O VZPOREDNEM MNOŽENJU MATRIK

INFORMATICA 1/91

Borut Robič, Polona Blaznik
Institut Jožef Stefan
Ljubljana

Keywords: parallel algorithm, matrix multiplication, parallel computer architecture.

Opisani so trije časovno optimalni algoritmi za vzporedno množenje kvadratnih matrik na modelih SIMD-MC^2, SIMD-CC ter SIMD-PS vzporednega računanja. Podane so ustrezne časovne zahtevnosti. Izpeljan je formalni dokaz pravilnosti algoritma na SIMD-PS.

ON PARALLEL MATRIX MULTIPLICATION: We describe three parallel matrix multiplication algorithms which are known to be time optimal on underlying models of parallel computation: SIMD-MC^2, SIMD-CC, and SIMD-PS. Their time complexities are analized. Also, a formal proof of the algorithm correctness for SIMD-PS is derived.

Keywords: parallel algorithm, matrix multiplication, parallel computer architecture.

1. Uvod

Množenje matrik je operacija, ki se zelo pogosto pojavlja v numeričnih ter nenumeričnih problemih, zato je bila in še vedno je predmet intenzivnih raziskav.

Osnovni zaporedni algoritem za množenje dveh kvadratnih matrik reda n ima časovno zahtevnost $O(n^3)$ aritmetičnih operacij. Slednji je veljal za najboljšega vse do odkritja Strassenovega algoritma leta 1969 s kompleksnostjo $O(n^{2.81})$, kar je sprožilo plaz raziskav, usmerjenih v iskanje časovno še učinkovitejših algoritmov. Poznan je že algoritem s kompleksnostjo $O(n^\omega)$, kjer je $\omega < 2.4955$ [3]. Ker ni znano, ali je $\Omega(n^2)$ hkrati tudi *največja* spodnja meja za časovno kompleksnost zaporednega množenja matrik, je iskanje časovno optimalnega zaporednega algoritma še vedno aktualno. Žal pa se prednost teh algoritmov pokaže šele pri razmeroma velikih matrikah, zato imajo predvsem teoretični pomen.

Po drugi strani pa hkrati z razvojem vzporednih arhitektur postajajo zanimivi tudi vzporedni algoritmi za množenje matrik. Vzporedno množenje matrik se pojavlja celo v problemih, ko jih običajno ne rešujemo z uporabo zaporednega matričnega množenja [4]. V nadaljevanju se bomo osredotočili na družino SIMD računalnikov (Single Instruction stream, Multiple Data stream).

Družina SIMD modelov vzporednega računanja obsega dve poddružini. V prvo spada model SIMD-SM (Shared Memory) skupaj z različnimi variantami, ki se med seboj razlikujejo po stopnji omejevanja in izvedbe sočasnega čitanja/vpisovanja v skupni pomnilnik. Modeli iz te poddružine omogočajo lažji razvoj vzporednih algoritmov, ker se ni potrebno ozirati na arhitekturne značilnosti. Uporabni so predvsem pri določanju spodnjih meja za časovne kompleksnosti algoritmov.

V drugo poddružino pa spadajo realnejši modeli, pri katerih so izpostavljene nekatere arhitekturne značilnosti - predvsem način povezovanja procesorjev ter porazdelitev pomnilnika. Mednje spadajo tudi modeli SIMD-MC (Mesh Connected), SIMD-CC (Cube Connected), ter SIMD-PS (Perfect Shuffle), ki so poimenovani po različnih medprocesorskih povezovalnih shemah. Pri razvoju algoritma za reševanje danega problema moramo upoštevati vse značilnosti modela, na katerem bo potekalo računanje. Zato lahko obstaja za dani problem več časovno različnih optimalnih algoritmov, ki so pač razviti za različne računske modele.

V nadaljevanju bomo opisali problem vzporednega množenja dveh kvadratnih matrik na omenjenih modelih. Matriki označimo z A in B; obe sta reda $n \times n$. Produkt naj bo matrika C z elementi $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$.

