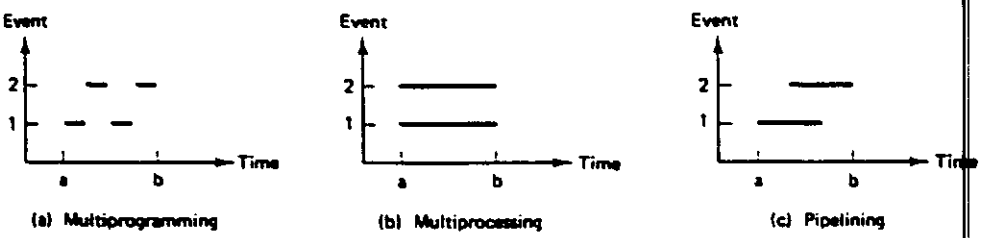
BAB 10

Pemrosesan Paralel

10.1 KEBUTUHAN AKAN PEMROSESAN PARALEL

Banyak pekembangan-perkembangan baru dalam arsitektur komputer yang didasarkan pada konsep pemrosesan paralel. **Pemrosesan paralel** dalam sebuah komputer dapat didefinisikan sebagai pelaksanaan instruksi-instruksi secara bersamaan waktunya. Hal ini dapat menyebabkan pelaksanaan kejadian-kejadian (1) dalam interval waktu yang sama, (2) dalam waktu yang bersamaan atau (3) dalam rentang waktu yang saling tumpang tindih, seperti diperlihatkan pada Gambar 10-1. Dalam bab ini kita belajar tentang implementasi paralelisme yang sesungguhnya dalam komputer.

Komputer yang cepat dan efisien banyak dibutuhkan oleh berbagai bidang dewasa ini. Sebagian menyertakan simulasi numerik bagi field-field yang berkesinambungan. Masalah-masalah semacam itu membutuhkan banyak perhitungan dan dapat ditemukan pada disiplin-disiplin ilmu seperti oceanografi, astrofisik, seismologi, meteorolgi, atomik, nuklir dan fisika plasma. Jumlah "pengolahan angka-angka" baik secara volume data dan jumlah perhitungan, sangat besar.



Gambar 10-1 Kesejajaran eksekusi kejadian-kejadian

Sebagai contoh, salah satu jenis eksplorasi seismik melibatkan proses pembuatan gelombang sonik di dalam tanah. Pantulan-pantulan (echo) gelombang ini ditangkap oleh beratus-ratus geophone di area tersebut. Kemudian data ini digunakan untuk membuat bagian-bagian perpotongan geometris dua dimensi untuk lapisan-lapisan bawah tanah. Dalam contoh ini, 3.000 nilai waktu yang berbeda, masing-masing berasal dari 48 lokasi yang berbeda, disimpan untuk masing-masing gelombang sonik. Perhitungan ini melibatkan 5 sampai 8 juta angka-angka floating-point per mil untuk suatu garis survei.

Akibat adanya keterlibatan matematika dalam simulasi komputer pada field-field yang berkesinambungan, proses perhitungan tersebut dapat kita bagi menjadi beberapa lajur independen yang dapat dikerjakan secara bersama-sama. Dengan demikian, kemampuan komputer bagi pemrosesan paralel dalam simulasi sangat dibutuhkan. Bidang-bidang lain yang diuntungkan oleh penggunaan komputer dengan kemampuan pemrosesan paralel meliputi inteligensia buatan (*artificial intelligence*), rekayasa genetik (*genetic engineering*), analisis elemen terbatas untuk perancangan struktur dan eksperimen terowongan angin (*wind tunnel*) untuk studi aerodinamika serta perancangan sirkuit LSI dan VLSI.

10.2 TEKNIK-TEKNIK PEMROSESAN PARALEL

Komputer-komputer yang telah kita pelajari selama ini secara keseluruhan memiliki suatu arsitektur yang kita gambarkan sebagai proses serial (kecuali pada Bab 9). Telah kita bahas pada Bagian 5.1, suatu komputer serial terdiri atas sebuah unit memori utama untuk penyimpanan data dan instruksi, sebuah CPU untuk menginterpretasikan dan menjalankan instruksi-instruksi tersebut serta piranti-piranti

I/O. Setiap operasi komputer, mulai proses fetch awal pada sebuah instruksi sampai eksekusinya untuk menyimpan atau menuliskan hasil-hasilnya, biasanya dilakukan secara sekuensial, yaitu satu per satu. Sebuah **komputer paralel** mempunyai kemampuan untuk saling tumpang tindih atau menjalankan beberapa operasi ini secara bersamaan waktunya.

Beberapa cara telah dikembangkan untuk memperkenalkan paralelisme ke dalam arsitektur komputer serial. Beberapa di antaranya adalah sebagai berikut:

1. *Pipelining*. Telah kita bahas di Bab 9, pipelining dapat kita anggap sebagai paralelisme seperti pada Gambar 10-1(c).
2. *Unit-unit fungsional berganda* (*multiple*). Daripada memiliki sebuah ALU dalam sebuah CPU untuk mejalankan semua fungsi aritmatika dan logika, kita dapat membangun unit-unit yang terpisah untuk menjalankan fungsi-fungsi yang berbeda. Kita dapat juga memiliki lebih dari satu unit untuk menjalankan suatu fungsi tertentu. Kedua pendekatan itu memungkinkan terjadinya perhitungan paralel.
3. *Tumpang tindih antara operasi-operasi CPU dan I/O*. Pada komputer-komputer terdahulu, CPU bertanggung jawab untuk mengarahkan semua operasi I/O dan untuk menjalankan instruksi-instruksi program di dalam memori. Dengan memiliki sebuah prosesor I/O khusus (*special purpose*) (lihat Bagian 6.5) yang menangani semua operasi I/O memungkinkan pelaksanaan instruksi-instruksi program lainnya dijalankan pada waktu yang sama.
4. *Interleaving memori*. Seperti telah digambarkan pada Bagian 7.3, interleaving pada memori memungkinkan lebih dari satu word yang di-fetch dari memori pada suatu waktu. Dalam hubungannya dengan teknik-teknik paralelisme lainnya, cara ini dapat digunakan untuk menyediakan instruksi dan data dalam kecepatan yang tinggi.
5. *Multiprograming*. Dari titik pandang sistem komputer, multiprograming merupakan jenis paralelisme yang diperlihatkan pada Gambar 10-1(a). Dalam kasus ini, kejadiannya merupakan program-program yang dijalankan secara sekuensial pada interval waktu yang sama.
6. *Multiprosesing*. Daripada kita memiliki sebuah CPU dalam sebuah sistem, kita dapat menggunakan beberapa prosesor yang bekerja bersama-sama pada permasalahan yang sama dan menghasilkan paralelisme yang diperlihatkan pada Gambar 10-1(b) (juga lihat Bagian 6.5). Dalam kasus ini, kejadian-kejadian itu dapat berupa bagian dari suatu program tunggal atau

suatu program yang benar-benar terpisah. Kejadian yang pertama lebih sulit, kecuali jika program-program tersebut ditulis dalam suatu bahasa yang sama.

Arsitektur yang dibahas dalam bab ini menggunakan metode-metode ini, baik secara individual maupun kombinasi, untuk menjalankan paralelisme dalam suatu komputer.

10.3 PERCEPATAN KOMPUTER PARALEL

Bayangkan tentang sebuah komputer paralel dengan n prosesor identik yang bekerja secara berdampingan pada sebuah program tunggal. Pada awalnya, sepertinya kita dapat mengerjakan semua permasalahan dengan n kali lebih cepat daripada menggunakan sebuah prosesor tunggal. Sayangnya hal ini tidak selalu demikian. Karena alasan-alasan tertentu, seperti konflik atas akses memori, konflik atas jalur komunikasi dan algoritma yang tidak efisien untuk implementasi yang sesungguhnya atas kesejajaran masalah-masalah tersebut, percepatan yang dihasilkan jauh lebih kecil daripada n.

Suatu batas bawah, dikenal sebagai **konjektur Minsky**, memperkirakan percepatan yang sesungguhnya yaitu sebesar $\log_2 n$. Sedangkan batas atas tergantung dari apakah keseluruhan program tersebut menyertakan bagian I/O (biasanya berupa kode sekuensial). Jika ya, batas atas umum kita hitung sebagai $n/\ln n$ (Hwang dan Briggs, 1984). Tabel 10.1 memberikan beberapa perkiraan batas atas dan bawah kecepatan untuk sistem yang terdiri atas sejumlah n prosesor.

Akibat adanya kedua batas tersebut, dapat kita lihat bahwa penambahan lebih banyak prosesor tidak selalu menjadi cara terbaik untuk membuat suatu komputer bekerja lebih cepat. Untuk setiap prosesor yang ditambahkan, semakin sedikit percepatan dan semakin kompleks bagi kita untuk mengatur semua prosesor tersebut. Untuk alasan ini, timbul usaha-usaha untuk mengeksploitasi kesejajaran dalam algoritma, yang dengan demikian semakin memperbaiki penggunaan prosesor berkecepatan tinggi dalam jumlah yang sedikit, daripada hanya mengandalkan pada prosesor berkecepatan rendah dalam jumlah yang besar, untuk percepatan program. Selain itu, semakin banyak pula penelitian yang dilakukan untuk memecahkan beberapa masalah konflik yang dihasilkan oleh prosesor berganda dalam suatu sistem, yang juga meningkatkan batas bawah dan batas atas percepatan.

TABEL 10.1 PERKIRAAN BATAS-BATAS ATAS DAN BAWAH UNTUK PERCEPATAN (*SPEEDUP*)

Jumlah prosesor	Batas bawah ($\log_2 n$)	Batas atas ($n/\ln n$)
2	1	2,89
4	2	2,89
8	3	3,85
16	4	5,77
32	5	9,23

10.4 KLASIFIKASI PADA ARSITEKTUR KOMPUTER PARALEL

Paralelisme dalam suatu komputer dapat diaplikasikan pada beberapa tingkatan, seperti berikut:

1. *Tingkat pekerjaan:* antara pekerjaan-pekerjaan atau fase-fase suatu pekerjaan. Hal ini menjadi prinsip dasar dari multiprograming.
2. *Tingkat prosedur:* antara prosedur-prosedur dan di dalam loop. Hal ini harus tercakup sebagai hal yang penting bagi suatu bahasa.
3. *Tingkat instruksi:* antara fase-fase sebuah siklus instruksi, yaitu fetch, decode dan eksekusi suatu instruksi.
4. *Tingkat aritmatika dan bit:* antara bit-bit dalam sirkuit aritmatika. Salah satu contohnya adalah adder paralel.

Paralelisme pada tingkat aritmatika dan bit sudah menjadi standar komputer-komputer dewasa ini. Penekanannya sekarang adalah untuk menerapkan paralelisme pada ketiga tingkatan lainnya. Meskipun beberapa penerapan dilakukan dengan perangkat lunak, pada bab ini kita akan memfokuskan diri pada penerapan perangkat keras.

Telah banyak usaha untuk mengklasifikasikan perancangan arsitektur komputer paralel. Namun tidak ada satupun yang mampu memisahkan semua jenis perancangan menjadi kelompok-kelompok yang berbeda. Skema klasifikasi yang paling umum digunakan adalah taksonomi Flynn. Kita akan membahas skema ini dan dua skema lainnya: dari Shore dan Feng.

