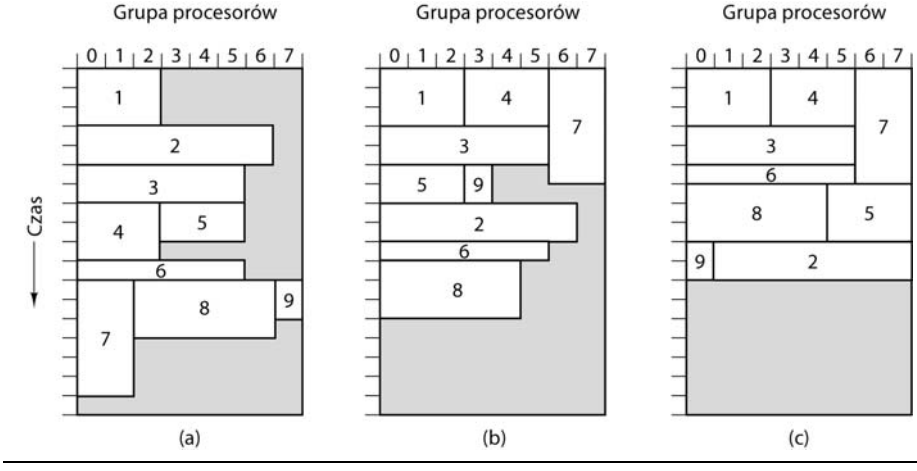
MPI-2 rozszerza funkcjonalność wersji MPI-1 o procesy dynamiczne, zdalny dostęp do pamięci, nieblokującą komunikację kolektywną, obsługę skalowalnego wejścia-wyjścia, przetwarzania w czasie rzeczywistym i wiele innych mechanizmów, których omówienie wykracza poza ramy niniejszej książki.

Środowisko naukowe przez długie lata podzielone było na zwolenników MPI i zwolenników PVM. Z najważniejszych zalet PVM wymieniano łatwość obsługi i łatwość nauczenia się go, stronnicy MPI wskazywali natomiast na jego większe możliwości oraz fakt, iż interfejs ten uznany został przez komitet normalizacyjny za standard i opisany w postaci oficjalnego dokumentu. Zdaniem sympatyków PVM wysoce zorganizowana biurokracja nie jest nikomu specjalnie potrzebna, a jej brak nie zawsze musi być mankamentem. Względy praktyczne zdecydowały ostatecznie o zwycięstwie MPI.

8.4.5. Szeregowanie zadań

Programiści korzystający z interfejsu MPI z łatwością mogą tworzyć aplikacje żądające przydziału wielu procesorów i wykonywane przez dłuższy czas. Gdy wiele niezależnych żądań tego typu napływa od różnych użytkowników, żądających przydziału rozmaitej liczby procesorów, klaster wyposażony być musi w pewien mechanizm planowania decydujący o tym, które zadania mają być wykonywane przez które procesory.

W najprostszym modelu planistycznym każde zadanie (*job*) określić musi a priori swe potrzeby, podając, ile procesorów potrzebować będzie do swej realizacji. System operacyjny wybiera do realizacji zadania w kolejności ich przybywania do kolejki wejściowej (FIFO), jak pokazano to na rysunku 8.38 (a)[27], i czyni to dopóty, dopóki nie zabraknie mu procesorów; wstrzymuje wówczas inicjowanie nowych zadań, czekając na zwolnienie wystarczającej liczby procesorów przez kończące się zadania.



RYSUNEK 8.38. Szeregowanie zadań w klastrze; obszary zacieniowane reprezentują bezczynne procesory: (a) tradycyjna metoda FIFO; (b) zoptymalizowana metoda FIFO; (c) „kafelkowanie"

Zwróćmy uwagę, że system wstrzymuje inicjowanie nowych zadań w momencie, gdy okaże się, że liczba dostępnych procesorów jest niewystarczająca dla **następnego zadania z kolejki**; fakt, że na dalszych pozycjach kolejki mogą oczekiwać zadania, dla których liczba ta byłaby wystarczająca, nie ma żadnego znaczenia.

Od mankamentu tego wolny jest algorytm, którego istotę przedstawiono na w części (b) rysunku 8.38. System stara się tu zagospodarować wolne procesory, wyszukując w kolejce zadania, dla których ich liczba jest wystarczająca (i „przeskakując" te zadania, dla których nie jest). Poszukiwanie „zdatnych do uruchomienia" zadań wykonywane jest — w kolejności FIFO — po każdym zakończeniu któregokolwiek zadania.

[27] Mimo iż z rysunku 8.38 mogłoby wynikać, że klaster złożony jest z 8 procesorów, w rzeczywistości poszczególne prostokąty oznaczać mogą nie pojedyncze procesory, lecz na przykład 16-procesorowe grupy klastra, który posiada ich 128.

Jeżeli w odniesieniu do każdego zadania określić można nie tylko wymaganą liczbę procesorów, lecz także maksymalny czas jego realizacji, możliwe jest zastosowanie jeszcze bardziej wymyślnego algorytmu szeregowania, którego istotę wyjaśniono w części (c) rysunku. Dwie wymienione wartości — wymaganą liczbę procesorów i maksymalny czas realizacji — potraktować można jako długości boków prostokąta reprezentującego zadanie; system operacyjny może sobie wówczas „wykafelkować" takimi prostokątami swą przestrzeń szeregowania, planując w ten sposób precyzyjnie (i optymalnie) momenty rozpoczynania kolejnych zadań. Metoda ta sprawdza się szczególnie dobrze w przypadku zadań wykonywanych w sposób cykliczny w określonych przedziałach czasowych (na przykład codziennie między 22:00 a 6:00 czy też ostatniego dnia miesiąca o dowolnej porze).

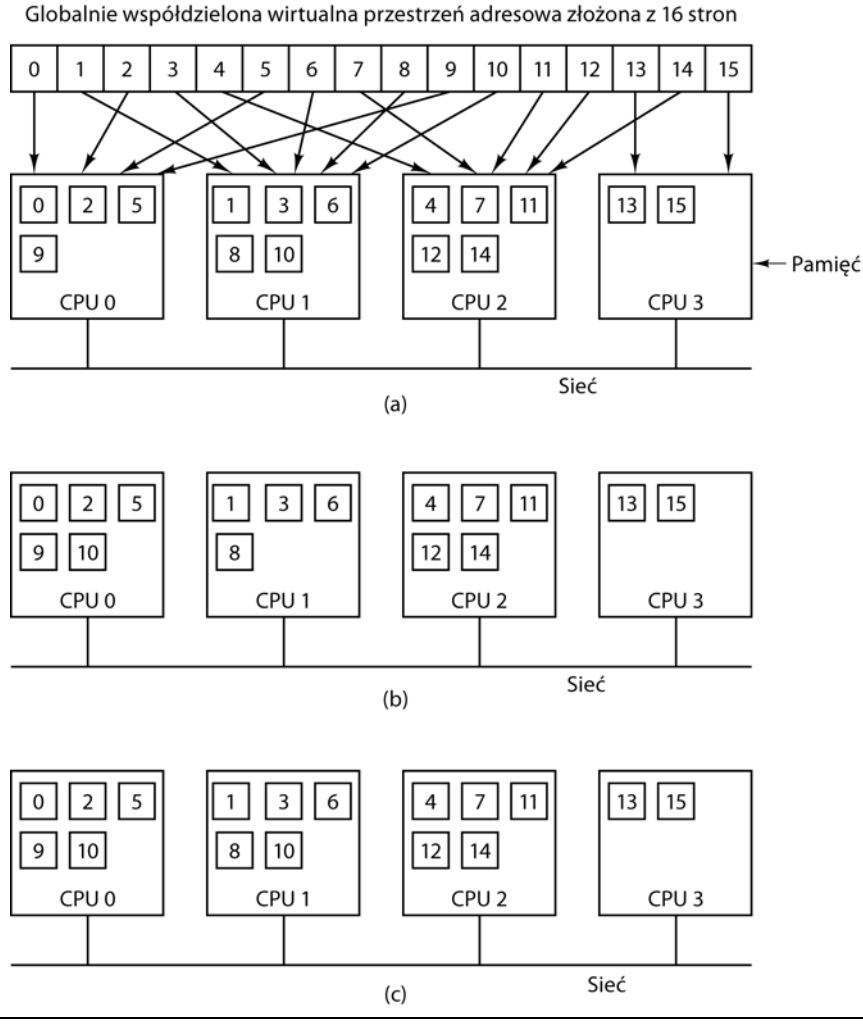
8.4.6. Implementacja współdzielonej pamięci na poziomie aplikacji

Jak już wielokrotnie pisaliśmy w niniejszym rozdziale, wielokomputery znacznie łatwiej poddają się skalowaniu w porównaniu z wieloprocesorami, lecz płaconą za to ceną jest rezygnacja z pamięci współdzielonej, na rzecz przekazywania komunikatów (na przykład za pomocą MPI) między poszczególnymi pamięciami, należącymi do różnych procesorów. Jakkolwiek programiści w zupełności rozumieją nieuchronność tego kompromisu, to jednak dla wielu z nich mechanizm współdzielenia pamięci jest na tyle atrakcyjny, że z nieuchronnością tą nie chcą się pogodzić do końca: skoro już tak jest, że nie mogą cieszyć się rzeczywistą pamięcią współdzieloną wielokomputera, to chcieliby doświadczyć przynajmniej jej iluzji. Iluzja jako taka jest istotą wszystkiego, co w swej nazwie zawiera przymiotnik „wirtualny", i te same mechanizmy, dzięki którym programy zyskują iluzję ogromnej wirtualnej przestrzeni adresowej (na komputerze, który fizycznie ma tej pamięci znacznie mniej), mogą być wykorzystane do stworzenia iluzji współdzielenia pojedynczej pamięci na wielokomputerze, który w istocie pamięć fizyczną ma podzieloną między poszczególne procesory. I w ten oto sposób daje się pogodzić dwa (wydawałoby się) sprzeczne wymagania: duży i niedrogi sprzęt („niedrogi" przynajmniej w przeliczeniu na jeden węzeł) i łatwość programowania.

Jak widzieliśmy na rysunku 8.17, symulację współdzielenia pamięci przeprowadzić można na wielu poziomach, od systemu operacyjnego poczynając, poprzez implementację języka programowania, a na samej aplikacji użytkowej kończąc. W niniejszym punkcie przyjrzymy się dokładniej wybranym przykładom wykorzystywania tych możliwości w praktyce.