2. Množenje na SIMD-SM-CRCW

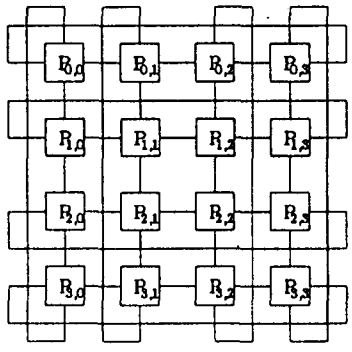
Najprej določimo spodnjo mejo za časovno kompleksnost vzporednega množenja matrik. Vzemimo model SIMD-SM-CRCW, kjer imajo vsi procesorji skupni pomnilnik, v katerega lahko sočasno vpisujejo ali iz njega čitajo. Reševanje konfliktnih situacij pri sočasnem posegu (vpisovanju, čitanju) v skupni pomnilnik je prepuščeno modelu (nekateri pristopi so opisani v [1]); algoritem bo zato enostavnejši, model pa toliko bolj skrivnosten. Pri analizi ne upoštevamo operacije prenašanja podatka iz enega procesorja v drugi. Prav tako predpostavljamo, da število procesorjev ni vnaprej omejeno: če se pri razvoju algoritma pojavi potreba po večjem številu procesorjev, lahko smatramo, da so že na voljo. Kljub svoji nerealnosti pa ta model omogoča določitev spodnje meje za časovno kompleksnost vzporednega množenja matrik, saj se pri razvoju algoritma osredotočimo le na bistvene operacije.

Denimo torej, da je na voljo vsaj n^3 procesorjev. V prvem koraku se sočasno izračuna vseh n^3 produktov $a_{i,k} b_{k,j}$, $0 \leq i,j,k \leq n-1$. Nato se sočasno izračuna vseh n^2 vsot $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$, $0 \leq i,j \leq n-1$. Vsako vsoto lahko izračunamo z $\frac{n}{2}$ procesorji v $\lceil \log n \rceil$ korakih, če izvršimo seštevanje v obliki binarnega drevesa. Skupaj je potrebnih $\lceil \log n \rceil + 1$ korakov, kar je spodnja meja za časovno kompleksnost vzporednega množenja matrik [2].

Za razvoj praktično uporabnega algoritma pa moramo izbrati enega izmed realnejših modelov vzporednega računanja. Obravnavali bomo vzporedno množenje matrik na modelih SIMD-MC^2, SIMD-CC in SIMD-PS.

3. Množenje matrik na SIMD-MC^2

SIMD-MC^2 vsebuje n^2 procesorjev $P_{i,j}$, $0 \leq i,j \leq n-1$, ki so ciklično povezani v dvodimenzionalno mrežo (Slika 1). Vsak procesor $P_{i,j}$ ima štiri sosede $P_{i\oplus 1,j}$, $P_{i,j\oplus 1}$, $P_{i\ominus 1,j}$ in $P_{i,j\ominus 1}$, kjer smo s $\oplus$ in $\ominus$ označili operaciji seštevanja in odštevanja po modulu n. Vsak $P_{i,j}$ vsebuje tri registre $A_{i,j}$, $B_{i,j}$ ter $C_{i,j}$.



Slika 1

Ob ustreznem ukazu lahko procesorji sočasno pošljejo vsebine izbranega registra v dani smeri svojim sosedom. Na primer, kadar se prenašajo vsebine registrov A v levo (oziroma navzgor), to zapišemo z $A_{i,j} \leftarrow A_{i,j\oplus 1}$ (oziroma $A_{i,j} \leftarrow A_{i\oplus 1,j}$).

3.1 Spodnja časovna meja

Gentleman je v [5] pokazal, da je treba pri analizi vzporednih algoritmov upoštevati tudi operacije prenašanja podatkov med procesorji; analize, ki tega ne upoštevajo, dajejo nerealne rezultate. Računski model, na katerem temelje Gentlemanove ugotovitve, združuje večje število procesorjev, ki imajo lahko tudi lokalne pomnilnike. Vsak procesor lahko neposredno pošlje podatek *omejenemu* številu sosednjih procesorjev. Procesor lahko pošlje podatek v enem koraku le *enemu* sosedu. Gentlemanov računski model je dovolj splošen, da je v njem zajet tudi SIMD-MC^2.