Klasifikasi Flynn

Michael J. Flynn (1966) memperkenalkan suatu skema untuk mengklasifikasikan arsitektur suatu komputer dengan melihat bagaimana mesinnya menghubungkan instruksi-instruksinya ke data yang sedang diproses. Ia menyatakan suatu aliran (*stream*) sebagai suatu rangkaian item-item, baik berupa instruksi maupun data, yang dijalankan atau dioperasikan oleh sebuah prosesor. Di bawah ini diberikan klasifikasinya:

1. *SISD*: single instruction stream, single data stream
2. *SIMD*: single instruction stream, multiple data stream
3. *MISD*: multiple instruction stream, single data stream
4. *MIMD*: multiple instruction stream, multiple data stream

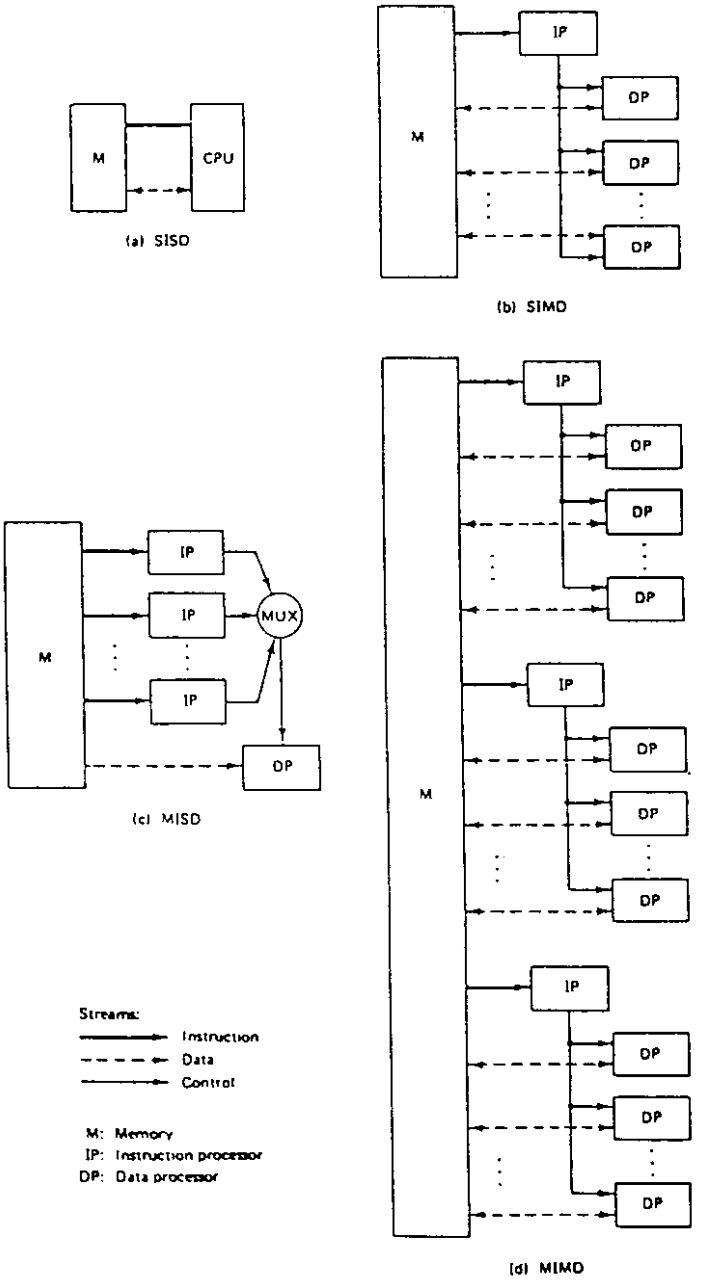
Organisasi ini, diperlihatkan pada Gambar 10-2, akan dibahas lebih terperinci.

SISD. Sebuah komputer SISD [Gambar 10-2(a)] merupakan suatu komputer serial konvensional yang telah dibahas pada Bagian 10.1, dimana instruksi-instruksi dijalankan satu per satu dan sebuah instruksi tunggal berhubungan dengan paling banyak satu operasi data. Kita dapat juga menggukan pipelining untuk mempercepat pemrosesan dan kebanyakan komputer SISD di-pipeline-kan ke beberapa saluran tambahan. Karakteristik SISD yang penting adalah pelaksanaan instruksi secara sekuensial.

SIMD. Dalam sebuah komputer SIMD [Gambar 10-2(b)], suatu instruksi tunggal mungkin mengawali sejumlah besar operasi. Instruksi vektor ini, seperti nama mereka, dilaksanakan satu per satuan waktu namun mampu bekerja pada beberapa aliran data sekaligus. Juga, kita dapat menggunakan pipelining untuk mempercepat pemrosesan. Kelas ini berhubungan dengan prosesor array yang akan dibahas pada Bagian 10-6.

MISD. Kelas MISD melaksanakan beberapa operasi instruksi secara bersamaan pada sebuah item data tunggal. Organisasi secara teoritisnya diperlihatkan pada Gambar 10-2(c), namun tidak ada komputer yang masuk dalam kategori ini.

MIMD. Sebuah komputer MIMD [Gambar 10-2(d)] dicirikan oleh eksekusi lebih dari satu instruksi pada saat yang bersamaan, dimana setiap instruksi beroperasi pada beberapa aliran data. Kelas ini mencakup sistem multiprosesor, mulai dari



Gambar 10-2 Organisasi arsitektur menurut Flynn

komputer mainframe yang dihubungkan ke suatu jaringan (linked) sampai array mikroprosesor yang besar.

Klasifikasi Shore

J.E. Shore (1973) membuat suatu klasifikasi arsitektur komputer yang didasarkan pada organisasi bagian-bagian penyusun suatu komputer dan membedakannya menjadi enam jenis mesin.

Mesin I. Pada komputer ini, satu instruksi dikerjakan pada suatu waktu dan masing-masing beroperasi pada satu word dalam suatu waktu. Unit pengolahan bisa berupa pipeline maupun tidak.

Mesin II. Komputer ini juga menjalankan satu instruksi pada suatu waktu, namun ia beroperasi pada sebuah irisan dari suatu bit dalam suatu waktu, bukannya semua bit dalam suatu word data. Namun, pipelining tidak relevan bagi klasifikasi ini.

Mesin III. Sebuah komputer dalam kelas ini memiliki dua unit pengolahan yang dapat beroperasi pada data, satu word dalam suatu waktu atau suatu *irisan bit* dalam suatu waktu. (Irisan bit ke-*i* dari *n* word merupakan bit ke-*i* dari masing-masing *n* word, jika dianggap sebagai suatu keseluruhan.) Mesin ini juga dikenal sebagai **komputer orthogonal**.

Mesin IV. Komputer jenis ini dicirikan oleh sejumlah elemen pengolahan (unit pengolahan dan unit memori), semua di bawah kendali sebuah unit kendali logika (CLU) tunggal. Komunikasi antara elemen-elemen pengolahan hanya dilakukan melalui unit kendali logika.

Mesin V. Mesin V dihasilkan dengan mengubah Mesin IV sedemikian sehingga elemen-elemen pengolahan dapat berkomunikasi dengan tetangga terdekat mereka. Karena itu, suatu elemen pengolahan dapat mereferensikan data di dalam daerah memorinya sendiri maupun daerah memori pada elemen pengolahan lain di dekatnya.

Mesin VI. Komputer ini, disebut sebagai array logika-dalam-memori, merupakan sebuah mesin dengan logika prosesor yang tersebar dalam memori. Suatu contoh mesin jenis ini adalah prosesor array asosiatif, yang akan dibahas pada Bagian 10.6.

Klasifikasi Feng

Tse-yum Feng (1972) menyarankan pengklasifikasian arsitektur komputer atas tingkatan paralelisme mereka. **Tingkatan paralelisme** (*degree of parallelism*) diwakili oleh pasangan (n, m) dimana n merupakan panjang word dan m adalah panjang irisan bit. Pasangan ini diklasifikasikan menjadi empat kelompok sebagai berikut:

1. Jika $n = 1$ dan $m = 1$ maka tidak terjadi paralelisme. Word dan bit diproses satu per satuan waktu. Hal ini disebut sebagai **word serial/bit serial (WSBS)**.
2. Jika $n > 1$ dan $m = 1$ maka paralelisme itu disebut sebagai **word paralel/bit serial (WPBS)**. Dalam hal ini, semua n irisan bit diproses satu per satuan waktu.
3. Paralelisme **word serial/bit paralel (WSBP)** terjadi jika $n = 1$ dan $m > 1$. Dengan demikian sejumlah n word diproses satu per satuan waktu tetapi sejumlah m bit dari masing-masing word diproses secara paralel.
4. Kategori terakhir disebut sebagai **word paralel/bit paralel (WPBP)** dan merupakan suatu paralelisme dimana $n > 1$ dan $m > 1$. Dalam hal ini, sejumlah nm bit diproses secara bersamaan.

Suatu diagram skematis dari kategori-kategori ini diperlihatkan pada Gambar 10-3. Perbedaan klasifikasi ini dengan dua klasifikasi sebelumnya adalah bahwa dengan mengklasifikasikan komputer dalam salah satu dari keempat kategori ini, tingkatan paralelisme mereka (n, m) juga ditentukan. Istilah yang lebih umum bagi klasifikasi Feng adalah pemorsesan **bit-serial** untuk WSBS, pemrosesan **bit-slice** untuk WPBS, pemrosesan **word-slice** untuk WSBBP dan pemrosesan **paralel penuh** untuk WPBP.

Perbandingan Klasifikasi

Klasifikasi Mesin I Shore berhubungan dengan kelas SISD Flynn dan Mesin II sampai IV dapat dianggap sebagai subdivisi kelas SIMD. Pada klasifikasi Feng, Mesin I merupakan WSBP, Mesin II merupakan WPBS dan Mesin II sampai VI merupakan WPBP.

10.5 PEMROSESAN VEKTOR

Sebuah **vektor** adalah kumpulan sejumlah n elemen yang berurutan. Jumlah elemen tersebut, n, disebut **panjang vektor**. Suatu operasi yang bekerja pada paling sedikit satu operand vektor disebut sebagai **instruksi vektor**.

Contoh 10.1

Anggaplah $X = (2, 5, 3)$ dan $Y = (1, 6, 4)$ merupakan dua buah vektor, masing-masing mempunyai panjang 3. Kita dapat mendefinisikan sebuah operasi vektor ADDV (A, B, C) dimana untuk setiap i, tambahkan elemen a_i dan b_i pada vektor A dan B dan simpan hasil penjumlahannya dalam c_i pada vektor C. Karena itu, pelaksanaan ADDV (X, Y, Z) akan menghasilkan vektor $Z = (3, 11, 7)$. Dalam contoh ini, X, Y dan Z merupakan operand vektor untuk instruksi vektor ADDV.

