VI. OPERASI MATRIKS (Part 2)

Oleh Dr. Asep Juarna

2. Perkalian Matriks dengan Vektor

Sebagai ilustrasi awal, diberikan matriks ***A*** dan vektor ***U*** masing-masing dengan ukuran 3 × 4 dan 4 × 1; hasilnya adalah vektor ***V*** yang tentu saja berukuran 3 × 1, seperti terlihat di bawah ini:

$$\boldsymbol{A} \times \boldsymbol{U} = \boldsymbol{V} \Leftrightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \times \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} a_{11}u_1 + a_{12}u_2 + a_{13}u_3 + a_{14}u_4 \\ a_{21}u_1 + a_{22}u_2 + a_{23}u_3 + a_{24}u_4 \\ a_{31}u_1 + a_{32}u_2 + a_{33}u_3 + a_{34}u_4 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

2.1. Algoritma Sekuensial

Ilustrasi di atas menunjukkan bahwa perkalian matriks dengan vektor hanya mungkin jika banyaknya kolom matriks sama dengan dimensi vektor, dan hasilnya adalah sebuah vektor berdimensi sama dengan jumlah baris vektor dengan komponen sebagai berikut:

$$v_i = \sum_{j=1}^{4} a_{i,j} \times u_j \text{ , untuk } i = 1, 2, \text{ dan } 3$$

Secara umum, untuk matriks ***A*** berukuran $m \times n$ dan vektor ***U*** berukuran $n \times 1$ komponen vektor ***V*** di atas adalah:

$$v_i = \sum_{j=1}^{n} a_{i,j} \times u_j \text{ , untuk } i = 1, 2, \ldots m$$

Dengan demikian, algoritma sekuensial perkalian matriks dengan vektor adalah sebagai berikut:

procedure *seq_mult-mat-vec* (*A*, *U*, *V*)
(1) **for** $i = 1$ **to** m **do**
(2) $v_i \leftarrow 0$
(3) **for** $j = 1$ **to** n **do**
(4) $v_i \leftarrow v_i + a_{i,j} \times u_j$
end for
end for

Sebanyak n iterasi pada *loop* (3) diulang sebanyak m kali pada *loop* (1) sehingga *running time* **procedure *seq_mult-mat-vec*** adalah $t(n) = n \times m$. Jika $m < n$ maka *running time* ini adalah $O(n^2)$; sesuai dengan definisi *big-Oh*, ini artinya berapapun m, selama $m < n$, perkalian $n \times m$ tidak akan melebihi n^2.

2.2. Algoritma Paralel

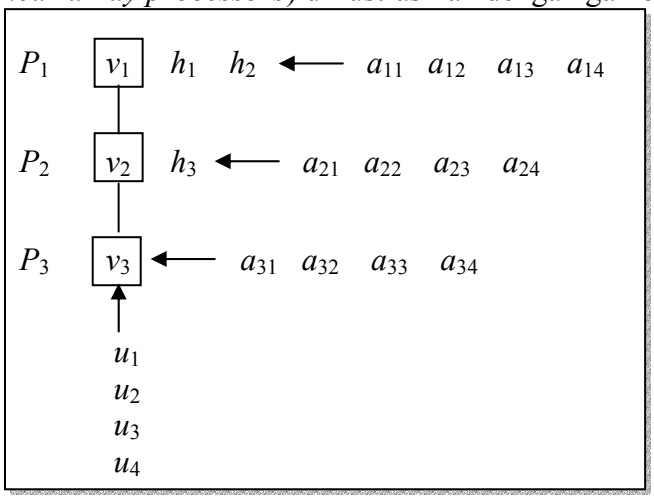
2.2.1. *Linear Array*

Banyaknya prosesor pada *linear array* (LA) adalah sebanyak jumlah baris matriks yang diperkalikan, yaitu n. Prosesor-prosesor tersebut disusun vertikal sedemikain rupa sehingga prosesor ke-n berada paling bawah dan prosesor pertama paling atas. Vektor diinputkan dari bawah sedangkan baris-baris matriks diinputkan dari samping seperti

terlihat pada Gambar 6.3. Setiap prosesor mempunyai 3 register a, u, dan v. Ketika prosesor P_i menerima 2 input $a_{i,j}$ dan u_j, prosesor tersebut melakukan hal-hal berikut:
(i) menyimpan $a_{i,j}$ dan u_j masing-masing di register a dan u
(ii) mengalikan isi register a dengan isi register u
(iii) menambahkan hasil (ii) ke isi register v
(iv) mengirim u_j ke P_{i-1} (kecuali $i = 1$, artinya prosesor P_1 tidak mengirim u_j karena P_1 adalah prosesor terakhir yang menerima u_j).