Razpršilna funkcija $\rho(k)$: N $\mapsto$ N naj bo definirana kot največje število procesorjev, ki lahko po k korakih prenašanja podatkov prejmejo podatek iz izbranega začetnega procesorja. Tu seveda predpostavljamo, da procesorji, ki so podatek že prejeli, sodelujejo pri njegovem nadaljnem širjenju v okolico. Funkcija ρ je seveda odvisna od povezovalne sheme procesorjev, torej od modela vzporednega računalnika. Neodvisno od povezovalne sheme pa velja sledeči

Izrek [5] *Na opisanem modelu računanja zahteva množenje dveh matrik reda $n \times n$ vsaj s korakov prenašanja podatkov, tako da je $\rho(2s) \geq n^2$* .

Dokaz. Opazujmo poljuben element $c_{i,j}$; nahaja se v procesorju $P_{i,j}$ in je enak $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k}b_{k,j}$. Vsak izmed faktorjev $a_{i,k}$ in $b_{k,j}$, $0 \leq k \leq n-1$, ki vstopajo v ta izraz, se na začetku nahaja v lastnem procesorju in med računanjem vrednosti $c_{i,j}$ opravi neko pot do procesorja $P_{i,j}$. Naj bo $s(i,j)$ dolžina najdaljše izmed poti, ki jo opravijo ti faktorji pri računanju $c_{i,j}$ in naj bo $s = \max_{i,j} s(i,j)$. Torej je pri računanju produkta matrik potrebnih vsaj s prenosov podatkov. Izberimo dva poljubna procesorja P_x in P_y. Element matrike A, ki se nahaja v procesorju P_x, naj bo $a_{i,j}$. Zanima nas zgornja meja za število potrebnih korakov pri prenašanju tega elementa v procesor P_y. Denimo, da se v P_y nahaja element $b_{u,v}$ matrike B. Tedaj lahko pri prenašanju elementa $a_{i,j}$ uberemo pot preko procesorja P_z, v katerem pričakujemo element $c_{i,v}$ produkta matrik. Takšna pot namreč mora obstajati, saj v nasprotnem ne bi mogli izračunati $c_{i,v}$: prvi del je pot, ki jo pri računanju $c_{i,v}$ opravi $a_{i,j}$ do P_z, drugi del pa pot, ki jo pri tem (v obratni smeri) opravi $b_{u,v}$. Iz definicije števila s sledi, da obe poti skupaj merita kvečjemu $2s$. Sledi, da je za prenos podatka iz enega procesorja v drugi potrebnih kvečjemu $2s$ korakov. Seveda pa mora biti s zadosti velik, da lahko iz danega procesorja podatek pride do kateregakoli izmed n^2 procesorjev. To pomeni, da mora biti $2s$ korakov dovolj za razpršitev podatka v vseh n^2 procesorjev, oziroma $\rho(2s) \geq n^2$. S tem je izrek dokazan. Opazimo, da dokazovanja nismo oprli na nobeno od medprocesorskih povezovalnih shem. □

Za model SIMD-MC^2 je enostavno videti, da je moč v k korakih začetni podatek razpršiti v $\rho(k) = 2k^2 + 2k + 1$ procesorjev. Od tod in iz zgornjega izreka sledi, da je za ta model $s \geq \frac{1}{2}(n-\frac{1}{4})^{\frac{1}{2}} - \frac{1}{4}$. Sledi, da je spodnja časovna meja za množenje dveh matrik reda $n \times n$ na modelu SIMD-MC^2 enaka $\Omega(n)$. Vsak algoritem s časovno kompleksnostjo $\Theta(n)$ je zato s stališča asimptotičnih časovnih kompleksnosti *optimalen na SIMD-MC^2*. Tak algoritem je opisan v nadaljevanju.