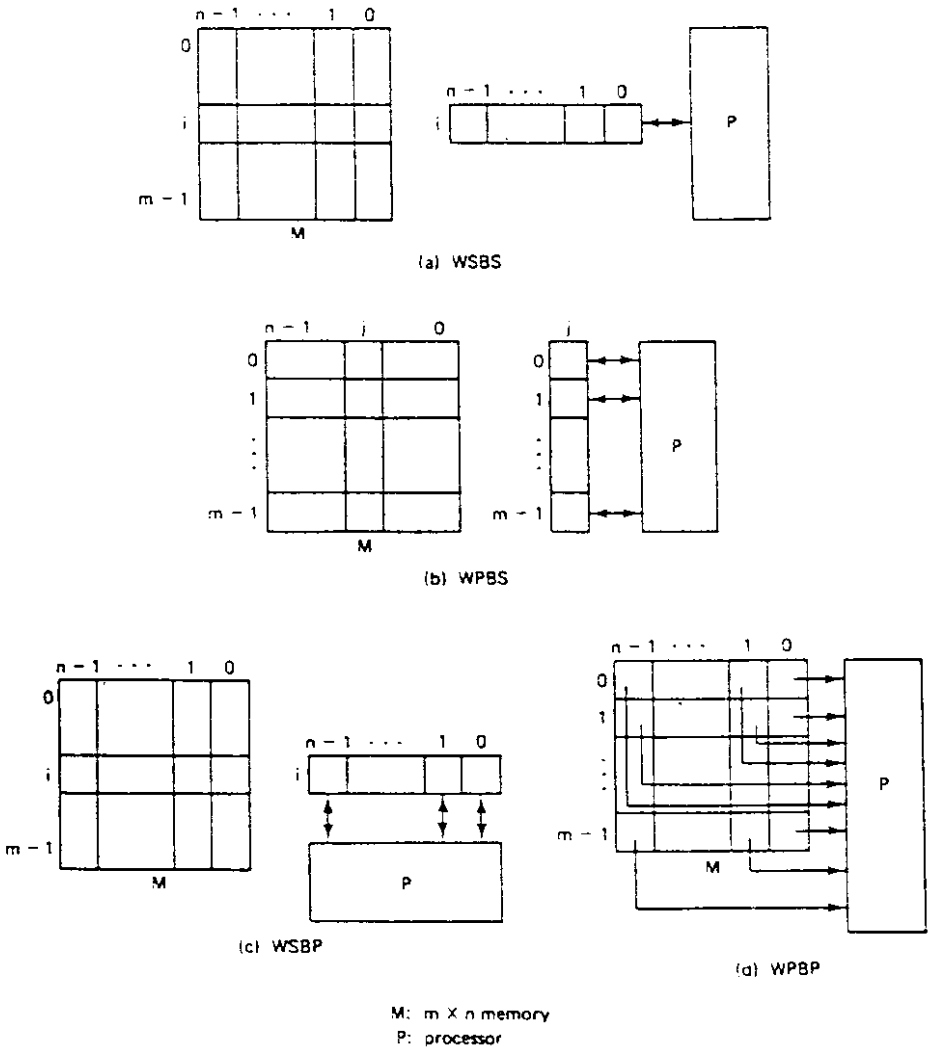
Instruksi vektor dalam sebuah komputer dijalankan oleh suatu prosesor vektor. Instruksi tersebut bisa segala jenis: sebagai contoh, operasi vektor integer, seperti pada Contoh 10.1; operasi vektor Boolean; dan operasi vektor karakter. Salah satu keuntungan utama pemrosesan vektor adalah bahwa tidak dibutuhkannya mekanisme pengendalian loop. Mereka dibangun ke dalam perangkat keras tersebut dan dengan demikian mengurangi overhead yang berhubungan dengan pemrograman sebuah loop dalam suatu komputer skalar konvensional.

Karakteristik Instruksi Vektor

Semua instruksi komputer harus memerinci informasi dasar tertentu untuk eksekusi. Informasi ini harus mencakup hal-hal di bawah ini, baik secara eksplisit maupun implisit:

1. Operasi yang akan dijalankan
2. Operand yang akan digunakan
3. Status yang akan direkam
4. Instruksi berikutnya yang akan dijalankan

Perbedaan utama antara sebuah instruksi skalar dan sebuah instruksi vektor adalah cara memerinci operand-operand tersebut. Kebanyakan operand vektor disimpan di dalam memori dan hampir selalu dialamati secara langsung. Sebagai tambahan bagi pemerincian address operand, sebuah instruksi vektor juga harus memerinci informasi seperti berikut:



Gambar 10-3 Representasi organisasi Feng

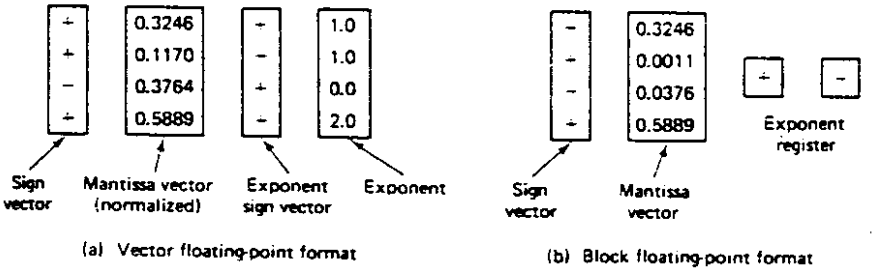
1. Dimensi vektor tersebut (sebuah matriks dianggap sebagai suatu vektor 2-dimensi dalam konteks ini.)
2. Panjang masing-masing dimensi vektor tersebut

3. Jenis data pada masing-masing dimensi vektor tersebut
4. Penyusunan elemen-elemen vektor tersebut di dalam memori (elemen-elemen tersebut tidak harus berkedudukan di dalam lokasi memori yang berdekatan, meskipun hal itu biasanya merupakan default).

Dua buah konsep baru yang unik pada pemrosesan vektor adalah format data **floating-point blok** dan **padding**. Kedua konsep ini tidak dapat diterapkan pada prosesor skalar dan jika diikutsertakan sangat mempengaruhi arsitektur sebuah prosesor vektor. Perhatikan kembali bahwa suatu angka floating-point mempunyai empat bagian: tanda magnitude, mantissa, tanda eksponen dan eksponen (lihat Bagian 2.4). Untuk menggambarkan suatu vektor floating-point, informasi yang sama harus disertakan untuk *masing-masing* elemen vektor. Hal ini biasanya dikerjakan dengan empat vektor, satu untuk setiap bagian elemen floating-point. Hal ini memungkinkan setiap elemen vektor dinormalisir secara individual oleh eksponen-nya sendiri yang memerinci posisi titik radix. Bagaimanapun juga, jika sebuah eksponen digunakan untuk seluruh vektor, dapat dihemat sejumlah besar memori. Ini prinsip dasar floating-point blok yang dibahas oleh Higbie (1976). Untuk dapat menggunakan hanya satu eksponen, kita perlu menyusun skala mantissa secara tepat. Namun jika hal ini mengakibatkan kerugian data yang signifikan untuk segala elemen tersebut, floating-point blok tidak boleh dipakai. Gambar 10-4 memperlihatkan sebuah contoh vektor dalam floating-point dan format floating-point blok. Dapat kita lihat, posisi titik desimal harus dirinci secara eksplisit untuk elemen mantissa sewaktu menggunakan format floating-point blok.

Padding merujuk pada proses pemanjangan sebuah vektor secara otomatis. Hal ini biasanya dilakukan dengan salah satu cara dari dua kemungkinan di bawah ini, untuk dua tujuan yang berbeda:

1. Ketika sebuah mesin dibangun untuk memproses vektor-vektor, biasanya ia dapat beroperasi dengan dua vektor lebih cepat daripada menggunakan satu vektor dan satu skalar. Untuk alasan tersebut, sewaktu dibutuhkan sebuah operasi seperti "kalikan sebuah vektor dengan suatu konstanta", konstanta tersebut disalin ulang untuk membuat sebuah vektor sesuai dengan panjang yang dibutuhkan, dimana setiap elemen sama dengan konstanta itu. Proses padding pada vektor konstanta ini dilakukan oleh unit kendali memori atau sebagai bagian dari suatu pipeline, yang dengan demikian hanya membutuhkan satu salinan konstanta yang benar-benar akan disimpan ke dalam memori.



Gambar 10-4 Dua format vektor floating-point

2. Alasan lain untuk padding adalah untuk memungkinkan terjadinya sebuah operasi pada vektor-vektor yang berbeda panjangnya. Dalam hal ini, vektor yang lebih pendek di-padding-kan dengan beberapa konstanta identitas untuk operasi tersebut. Jika operasi itu merupakan penambahan maka vektor yang lebih pendek di-padding-kan dengan nol; jika operasinya merupakan perkalian maka vektor yang lebih pendek di-padding-kan dengan satu.

Contoh Arsitektur Perangkat Keras

Banyak arsitektur komputer yang mampu menjalankan pemrosesan vektor. Namun, secara umum prosesor vektor memiliki salah satu dari dua pendekatan perancangan yang berbeda: pemrosesan vektor paralel dan pemrosesan vektor pipeline. Dalam suatu prosesor **vektor paralel**, untuk setiap operasi digunakan sejumlah **elemen perhitungan** (*CE* atau *computational element*). Semua CE tersebut berada di bawah kendali satu unit prosesor vektor, seperti diperlihatkan pada Gambar 10-5(a). Sistem prosesor array dan sistem multiprosesor, yang akan dibahas nanti, merupakan prosesor vektor paralel.

Daripada memiliki beberapa CE yang terpisah untuk elemen operand vektor yang terpisah-pisah, suatu prosesor **vektor pipeline** hanya memiliki sedikit CE namun masing-masing dari mereka di-pipeline-kan. Karena instruksi vektor perlu menjalankan operasi yang sama berkali-kali, mereka akan cocok untuk struktur pipeline yang dibahas di Bab 9. Gambar 10-5(b) memperlihatkan diagram blok untuk prosesor vektor pipeline.

Perhatian terhadap Pengalamatan Vektor

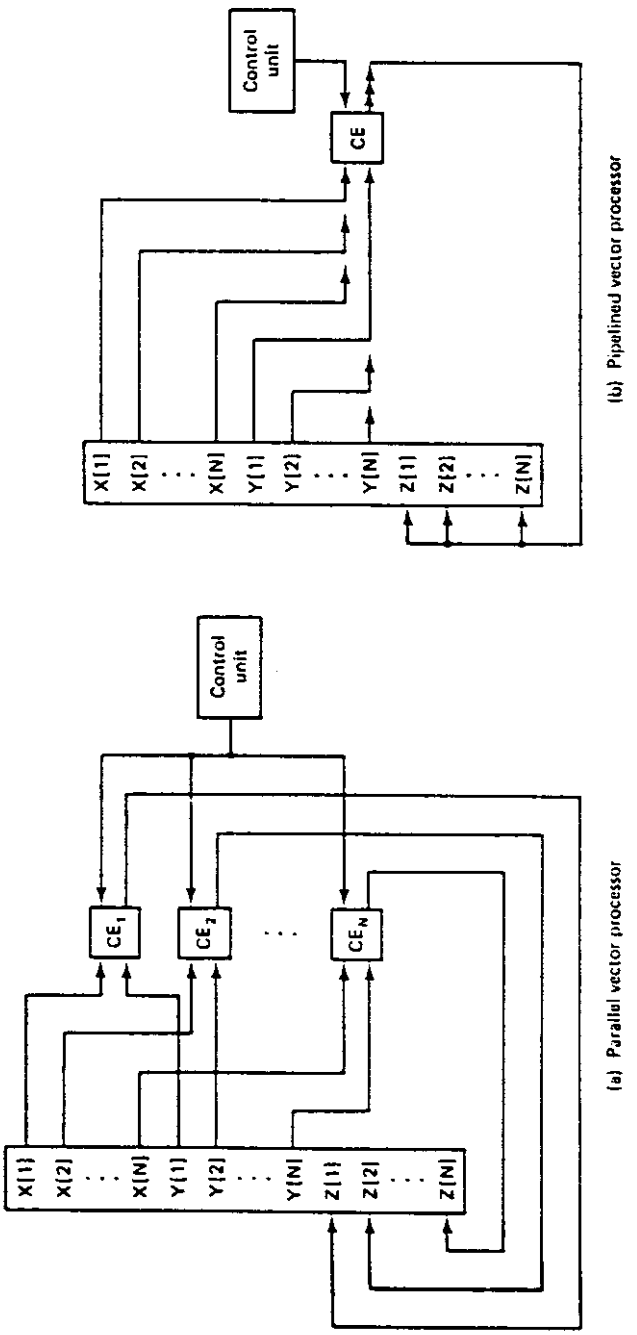
Meskipun hanya terdapat beberapa modus pengalamatan untuk komputer serial (lihat Bab 5), ada bermacam-macam kemungkinan modus pengalamatan dalam sebuah prosesor vektor. Namun, kecepatan pengalamatan sebuah operand dari memori menjadi lebih kritis terhadap kecepatan eksekusi pada suatu prosesor vektor dibandingkan dengan pada sebuah komputer serial. Karena setiap elemen pada operand vektor merupakan sebuah word yang terpisah, dibutuhkan sejumlah n pembacaan (proses read) untuk mendapatkan kembali vektor n-elemen dari sebuah RAM konvensional. Waktu yang dipergunakan di sini dapat dengan mudah merusak penghematan yang kita lakukan sehubungan dengan pemrosesan vektor secara paralel atau pipeline. Karena itu mari kita perhatikan contoh implementasi pengalamatan pada prosesor vektor, yang dapat dibagi menjadi dua kelas: pengalamatan rapat-rapat (teratur) dan pengalamatan jarang-jarang.