Rozproszona pamięć współdzielona

Obecny w wielu procesorach mechanizm stronicowania stwarza wiele możliwości w zakresie symulowania cech, których dany system fizycznie nie posiada, między innymi pamięci współdzielonej wielokomputera. Sam pomysł nie jest niczym nadzwyczajnym: wszystkie procesory wielokomputera współdzielą pojedynczą wirtualną przestrzeń adresową. W najprostszym przypadku każda strona tej przestrzeni odwzorowana jest w dokładnie jednej pamięci lokalnej; na rysunku 8.39 (a) widzimy taką przykładową przestrzeń złożoną z 16 stron, współdzieloną przez cztery procesory.



RYSUNEK 8.39. Wirtualna 16-stronicowa przestrzeń adresowa współdzielona przez cztery procesory wielokomputera: (a) sytuacja początkowa; (b) rezultat odwołania się procesora 0 do strony 10; (c) rezultat odwołania się procesora 1 do strony 10, która jest stroną tylko do odczytu

Gdy procesor odwołuje się do strony odwzorowywanej w jego lokalną pamięć, nie dzieje się nic niezwykłego. Odwołanie się tego procesora do strony odwzorowanej w lokalną pamięć któregoś z innych procesorów powoduje wystąpienie wyjątku braku strony. Do akcji wkracza oprogramowanie obsługujące ten wyjątek (system operacyjny lub aplikacja użytkowa) i następuje „przeniesienie" strony: przestaje być ona odwzorowana w swą macierzystą pamięć i zamapowana zostaje w pamięć lokalną procesora wywołującego. Mamy tu do czynienia z wariantem stronicowania na żądanie, żądana strona nie jest jednak wczytywana z dysku, lecz kopiowana z innej pamięci lokalnej. Na rysunku 8.39 (b) widzimy rezultat przeniesienia w opisany sposób strony nr 10 z pamięci lokalnej procesora 1 do pamięci lokalnej procesora 0.

Opisany mechanizm zaimplementowany został po raz pierwszy w systemie IVY (Li i Hudak, 1986 i 1989). W systemie tym zasymulowano w pełni współdzieloną pamięć o spójności sekwencyjnej (patrz 8.3.2). Początkowa wersja tej implementacji pozostawiała wiele do życzenia pod względem efektywności, dlatego pierwszym zabiegiem optymalizacyjnym była możliwość wyróżniania wybranych stron jako przeznaczonych tylko do odczytu. Strony takie mogą być bowiem reprezentowane w wielu pamięciach lokalnych jednocześnie, bez zagrożenia utratą spójności. W sytuacji przedstawionej na rysunku 8.39 (c) procesory 0 i 1 wykorzystują oddzielne kopie strony nr 10.

Zabieg ten nie rozwiązuje jednak innego problemu, odbijającego się na efektywności w sposób drastyczny. Mowa tu o sytuacji, gdy dwa różne procesory dokonują jednocześnie modyfikacji dwóch rozłącznych obszarów tej samej strony (na przykład kilku jej początkowych słów i kilku słów końcowych). Ponieważ modyfikowana strona może być reprezentowana wyłącznie w pamięci lokalnej procesora dokonującego modyfikacji (po uprzednim unieważnieniu innych jej kopii), więc w efekcie obydwa procesory nawzajem przerzucają stronę między sobą niczym piłeczkę pingpongową. Sytuacja ta, stanowiąca odpowiednik migotania stron pamięci wirtualnej, nazywa się potocznie (i całkowicie słusznie) **zdradliwym współdzieleniem** (*false sharing*).

Problem zdradliwego współdzielenia próbowano rozwiązać na wiele sposobów. W systemie Treadmarks zrezygnowano w tym celu ze spójności sekwencyjnej na rzecz spójności zwalniania (Amza, 1996): potencjalnie modyfikowalne strony mogą być reprezentowane w postaci wielu kopii (w różnych węzłach), jednak proces zamierzający którąś z tych kopii modyfikować musi uprzednio wywołać funkcję `acquire`; następuje wówczas unieważnienie wszystkich innych kopii poza tą modyfikowaną. Po zakończeniu modyfikacji proces wywołuje funkcję `release`, zezwalając tym samym na ponowne utworzenie kilku kopii zmodyfikowanej strony.

Drugi zabieg optymalizacyjny, jaki zastosowano w systemie Treadmarks, sprowadza się do przesyłania między poszczególnymi procesorami nie kompletnej zawartości strony, lecz jedynie informacji o wprowadzonych do strony modyfikacjach. Początkowo każda strona oznaczona jest jako strona tylko do odczytu; przy pierwszej próbie jej modyfikacji występuje wyjątek ochrony, w ramach obsługi którego utworzona zostaje kopia strony, zwana popularnie jej **bliźniaczką** (*twin*), sama zaś strona oryginalna może być odtąd bez przeszkód modyfikowana. Gdy jednak do strony tej nastąpi zdalne odwołanie, jej zawartość porównywana jest z zawartością siostry-bliźniaczki i do odwołującego się procesora przesłany zostaje komunikat zawierający informację o różnicach między nimi. Redukuje to rozmiar komunikatów i poprawia przepustowość sieci połączeniowej.

Pozostaje jeszcze problem zlokalizowania brakującej strony w przypadku wystąpienia wyjątku. Jak widzieliśmy już wcześniej w niniejszym rozdziale, istnieją różne sposoby rozwiązania tego problemu, między innymi wykorzystywana w komputerach NUMA i COMA koncepcja węzłów macierzystych i katalogów. W istocie wiele rozwiązań wykorzystywanych w realizacji DSM da się również zastosować w przypadku komputerów NUMA i COMA, bowiem DSM jest tak naprawdę programową implementacją tego, co w komputerach NUMA i COMA dzieje się z pamięciami podręcznymi.

Tematyka DSM jest wciąż obszarem intensywnych badań. Spośród praktycznych rozwiązań, które powstały na bazie tej koncepcji, wymienić należy między innymi systemy CASHMERE (Kontothanassis i in., 1997) oraz (Stets i in., 1997), CRL (Johnson i in., 1995), Shasta (Scales i in., 1996) oraz Trademarks (Amza, 1996) oraz (Lu i in., 1997).

Linda

Bazujące na koncepcji DSM systemy IVY i Treadmarks wykorzystują sprzęt jednostki MMU do przechwytywania odwołań do brakujących stron. Mimo skuteczności tego mechanizmu i różnych jego udoskonaleń — jak przesyłanie informacji o zmianach zamiast kompletnej zawartości strony — niezmiennym pozostaje fakt, że strona, jako wynik fizycznego podziału pamięci wirtualnej, nie ma żadnego związku z logiką programu i wykorzystywanie jej jako jednostki współdzielenia informacji między procesami jest cokolwiek nienaturalne.

Jednym z rozwiązań mających tę nienaturalność zniwelować jest narzucenie określonej struktury na współdzieloną pamięć, będącą dotąd jedynie zbiorem stron. W języku Linda (Carriero i Gelernter, 1989) pamięć ta traktowana jest jako zbiór **krotek** (*tuples*) i nazywana wprost **przestrzenią krotek** (*tuple space*). Obsługę tej przestrzeni zapewniają rozszerzenia dodawane do istniejących języków programowania — między innymi C i FORTRAN-u — czyniące z tych języków odmiany przeznaczone do programowania współbieżnego, o nazwach C-Linda i FORTRAN-Linda.

Sama krotka jest pojedynczym polem lub ciągiem kilku pól; w języku C-Linda pola mogą być liczbami całkowitymi, liczbami zmiennopozycyjnymi, tablicami, łańcuchami lub strukturami (ale nie innymi krotkami). Trzy przykładowe krotki przedstawiliśmy na listingu 8.1.

LISTING 8.1.
Przykładowe krotki języka Linda

```
("abc", 2, 5)
("matrix-1", 1, 6, 3.14)
("rodzina", "jest bratem", Karol, Edward)
```

Na krotkach wykonywać można cztery operacje. Za pomocą operacji `out` można dodać nową krotkę do przestrzeni krotek, na przykład wywołanie:

```
out("abc", 2, 5);
```

dodaje do tej przestrzeni nową krotkę `("abc", 2, 5)`. Parametry wywołania funkcji `out` mogą być stałymi, zmiennymi lub wyrażeniami, jak w poniższym wywołaniu:

```
out("matrix-1", i, j, 3.14);
```

powodującym dodanie nowej krotki, której wartość drugiego i trzeciego pola zależna jest od bieżącej wartości zmiennych *i* oraz *j*.

Wydobywanie krotek z przestrzeni odbywa się za pomocą wywołań funkcji `in`. Żądana krotka identyfikowana jest przez zawartość, nie przez nazwę czy adres. Argumenty wywołania funkcji `in` mogą być wyrażeniami lub parametrami formalnymi. Przykładowo wywołanie:

```
in("abc", 2, ? i);
```

powoduje „wyszukanie" dowolnej krotki zawierającej (kolejno) łańcuch „abc", liczbę 2 oraz dowolną liczbę całkowitą (przy założeniu, że zmienna *i* zadeklarowana została jako całkowita). Znaleziona krotka jest jednocześnie *usuwana* z przestrzeni krotek,

a do zmiennej *i* wpisywana jest wartość jej trzeciego pola. Wyszukiwanie i usuwanie znalezionej krotki realizowane jest jako niepodzielna (atomowa) sekwencja, jeśli zatem dwa różne procesy prowadzić będą identyczne poszukiwanie, tylko jeden z nich żądaną krotkę znajdzie (chyba że w przestrzeni znajdować się będzie kilka krotek spełniających kryteria poszukiwania). Nic nie stoi na przeszkodzie, by w przestrzeni krotek obecnych było wiele kopii tej samej krotki.

Algorytm dopasowywania, wykorzystywany przez funkcję `in`, nie jest zbyt skomplikowany. Podany w wywołaniu funkcji zestaw parametrów traktowany jest jako **szablon** (*template*) i (koncepcyjnie) porównywany z zawartości każdej krotki znajdującej się w przestrzeni. Krotkę uznaje się za pasującą do szablonu, jeśli spełnione są wszystkie trzy następujące kryteria:

(1) Krotka posiada tę samą liczbę pól co szablon.
(2) Typy poszczególnych pól krotki są identyczne z typami odpowiadających im pól szablonu.
(3) Wartość każdego pola krotki musi być zgodna z wartością odpowiadającego mu pola szablonu.