3.2 Algoritem

Videli smo, da se na modelu SIMD-MC^2 dveh matrik reda $n \times n$ ne da zmnožiti hitreje kot v linearnem času glede na n. L.E.Cannon je razvil algoritem, ki to stori v času $\Theta(n)$ in je zato časovno optimalen na modelu SIMD-MC^2 [4].

Na začetku velja $A_{i,j} = a_{i,j}$, $B_{i,j} = b_{i,j}$ ter $C_{i,j} = 0$, $0 \leq i,j \leq n-1$. Na koncu pa je $C_{i,j} = c_{i,j} = \sum_{k=0}^{n-1} a_{i,k}b_{k,j}$, $0 \leq i,j \leq n-1$, torej element produkta matrik. Algoritem se glasi:

```
1  begin
2   for k = 1 to n-1 do
3    forall P(i,j) do
4     if i≥k then A(i,j) ← A(i,j⊕1)
5     if j≥k then B(i,j) ← B(i⊕1,j)
6    endforall
7   endfor;
8   forall P(i,j) do
9    C(i,j) := A(i,j)*B(i,j)
10  endforall;
11  for k = 1 to n-1 do
12   forall P(i,j) do
13    A(i,j) ← A(i,j⊕1)
14    B(i,j) ← B(i⊕1,j)
15    C(i,j) := C(i,j)+A(i,j)*B(i,j)
16   endforall
17  endfor
18 end.
```

Tu pomeni znak $\leftarrow$ prenašanje podatkov med registri različnih procesorjev, znak := pa znotraj procesorja.

Algoritem sestavljajo trije deli. V prvem, ki ga sestavljajo vrstice 2...7, se vsebine registrov $A_{i,j}$ in $B_{i,j}$ premestijo: v i-ti vrstici se začetne vsebine registrov A i-krat premestijo ciklično v levo, v j-tem stolpcu pa se začetne vsebine registrov B j-krat ciklično premestijo navzgor. Po izvršitvi prvega dela velja torej za vsak $P_{i,j}$: $A_{i,j} = a_{i,i\oplus j}$ in $B_{i,j} = b_{i\oplus j,j}$. Vsebini registrov $A_{i,j}$ in $B_{i,j}$ se lahko takoj pomnožita, saj se nahajata v istem procesorju $P_{i,j}$, njun produkt pa je eden od členov v končnem $c_{i,j}$. Sočasno množenje v vseh procesorjih opisujejo stavki 8...10, ki tvorijo drugi del algoritma. V tretjem delu algoritma (vrstice 11...17) prištejemo trenutni vsebini registra $C_{i,j}$ še ostalih $n-1$ produktov, ki so členi končnega $c_{i,j}$. V vsakem procesorju dobimo nova faktorja, ki sestavljata takšen produkt, če vsebine vseh registrov A preslikamo enkrat ciklično v levo, vsebine vseh registrov

B pa enkrat ciklično navzgor. Na primer, po k izvršitvah zanke 11 velja za vsak $P_{i,j}$, $0 \le i,j \le n-1$:

$$A_{i,j} = a_{i,i\oplus j\oplus k}$$
$$B_{i,j} = b_{i\oplus j\oplus k,j}$$
$$C_{i,j} = c_{i,j} = \sum_{p=0}^{k} a_{i,i\oplus j\oplus p} b_{i\oplus j\oplus p,j}$$

Po $k = n-1$ ponovitvah zanke 11 se v $C_{i,j}$ nahaja rezultat $c_{i,j}$.

Kakšna je časovna kompleksnost algoritma? Prvi del zahteva $2(n-1)$ operacij sočasnega premikanja vsebin registrov. Drugi del algoritma zahteva eno sočasno množenje po vseh procesorjih. Tretji del pa zahteva $2(n-1)$ operacij sočasnega premikanja vsebin registrov, ter $2(n-1)$ operacij sočasnega množenja/seštevanja. Skupaj je potrebnih $4(n-1)$ operacij sočasnega premikanja vsebin registrov ter $2(n-1)+1$ aritmetičnih operacij. Algoritem ima časovno kompleksnost $\Theta(n)$ in je zato asimptotično časovno optimalen na SIMD-MC^2.