Pengalamatan rapat-rapat (teratur). Salah satu karakteristik umum pada pengalamatan **rapat-rapat** (*dense*) atau **teratur** (*regular*) adalah bahwa pola di dalam memori kurang lebih telah kita ketahui sebelumnya. Secara dasarnya, elemen-elemen tetangga pada suatu vektor disimpan pada suatu lokasi sedemikian sehingga address-address tersebut saling rapat/berdekatan atau terstruktur dalam cara pengurutan lainnya. Karakteristik umum lainnya pada pengalamatan jenis ini adalah bahwa semua elemen vektor tersebut pada dasarnya tersimpan bersama-sama sebagai sebuah kelompok. Hal ini akan menjadi lebih jelas sambil kita bahas. Tiga pola paling umum pada jenis ini adalah:

1. Sekuensial
2. Tidak sekuensial tapi teratur
3. Submatriks

Kita akan membahas ketiga pola ini dalam bentuk matriks 4 x 3 M, yang tersimpan di dalam memori seperti diperlihatkan pada Gambar 10-6.

Pada pengalamatan **sekuensial**, elemen-elemen tetangga pada vektor tersebut disimpan dalam lokasi memori yang rapat/berdekatan. Mengacu pada matriks M pada Gambar 10-6(a), perhatikan vektor pada kolom kedua $X = (M_{12}, M_{22}, M_{32}, M_{42})$. Dengan melihat lokasinya di dalam memori [Gambar 10-6(b)], kita lihat bahwa elemen $X[i + 1]$ disimpan di sebelah kanan tepat setelah elemen $X[i]$ (untuk $1 <= i < 4$). Dengan demikian perbedaan address mereka adalah 1.

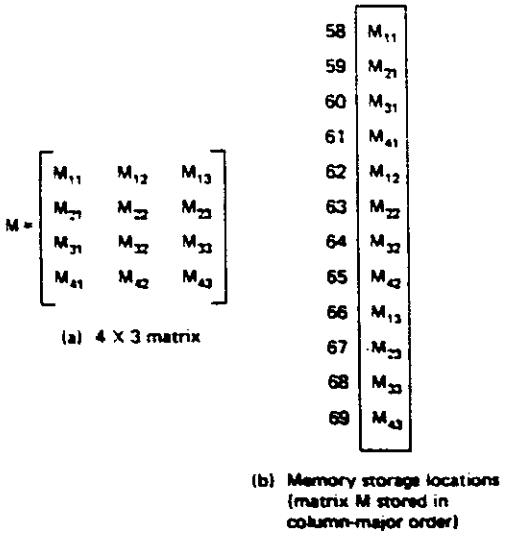


Gambar 10-5 Arsitektur prosesor vektor

Jika kita memperhatikan sebuah baris pada matriks *M*, bagaimanapun juga kita menemukan bahwa baris itu tidak tersimpan pada blok memori yang berdekatan. Anggaplah *Y* adalah vektor pada baris kedua, $Y = (M_{21}, M_{22}, M_{23})$. Dalam hal ini, untuk suatu matriks $m \times n$, elemen $Y[i]$ disimpan pada m lokasi jauhnya dari $Y[i + 1]$. Dengan demikian, meskipun pola address untuk vektor *Y* tidak sekuensial, namun ia teratur (**regular**) dan merupakan jenis pola ke-2.

Jenis yang ketiga, **submatriks**, benar-benar merupakan sebuah kombinasi dari kedua pola lainnya. Perhatikan vektor $Z = (M_{32}, M_{42}, M_{33}, M_{43})$, yang merupakan submatriks dari *M*. Bagian dari vektor ini disimpan secara sekuensial dan bagian lainnya disimpan dalam pola nonsekuensial namun teratur. Bagaimanapun juga, vektor tersebut masih disimpan dalam suatu pola yang tetap dan keseluruhan vektor tersebut ada dalam kelompok word yang menyimpan matriks *M*. Karena itu, ia masih dianggap sebagai salah satu jenis pengalamatan rapat-rapat (teratur).

Pada masing-masing ketiga kasus ini, waktu yang dibutuhkan untuk mengakses suatu operand vektor secara pokok tergantung pada arsitektur dan rancangan memori, bukan pada perhitungan address. Dalam contohnya, digunakan beberapa kombinasi perancangan memori berkecepatan tinggi pada Bab 7 dan menyertakan *memori interleave* dan *memori asosiatif*.



Gambar 10-6 Konfigurasi penyimpanan matriks *M*

Pengalamatan jarang-jarang. Modus pengalamatan ini merujuk pada kasus dimana pola alamat operand vektor harus dihitung secara dinamis. Meskipun tidak sering digunakan seperti pengalamatan jarang-jarang (teratur), ada kasus-kasus tertentu dimana pengalamatan ini lebih cocok dan lebih efisien digunakan. Dua metode umum kelas ini adalah vektor bit dan vektor indeks.

Seperti namanya, suatu **vektor bit** adalah sebuah vektor dimana setiap element terdiri atas sebuah bit tunggal, bisa 0 atau 1. Hal ini berarti bahwa suatu word dapat menyimpan banyak elemen vektor daripada menggunakan word-word yang terpisah untuk setiap elemen. Karena itu pengambilan kembali vektor semacam itu dari memori sesuai urutan magnitudenya lebih cepat dari vektor-vektor yang lebih umum. Vektor bit dapat digunakan dalam pengalamatan dengan menyimpan track elemen dari beberapa vektor lain yang akan membentuk operand vektor tertentu dalam tanda tanya.

Contoh 10.2

Kembali pada Gambar 10-6, anggap bahwa operand vektor yang akan digunakan dalam suatu operasi adalah $W = (M_{11}, M_{31}, M_{32}, M_{43})$. Sesuai urutan penyimpanan matriks dalam memori, vektor bit $B = (101000100001)$ menggambarkan vektor W.

Dalam kasus seperti Contoh 10.2, tidak ada pola alami yang dapat digunakan untuk mengalamati elemen-elemen vektor elemen. Namun, vektor bit alamat dapat digunakan untuk menurunkan perhitungan alamat yang diperlukan.

Metode kedua, menggunakan sebuah **vektor indeks** yang secara prinsip mirip dengan pemakaian vektor bit. Daripada menyimpan bit-bit yang merujuk ke vektor yang ada lainnya, di sini yang disimpan adalah alamat-alamat elemen operand vektor. Address yang tersimpang bisa berupa alamat langsung atau offset dari alamat dasar tertentu.

Contoh 10.3

Kita gunakan lagi matriks M pada Gambar 10-6 dan vektor W pada Contoh 10.2, suatu vektor indeks I dapat digunakan untuk menggambarkan W, dimana $I = (58, 60, 64, 69)$

Meskipun sebuah vektor indeks membutuhkan ruang di dalam memori yang lebih banyak daripada sebuah vektor bit dan dengan demikian membutuhkan waktu yang lebih lama untuk mengambilnya kembali dari dalam memori, dalam beberapa kasus hal ini merupakan satu-satunya pilihan. Ia mungkin satu-satunya cara yang efisien untuk mengimplementasikan tabel *lookup* atau fungsi interpolasi pada sebuah komputer vektor. Untuk penyimpanan vektor indeks, ia dapat disim-

pan sedemikian seolah-olah akan dialamati di dalam salah satu pola rapat-rapat (teratur). Hal ini akan meningkatkan kecepatan waktu akses memori.

10.6 PROSESOR ARRAY

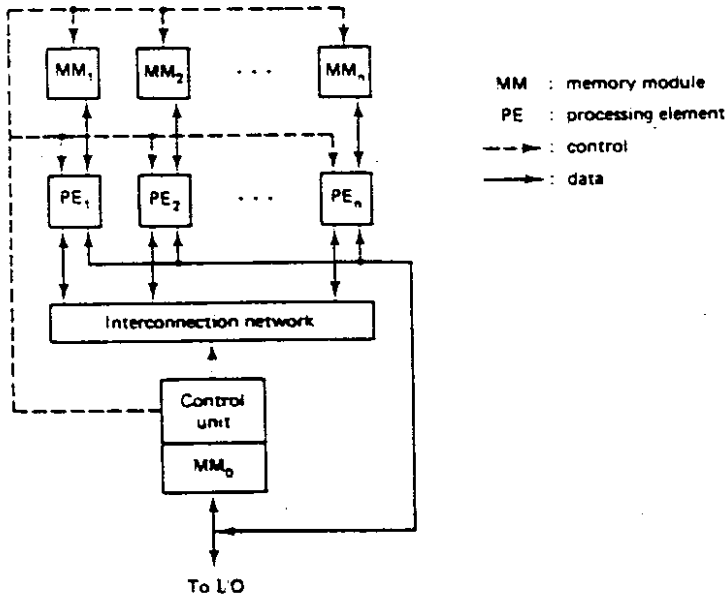
Sebuah **prosesor array** merupakan sebuah komputer synchronous dengan lebih dari satu buah elemen pemrosesan yang beroperasi secara paralel. **Elemen pemrosesan** (*PE* atau *processing element*), masing-masing terdiri atas sebuah ALU dan register-register, semuanya di bawah kendali sebuah unit kendali logika tunggal (CLU). ALU di dalam PE mungkin di-pipeline-kan, mungkin juga tidak. Karena PE beroperasi secara paralel, prosesor array mampu melakukan pemrosesan vektor. Sebuah prosesor array memiliki dua karakteristik yang berbeda:

1. PE-PE tersebut dirancang sebagai piranti yang pasif tanpa kemampuan decoding atau kendali atas instruksi.
2. Semua PE tersebut menjalankan fungsi yang sama pada waktu yang sama pula, di bawah kendali sebuah CLU tunggal.

Organisasi pada Prosesor Array

Setiap PE dalam suatu prosesor array dapat dihubungkan untuk berkomunikasi dengan suatu bagian memori tertentu, ataupun dapat dihubungkan secara dinamis ke bagian memori manapun yang diatur oleh CLU. Pada dasarnya, kedua organisasi prosesor array tersebut saling dibedakan oleh kepemilikan PE-nya. Dalam kedua kasus tersebut, PE mencakup sebuah ALU dan semua register yang diperlukan untuk berkomunikasi dengan komponen lainnya di dalam komputer. Sekarang mari kita lihat kedua organisasi tersebut secara terperinci.