Parametry formalne — zapisywane z użyciem znaku zapytania, po którym następuje nazwa zmiennej lub nazwa typu — uczestniczą w procesie dopasowywania jedynie w tym sensie, że sprawdzany jest ich typ. Jeśli parametr formalny określa zmienną (nie typ), to do zmiennej tej wpisywana jest wartość odpowiedniego pola znalezionej krotki.

Jeśli w przestrzeni krotek nie ma ani jednej krotki, która spełniałaby kryteria wyszukiwania, proces wywołujący funkcję `in` zostaje zawieszony w oczekiwaniu na umieszczenie w przestrzeni choć jednej takiej krotki przez inny proces. Wykonywanie funkcji `in` zostanie wtedy zakończone i da w wyniku tę właśnie nową krotkę (z jednoczesnym usunięciem jej z przestrzeni). Dzięki takiemu rozwiązaniu obojętne jest, który z procesów — ten umieszczający krotkę w przestrzeni czy ten wydobywający ją stamtąd — pierwszy wywoła stosowną funkcję, odpowiednio `out` i `in`.

Możliwe jest odczytywanie krotek bez usuwania ich z przestrzeni — dokonuje się tego za pomocą funkcji `read`. Ostatnia ze wspomnianych operacji na krotkach — `eval` — podobna jest do operacji `out`: powoduje ona utworzenie nowych procesów do obliczenia wartości każdego z parametrów wywołania, a utworzona z tych wartości krotka dodawana jest do przestrzeni krotek. W taki oto sposób można w języku Linda programować obliczenia równoległe.

Powszechnie wykorzystywanym paradygmatem programistycznym języka Linda jest **model powielonego wykonawcy** (*replicated worker model*). Model ten bazuje na koncepcji **walizki z zadaniami** (*task bag*) pełnej prac (*jobs*) do wykonania. „Walizka" ta jest de facto przestrzenią krotek, a znajdujące się w niej zadania — poszczególnymi krotkami. Główny proces („ekonom") wykonuje pętlę zawierającą wywołania:

```
out("task-bag", job);
```

powodujące dodawanie do „walizki" nowych zadań, zaś każdy z procesów-wykonawców rozpoczyna swą pracę od wywołania:

```
in("task-bag", ?job)
```

udostępniającego kolejną krotkę opisującą zadanie do wykonania. Po wykonaniu zadania proces-wykonawca powraca do wywołania funkcji `in`. Oczywiście wykonanie zadania może się wiązać z dodawaniem nowych prac do wspomnianej „walizki". W ten oto prosty sposób praca dzielona jest dynamicznie między wykonawców, z których każdy zajęty jest przez cały czas. Odbywa się to przy stosunkowo niewielkich zabiegach programistycznych.

Opracowano różne implementacje języka Linda dla wielokomputerów. W każdej z nich podstawowym problemem jest sposób rozdzielenia krotek między komputery i ich wyszukiwanie w przestrzeni — to ostatnie odbywa się na przykład z wykorzystaniem rozgłaszania (*broadcasting*) lub przy użyciu systemu katalogów. Równie istotnym problemem jest replikacja procesów. Zagadnienia te dyskutowane są w pracy (Bjornson, 1993).

Orca

Nieco odmienne podejście do współdzielenia pamięci na poziomie aplikacji opiera się na pełnowartościowych obiektach (zamiast na krotkach) jako jednostkach współdzielenia informacji. Każdy z obiektów zawiera (ukrytą) informację o stanie oraz metody, które mogą ten stan zmieniać i odczytywać. Dzięki uniemożliwieniu programistom bezpośredniego operowania na stanie obiektu możliwe stało się współdzielenie obiektów między komputerami nie dysponującymi pamięcią współdzieloną.

Jednym z „obiektowych" systemów stwarzających iluzję współdzielonej pamięci wielokomputera jest Orca (Bal, 1991), (Bal i in. 1992) oraz (Bal i Tanenbaum, 1988). Orca jest tradycyjnym językiem programowania, bazującym na języku Modula 2, wzbogaconym o dwie nowości: obiekty i tworzenie nowych procesów. Obiekty języka Orca są abstrakcyjnymi typami danych, podobnie jak obiekty w języku Java czy pakiety w języku Ada. Realizują one enkapsulację (hermetyzację) wewnętrznych struktur danych i stworzonych przez użytkownika metod, zwanych **operacjami**. Obiekty są tworami biernymi (pasywnymi) — nie zawierają wątków, które mogłyby wysyłać komunikaty; procesy mogą operować na obiektach, wywołując ich metody.

Każda ze wspomnianych metod składa się z pary (*wartownik*, *ciało*). **Wartownik** (*guard*) to słowo kluczowe **guard**, po którym występuje wyrażenie boolowskie i słowo kluczowe **do**. Wykonywanie **ciała**, będącego w istocie blokiem instrukcji zakończonym słowem **od**, nie może rozpocząć się wcześniej, niż wyrażenie to będzie miało wartość `true`. Na listingu 8.2 przedstawiono implementację stosu, jako obiektu języka Orca z dwiema operacjami `push` i `pop`, powodującymi (odpowiednio) umieszczenie na stosie elementu typu `integer` i zdjęcie tego elementu ze stosu.

LISTING 8.2.
Uproszczony obiekt stosu języka Orca, zawierający wewnętrzne dane i dwie operacje

```
Object implementation stack;
  top: integer;                          # pamięć stosu
  stack: array[integer 0..N-1] of integer;

  operation push(item: integer);         # funkcja nie zwracająca
                                         # wyniku
  begin
    guard top < N-1 do
```

```
      stack[top] := item;                # umieszczenie elementu na
                                         # stosie
      top := top + 1;                    # inkrementacja wskaźnika
                                         # stosu
    od;
  end;

  operation pop(): integer;              # funkcja zwracająca wartość
                                         # typu integer
  begin
    guard top > 0 do                     # oczekiwanie, gdy stos jest
                                         # pusty
      top := top - 1;                    # dekrementacja wskaźnika
                                         # stosu
      return stack[top];                 # zwrócenie szczytowego
                                         # elementu stosu
    od;
  end;

begin
  top := 0;                              # inicjacja
end;
```

Gdy tylko obiekt `stack` zostanie zdefiniowany, można deklarować zmienne reprezentujące jego egzemplarze:

```
s, t: stack;
```

Każda z tak zadeklarowanych zmiennych zostaje automatycznie zainicjowana przez ustawienie (wewnętrznej) wartości `top` na 0. Umieszczenie wartości całkowitej `k` na stosie reprezentowanym przez zmienną *s* odbywa się w wyniku wykonania instrukcji:

```
S$push(k);
```

Wartownicy w obydwu operacjach zapewniają spójność stosu: nie jest możliwe umieszczenie elementu na stosie całkowicie zapełnionym ani zdjęcie elementu z pustego stosu. Jeśli stos `s` jest stosem pustym, to wykonanie instrukcji:

```
k := s$pop();
```

zostanie zawieszone do momentu, aż któryś proces umieści jakiś element na stosie `s`.

W języku Orca można kreować nowe procesy na wskazanych procesorach. Służy do tego instrukcja **fork** o postaci:

```
fork procedura on procesor;
```

Do nowego procesu mogą być przekazywane parametry, w tym obiekty — to naturalny sposób rozpraszania obiektów między komputerami klastra. Przykładowo, instrukcja:

```
for i in 1 .. n do fork foobar(s) on i; od;
```

generuje nowe procesy na każdym z komputerów od 1 do `n`, a każdy z tych procesów realizuje obliczenie będące treścią procedury `foobar` wywołanej z parametrem `s`. Ponieważ wszystkie te procesy — wraz z kreującym je procesem macierzystym — współdzielą ten sam obiekt `s` reprezentujący stos, wszystkie one mogą umieszczać na tym stosie elementy i zdejmować je z niego tak, jak gdyby wykonywane były na wieloprocesorze współdzielącym pojedynczą pamięć. Za stwarzanie tej iluzji odpowiedzialne są biblioteki wykonywalne (*runtime*) implementacji języka.

Operacje wykonywane na obiektach są niepodzielne i spójne sekwencyjnie. Oznacza to gwarancję ze strony systemu, że jeżeli kilka procesów operować będzie prawie równocześnie na tym samym obiekcie, to z punktu widzenia każdego z tych procesów kolejność poszczególnych operacji będzie identyczna (choć ustalona dowolnie przez system).

Język Orca integruje współdzielenie danych i synchronizację w sposób niespotykany w „stronicowanych" systemach DSM. Wykonywane równolegle procesy wymagają dwóch rodzajów synchronizacji. Pierwszy z nich wiąże się z wzajemnym wykluczaniem w przypadku próby jednoczesnego dostępu do krytycznych regionów kodu. W języku Orca każda operacja na obiekcie jest w istocie takim krytycznym regionem, istnieje bowiem gwarancja, że końcowy efekt operowania na obiekcie będzie taki sam, jak gdyby wszystkie krytyczne operacje wykonywane były sekwencyjnie, bez nakładania się w czasie ze strony różnych procesów. Pod tym względem obiekty języka Orca stanowią rozproszoną formę monitorów (Hoare, 1975).

Drugi ze wspomnianych rodzajów synchronizacji to synchronizacja zapewniana przez wartowników, polegająca na zawieszaniu procesów w oczekiwaniu na spełnienie określonych warunków. W przykładzie przedstawionym na listingu 8.2 proces usiłujący zdjąć element z pustego stosu zostaje zawieszony do momentu, aż na stosie tym znajdzie się choć jeden element.

Biblioteki wykonywalne (*runtime*) języka Orca zarządzają replikacją obiektów, ich migracją, spójnością i wywoływaniem operacji. Każdy obiekt może znajdować się w jednym z dwóch stanów: może być unikalny (*single-copy*) lub powielony (*replicated*). Obiekt unikalny istnieje na jednej tylko maszynie, więc wszelkie odwołania do niego mają charakter lokalny. Kopie obiektu powielonego znajdują się natomiast na każdej maszynie, na której realizowany jest proces odwołujący się do tego obiektu; umożliwia to efektywniejsze odczytywanie obiektu (bo dokonywane w sposób lokalny) za cenę bardziej skomplikowanej i bardziej czasochłonnej jego modyfikacji. Gdy proces zamierza wywołać operację modyfikującą powielony obiekt, musi wpierw uzyskać unikalny numer sekwencyjny (za pośrednictwem centralnego procesu, który zajmuje się przydzielaniem takich numerów). Numer ten stanowi integralną część komunikatu wysyłanego do każdej maszyny, związanego z uaktualnianiem kopii obiektu. Uporządkowanie tych komunikatów (przez każdą z maszyn docelowych) w kolejności rosnących numerów sekwencyjnych gwarantuje identyczna dla wszystkich procesów kolejność wykonywania operacji modyfikujących, co stanowi istotę spójności sekwencyjnej.