Perhatikan pada konfigurasi di atas bahwa baris ke i matriks $\boldsymbol{A}$ terlambat satu langkah daripada baris ke $i + 1$ untuk $1 \le i \le m$ -1, sebagaimana ditunjukkan oleh adanya h_1, h_2 dan h_3; hal ini sengaja dilakukan untuk memastikan $a_{i,j}$ diperkalikan dengan u_j pada waktu yang tepat.

Procedure paralel perkalian matriks 3 × 4 dengan vektor 4 × 1 menggunakan 3 prosesor larik linier (*linear array processors*) diilustrasikan dengan gambar berikut:



Gambar 6.3. Ilustrasi prosedur paralel perkalian matriks 3 × 4 dengan vektor kolom berdimensi 4 menggunakan *array linear* berprosesor 3

Algoritma sekuensial perkalian matriks dengan vektor, dengan demikian, adalah sebagai berikut:

procedure *par_LA-mult-mat-vec* (*A*, *U*, *V*)
for $i = 1$ **to** m **do in parallel**
  $v_i \leftarrow 0$
  while prosesor P_i menerima dua input a dan u **do**
    (1) $v_i \leftarrow v_i + (a + u)$
    (2) **if** $i > 1$ **then** prosesor P_i mengirim u ke prosesor P_{i-1}
        end if
  end while
end for

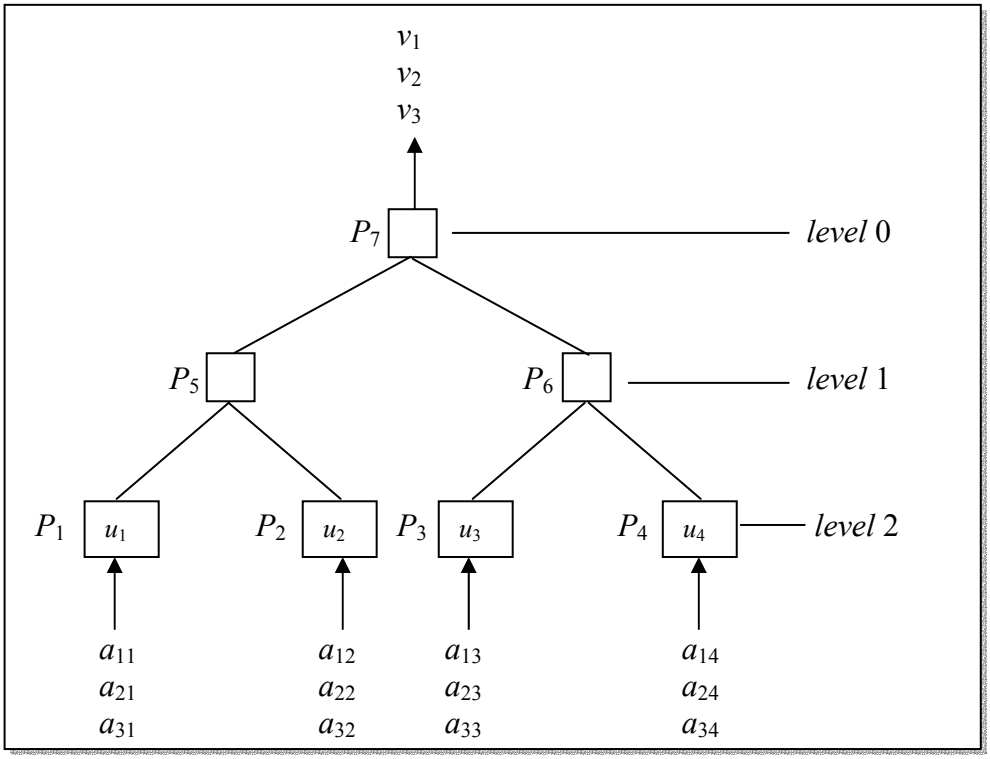
Elemen matriks $a_{1,n}$ adalah elemen terakhir yang diterima prosesor, yaitu prosesor P_1. Dari posisi awalnya, elemen $a_{1,n}$ ini membutuhkan $m + n - 1$ langkah untuk sampai ke prosesor P_1. Dengan demikian, jika $m < n$ maka *running time* **procedure *par_LA-mult-mat-vec*** adalah $t(n) = O(n)$. Banyaknya prosesor yang digunakan **procedure *par_LA-***