Elemen pemrosesan dengan memori lokal. Konfigurasi yang dilukiskan pada Gambar 10-7 mewakili sebuah prosesor array dengan sejumlah n PE, masing-masing dengan memori lokalnya sendiri-sendiri. Kita perhatikan bahwa unit memori terbagi menjadi sejumlah $n + 1$ modul yang berbeda, satu modul dihubungkan ke masing-masing PE dan satu modul dihubungkan ke CLU. Prosesor array ini sering disebut sebagai suatu komputer SIMD. Yaitu, komputer yang menangani instruksi tunggal satu per satuan waktu, beroperasi pada aliran data berganda.



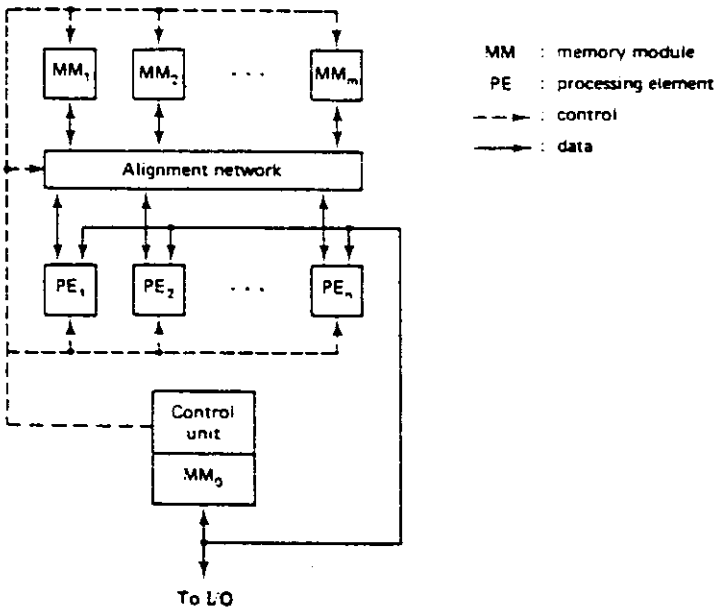
Gambar 10-7 Organisasi prosesor array dengan memori lokal

Bagian memori pada CLU berisi program sistem dan pemakai. CLU mem-fetch instruksi dari memori ini, men-decode-nya dan kemudian menentukan dengan cara bagaimana PE harus dijalankan sesuai dengan lokasi operandnya. Meskipun semua PE dapat beroperasi sekaligus, CLU dapat membuat beberapa di antaranya menjadi tidak mungkin jika mereka tidak diperlukan dalam instruksi saat ini. Dalam prosesor array jenis ini, setiap PE dipasang ke sebuah modul memori yang terpisah. Data diteruskan melalui mereka hanya melalui **jaringan antarhubungan** (*interconnection network*) dan hanya jika diperintah oleh CLU. Kita akan membahas jaringan penghubung untuk prosesor array jenis ini di bagian berikut ini.

Elemen pemrosesan dengan memori paralel. Suatu cara lain untuk merancang sebuah prosesor array adalah dengan mengorganisir PE dan memori sedemikian sehingga PE-PE yang berbeda dapat berkomunikasi dengan modul memori yang berbeda-beda (*MM* atau *memory module*). Untuk melayani pengaturan ini, dibutuhkan suatu komponen **jaringan penjajaran** (***alignment network***) untuk menghubungka PE dan MM. Gambar 10-8 melukiskan organisasi jenis ini.

Semua PE dan MM masih berada di bawah kendali CLU dan CLU tetap memiliki memorinya sendiri. Jaringan penjajaran, juga di bawah kendali CLU, menyediakan hubungan antara PE dan MM. Karena hubungan yang dinamis ini, kita dapat mempunyai sejumlah MM dan PE yang berbeda dalam suatu sistem. Sebagai contoh, dua buah PE dapat dihubungkan ke MM yang sama atau sebaliknya. Organisasi yang diperlihatkan oleh Gambar 10-8 memberikan beberapa karakteristik sistem multiprosesing dan karenanya kita menunda pembahasan mengenai jaringan penjajaran ini pada Bagian 10.7.

Contoh prosesor array. Mungkin tiga prosesor array yang paling terkenal adalah ILLIAC IV, BSP (Burroughs Scientific Processor) dan MPP (Massively Parallel Processor). Sistem ILLIAC IV dibangun di Universitas Illinois pada tahun 1960-an dan kemudian dibangun oleh Burroughs Corporation pada tahun 1972. Sistem ini memiliki 64 buah PE dengan unit memori lokal (seperti pada Gambar 10-7) yang dihubungkan melalui suatu jaringan penghubung *berlubang*



Gambar 10-8 Organisasi prosesor array dengan memori paralel

(*mesh interconnection network*). (Jaringan jenis ini akan dibahas pada bagian selanjutnya)

Sistem BSP juga dibangun oleh Burroughs Corporation sebagai sebuah sistem komersial yang meningkatkan beberapa aspek generasi ILLIAC IV. Produksinya tertunda pada tahun 1979, sistem ini memiliki 16 buah PE dan 17 modul memori yang dihubungkan bersama-sama (seperti pada Gambar 10-8) melalui sebuah jaringan penjajaran **crossbar**. Tidak seperti ILLIAC IV, sistem BSP diperluas menjadi sebuah mesin pemvektor FORTRAN.

Sistem MPP, yang terbaru di antara ketiganya, dibangun pada awal tahun 1980-an. Sistem ini dibangun pada Goddard Space Flight Center milik NASA yang tugas pokoknya memproses citra (image) satelit. Seperti ILLIAC IV, sistem ini memiliki PE-PE dengan memori lokal yang terhubung melalui sebuah jaringan antarhubungan berlubang (*mesh interconnection network*). Tidak seperti ILLIAC IV, sistem ini memiliki 16.384 buah PE! Ia merupakan sebuah prosesor array irisan-bit yang dibangun dengan teknologi VLSI.

Jaringan Penghubung SIMD

Jaringan penghubung suatu prosesor array SIMD pada dasarnya adalah sebuah **jaringan routing data** (*data routing network*) untuk register-register tertentu antar PE-PE. Untuk lebih sederhana, perhatikan sebuah register R_j, masing-masing dalam PE_i, yang terhubung ke jaringan routing data ini. Fungsi jaringan penghubung adalah mengambil data dari register *j* dan meneruskannya ke register *k* lainnya. Tidak perlu semua *n* register dari *n* PE saling dihubungkan dan, dalam kenyataannya, biasanya semuanya tidak dihubungkan. Jika perlu meneruskan data antara dua PE yang tidak saling dihubungkan melalui jaringan, data tersebut dapat diteruskan melalui PE perantara yang saling dihubungkan. Mari kita lihat beberapa jaringan tertentu dalam konteks ini.

Jaringan statis dan dinamis. Kita dapat membagi jaringan routing data menjadi dua kelas: statis dan dinamis. Jaringan statis ditandai oleh hubungan yang tetap (*fixed connection*) sedangkan jaringan dinamis mempunyai hubungan variabel yang dihubungkan pada saat diperlukan.

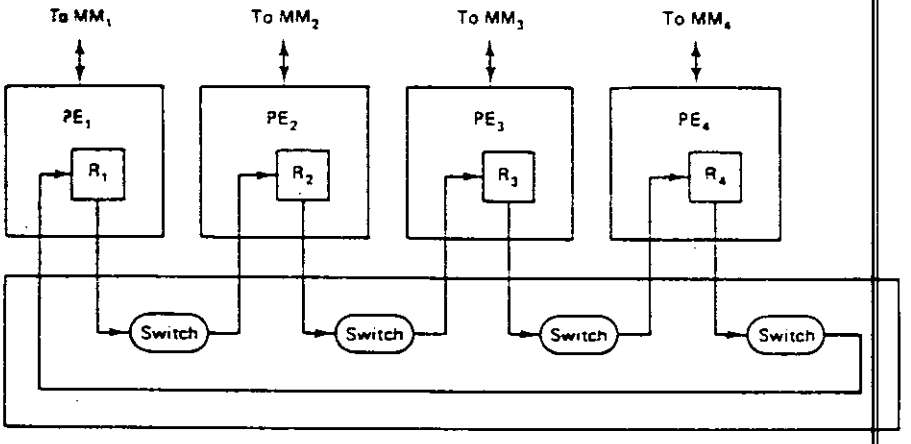
Sebagai contoh jaringan statis, perhatikan **jaringan lingkaran** (*ring network*) sederhana yang diperlihatkan pada Gambar 10-9. Untuk contoh pertama ini, kita juga menyertakan rincian PE. Register R_1 dihubungkan melalui sebuah saklar ke register R_2, register R_2 dihubungkan ke register R_3 dan kemudian ke R_1.

Lingkaran dan contoh jaringan statis lainnya yang dapat digunakan sebagai jaringan penghubung diperlihatkan pada Gambar 10-10. Untuk lebih sederhana, Gambar 10-10 hanya memperlihatkan saklar dalam jaringan penghubung.

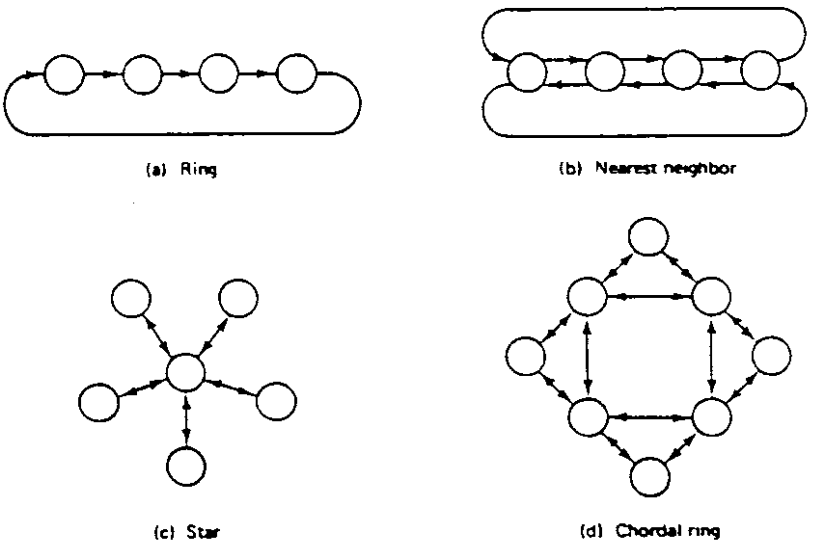
Jaringan dinamis lebih lanjut dapat diklasifikasikan sebagai stage-tunggal atau multistage. Suatu jaringan dinamis stage-tunggal pada dasarnya merupakan kotak saklar N x N (sebuah matriks saklar). Gambar 10-11(a) memperlihatkan diagram skematik dari sebuah kotak saklar 2 x 2 dengan empat kemungkinan stage hubungan [Gambar 10-11(b)]. Implementasi perangkat keras sebuah kotak saklar N x N memerlukan sejumlah N demultiplexer dan N multiplexer, seperti diperlihatkan pada Gambar 10-11(c). Hubungan yang dibutuhkan dibuat dengan mengendalikan pemilihan baris demultiplexer ke multiplexer. Jika saklar tersebut dihubungkan *sepenuhnya* (yaitu, jika semua input dapat dihubungkan ke semua output), seperti diperlihatkan pada Gambar 10-11(c), saklar semacam itu disebut sebagai suatu **saklar crossbar**. (Saklar crossbar akan dibahas secara lebih terperinci pada Bagian 10.7). Jika sebuah saklar tidak dihubungkan sepenuhnya, kita perlu memutar item-item data melalui saklar tersebut sampai mereka diteruskan ke output yang tepat.