Globe

Większość systemów DSM oraz systemów opartych na językach Linda i Orca funkcjonuje na bazie lokalnych sieci obejmujących swym zasięgiem pojedynczy budynek lub osiedle. Symulowane współdzielenie pamięci jest jednak możliwe także w większej skali, nawet w skali całego świata. W systemie Globe obiekt może być zlokalizowany

w przestrzeni adresowej kilku procesów jednocześnie, nawet jeśli procesy te realizowane będą na różnych kontynentach (Kermarrec i in., 1998), (Popescu i in., 2002) oraz (van Steen i in., 1999). Dostęp do obiektu możliwy jest jedynie za pośrednictwem jego metod, co umożliwia ukrycie szczegółów implementacyjnych przed użytkownikami i implementowanie różnych obiektów według różnych strategii. Przykładowo, strategia dostępu do danych obiektu może być kwestią wyboru między utrzymywaniem tych danych w jednym egzemplarzu i dynamicznym ich pobieraniem, gdy będą potrzebne (wariant korzystny w przypadku częstych aktualizacji przez jednego właściciela), a utrzymywaniem ich kopii w każdym obiekcie i uaktualnianiem ich za pomocą efektywnych protokołów typu *multicast*.

Tym, co w systemie Globe imponujące, są projekty jego skalowania do rozmiaru *miliarda* użytkowników i *biliona* obiektów (w większości mobilnych). Lokalizacja obiektów, zarządzanie nimi i w ogóle realizacja samego skalowania to niebagatelne wyzwania stojące przed projektantami. Globe sprostać ma tym wyzwaniom poprzez ustanowienie pewnych ram, w zakresie których każdy obiekt realizuje swe własne strategie powielania, bezpieczeństwa itp. Pozwoli to uniknąć problemów typowych dla rozwiązań „uniwersalnych", stosowanych w innych systemach, z jednoczesnym zachowaniem prostoty programowania oferowanej przez współdzieloną pamięć.

Spośród innych systemów o dużym rozproszeniu geograficznym wymienić należy Globus (Foster i Kesselman, 1998a i 1998b) i Legion (Grimshaw i Wulf, 1996 i 1997); w przeciwieństwie do systemu Globe systemy te nie oferują iluzji współdzielonej pamięci.

8.4.7. Wydajność

Głównym celem budowania komputerów równoległych jest możliwość wykonywania szybszych obliczeń niż te możliwe do wykonania na komputerze z jednym procesorem. Rozwiązania niespełniające tego oczywistego warunku nie są w ogóle warte, by się nimi zajmować. Przyspieszenie obliczeń musi być także dokonane w sposób ekonomicznie uzasadniony: komputer dwukrotnie szybszy, lecz kosztujący 50 razy więcej niż komputer jednoprocesorowy nie ma raczej szans na to, by stać się hitem sezonu. W niniejszym punkcie rozważymy niektóre aspekty związane z wydajnością architektury komputerów równoległych.

Metryki sprzętowe

Z perspektywy sprzętowej jedynymi interesujący miernikami wydajności są: szybkość procesora, szybkość wejścia-wyjścia oraz szybkość połączeń. Dwóch pierwszych wartości nie sposób zmienić przez zrównoleglenie, najważniejszym parametrem postrzeganej sprzętowo wydajności pozostaje więc szybkość sieci połączeniowej oraz czynniki, które tę szybkość kształtują — przede wszystkim opóźnienie i przepustowość.

Opóźnienie obustronne (*round-trip latency*) to czas, jaki upływa od wysłania przez procesor pakietu do uzyskania odpowiedzi. Gdy pakiet wysyłany jest do pamięci, opóźnienie to jest miarą czasu zapisu lub odczytu słowa bądź bloku słów. Gdy pakiet wysyłany jest do innego procesora, opóźnienie to stanowi miarę czasu komunikacji międzyprocesorowej dla pakietów o takim, a nie innym rozmiarze. Zwykle interesująca jest wartość opóźnienia dla bardzo małych pakietów, wielkości jednego słowa lub jednego wiersza pamięci cache.

Wielkość opóźnienia kształtowana jest przez wiele czynników, różnych dla sieci z komutacją obwodów (*circuit-switched*), przekazywania z zapamiętywaniem (*store-and-forward*), wirtualnego przecinania (*virtual cut-thorugh*) i sieci tunelowych (*wormhole*). W przypadku komutacji obwodów wielkość opóźnienia jest sumą czasu zestawiania połączenia i czasu transmisji. Aby zestawić połączenie, konieczne jest wysłanie pakietu próbnego, w celu zarezerwowania odpowiednich zasobów, i uzyskanie odpowiedzi na ten pakiet. Potem można już poskładać pakiet i w końcu wysłać go z pełną prędkością. Jeśli więc czas zestawiania połączenia równy jest T_s, a wielkość pakietu wynosi p bitów, to przy przepustowości b jednostronne opóźnienie będzie mieć wielkość równą $T_s + \frac{p}{b}$. Jeśli obwód jest obwodem pełnodupleksowym, wysłanie odpowiedzi nie wymaga powtórnego nawiązywania połączenia, zatem minimalne opóźnienie związane z wysłaniem p-bitowego pakietu i uzyskaniem p-bitowej odpowiedzi wynosi $T_s + \frac{2p}{b}$.

W przypadku komutacji pakietów (*packet switching*) nie ma potrzeby wysyłania pakietu próbnego a priori, w dalszym ciągu występuje jednak opóźnienie T_a związane ze składaniem pakietu. Opóźnienie jednostronne równe jest $T_a + \frac{p}{b}$, jest to jednak de facto tylko opóźnienie związane z przekazaniem pakietu do pierwszego przełącznika. Sam węzeł wnosi pewne opóźnienie T_d, związane głównie z czasem przetwarzania pakietu i jego przebywania w kolejce w oczekiwaniu na zwolnienie portu wyjściowego. Pakiet przekazywany jest następnie do kolejnego przełącznika i sytuacja się powtarza. Jeśli więc mamy n przełączników, to całkowita wartość jednostronnego opóźnienia wynosi $T_a + n\left(\frac{p}{b} + T_d\right) + \frac{p}{b}$, gdzie ostatni składnik związany jest z transmisją pakietu od ostatniego przełącznika do miejsca przeznaczenia.

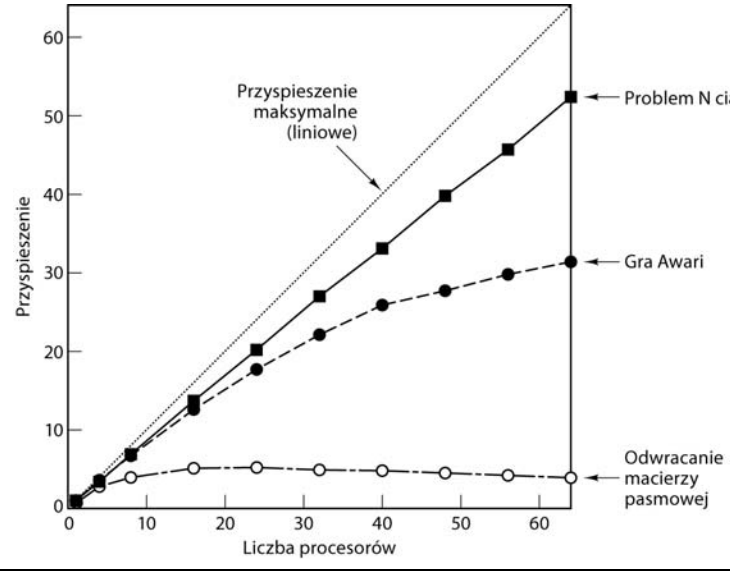
Opóźnienie jednostronne w przypadku wirtualnego przecinania i sieci typu *wormhole* w najlepszym przypadku bliskie jest $T_a + \frac{p}{b}$, nie jest bowiem konieczne wysyłanie pakietu próbnego, nie ma też opóźnień związanych z zapamiętywaniem pakietu w węźle. Występuje jedynie opóźnienie związane ze składaniem pakietu i wysyłaniem go przez port. Opóźnienie wynikające z propagacji pakietu ma w tym przypadku wartość pomijalną.

Kolejną sprzętową metryką wydajności jest przepustowość (*bandwidth*). Wiele programów równoległych, głównie w badaniach przyrodniczych, dokonuje przetwarzania ogromnych ilości danych, więc z punktu widzenia tych programów liczba bajtów, jakie system przetransmitować może w czasie jednej sekundy, jest zwykle czynnikiem krytycznym. Przepustowość mierzyć można za pomocą różnych metryk: o jednej z nich — przepustowości przepołowienia (*bisection bandwidth*) — pisaliśmy już wcześniej w niniejszym rozdziale. Dodając do siebie przepustowość wszystkich łączy wychodzących z danego węzła otrzymujemy inną metrykę — **przepustowość zagregowaną** (*aggregated bandwidth*) — odzwierciedlającą maksymalną szybkość, z jaką strumienie bitów mogą wpływać do danego węzła i wypływać z niego. Bardzo istotną metryką jest także **średnia przepustowość** (*average bandwidth*) każdego z węzłów: jeśli procesor zdolny jest wysyłać dane z szybkością 1 MB/s, to fakt, że przepustowość przepołowienia kształtuje się w granicach 100 GB/s, ma raczej niewielkie znaczenie, bo same procesory są czynnikiem silnie tę przepustowość ograniczającym.

W praktyce jednak osiągnięcie przepustowości zbliżonych choćby do wartości teoretycznych odzwierciedlanych przez metryki jest rzeczą niesłychanie trudną, ze względu na liczne źródła potencjalnych opóźnień. Poskładanie pakietu, opatrzenie go nagłówkiem, wysłanie — to wszystko wymaga czasu: wysyłając 1024 pakiety o rozmiarze (netto — bez nagłówka) 4-bajtów każdy, nigdy nie osiągniemy takiej przepustowości jak w przypadku wysyłania jednego pakietu o rozmiarze 4096 bajtów. Nie oznacza to jednak, że duże pakiety są po prostu lepsze, bo wysyłanie małych pakietów wiąże się z mniejszymi opóźnieniami: małe pakiety krócej blokują linie transmisyjne i szybciej ulegają przełączaniu. Prowadzi to do konieczności rozstrzygania nieuchronnych kompromisów między opóźnieniem a przepustowością, a ostatecznym, rozstrzygającym kryterium jest zwykle specyfika samej aplikacji. Warto jednak zwrócić uwagę na oczywisty skądinąd fakt, że przepustowość zawsze można zwiększyć, dokupując dodatkowy sprzęt (dodatkowe łącza) lub modyfikując istniejący (szybsze łącza). Redukcji opóźnienia niestety „dokupić" się nie da.