mult-mat-vec ini adalah $p(n) = n$, dengan demikian *cost* procedure ini adalah $c(n) = p(n) \times t(n) = O(n^2)$; nilai ini adalah *cost optimal*.

Latihan: Diberikan matriks ***A*** berukuran $m \times n$ dan vektor ***U*** berukuran $n \times 1$ sebagai input **procedure *par_LA-mult-mat-vec*** menggunakan n prosesor. Tentukan selisih langkah antara selesainya pekerjaan prosesor P_n dengan selesainya pekerjaan prosesor P_1! Bagaimana dengan selisih langkah antara selesainya pekerjaan prosesor P_n dengan selesainya pekerjaan prosesor P_k, $1 < k < n$?

2.2.2. *Tree*

Prosedur paralel perkalian matriks dengan vektor akan diimplementasikan dengan menggunakan *tree of processor*, tepatnya *complette binary tree of processor* dengan *d level* (bernomor 0 s.d. *d* - 1) dan jumlah prosesor $N = 2^d - 1$. Matriks yang dikalikan berukuran $m \times n$ sedangkan dimensi vektornya adalah n di mana $n = 2^{d-1}$. Untuk $m = 3$ dan $d = 3$ (yaitu $n = 2^{3-1} = 2^2 = 4$) situasinya terlihat pada gambar berikut:



Gambar 6.4. Ilustrasi prosedur paralel perkalian matriks 3×4 dengan vektor kolom berdimensi 4 menggunakan *tree* berprosesor 17 atau dengan *level* $d = 3$.

Pada Gambar 6.4 di atas, P_1 s.d. P_4, yang merupakan prosesor-prosesor daun (*leaf*), bertugas mengalikan sebuah elemen matriks dengan sebuah elemen vektor dan mengirimkan hasilnya ke prosesor induknya (*parent*). P_5 dan P_6, yang merupakan prosesor-prosesor dalam (*inner*), bertugas menjumlahkan bilangan yang diperoleh dari kedua anaknya (*children*) dan mengirimkan hasilnya ke induknya. P_7, yang merupakan akar (*root*) bertugas menjumlahkan bilangan yang diperoleh dari kedua anaknya dan mengeluarkannya sebagai *output*. Perhatikan bahwa ketika P_1 s.d. P_4 mengalikan sebuah elemen matriks dengan sebuah elemen vektor dan mengirimkan hasilnya ke prosesor induknya, pada saat yang sama P_5 dan P_6 melakukan penjumlahan dua bilangan, jika ada, yang sebelumnya diterima dari kedua anaknya mengirimkan hasilnya P_1, dan pada saat yang sama pula P_7 menjumlahan dan mengirimkan hasilnya

sebagai output prosedur. Dengan demikian, algoritma prosedur paralel perkalian matriks menggunakan *tree* tersebut adalah sebagai berikut:

procedure *par_Tree-mult-mat-vec* (*A*, *U*, *V*)
do step 1 dan 2 **in parallel**
(1) **for** $i = 1$ **to** n **do in parallel**
  for $j = 1$ to m **do in parallel**
  (1.1) hitung $u_i \times a_{i,j}$
  (1.2) kirim hasilnya ke induk (*parent*)
  end for
 end for
(2) **for** $i = n + 1$ **to** $2n - 1$ **do in parallel**
  while P_i menerima 2 input **do**
  (2.1) hitung jumlah kedua input
  (2.2) **if** $i < 2n - 1$ **then** kirim hasilnya ke induk
  else kirim hasilnya sebagai output algoritma
  end if
 end while
end for

Diperlukan log n langkah setelah baris pertama matriks ***A*** diterima prosesor-prosesor daun (*leaf processors*) untuk dikeluarkan prosesor akar (*root processor*) sebagai *output* v_1. Karena matriks ***A*** mempunya m baris, diperlukan $m - 1$ langkah berikutnya sehingga semua baris matriks ***A*** dikeluarkan prosesor akar sebagai *output*. Dengan demikian *running time* **procedure *par_Tree-mult-mat-vec*** adalah $t(n) = (m - 1) + \log n$. Jumlah prosesor yang digunakan adalah $p(n) = N = 2^d - 1 = 2 \times 2^{d-1} - 1 = 2n - 1$. Jika $m < n$ maka $t(n) = O(n)$, $p(n) = O(n)$, dan $c(n) = t(n) \times p(n) = O(n^2)$, dan ini adalah *cost optimal*.