Contoh 10.4

Perhatikan saklar 4 x 4 pada Gambar 10-12. Jika kita ingin menghubungkan input 4 ke output 2, pertama-tama input 4 harus diteruskan ke output 1, diputar kembali ke input 1 dan kemudian diteruskan ke output 2.



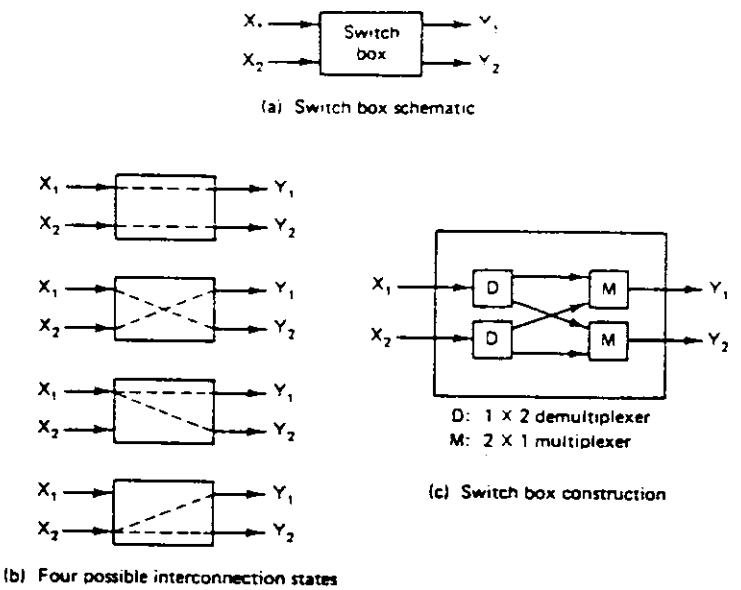
Gambar 10-9 Jaringan penghubung struktur ring



Gambar 10-10 Topologi jaringan statis

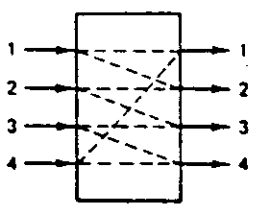
Kebanyakan jaringan penghubung stage dalam prosesor array merupakan jenis resirkulasi stage-tunggal.

Suatu jaringan dinamis **multistage** terbentuk dari sejumlah kotak saklar dan dapat menghubungkan semua input ke output manapun tanpa menjadi terhubung penuh. Kotak-kotak saklar itu dihubungkan dalam sebuah topologi jaringan yang dapat diatur kembali untuk membentuk semua kombinasi yang mungkin. Sebuah contoh jaringan semacam itu digambarkan secara jelas dalam **jaringan Benes**. Gambar 10-13 memperlihatkan sebuah jaringan Benes 8 x 8 yang terbentuk dari 20 buah kotak saklar 2 x 2. Dalam jaringan ini, masing-masing dari kedelapan input tersebut dapat dihubungkan salah satu output manapun pada suatu waktu tertentu. Sebagai contoh, untuk menghubungkan input 1 ke output 7, jalur *a-g-l-p-t* dapat diikuti melalui saklar-saklar tersebut dan untuk menghubungkan input 1 ke output 3 kita mengikuti jalur *a-e-i-m-r*. Hal yang penting di sini adalah terdapatnya lebih dari satu kemungkinan jalur untuk suatu hubungan tertentu. Jadi, misalnya, jika hubungan input 8 ke output 6 saat ini menggunakan jalur *d-h-l-p-s*, hubungan input 1 ke output 7 dapat mengunakan jalur *a-e-i-n-t*, bukannya jalur *a-g-l-p-t*. Dengan cara ini, kedua hubungan tersebut dapat terjadi dalam

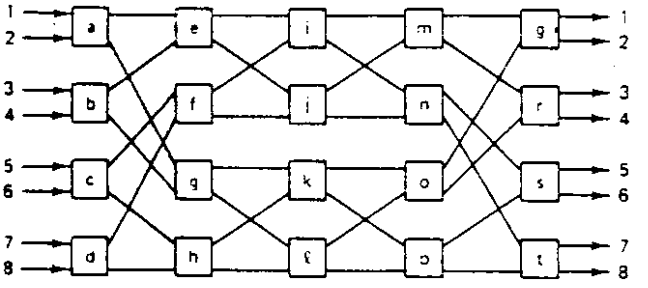


Gambar 10-11 Kotak saklar 2 x 2

waktu yang bersamaan. Selama tidak ada dua input yang perlu diteruskan ke output yang sama pada waktu yang sama pula, semua hubungan yang diminta dapat dilaksanakan secara bersamaan waktunya dengan menggunakan suatu jaringan dinamis multistage. Dengan demikian, jalur *a-g-l* dan *b-g-l* tidak dapat dikerjakan secara bersamaan waktunya, namun jalur *a-g-l* dan *b-g-k* dapat dikerjakan bersamaan.



Gambar 10-12 Kotak saklar 4 x 4



Gambar 10-13 Jaringan Benes 8 x 8

Prosesor Array Asosiatif

Pada semua organisasi paralel yang telah dibahas sejauh ini, kita secara implisit mengasumsikan bahwa unit memori merupakan RAM. Ini berarti bahwa setiap word harus diakses dengan memerinci alamat word tersebut dan, yang lebih penting, hanya satu word dalam sebuah modul memori yang dapat diakses pada suatu waktu. Karena itu, tidak perduli seberapa cepat data dapat diproses di dalam unit pemrosesan paralel, kecepatan arsitektur ini *yang sesungguhnya* tergantung pada kecepatan unit RAM.

Jenis memori lainnya yang sangat cocok untuk arsitektur paralel adalah **memori asosiatif** yang diperkenalkan pada Bab 7. Memori ini berbeda dengan sebuah RAM, dimana memori ini *isinya dapat dialamati* (*content-addressable*) dan seluruh memori dapat ditelusuri secara bersamaan waktunya. Jika suatu memori asosiatif digunakan dalam sebuah prosesor array, komputernya diberi nama **prosesor asosiatif**.

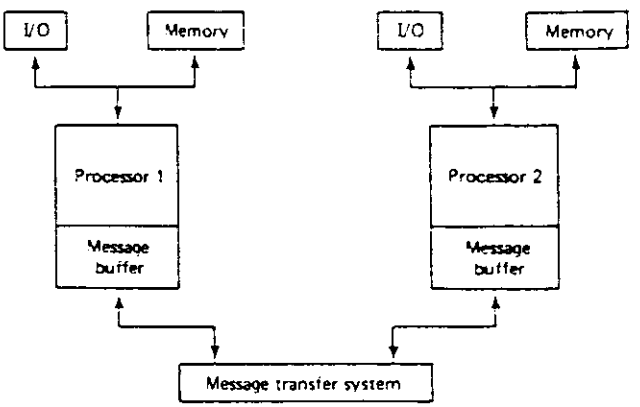
Kerugian utama suatu prosesor asosiatif adalah harga unit memori. Saat ini, rasio harga-per-unjukkerja dianggap terlalu tinggi untuk penerapan mesin-mesin ini secara komersial. Namun ada beberapa prosesor asosiatif yang telah dibangun untuk aplikasi militer. Komputer PEPE (Parallel Element Processing Ensemble) merupakan suatu komputer asosiatif yang digunakan untuk aplikasi pemrosesan sinyal radar. Komputer STARAN, dibangun oleh Goodyear Aerospace, juga dianggap sebagai suatu komputer asosiatif. STARAN pertama diatur untuk pemrosesan citra digital pada tahun 1975. Sejak saat itu, telah banyak dipasang model-model yang diperbaharui dengan unit memori yang lebih besar dan kecepatan pemrosesan yang lebih tinggi.

10.7 SISTEM MULTIPROSESOR

Sistem multiprosesor merupakan sebuah sistem dimana sekumpulan prosesor dalam suatu *komputer tunggal* berhubungan dan bekerja sama satu sama lain untuk memecahkan suatu permasalahan. (Jangan campur adukkan dengan sistem multiprograming, dimana lebih dari satu program berjalan pada sistem tersebut. Multiprograming disediakan oleh sistem operasi dan dapat diimplementasikan pada sistem uniprosesor ataupun multiprosesor.) Prosesor tersebut dapat berkomunikasi melalui baris data langsung melalui memori yang terbagi-bagi atau dengan perantaraan kombinasi memori itu. Kita menggunakan frase "dalam suatu komputer tunggal" yang berarti bahwa kita tidak membicarakan komputer berganda yang dihubungkan bersama-sama ke suatu sistem multiprosesor. Jenis sistem itu lebih umum disebut sebagai **sistem terdistribusi** (*distributed system*). Yang lebih penting, pembatasan ini berarti bahwa suatu sistem operasi digunakan untuk semua prosesor dalam sistem tersebut.

Sistem Memori-Pribadi dan Memori-Terbagi

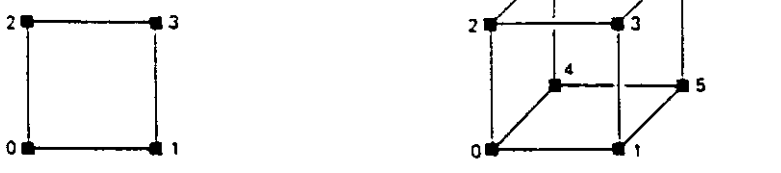
Kita dapat mengelompokkan sistem multiprosesor menjadi dua klasifikasi umum yang didasarkan pada jenis hubungan antara prosesor-prosesor tersebut. Suatu sistem prosesor dengan **memori pribadi** (*private-memory*) (terkadang disebut sebagai **sistem pasangan loggar**/*loosely coupled*) merujuk pada sebuah konfigurasi dimana prosesor itu memiliki memori lokal yang besar dan mungkin juga



Gambar 10-14 Sistem prosesor dengan memori pribadi

kelompok piranti I/O mereka. Prosesor itu saling berkomunikasi satu sama lain melalui suatu **sistem transfer pesan** (*message transfer system*) seperti diperlihatkan pada Gambar 10-14. Setiap prosesor juga menyertakan suatu area buffer untuk menyimpan pesan-pesan yang tiba sewaktu yang satunya dilayani. Sistem transfer pesan dalam sebuah sistem pasangan longgar mempunyai efek terbatas bagi unjuk rasa sistem tersebut. Ia dapat sesederhana suatu bus terbagi (*shared bus*) atau serumit suatu sistem memori-terbagi yang efisien dengan kendali komunikasi yang terpasang di dalamnya (*built-in*). Contoh yang paling umum tentang multiprosesor memori pribadi adalah mesin hipercube. Dalam suatu mesin hipercube, pasangan prosesor-memori (atau **node**) dihubungan dengan suatu cara khusus sedemikian sehingga, untuk sebuah sistem dengan n buah prosesor, setiap node dihubungkan secara langsung ke $\log_2 n$ node lainnya dan jalur terpanjang antara dua sembarang node adalah $\log_2 n$. Gambar 10-15 memperlihatkan dua contoh hipercube: sebuah hipercube dua-dimensi pada bagian (a) dan sebuah hipercube tiga-dimensi pada bagian (b).