Metryki programowe

Sprzętowe metryki wydajności w rodzaju przepustowości i opóźnienia są dobrą miarą możliwości sprzętu, programistów interesują jednak raczej pośrednio. Tym bowiem, co dla programisty najważniejsze, jest kwestia, na ile szybciej wykonywać się będą tworzone przez niego programy, gdyby przenieść je z komputera z jednym procesorem do środowiska wieloprocesora lub wielokomputera z *n* procesorami. Przede wszystkim należy zdawać sobie sprawę z faktu, że liczba procesorów w komputerze to tylko jeden z czynników, najwięcej bowiem zależy od specyfiki samego programu. Na rysunku 8.40 widoczny jest wykres ilustrujący stopień przyspieszenia — w funkcji liczby procesorów, od 1 do 64 — trzech różnych programów równoległych: programu symulującego oddziaływanie N ciał (*N-body problem*), programu rozgrywającego



RYSUNEK 8.40. Przyspieszenie szybkości obliczeń trzech różnych typów programów wskutek zwiększania liczby procesorów

afrykańską grę planszową Awari i programu obliczającego macierz odwrotną do zadanej macierzy pasmowej. Zaznaczono także teoretyczną granicę możliwego przyspieszenia, identycznego ze stopniem zwielokrotnienia liczby procesorów. Jak wykazały eksperymenty, najbardziej wrażliwym na sprzętowe zrównoleglenie jest problem oddziaływania N ciał; w programie rozgrywającym grę planszową zwiększanie liczby procesorów odnosi wyraźny skutek tylko do pewnej granicy, natomiast odwracanie macierzy zdaje się być na zwiększanie liczby procesorów zupełnie niewrażliwe. Więcej informacji na temat samych programów oraz wyników eksperymentu znaleźć można w pracy (Bal i in., 1998).

Podstawowym powodem tego, że różne programy równoległe różnie reagują na zwiększanie liczby procesorów (w sposób daleki od liniowego przyspieszenia) jest oczywisty fakt, że nawet program wysoce równoległy musi posiadać w swym ciele fragmenty ściśle sekwencyjne: nadawanie początkowych wartości zmiennym, wczytywanie danych, formatowanie i drukowanie wyników, itp. Fragmenty takie stanowią coś na kształt sekcji krytycznych, które w danej chwili mogą być wykonywane tylko przez jeden proces niezależnie od tego, ile równoległych procesów w ramach danego zadania faktycznie zostało uruchomionych. Zależność możliwego do uzyskania przyspieszenia programu równoległego w zależności od jego specyfiki (i oczywiście liczby procesorów) ujmuje ilościowo **prawo Amdahla**. Oznaczmy przez T_s i T_r łączny czas wykonywania (odpowiednio) ściśle sekwencyjnego i pozostałego („równoległego") kodu danego programu, w przypadku wykonywania go na komputerze z jednym procesorem. Gdy procesorów będzie *N*, czas wykonywania kodu sekwencyjnego nie zmieni się, natomiast czas wykonania kodu „równoległego" zmniejszy się *N*-krotnie, co schematycznie zilustrowano na rysunku 8.41. Uzyskamy tym samym przyspieszenie wynoszące:

$$\eta = \frac{T_s + T_r}{T_s + \frac{T_r}{N}}$$

Oznaczmy przez *f* „stopień sekwencyjności" programu, a przez *T* łączny czas jego wykonywania:

$$T = T_s + T_r \qquad f = \frac{T_s}{T}$$

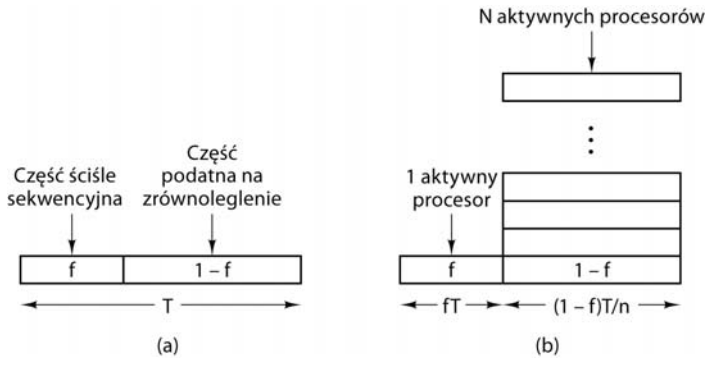
Mamy wówczas:

$$\eta = \frac{T}{T_s + \frac{T_r}{N}} = \frac{T}{Tf + \frac{T(f-1)}{N}} = \frac{1}{f + \frac{1-f}{N}} = \frac{N}{Nf + 1 - f}$$

i ostatecznie:

$$\eta = \frac{N}{1 + (N-1)f}$$

Dla $f = 0$ otrzymujemy przyspieszenie liniowe ($\eta = N$), dla $f > 0$ przyspieszenie jest z konieczności mniejsze od liniowego. Dla programu ściśle sekwencyjnego ($f = 1$) przyspieszenie jest w ogóle niemożliwe ($\eta = 1$).



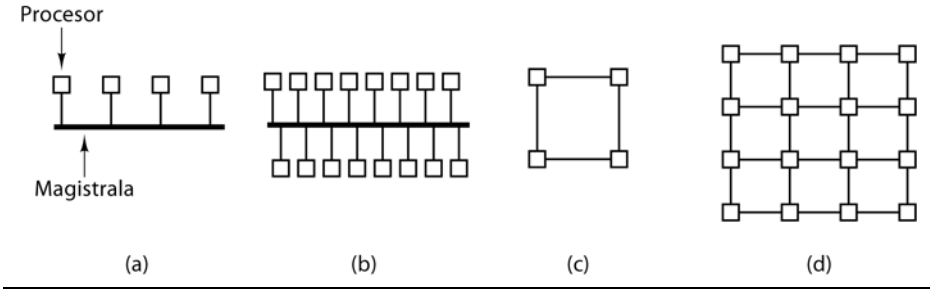
RYSUNEK 8.41. Przyspieszenie programu równoległego w zależności od stopnia jego sekwencyjności

Ograniczenie wyrażone przez prawo Amdahla nie jest jedyną przyczyną praktycznej nieosiągalności maksymalnego przyspieszenia. Oprócz innych tak oczywistych przyczyn jak niezerowa wartość opóźnień i skończona wartość pasma transmisji dużą rolę grają ograniczenia wynikające ze sposobu tworzenia programów. Rzadko który program napisany jest w ten sposób, że w nieskończoność sięgać może po kolejne procesory, jeśli tylko są dostępne. Graniczna wartość liczby używanych procesorów wynikać może na przykład z rozmiaru zadeklarowanych struktur danych, lecz tkwi ona organicznie w wielu wykorzystywanych algorytmach: przykładowo, poszukiwanie wartości maksymalnej n liczb bądź obliczanie iloczynu skalarnego dwóch wektorów n-elementowych nie daje się podzielić na więcej niż n równoległych obliczeń i zwiększanie liczby procesorów ponad tę granicę jest z punktu widzenia takich programów mało interesujące. Co do samych algorytmów, często bywa tak, że najbardziej optymalny (pod względem złożoności obliczeniowej) algorytm rozwiązujący dany problem w ogóle nie poddaje się zrównolegleniu, daje się natomiast zrównoleglić inny algorytm dla tego problemu, lecz o znacznie większej złożoności. W efekcie zysk wynikający ze zrównoleglenia osłabiany jest konsekwencjami użycia suboptymalnego algorytmu. Można mówić o szczęściu, gdy dany program wolny jest od tego rodzaju ograniczeń algorytmicznych i udaje się uzyskać jego n-krotne przyspieszenie przez zastosowanie $2n$ czy $3n$ procesorów. W końcu procesory są tanie, a poza tym — ile firm tak naprawdę uzyskuje 100-procentową efektywność w swej działalności biznesowej?

Uzyskiwanie dużej wydajności

Oczywistym zabiegiem mogącym zwiększyć wydajność systemu komputerowego jest zwiększenie liczby wchodzących w skład niego procesorów. Zabieg ten okaże się jednak skuteczny tylko wówczas, gdy zwiększanie liczby procesorów nie będzie prowadzić do powstawania różnego rodzaju wąskich gardeł. System, w którym zasada i „dołożenie" nowych procesorów faktycznie skutkuje większą wydajnością, nazywamy **systemem skalowalnym**.

By lepiej zrozumieć niektóre implikacje skalowalności, rozpatrzmy najpierw zestaw czterech procesorów połączonych wspólną magistralą, jak na rysunku 8.42 (a). Wyobraźmy sobie, że w zamiarze zwiększenia wydajności systemu dołączono do wspomnianej magistrali kolejnych 12 procesorów, co zilustrowano w części (b) rysunku.



RYSUNEK 8.42. Przykładowe konfiguracje w kontekście skalowalności: (a) 4 procesory połączone wspólną magistralą; (b) 16 procesorów połączonych wspólną magistralą; (c) 4 procesory połączone w kratę; (d) 16 procesorów połączonych w kratę

Jeśli przepustowość magistrali równa była *b*, to zwiększając liczbę procesorów z 4 do 16, powodujemy jednocześnie zredukowanie przypadającego na jeden procesor pasma z $\frac{b}{4}$ do $\frac{b}{16}$. System wielu procesorów połączonych jedną magistralą **nie** jest więc skalowalny.

Przeprowadźmy analogiczne rozważania w odniesieniu do czterech procesorów połączonych w kratę, jak w części (c) rysunku 8.42. Zwiększanie liczby procesorów powoduje jednocześnie dodawanie nowych łączy, dzięki czemu skalowanie systemu nie powoduje spadku zagregowanej przepustowości, jak to miało miejsce w przypadku pojedynczej magistrali. Wręcz przeciwnie: średnia liczba łączy przypadających na jeden procesor zwiększa się z 1 (4 łącza, 4 procesory) do 1,5 (24 łącza, 16 procesorów), a więc dołączenie nowych procesorów powoduje zwiększenie przepustowości zagregowanej.