3. Perkalian Matriks dengan Matriks

Diberikan dua matriks ***A*** berukuran $m \times n$ dan matriks ***B*** berukuran $n \times k$. Perkalian matriks ***A*** dengan ***B*** adalah matriks ***C*** berukuran $m \times k$ di mana:

$$c_{i,j} = \sum_{s=1}^{n} a_{i,s} \times b_{s,j}$$

Contoh:
Diberikan matriks ***A*** berukuran 3×2 (yaitu 3 baris 2 kolom) dengan matriks ***B*** berukuran 2×3 (yaitu 2 baris dan 3 kolom) beserta matriks $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$ sebagai berikut:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix},\ B = \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \Rightarrow C = \begin{bmatrix} (1\times 7)+(2\times 10) & (1\times 8)+(2\times 11) & (1\times 9)+(2\times 12) \\ (3\times 7)+(4\times 10) & (3\times 8)+(4\times 11) & (3\times 9)+(4\times 12) \\ (5\times 7)+(6\times 10) & (5\times 8)+(6\times 11) & (5\times 9)+(6\times 12) \end{bmatrix} = \begin{bmatrix} 27 & 30 & 33 \\ 61 & 68 & 75 \\ 95 & 106 & 117 \end{bmatrix}$$

Perhatikan bahwa, misalnya, $c_{2,3} = 75 = (3 \times 9) + (4 \times 12) = (a_{2,1} \times b_{1,3}) + (a_{2,2} \times b_{2,3}) = \sum_{s=1}^{2} a_{2,s} \times b_{s,3}$, $c_{3,1} = 95 = (5 \times 7) + (6 \times 10) = (a_{3,1} \times b_{1,1}) + (a_{3,2} \times b_{2,1}) = \sum_{s=1}^{2} a_{3,s} \times b_{s,1}$, dan seterusnya.

3.1. Algoritma Sekuensial

Berdasarkan contoh di atas, algoritma sekuensial perkalian matriks ***A*** berukuran $m \times n$ dengan matriks ***B*** berukuran $n \times k$ (hasilnya adalah matriks ***C*** berukuran $m \times n$) adalah sebagai berikut:

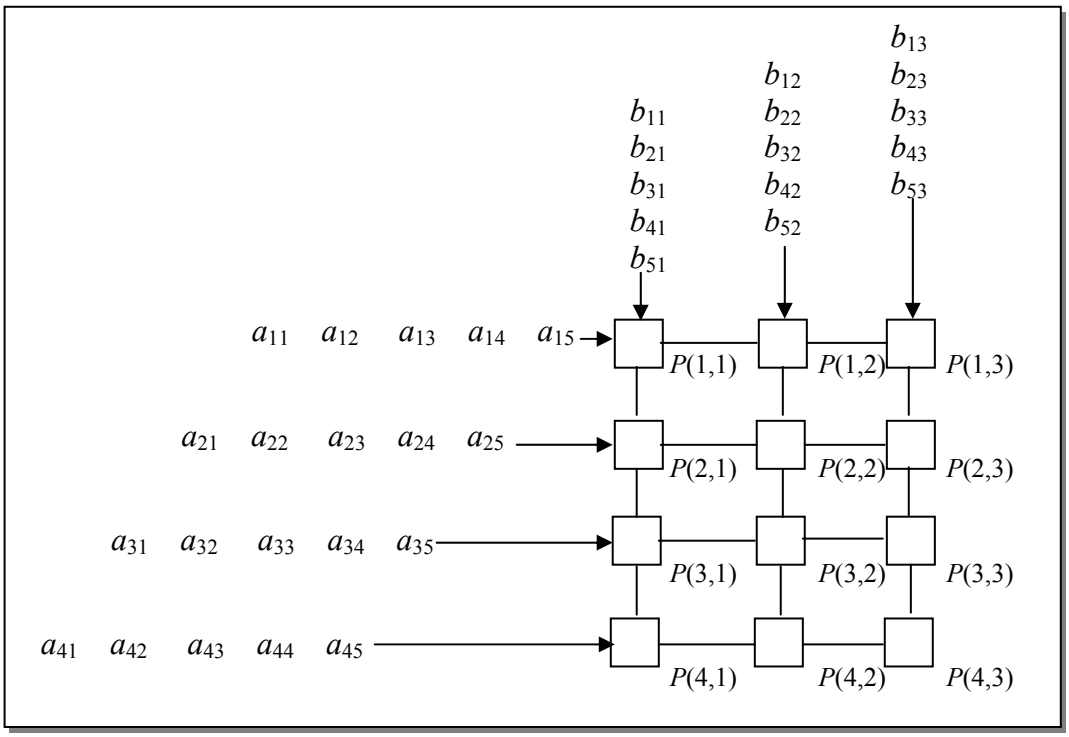
procedure *seq_mult-mat* (*A*, *B*, *C*)
{Input: ***A***($m \times n$), ***B***($n \times k$)
Output: ***C***($m \times k$)}
(1) **for** $i = 1$ **to** m **do**
(2) **for** $j = 1$ **to** k **do**
(3) $c_{i,j} \leftarrow 0$
(4) **for** $s = 1$ **to** n **do**
(5) $c_{i,j} \leftarrow c_{i,j} + (a_{i,s} \times b_{s,j})$
(6) **end for**
(7) **end for**
(8) **end for**

Running time **procedure *seq_mult-mat*** di atas adalah $n \times k \times m$. Jika $m \leq n$ dan $k \leq n$ maka *running time* procedure ini adalah $O(n^3)$.

3.2. Algoritma Paralel

3.2.1. *Mesh*

Diberikan matriks ***A*** berukuran $m \times n$ dengan matriks ***B*** berukuran $n \times k$; algoritma paralel perkalian matriks berikut adalah pada struktur prosesor *mesh* dengan $m \times k$ prosesor. Sebagai ilustrasi awal, diberikan matriks ***A*** berukuran 4×5 dengan matriks ***B*** berukuran 5×3 menggunakan *mesh* prosesor 4×3.



Gambar 6.5. Ilustrasi prosedur paralel perkalian matriks ***A*** berukuran 4×5 dengan matriks ***B*** berukuran 5×3 menggunakan *mesh* berprosesor 4×3.

Seperti pada perkalian matriks dengan vektor menggunakan *linear array*, pada perkalian matriks dengan matriks menggunakan *mesh* terjadi proses keterlambatan yang memang disengaja agar perkalian $a_{i,s}$ dengan $b_{s,j}$, yang merupakan prosedur utama perkalian matriks sesuasi formula $c_{i,j} = \sum_{s=1}^{n} a_{i,s} \times b_{s,j}$, terjadi pada waktu yang tepat. Kali ini keterlambatan tersebut terjadi saat penginputan baris-baris matriks ***A*** dan kolom-kolom matriks ***B***. Baris ke-*m* matriks ***A*** dan kolom ke-*k* matriks ***B*** adalah baris dan kolom terakhir yang diinput ke *mesh*, sedangkan dua elemen terakhir yang diinput adalah $a_{m,1}$ dan $b_{1,k}$. Pada contoh di atas, seperti terlihat pada Gambar 6.5, baris ke-4 matriks ***A***, kolom ke-3 matriks ***B*** adalah baris dan kolom yang terakhir diinput ke *mesh*, sedangkan dua elemen terakhir yang diinput adalah $a_{4,1}$ dan $b_{1,3}$.