Pada sistem multiprosesor jenis kedua, disebut sebagai suatu sistem **memori terbagi** (*shared-memory*) (atau **pasangan ketat**/*tightly coupled*), semua prosesor membagi-bagi sebuah memori utama umum, seperti diperlihatkan pada Gambar 10-16. Biasanya setiap prosesor memiliki piranti I/O yang terpisah dan mungkin memiliki cache yang berbeda pula. Setiap prosesor dihubungkan ke memori utama komputer via sebuah jaringan penghubung, seperti diperlihatkan pada Gambar 10-16(a) atau dengan menggunakan sebuah **memori multiport**, seperti diperlihatkan pada Gambar 10-16(b). Perhatikan bahwa dalam konteks sistem

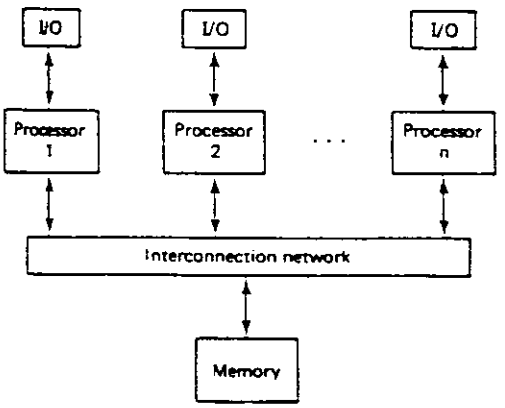


(a) Four-node, two-dimensional hypercube (b) Eight-node, three-dimensional hypercube

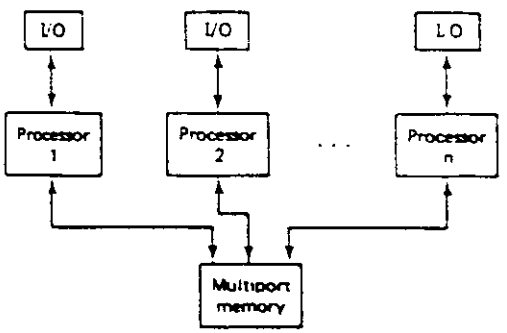
Gambar 10-15 Arsitektur hipercube

multiprosesor, jaringan penjajaran (*alignment network*) yang disebut-sebut pada Bagian 10.6 dirujuk sebagai jaringan penghubung. Dalam kasus memori-terbagi, sistem tersebut kurang fleksibel untuk pengembangan (ekspansi). Juga harus ada sebuah sirkuit logika untuk berhubungan dengan usaha-usaha yang berkesinambungan (percekcokan) oleh dua atau lebih prosesor untuk mengakses memori terbagi. Hal ini akan dibahas pada subbagian berikutnya.

Seperti pada mesin-mesin yang sesungguhnya, hipercube komersial yang tersedia menyertakan mesin FPS seri-T, Intel iPSC dan NCUBE. Ada berbagai macam mesin memori terbagi, tapi dua yang paling baik adalah superkomputer ETA 10 dan Cray-3.



(a) Connections via an interconnection network



(b) Connections using multiport memory

Gambar 10-16 Sistem multiprosesor memori-terbagi

Organisasi Perangkat Keras Multiprosesor

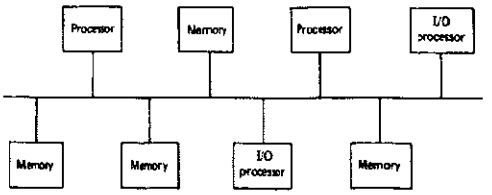
Banyak keuntungan sistem multiprosesor. Di antaranya adalah kehandalan yang tinggi, meningkatnya ketersediaan unit memori dan piranti I/O dan meningkatnya kemampuan hitung. Tidak peduli apakah sistem tersebut merupakan pasangan longgar atau pasangan ketat, hubungan antara prosesor-prosesor tersebut, juga unit memori dan piranti I/O, menentukan unjuk kerja dan efisiensi keseluruhan sistem. Tiga macam organisasi yang paling umum adalah bus umum, saklar crossbar dan memori multiport.

Bus umum. Bus umum (*common bus*), juga disebut sebagai **bus dengan waktu terbagi** (*time-shared bus*), merupakan skema hubungan yang paling sederhana untuk suatu sistem multiprosesor. Seperti diperlihatkan pada Gambar 10-17, ia merupakan jalur komunikasi tunggal antara komponen-komponen fungsional. Bus itu sendiri merupakan suatu piranti pasif, dimana operasi transfer dikendalikan oleh interface bus dalam komponen tersebut.

Keuntungan utama organisasi bus umum adalah relatif sederhananya kita untuk menambahkan atau melepaskan komponen-komponennya. Meskipun diperlukan suatu protokol komunikasi yang tepat bagi setiap komponen untuk mengetahui komponen-komponen lainnya yang ada di dalam bus, hal ini ditangani oleh perangkat lunak dan dapat dengan mudah kita modifikasi.

Sayangnya, jalur komunikasi tunggal, yang menghasilkan sistem sederhana yang murah, juga menjadi penyebab beberapa masalah yang serius. Sebagai contoh, jika bus tidak berhasil/gagal maka seluruh sistem ikut gagal. Juga, dengan semakin bertambah sibuknya sistem tersebut, pertarungan memperebutkan bus menyebabkan suatu penurunan unjuk kerja yang serius. Variasi-variasi pada skema ini, seperti dua bus unidirectional atau multiple bus bidirectional, tidak mengatur bagaimana membenahi kekurangan tersebut tanpa mengorbankan kesederhanaan sistem itu. Secara umum, organisasi jenis ini hanya digunakan dalam sistem multiprosesor yang kecil.

Saklar crossbar. Konsep suatu saklar crossbar diperkenalkan pada Bagian 10.6. Dalam konteks sistem multiprosesor, hal ini berarti bahwa suatu jalur yang terpisah menghubungkan setiap prosesor ke setiap unit memori. Karena setiap unit memori diakses oleh jalur-jalur yang berbeda, tidak akan terjadi bloking yang disebabkan oleh adanya transmisi-transmisi yang bersamaan waktunya. Suatu organisasi saklar crossbar untuk sebuah sistem multiprosesor diperlihatkan pada Gambar 10-18. Jumlah prosesor. piranti I/O dan unit memori bersifat arbitrer dan tidak harus sama.



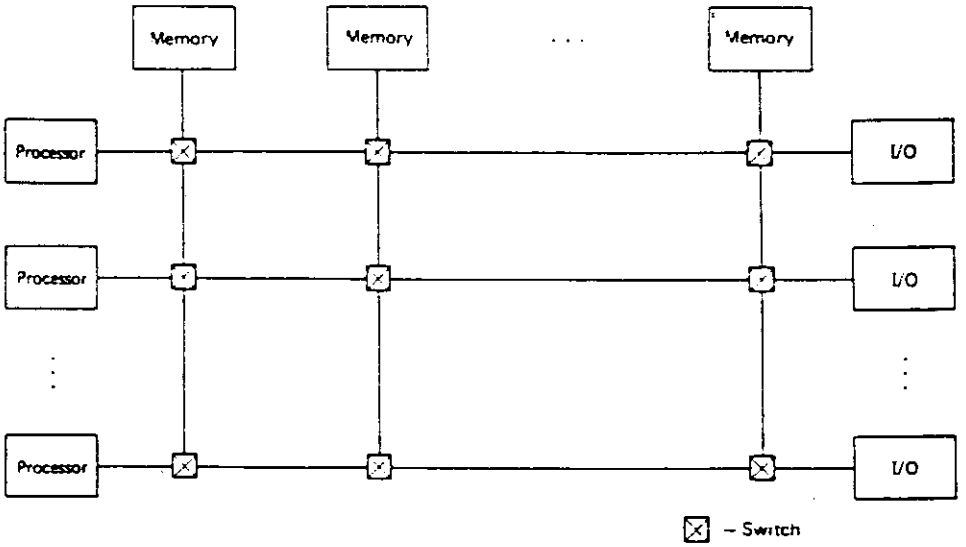
Gambar 10-17 Organisasi multiprosesor bus umum

Meskipun transmisi ke unit memori yang berbeda-beda di waktu yang bersamaan dapat terjadi tanpa ada masalah, saklar itu harus dapat memecahkan konflik yang dihasilkan dari akses yang bersamaan waktunya ke unit memori yang sama. Permintaan-permintaan yang menimbulkan konflik ini biasanya ditangani dengan suatu dasar prioritas yang telah ditentukan sebelumnya. Sewaktu hal ini terjadi, perangkat keras yang meminta implementasi saklar-saklar ini dapat menjadi agak kompleks. Namun kompleksitas ini sebenarnya menjadikan interface-interface pada komponen fungsional menjadi lebih sederhana. Selain itu juga mudah bagi kita untuk melepaskan komponen yang salah berfungsi tanpa menghentikan atau menurunkan kemampuan keseluruhan sistem. Organisasi ini biayanya paling efektif untuk sistem yang berukuran sedang. Namun, jika jumlah total hubungan prosesor-memori menjadi besar maka organisasi ini menjadi sangat mahal.

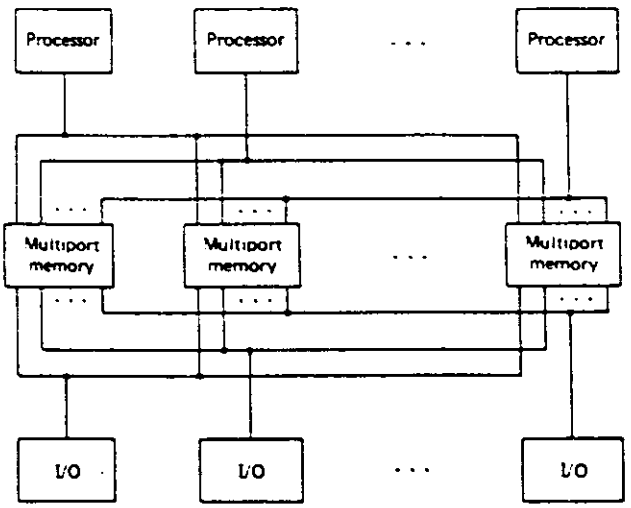
Memori multiport. Organisasi ketiga didapat dengan mengeluarkan logika kendali, logika saklar arbitrasi prioritas dari saklar crossbar dan meletakkan mereka di dalam interface masing-masing unit memori. Kemudian semua komponen fungsional lainnya mengakses unit memori melalui suatu port tertentu, seperti diperlihatkan pada Gambar 10-19. Sebuah port dipasok untuk masing-masing komponen fungsional.

Seperti pada saklar crossbar, konflik-konflik biasanya dipecahkan menurut prioritasnya. Salah satu ciri khas organisasi memori multiport yang cukup baik adalah bahwa akses ke unit memori tertentu dapat dilarang cukup dengan tidak menghubungkan komponen yang akan dilarang tersebut ke port memori. Hal ini digunakan untuk menciptakan unit penyimpanan pribadi, biasanya untuk alasan keamanan.

Organisasi memori multiport juga merupakan memori yang paling cocok bagi sistem berukuran menengah. Dimana terhambatnya sistem multiprosesor yang besar paling dipengaruhi oleh ketidakmampuan saklar-saklar untuk menghasilkan biaya yang wajar dan unjuk kerja yang baik.