Przepustowość nie jest jednak jedynym kryterium wydajności, równie istotne jest bowiem opóźnienie, ściśle związane ze średnicą sieci. Dołączanie nowych procesorów do magistrali nie zwiększa tej średnicy i opóźnienie sygnału wysyłanego na „pustą" magistralę jest od liczby procesorów niezależne. Średnica sieci o topologii kraty i rozmiarze $n \times n$ węzłów równa jest $2(n–1)$, a więc opóźnienie (w najgorszym przypadku) zwiększa się w przybliżeniu proporcjonalnie do pierwiastka kwadratowego z liczby węzłów (procesorów). Dla 400 połączonych w kratę procesorów średnica sieci równa jest 38, zaś dla 1600 procesorów równa jest już 78; czterokrotne zwiększenie liczby procesorów powoduje dwukrotne (w przybliżeniu) zwiększenie średniego opóźnienia.

W idealnym przypadku skalowalny system powinien charakteryzować się jednakową wartością przepustowości dla każdego z procesorów i stałą — czyli niezależną od liczby procesorów — wartością średniego opóźnienia. W praktyce udaje się zwykle zapewnić wystarczającą przepustowość, lecz zwykle dzieje się to za cenę zwiększania opóźnienia. Ograniczenie tempa tego wzrostu do logarytmicznego względem przepustowości — jak ma to miejsce w topologii hipersześcianu — jest obecnie najlepszą rzeczą, jaką jesteśmy w stanie zrobić pod tym względem.

Problem zwiększającego się opóźnienia jest szczególnie dotkliwy dla aplikacji „drobnoziarnistych" czyli takich, które operują na małych porcjach danych często odczytywanych z pamięci i zapisywanych w niej. Jeżeli większość odwołań będzie miała charakter zdalny — czyli odnosić się będzie do pamięci innej niż lokalna pamięć odwołującego się procesora — skumulowane opóźnienia spowalniać będą pracę

aplikacji w znaczącym stopniu, tym większym, im bardziej rozległy jest sam system. Problem ten w równym stopniu dotyczy wieloprocesorów i wielokomputerów, bowiem w obydwu tych architekturach pamięć podzielona jest na rozłączne, odległe od siebie moduły.

W związku z tym projektanci systemów starają się minimalizować opóźnienie, a co najmniej maskować jego występowanie. Jedną z technik takiego maskowania jest powielanie (replikacja) danych. Jeśli kopie tych samych danych utrzymywane będą w różnych częściach systemu, dostęp do tych danych będzie o wiele szybszy niż w przypadku, gdyby utrzymywane były w jednym egzemplarzu. Przykładem takiego maskowania jest cache'owanie, w ramach którego kilka kopii oryginalnych danych z pamięci znajduje się w pamięciach podręcznych procesorów, które z danych tych korzystają. Mamy tu do czynienia z pewną asymetrią — dane w pamięci są „oryginalne", zaś ich kopie w pamięciach podręcznych mają jedynie charakter „wtórny"; możliwa jest jednak także inna strategia, w której wszystkie kopie danych są równouprawnione, kluczowym zagadnieniem jest wówczas sprawowanie kontroli nad tym, kto, kiedy i gdzie dokonuje odczytywania i modyfikowania bloków danych. Istnieją rozmaite rozwiązania w tym względzie, od dynamicznego udostępniania danych na żądanie przez sprzęt, do jawnego odczytywania danych, będącego konsekwencją użycia stosownych opcji kompilacji.

Inną techniką maskowania opóźnienia jest **wyprzedzający odczyt danych** (*prefetching*). Jeśli dane odczytywane są z pamięci wcześniej, niż okazują się faktycznie potrzebne, ich odczytywanie odbywa się równocześnie z wykonywaniem programu. Wyprzedzający odczyt może być dokonywany samoczynnie. Przenoszenie danych z pamięci głównej do pamięci podręcznej odbywa się nie pojedynczymi bajtami, lecz w większych porcjach; jest to przykład zrobienia użytku z powszechnego dla typowych programów zjawiska lokalności (przestrzennej) odwołań polegającego na tym, że w pewnym interwale czasowym odwołania do pamięci mają tendencję do skupiania się w małych obszarach przestrzeni adresowej.

Wyprzedzające odczytywanie danych może również odbywać się w sposób kontrolowany. Odpowiednio „inteligentny" kompilator może mianowicie wplatać w generowany kod rozkazy wstępnego pobierania danych, jeżeli uzna, że dane te mogą okazać się potrzebne podczas realizacji następnego fragmentu kodu. Inteligencja kompilatora dotyczyć musi przy tym nie tylko samego języka programowania, lecz także wielu szczegółów technicznych (w tym zależności czasowych) maszyny wynikowej. Spekulatywny z konieczności charakter automatycznego generowania rozkazów pobierania wstępnego niesie ze sobą ryzyko pewnego pogorszenia wydajności: czasochłonne w obsłudze wyjątki braku strony powodowane przez rozkazy, które (w ostatecznym rozrachunku) wykonywane nie będą, stanowią czystą stratę czasu.

Trzecią technikę maskowania opóźnień, jaką jest wielowątkowość na poziomie sprzętu, omawialiśmy już wcześniej w niniejszym rozdziale. Jeżeli przełączanie między wątkami będzie dostatecznie szybkie, a liczba działających wątków odpowiednio duża, fakt zablokowania jednego wątku (w oczekiwaniu na odczytanie danych z pamięci) może być maskowany przez nieskrępowane wykonywanie w tym czasie innych wątków. Procesor, wykonujący na przemian po jednym rozkazie z każdego wątku, jest wówczas w pełni odciążony niezależnie od opóźnień w dostępie do pamięci ze strony poszczególnych wątków.

Czwarta technika maskowania opóźnień opiera się na nieblokujących rozkazach zapisu. Normalnie proces wykonujący rozkaz `WRITE` zawieszany jest w oczekiwaniu na zakończenie jego wykonywania. W wariancie nieblokującym wykonywanie procesu

jest kontynuowane, równolegle z realizacją rozkazu. Jeżeli dopuścimy wykonywanie rozkazów w kolejności innej niż ich naturalna kolejność w programie, to możliwe są także nieblokujące rozkazy odczytu (READ), ich realizacja jest jednak znacznie bardziej skomplikowana.

8.5. OBLICZENIA SIATKOWE[28]

Wiele współczesnych wyzwań na gruncie nauki, inżynierii, przemysłu, ochrony środowiska itp. ma charakter interdyscyplinarny i wymaga współdziałania na ogromną skalę. Skuteczne stawianie czoła tym wyzwaniom wymaga doświadczenia, wiedzy, kwalifikacji, środków, oprogramowania i danych ze strony wielu organizacji znajdujących się w różnych zakątkach naszego globu. Przykładami takich działań mogą być:

(1) planowanie naukowej wyprawy na Marsa;
(2) budowa złożonego produktu w olbrzymiej skali (dużej zapory wodnej lub promu kosmicznego);
(3) międzynarodowa pomoc ofiarom kataklizmów i klęsk żywiołowych[29].

Niektóre z takich inicjatyw mają charakter doraźny, inne realizowane są długoterminowo; tak czy inaczej, wszystkie one wymagają jednak zaangażowania wielu organizacji z ich własnymi zasobami i procedurami, wykorzystywanymi w dążeniu do osiągnięcia wspólnego celu.

Jeszcze nie tak dawno współpraca dwóch organizacji, posiadających różne komputery sterowane różnymi systemami operacyjnymi, implementujące różne bazy danych funkcjonujące w oparciu o różne protokoły była przedsięwzięciem na wskroś trudnym. Nieustannie rosnące potrzeby w zakresie kooperacji różnych organizacji w skali całego świata doprowadziły jednak z czasem do wypracowania systemów i technologii umożliwiających połączenie zróżnicowanych, odległych komputerów w formę **siatki** (*grid*), zwanej także niekiedy **kratą**. Zgodnie z hierarchią przedstawioną na rysunku 8.1, siatka ta stanowi przykład ekstremalnie luźnego połączenia węzłów i może być uważana za gigantyczny, międzynarodowy, heterogeniczny klaster.

Zadaniem wspomnianej siatki jest dostarczenie technicznej infrastruktury umożliwiającej zorganizowanie odrębnych organizacji w formę olbrzymiej **organizacji wirtualnej**, podporządkowanej określonemu celowi. Taka wirtualna organizacja musi mieć elastyczny charakter, radzić sobie z rozmaitymi konsekwencjami przybywania nowych uczestników (i ubywania innych) oraz umożliwiać poszczególnym członkom-organizacjom współpracę w obszarach, które uznają oni za nadające się do tego. Jednocześnie każda organizacja musi zachować kontrolę nad swymi własnymi zasobami w stopniu, w jakim sobie tego życzy. W tym celu projektanci siatek opracowują usługi, narzędzia i protokoły umożliwiające funkcjonowanie takich wirtualnych organizacji.

[28] Często spotyka się także określenie „Obliczenia gridowe" — *przyp. tłum.*

[29] Spośród innych przykładów tego rodzaju skoordynowanych działań w skali całego świata przywołać można poszukiwanie przejawów inteligencji we Wszechświecie (projekt SETI@home — *Search for ExtraTerrestrial Intelligence*, oficjalnie zakończony 15 grudnia 2005) oraz znajdowanie coraz większych liczb pierwszych Mersenne'a (projekt GIMPS — *Great Internet Mersenne Prime Search*) — *przyp. tłum.*

Siatka ma organiczną strukturę wielostronną, bowiem wszyscy członkowie wirtualnej organizacji korzystają z niej na równych prawach. Stanowi to istotną różnicę w stosunku do „dwustronnych" systemów typu „klient-serwer" lub „peer-to-peer"(p2p). W modelu „klient-serwer" — którego przykładem może być sieć WWW — wyróżnić można dwóch uczestników: serwer, świadczący usługi, oraz klienta z tych usług korzystającego. W przypadku aplikacji typu p2p dwóch równoprawnych partnerów wymienia między sobą pliki (lub inną informację); przykładem systemu p2p jest poczta elektroniczna. Skoro siatka jest koncepcyjnie i architektonicznie różna od obydwu tych modeli, jest zrozumiałe, iż wymaga ona zupełnie innych protokołów i technologii.