Dengan demikian, algoritma paralel perkalian matriks ***A*** berukuran $m \times n$ dengan matriks ***B*** berukuran $n \times k$ (hasilnya adalah matriks ***C*** berukuran $m \times n$) menggunakan *mesh* adalah sebagai berikut:

procedure *par_Mesh_mult-mat* (*A*, *B*, *C*)
{Input: ***A***($m \times n$), ***B***($n \times k$)
Output: ***C***($m \times k$)}
for $i = 1$ **to** m **do in parallel**
  for $j = 1$ **to** k **do in parallel**
    (1) $c_{i,j} \leftarrow 0$
    (2) **while** $P(i,j)$ menerima dua input a dan b
        (a) $c_{i,j} \leftarrow c_{i,j} + (a \times b)$
        (b) **if** $i < m$ **then** kirim b ke $(i+1,j)$
            end if
        (c) **if** $j < k$ **then** kirim a ke $(i,j+1)$
            end if
      end while
  end for
end for

Elemen-elemen $a_{1,n}$ dan $b_{n,1}$ membutuhkan $(m + k + n - 2)$ langkah sejak awal komputasi (yaitu sejak elemen-elemen $a_{m,1}$ dan $b_{1,k}$ masuk *mesh*) sampai $a_{1,n}$ dan $b_{n,1}$ mencapai prosesor yang tepat yaitu $P(m,k)$. Dengan asumsi $m \leq n$ dan $k \leq n$, *running time* **procedure *par_Mesh_mult-mat*** adalah $t(n) = O(n)$, $p(n) = O(n^2)$, sehingga $c(n) = t(n) \times p(n) = O(n^3)$, dan ini adalah *cost optimal* jika mengacu kepada algoritma sekuensial dengan *running time* $t(n) = O(n^3)$.

3.2.2. CRCW SM-SIMD

Algoritma paralel perkalian matriks dengan matriks menggunakan CRCW SM-SIMD adalah implementasi paralelisasi langsung dari algoritma sekuensial problem yang sama. Jumlah prosesor yang digunakan adalah $m \times k \times n$. Setiap prosesor tersebut mempunyai 3 indeks, sehingga notasinya adalah $P(i,j,s)$. CRCW mempunya masalah *write conflict*. Prosedur utama perkalian matriks adalah formula $c_{i,j} = \sum_{s=1}^{n} a_{i,s} \times b_{s,j}$. Mengacu kepada formula ini, *write conflict* pada CRCW diselesaikan dengan prosedur berikut: Jika terdapat lebih dari satu prosesor (ditandai dengan indeks *s* yang berbeda) mencoba menuliskan hasil ke $c_{i,j}$ maka yang akhirnya dituliskan adalah jumlah semua

hasil yang mau dituliskan semua prosesor tersebut. Dengan fakta dan pemikiran tersebut maka algoritma paralel perkalian matriks $\boldsymbol{A}$ berukuran $m \times n$ dengan matriks $\boldsymbol{B}$ berukuran $n \times k$ (hasilnya adalah matriks $\boldsymbol{C}$ berukuran $m \times n$) adalah sebagai berikut:

procedure *par_Mesh_mult-mat* (*A*, *B*, *C*)
{Input: $\boldsymbol{A}(m \times n)$, $\boldsymbol{B}(n \times k)$
Output: $\boldsymbol{C}(m \times k)$}
for $i = 1$ **to** m **do in parallel**
  for $j = 1$ **to** k **do in parallel**
    for $s = 1$ **to** n **do**
      (1) $c_{i,j} \leftarrow 0$
      (2) $c_{i,j} \leftarrow a_{i,s} \times b_{s,j}$
    end for
  end for
end for

Running time **procedure *par_Mesh_mult-mat*** di atas adalah $t(n) = O(1)$. Dengan asumsi $m \leq n$ dan $k \leq n$, jumlah prosesor yang digunakan adalah $t(n) = O(n^3)$, sehingga $c(n) = t(n) \times p(n) = O(n^3)$, dan ini adalah *cost optimal* jika mengacu kepada algoritma sekuensial dengan *running time* $t(n) = O(n^3)$.

Latihan: Pada **procedure *seq_mult-mat***, inisialisasi $c_{i,j} \leftarrow 0$ dilakukan sebelum *loop* **for** $s = 1$ **to** n **do** dan $c_{i,j} \leftarrow c_{i,j} + (a_{i,s} \times b_{s,j})$. Pada **procedure *par_Mesh_mult-mat*** inisialisasi $c_{i,j} \leftarrow 0$ dilakukan sesudah *loop* **for** $s = 1$ **to** n **do** dan $c_{i,j} \leftarrow c_{i,j} + (a_{i,s} \times b_{s,j})$. Jelaskan bahwa bahwa **procedure *par_Mesh_mult-mat*** tersebut menyelesaikan masalah *write conflict* CRCW.