Gambar 10-18 Organisasi multiprosesor saklar crossbar



Gambar 10-19 Organisasi multiprosesor memori multiport

Multicache dalam Sistem Multiprosesor

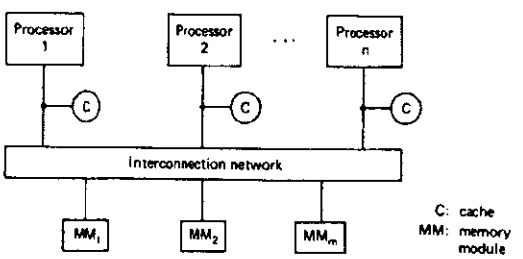
Telah kita singgung dalam pembahasan mengenai sistem multiprosesor berpasangan ketat, cache yang terpisah dapat menjadi sebuah bagian dari suatu sistem. Alasan yang mengharuskan kita memiliki cache sama seperti pembahasan di Bab 7 dan cache juga dapat digunakan dalam sistem berpasangan longgar. Sayangnya, kehadiran cache pribadi dalam suatu sistem multiprosesor, seperti diperlihatkan pada Gambar 10-20, menimbulkan masalah bagi konsistensi data. Hal ini dikenal sebagai masalah **perpautan cache** (*cache coherence*).

Masalah perpautan cache hanya ada jika cache dihubungkan dengan prosesor individual. Dalam hal ini, bisa terdapat beberapa salinan data yang sama di cache yang berlainan pada saat yang bersamaan. Jika salah satu prosesor kemudian menulis ke cachenya maka cache lain akan berisi data yang tidak ikut termodifikasi, dengan demikian data menjadi tidak tepat. Suatu kebijakan write-through yang sederhana tidak akan memecahkan masalah ini, karena hanya unit memori utama yang akan termodifikasi dan tidak semua cache akan berisi data tersebut.

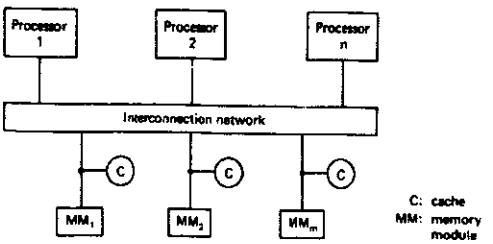
Contoh 10.5

Anggap bahwa word X di dalam memori utama mengandung nilai 76 dan anggap bahwa terdapat sebuah salinan word ini di dalam cache pribadi pada prosesor P_1, P_2 dan P_3. Lebih jauh, asumsikan bahwa disini berlaku suatu kebijakan write-through. Jika prosesor P_1 memodifikasi salinan word X-nya menjadi 85, word X di dalam memori utama juga berubah menjadi 85. Namun P_2 dan P_3 tetap berisi nilai 76, bukannya nilai yang baru 85, sehingga menyebabkan data tidak konsisten.

Salah satu cara menghindari masalah perpautan cache adalah dengan menghubungkan cache dengan memori terbagi (*shared*) daripada dengan prosesor. Seperti diperlihatkan pada Gambar 10-21, prosesor tersebut kemudian akan harus melewati



Gambar 10-20 Sistem multiprosesor dengan cache-cache pribadi



Gambar 10-21 Sistem multiprosesor dengan cache-cache terbagi.

jaringan penghubung untuk mengakses cache itu, sehingga menghilangkan sebagian besar keuntungan percepatan yang diperoleh dengan mengunakan cache di tempat pertama. Ada dua metode berbeda yang telah diusulkan untuk mengalamati masalah perpautan cache: pengujian perpautan statis dan pengujian perpautan dinamis.

Konsep dasar metode **pengujian perpautan statis** adalah memisahkan data menjadi kelas *umum/terbagi* (*shared*) dan *pribadi*. Dengan demikian dapat ditangani dengan cara yang berbeda untuk menghindari setiap ketidakkonsistenan. Salah satu metode untuk berhubungan dengan data yang terpisah-pisah adalah dengan menyimpan data yang terbagi (*shared*), yang dapat dimodifikasi (*modifiable*), di dalam memori utama dan hanya menyimpan data pribadi, yang hanya akan digunakan oleh satu prosesor tertentu, dalam cache yang semestinya. Metode lainnya mencakup pengelolaan suatu cache terbagi (shared) yang khusus untuk data yang terbagi dan ,juga, menyimpan data pribadi dalam cache.

Pada metode **pengujian perpautan dinamis**, beberapa salinan dari sembarang data diperbolehkan. Kapanpun sebuah prosesor mengubah data di dalam cachenya sendiri, ia harus membuat semua salinan data dalam cache lainnya menjadi tidak valid. Namun penerapan metode ini menghasilkan kerugian-kerugian berikut ini:

1. Penurunan rata-rata rasio hit (lihat Bagian 7.5) sehubungan dengan proses pembuatan data menjadi tidak valid.
2. Lalulintas antarcache mendesak konsistensi.
3. Konflik-konflik yang dihasilkan dari pengaksesan tabel global yang berurutan digunakan untuk menyimpan track dari cache mana yang mengandung salinan word memori yang mana.

4. Lebih banyak write-back sehubungan dengan hal dimana proses pembuatan data yang termodifikasi menjadi tidak valid tidak tulis kembali ke memori utama kecuali jika ada word-word yang belum termodifikasi untuk menggantikannya.

10.8 ORGANISASI PARALEL SECARA UMUM

Dalam bab ini kita telah membahas beberapa organisasi yang digunakan untuk menghasilkan pemrosesan paralel. Meskipun terdapat beberapa jenis prosesor paralel yang berbeda (seperti kita saksikan dengan adanya skema klasifikasi yang berbeda), mereka semua masuk ke dalam tiga kategori umum: sistem prosesor array, multiprosesor dan arus data.

Sistem **prosesor array** terdiri atas sejumlah elemen pemrosesan yang beroperasi secara paralel di bawah pengendalian sebuah unit kendali tunggal. Masing-masing elemen pemrosesan melakukan operasi yang sama pada saat yang sama, pada elemen data yang berbeda-beda. Sistem **multiprosesor** terdiri atas sejulah prosesor yang lengkap yang dapat memproses data secara independen.

Suatu jenis organisasi paralel yang benar-benar berbeda adalah arsitektur **arus data**. Dalam sebuah arsitektur arus data, instruksi secara otomatis dimungkinkan untuk dieksekusi begitu operand dta yang mereka minta tersedia, bukannya sebuah program counter menentukan kapan instruksi akan dilaksanakan. Tidak ada mekanisme arus pengendalian (program counter) dan urutan eksekusi instruksi tergantung sepenuhnya pada dependensi data antara instruksi tersebut; dari situ diperoleh istilah *arus data*. Secara teoritis, perpautan yang maksimal mungkin diperoleh pada jenis arsitektur ini, dengan sumberdaya yang mencukupi (prosesor, port memori, piranti I/O, dan lain-lain). Untuk pembahasan yang lebih mendalam mengenai sistem arus data, lihat Hwang dan Briggs (1984) dan IEEE Computer Magazine edisi Februari 1982.

SOAL

1. Berikan definisi istilah-istilah di bawah ini:
   - **(a)** SIMD
   - **(b)** MIMD
   - **(c)** irisan bit
   - **(d)** instruksi vektor
   - **(h)** sistem prosesor array
   - **(i)** jaringan penghubung
   - **(j)** saklar crossbar
   - **(k)** prosesor asosiatif

(e) floating point blok
(f) padding
(g) pengalamatan rapat-rapat
(l) multiprosesor
(m) perpautan cache

2. Klasifikasikan jenis-jenis paralelisme berikut ini apakah sebagai (A) selama interval yang sama [Gambar 10-1(a)], (B) pada waktu yang bersamaan [Gambar 10-1(b)] atau (C) pada rentang waktu yang saling tumpang tindih [Gambar 10-1(c)].

(a) Pipelining
(b) Beberapa unit fungsional
(c) Operasi CPU dan I/O yang tumpang tindih
(d) Interleaving memori
(e) Multiprograming
(f) Multiprosesing

3. Gambarkan batas bawah (konjektur Minsky) dan batas atas untuk percepatan sebuah program sebagai suatu fungsi dari jumlah prosesor dari 2^0 sampai 2^{10}.

4. Buatlah suatu format floating-point blok dari vektor-vektor floating-point berikut ini:

(a) $(0{,}826 \times 10^3, -0{,}1530 \times 10^1, 0{,}2649 \times 10^2)$
(b) $(-0{,}1837 \times 10^{-2}, 0{,}43 \times 10^{-3}, 0{,}205 \times 10^{-2})$
(c) $(0{,}686 \times 10^1, -0{,}7060 \times 10^0, 0{,}111 \times 10^{-2})$

5. Anggaplah A = (5, 234, 100, 97, 3) dan B = (802, 311, 65). Tunjukkan bagaimana vektor-vektor ini dapad di-padding-kan untuk meningkatkan efisiensi operasi-operasi berikut ini:

(a) $A \times B$
(b) $A + B$
(c) $3 \times A$

6. Berdasarkan matriks pada Gambar 10-6, nyatakanlah apakah vektor-vektor pengalamatan regular berikut ini sekuensial, nonsekuensial atau submatriks.

(a) $(M_{31}, M_{41}, M_{12}, M_{22})$
(b) $(M_{43}, M_{13}, M_{22}, M_{31})$
(c) (M_{23}, M_{33}, M_{43})
(d) $(M_{22}, M_{23}, M_{32}, M_{33}, M_{42}, M_{43})$

7. Mengacu pada matriks dalam Gambar 10-6, untuk setiap vektor berikut ini, berikan vektor bit yang menggambarkan lokasinya di dalam memori dan tentukan apakah vektor tersebut berupa pengalamatan rapat-rapat (dense) atau jarang-jarang (sparse).

(a) $(M_{42}, M_{32}, M_{22}, M_{12})$

(b) (M_{12}, M_{13}, M_{11})
(c) (M_{11}, M_{21}, M_{31}, M_{41}, M_{43})
(d) (M_{31}, M_{32}, M_{41}, M_{42})

8. Hitung vektor indeks untuk vektor pada soal nomor 7(c).

9. Gambarkan sebuah diagram logika untuk kotak saklar 3 x 3.

10. Merujuk pada saklar sirkulasi 4 x 4 pada Gambar 10-12, jelaskan apa yang diperlukan untuk menghubungkan hal-hal berikut ini:
(a) input 3 ke output 2
(b) input 1 ke output 4
(c) input 2 ke output 3

11. Tentukan jalur (path) yang dapat diikuti pada jaringan Benes 8 x 8 dalam Gambar 10-13 untuk menghasilkan sekumpulan hubungan secara simultan berikut ini:
(a) 1-8, 8-1
(b) 2-5, 4-3, 7-2
(c) 6-6, 7-1, 2-2

12. Untuk jaringan Benes pada Gambar 10-13, terangkan mengapa benar bahwa jalur *a-g-l* dan *b-g-k* dapat diproses secara bersamaan tetapi jalur *a-g-l* dan *b-g-l* tidak dapat.

13. Bagaimana proses yang berjalan pada prosesor yang berbeda berkomunikasi dalam setiap prosesor?
(a) Multiprosesor dengan memori pribadi (*private*)
(b) Multiprosesor dengan memori terbagi (*shared*)

14. Bandingkan dan pertentangkan unjuk kerja dan efisiensi suatu sistem multiprosesor yang hubungan antarprosesornya melalui:
(a) bus umum
(b) saklar crossbar
(c) memori multiport

15. Bandingkan dan pertentangkan organisasi cache pada Gambar 10-20 dan 10-21.

16. Klasifikasikan mesin-mesin berikut ini menurut taksonomi Flynn:
(a) BSP
(b) CYBER 205
(c) Intel iPSC
(d) NCUBE
(e) STARAN
(f) Cray-3
(g) ILLIAC IV
(h) MPP
(i) PEPE
(j) FPS T-series.