Siatka umożliwiać musi dostęp do szerokiej gamy zasobów. Każdy z tych zasobów zarządzany jest przez firmę-właściciela decydującą o tym, komu, kiedy i ile poszczególnych zasobów udostępnić. W sensie abstrakcyjnym siatka służy więc do zarządzania zasobami i do ich udostępniania.

Jednym ze sposobów abstrakcyjnego ujęcia siatki jest model warstwowy przedstawiony na rysunku 8.43. Najniższa z warstw — **warstwa konstrukcyjna** (*fabric layer*) jest fizycznie zbiorem komponentów, z których siatka jest zbudowana: procesorów, dysków, sieci i czujników na poziomie sprzętowym oraz aplikacji i danych na poziomie oprogramowania. Komponenty te są zasobami podlegającymi udostępnianiu w kontrolowany sposób.

Warstwa	Funkcja
Aplikacyjna	Współdzielenie zasobów udostępnianych w sposób kontrolowany
Kolektywna	Wykrywanie i monitorowanie zasobów, pośredniczenie w ich udostępnianiu, sterownie grupami zasobów
Zasobowa	Zapewnianie bezpiecznego, kontrolowanego dostępu do poszczególnych zasobów
Konstrukcyjna	Udostępnianie fizycznych komponentów: komputerów, pamięci, sieci, czujników, programów i danych

RYSUNEK 8.43. Warstwowy model siatki obliczeniowej

W warstwie bezpośrednio wyższej, zwanej **warstwą zasobową**, odbywa się zarządzanie indywidualnymi zasobami. W wielu przypadkach dostęp do wykorzystywanych w siatce zasobów organizowany jest za pośrednictwem lokalnych procesów, zapewniających kontrolowane udostępnianie tych zasobów zdalnym użytkownikom. Warstwa zasobowa pełni rolę interfejsu umożliwiającego warstwom wyższym uzyskiwanie informacji o charakterystyce i statusie indywidualnych zasobów, monitorowanie tych zasobów i wykorzystywanie ich w bezpieczny sposób.

Zazwyczaj indywidualne zasoby siatki łączone są w grupy — grupy dysków, grupy dostępnych cykli procesorów, grupy specyficznych danych itp. Zarządzanie tymi grupami dokonuje się **warstwie kolektywnej**, w ramach której użytkownicy mają możliwość wykrywania dostępnych zasobów w określonych grupach, w oparciu o katalogi lub inne bazy danych. W warstwie tej mogą być także realizowane usługi pośrednictwa, polegające na udostępnianiu przez dostawców zasobów o unikalnym, rzadkim charakterze (wśród użytkowników rywalizujących ze sobą o dostęp do tychże zasobów). Warstwa kolektywna odpowiedzialna jest także za powielanie (replikację) danych, wprowadzanie nowych uczestników i zasobów do siatki, ustalanie reguł dostępu użytkowników do poszczególnych grup zasobów oraz ewidencjonowanie wykorzystywania zasobów (głównie na potrzeby rozliczeń).

Najwyższą z warstw — **warstwę aplikacyjną** — tworzą aplikacje użytkowników, korzystające z usług niższych warstw w celu uzyskiwania uwierzytelnionego dostępu do zasobów, formułowania żądań w stosunku do tych zasobów, śledzenia postępu w realizacji tych żądań i informowania użytkowników o ich rezultatach, a także wykonywania procedur zarządzających przypadkami wyjątkowymi (awariami).

Zagadnieniem o kluczowym znaczeniu dla efektywnego wykorzystywania siatek obliczeniowych jest bezpieczeństwo wykorzystywanych zasobów. Właściciele zasobów pragną zazwyczaj zachować ścisłą kontrolę nad tym, kto, kiedy, w jakim zakresie i jak długo uzyskuje dostęp do poszczególnych zasobów. Bez zapewnienia należytego zabezpieczenia żadna organizacja nie zdecyduje się na udostępnienie swych zasobów. Z drugiej jednak strony, gdyby każdy użytkownik siatki zmuszony był posiadać inny login i inne hasło na każdym komputerze, korzystanie z siatki stałoby się dla niego koszmarem. Jest więc zrozumiałe, iż potrzebne są inne rozwiązania w tym zakresie.

Kluczowym elementem wszystkich modeli bezpieczeństwa siatek obliczeniowych jest użycie pojedynczego uwiarygodnienia (*single sign-on*). Dostęp do siatki poprzedzany jest uwierzytelnieniem, koniecznym do identyfikacji tego, w czyim imieniu wykonywane będą zamierzone działania. Identyfikacja ta dziedziczona jest przez procesy potomne, jakie mogą być uruchamiane przez główne obliczenia. Gdy identyfikacja ta prezentowana jest w zdalnym komputerze, podlega ona mapowaniu w jego lokalny mechanizm bezpieczeństwa: systemy platformy uniksowe posługują się w tym celu 16-bitowymi identyfikatorami użytkowników, lecz inne systemy stosować mogą odmienne rozwiązania w tym zakresie. Tak czy inaczej, w każdej siatce istnieć muszą mechanizmy umożliwiające ustanawianie, egzekwowanie i modyfikowanie reguł dostępu do zasobów.

W celu zapewnienia współdziałania między zróżnicowanymi platformami sprzętowymi i programowymi różnych organizacji, konieczne stało się opracowanie określonych standardów, dotyczących zarówno oferowanych usług, jak i protokołów umożliwiających korzystanie z nich. W celu zarządzania procesem standaryzacji powołano grupę roboczą o nazwie Global Grid Forum. Jednym z rezultatów prac tej grupy jest wypracowanie ramowych założeń o nazwie **Open Grid Services Architecture** (otwarta architektura usług siatkowych), w skrócie OGSA, obowiązujących w procesie formułowania nowych standardów. Wszędzie, gdzie to możliwe, zaleca się wykorzystywanie istniejących standardów, na przykład języka WSDL (*Web Services Definition Language* — język definiowania usług webowych) do opisywania usług OGSA. W chwili obecnej założenia ramowe OGSA wyróżniają osiem następujących grup usług, jednak prędzej czy później zestaw ten z pewnością ulegnie poszerzeniu.

(1) Usługi infrastruktury (umożliwienie komunikacji między zasobami).
(2) Usługi zarządzania zasobami (rezerwowanie i przydzielanie zasobów).
(3) Usługi związane z danymi (przenoszenie i replikacja danych).
(4) Usługi kontekstowe (opisywanie żądanych zasobów i ustanawianie polityki ich wykorzystywania).
(5) Usługi informacyjne (informowanie o dostępnych zasobach).
(6) Usługi samozarządzania (zapewnianie założonej jakości usług).
(7) Usługi zabezpieczeń (egzekwowanie polityki bezpieczeństwa).
(8) Usługi zarządzania wykonywaniem zadań (zarządzanie przepływem pracy).

O siatkach obliczeniowych napisano bardzo wiele, znacznie więcej, niż mogliśmy przedstawić na kilku stronach niniejszej książki. Zainteresowanych tą tematyką Czytelników odsyłamy do prac (Berman i in., 2003), (Foster i Keselman, 2003) oraz (Foster i in., 2002).

8.6. PODSUMOWANIE

Przyspieszanie komputerów wyłącznie za pomocą zwiększania częstotliwości taktowania procesorów napotyka na coraz większe problemy związane między innymi z chłodzeniem (oprócz innych, nie mniej poważnych problemów), projektanci postanowili więc poszukiwać sposobów realizacji tego przyspieszenia w zrównolegleniu obliczeń. Zrównoleglenie to może być wprowadzane na różnych poziomach organizacji komputera, od najniższego, gdzie elementy obliczeniowe są ze sobą bardzo ściśle powiązane, do najwyższego, gdzie związek między nimi ma bardzo luźny charakter.

I tak na najniższym poziomie mamy zrównoleglenie na poziomie pojedynczego układu scalonego (*on-chip parallelism*), gdzie poszczególne komponenty tego układu wykonują niezależne działania. Zrównoleglenie to może występować na poziomie rozkazów — wówczas jeden rozkaz lub grupa rozkazów zapoczątkowuje grupę identycznych działań wykonywanych przez wiele odrębnych jednostek funkcjonalnych — lecz może także przyjąć formę (sprzętowej) wielowątkowości, w ramach której poszczególne rozkazy wykonywane są naprzemiennie w ramach różnych wątków, realizując w ten sposób wirtualny wieloprocesor. Trzecia forma zrównoleglenia na poziomie układu scalonego polega na umieszczaniu w tym układzie kilku niezależnych rdzeni, z których każdy wykonuje swe funkcje niezależnie od pozostałych.

Przejawem realizacji zrównoleglenia na poziomie bezpośrednio wyższym są koprocesory, mające zwykle formę kart rozszerzających i odciążające procesor główny od wykonywania skomplikowanych obliczeń pewnych kategorii. Przykładami obecnie stosowanych koprocesorów są procesory sieciowe i procesory multimedialne.

Kolejny poziom zrównoleglenia urzeczywistniony został pod postacią wieloprocesorów. Wieloprocesor to zespół dwóch lub więcej procesorów współdzielących pojedynczą pamięć. Wieloprocesory kategorii UMA charakteryzują się jednakowym czasem dostępu wszystkich procesorów do wszystkich modułów pamięci i wykorzystują w charakterze medium połączeniowego współdzielone magistrale (z podsłuchiwaniem), przełącznice krzyżowe lub wielostopniowe sieci przełączające. Dla odmiany, w wieloprocesorach kategorii NUMA dostęp każdego procesora do jego własnej (lokalnej) pamięci trwa znacznie krócej niż dostęp do pamięci zdalnych, czyli modułów pamięci należących do innych procesorów. Obydwie kategorie wieloprocesorów współdzielą jednak pojedynczą przestrzeń adresową. Odrębną kategorią wieloprocesorów są wieloprocesory COMA, w których nie istnieje przypisanie pamięci do procesorów, a przepływ informacji odbywa się drogą przenoszenia (na żądanie) wierszy pamięci podręcznej między procesorami.

Wielokomputery, w przeciwieństwie do wieloprocesorów, nie posiadają wspólnej pamięci. Każdy procesor dysponuje swą własną pamięcią, a komunikacja między procesorami odbywa się za pośrednictwem komunikatów. Wielokomputery podzielić można na dwie główne kategorie: masywnie zrównoleglone procesory (MPP), będące dużymi superkomputerami wykorzystującymi specjalizowane superszybkie sieci połączeniowe, oraz klastry, oferujące zbliżoną funkcjonalność, jednak za znacznie niższą cenę i na bazie typowych komponentów. Przykładami superkomputerów MPP mogą być BlueGene/L oraz Red Storm, natomiast przykład udanej realizacji klastra stanowi popularna wyszukiwarka Google.

Wielokomputery programowane są najczęściej z wykorzystaniem pakietów komunikacyjnych, których przykładem jest interfejs MPI. Alternatywnym podejściem do programowania wielokomputerów jest stworzenie iluzji współdzielonej pamięci, której wielokomputery te w sensie fizycznym nie posiadają. Przykładami systemów

stwarzających taką iluzję są: oparty na stronicowaniu system DSM, przestrzeń krotek języka Linda oraz obiekty języków Orca i Globe. Różne odmiany systemu DSM realizują pamięć współdzieloną na poziomie sprzętowego stronicowania, co upodabnia je do wieloprocesorów NUMA, od których jednak różnią się znacznie dłuższym czasem dostępu do zdalnych modułów pamięciowych.

Najwyższy stopień zrównoleglenia realizowany jest w postaci siatek obliczeniowych. Siatki te służą do integracji (za pośrednictwem internetu) niezależnych organizacji, nawzajem udostępniających sobie moc obliczeniową, dane i inne zasoby, w dążeniu do realizacji wspólnego celu.

ZADANIA

(1) Rozkazy procesora Pentium mogą mieć długość do 17 bajtów. Czy wobec tego można zaliczyć Pentium do procesorów typu VLIW?
(2) Jakie będą wyniki obcięcia wartości `96`, `-9`, `300` i `256` do przedziału `0-255`?
(3) Czy dozwolone są następujące rozkazy procesora TriMedia? Jeśli nie, to dlaczego?
    (a) Dodawanie całkowite, odejmowanie całkowite, odczyt, dodawanie zmienno-pozycyjne, odczyt natychmiastowy;
    (b) Odejmowanie całkowite, mnożenie całkowite, odczyt natychmiastowy, przesunięcie, przesunięcie;
    (c) Odczyt natychmiastowy, dodawanie zmiennopozycyjne, mnożenie zmienno-pozycyjne, skok, odczyt natychmiastowy.
(4) Na rysunkach 8.5 (d) oraz 8.5 (e) pokazano 12 cykli wykonywania kilku rozkazów. Jak wyglądać będą trzy następne cykle każdego z nich?
(5) W pewnym procesorze kompletne wykonanie rozkazu powodującego chybienie pamięci *cache* na poziomie L1 i trafienie na poziomie L2 zajmuje (średnio) k cykli zegara. Jeśli chybienia na poziomie L1 maskowane są przez wielowątkowość, jak wiele drobnoziarnistych wątków należałoby uruchomić, by uniknąć martwych cykli?
(6) Pewnego ranka królowa pszczół w pewnym ulu zwołała wszystkie pszczoły-robotnice i poleciła im udać się na całodzienne zbieranie nektaru z nagietków. Robotnice wyfrunęły następnie z ula w różnych kierunkach w poszukiwaniu nagietków. Z jakim rodzajem zrównoleglenia mamy tu do czynienia: SIMD czy MIMD?
(7) W trakcie naszej dyskusji na temat modeli spójności stwierdziliśmy, że model spójności jest rodzajem kontraktu między oprogramowaniem a pamięcią. Dlaczego taki kontrakt jest w ogóle potrzebny?
(8) Rozpatrzmy wieloprocesor wykorzystujący wspólną magistralę. Co stanie się, gdy dwa procesory spróbują uzyskać dostęp do globalnej pamięci *dokładnie* w tym samym momencie?
(9) Załóżmy, że — ze względów technicznych — pamięć cache może podsłuchiwać jedynie linie adresowe, lecz nie linie danych. Czy okoliczność ta powoduje konieczność modyfikacji protokołu zapisu jednoczesnego (*write through*)?
(10) Załóżmy, że w pewnym wieloprocesorze opartym na wspólnej magistrali, bez cache'owania, jeden na cztery z wykonywanych rozkazów wiąże się z dostępem do pamięci i że dostęp ten powoduje zajętość magistrali przez cały czas wykonywania

tego rozkazu. Ewentualne odwołania ze strony innych procesorów w tym czasie umieszczane są w kolejce FIFO. O ile szybszy będzie taki system złożony z 64 procesorów w porównaniu z systemem z jednym procesorem?

(11) Protokół MESI posiada cztery stany, podczas gdy inne protokoły spójności cache z zapisem opóźnionym posiadają tylko trzy. Z którego ze stanów protokołu MESI można by wobec tego zrezygnować? Jakie byłyby konsekwencje rezygnacji z poszczególnych stanów? Gdybyś miał wybrać tylko trzy stany protokołu MESI, z którego byś zrezygnował?

(12) Czy w realizacji protokołu MESI może się tak zdarzyć, że pewien wiersz cache obecny jest w lokalnej pamięci podręcznej, lecz do wiersza tego nie było (zdalnych) odwołań ze strony innych procesorów? Jeśli tak, wyjaśnij, w jakich okolicznościach jest to możliwe.

(13) Rozpatrzmy wieloprocesor złożony z *n* procesorów połączonych wspólną magistralą. Prawdopodobieństwo, że którykolwiek procesor będzie próbował uzyskać dostęp do magistrali w konkretnym cyklu, równe jest *p*. Jakie jest prawdopodobieństwo tego, że:

   (a) magistrala jest wolna (brak jest żądań ze strony procesorów);
   (b) występuje dokładnie jedno żądanie;
   (c) występuje kilka żądań.

(14) Jak wiele przełącznic krzyżowych posiada Sun Fire 25K w pełnej konfiguracji?

(15) Załóżmy, że w sieci omega zerwane zostaje łącze między przełącznikami `2A` i `3B`. W następstwie tego wypadku, między którymi jeszcze parami przełączników zerwane zostają połączenia?

(16) Często wykorzystywane obszary pamięci (*hot spots*) są niewątpliwie poważnym problemem w wielostopniowych sieciach przełączających. Czy są one również problemem w przypadku systemów opartych na magistralach?

(17) Sieć omega wykorzystywana jest do połączenia 4086 procesorów RISC o cyklu 60-nanosekundowym z 4096 nieskończenie szybkimi modułami pamięci. Każdy z przełączników wnosi 5-nanosekundowe opóźnienie. Jakie całkowite opóźnienie wiązać się może z wykonywaniem rozkazu `LOAD` w tym systemie?

(18) Rozpatrzmy komputer wykorzystujący sieć omega podobną do przedstawionej na rysunku 8.25. Załóżmy, że program i stos procesora `i` przechowywany jest w module pamięci o numerze `i`. Zaproponuj niewielką zmianę w topologii, której konsekwencją będzie jednak znaczna różnica w wydajności (taką zmodyfikowaną topologię wykorzystano w systemach IBM RP3 oraz BBN Butterfly). Jakie mankamenty ma owa zmodyfikowana topologia w porównaniu z pierwotną?

(19) W pewnym wieloprocesorze NUMA odwołanie do pamięci lokalnej zajmuje 20 nanosekund, a do pamięci zdalnej — 120 nanosekund. Pewien program wykonywany na tym wieloprocesorze dokonuje łącznie *N* odwołań do pamięci, z których 1 procent to odwołania do konkretnej strony *P*. Strona ta jest początkowo strona zdalną, a jej skopiowanie do pamięci lokalnej zajmuje *C* nanosekund. Pod jakimi warunkami korzystne jest skopiowanie tej strony do pamięci lokalnej przy założeniu, że odwołania do niej ze strony innych procesorów będą miały charakter sporadyczny?

(20) Rozpatrzmy wieloprocesor CC-NUMA podobny do tego z rysunku 8.27, z tą różnicą, że posiada on 512 węzłów z 8 megabajtami pamięci w każdym z nich. Jeśli wiersz pamięci cache ma wielkość 64 bajtów, jaki jest procentowy narzut pojemności pamięci ze strony katalogów? Jaki wpływ na wielkość tego narzutu będzie mieć zwiększenie liczby węzłów?

(21) Dla każdej z topologii widocznych na rysunku 8.31 oblicz średnicę sieci.
(22) Dla każdej topologii widocznych na rysunku 8.31 określ stopień odporności na awarie, zdefiniowany jako maksymalna liczba krawędzi, których usunięcie nie powoduje jeszcze rozpadu sieci na dwie sieci niezależne.
(23) Rozważmy topologię podwójnego torusa, widoczną na rysunku 8.31 (f), lecz rozszerzoną do rozmiarów $k \times k$. Jaka jest średnica tej sieci? *Wskazówka*. Rozważ problem z osobna dla parzystego i nieparzystego k.
(24) Sieć połączeniowa ma formę sześcianu o rozmiarach $8 \times 8 \times 8$. Każde łącze jest pełnodupleksowym łączem o przepustowości 1 GB/s. Jaka jest przepustowość przepołowienia dla tej sieci?
(25) Prawo Amdahla wyraża ograniczenie potencjalnego przyspieszenia obliczeń uzyskanego dzięki zrównolegleniu architektury komputera. Wyraź — w funkcji f — maksymalne zrównoleglenie w sytuacji, gdy liczba procesorów zbliżą się do nieskończoności. Jakie są konsekwencje tego ograniczenia w przypadku $f = 0{,}1$?
(26) Analizując rysunek 8.42, można zrozumieć, dlaczego skalowanie jest niewykonalne w przypadku magistrali, lecz całkowicie realne w przypadku kraty. Przy założeniu, że każde łącze (lub magistrala) ma przepustowość b, oblicz średnią przepustowość przypadającą na 1 procesor w każdym z czterech przypadków. Zwiększ następnie liczbę procesorów do 64 i powtórz obliczenia. Jaka jest teoretyczna granica przepustowości, gdy liczba procesorów zdąża do nieskończoności?
(27) W treści rozdziału dyskutowane były trzy warianty realizacji operacji `send`: synchroniczny, blokujący i nieblokujący. Zaproponuj czwarty wariant, podobny do wariantu blokującego, lecz posiadający nieco inne niż on własności. Rozważ zalety i wady Twojego rozwiązania w porównaniu z wariantem blokującym.
(28) Rozważmy wielokomputer wykorzystujący sieć z rozgłaszaniem sprzętowym, na przykład Ethernet. Dlaczego wartość stosunku liczby operacji odczytu (czyli tych niemodyfikujących stanu zmiennych lokalnych) do liczby operacji zapisu (czyli tych modyfikujących ów stan) ma w tych sieciach znaczenie?