4. Množenje matrik na SIMD-CC

Matriki reda $n \times n$ lahko teoretično zmnožimo v času $\Theta(\log n)$, če se ne oziramo na model vzporednega računanja. Z izborom modela vzporednega računanja, na katerem želimo množiti matriki, pa je potrebno pri snovanju algoritma upoštevati tudi arhitekturne značilnosti modela. Prav te pa lahko zelo poslabšajo časovno kompleksnost algoritma, pa čeprav je ta optimalen na izbranem modelu. Takšen primer smo videli v prejšnji točki, kjer je imel optimalni algoritem na SIMD-MC^2 časovno kompleksnost samo $\Theta(n)$. V tej točki pa bomo opisali model SIMD-CC, ki dovoljuje zasnovo algoritma s časovno kompleksnostjo $\Theta(\log n)$. Model SIMD-CC imenujemo tudi *hiperkocka*.

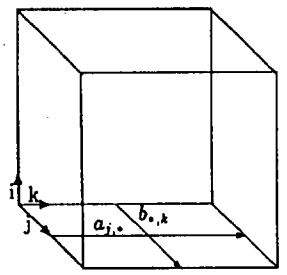
V tej točki naj bo dimenzija matrik $n = 2^q$, za nek $q \in \mathbf{N}$. V modelu SIMD-CC naj bo $p = n^3 = 2^{3q}$ procesorjev $P_0, \ldots, P_{p-1}$. V vsakem procesorju P_r se nahajajo registri A_r, B_r in C_r. Vsak procesor P_r je povezan z $\log(p) = 3\log n = 3q$ procesorji, katerih indeksi so števila $r^{(0)}, r^{(1)}, \ldots, r^{(3q-1)}$, kjer $r^{(b)}$ dobimo tako, da v r invertiramo bit r_b.

Če indeks r, $0 \le r \le p-1$, zapišemo v binarni obliki (s $3q$ biti) in zaporedne q-terke bitov opazujemo kot števila i,j,k, $0 \le i,j,k \le n-1$,

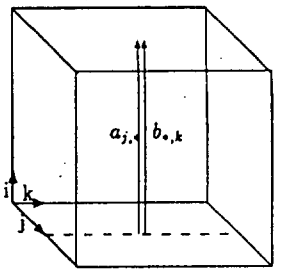
$$r = \underbrace{r_{3q-1}\cdots r_{2q}}_{i}\ \underbrace{r_{2q-1}\cdots r_{q}}_{j}\ \underbrace{r_{q-1}\cdots r_{0}}_{k}$$

lahko procesor P_r označimo tudi s $P_{[i,j,k]}$, pripadajoče registre pa z $A_{[i,j,k]}$, $B_{[i,j,k]}$ in $C_{[i,j,k]}$. Oba zapisa sta ekvivalentna in ju bomo uporabljali izmenično po potrebi. Prednost drugega zapisa je, da si lahko SIMD-CC predstavljamo kot tridimenzionalno polje procesorjev; števila i,j,k povedo, kje v polju se nahaja $P_{[i,j,k]}$. Opozoriti pa moramo, da procesorja, ki sta v tem polju sosednja, v splošnem nista povezana, saj so povezave med procesorji vspostavljene tako, kot je bilo opisano zgoraj.

Na začetku predpostavljamo, da sta matriki A in B vpisani v registre A oziroma B "na dnu" tridimenzionalnega polja, tako da je $A_{[0,j,k]} = a_{j,k}$ ter $B_{[0,j,k]} = b_{j,k}, 0 \le j,k \le n-1$. Zahtevamo, da nam algoritem vrne $C_{[0,j,k]} = c_{j,k} = \sum_{m=0}^{n-1} a_{j,m} b_{m,k}, 0 \le j,k \le n-1$. Opišimo postopek najprej v splošnem. Vektorja $a_{j,*}$ in $b_{*,k}$, katerih skalarni produkt je $c_{j,k}$, sta na dnu polja pravokotna eden na drugega (Slika 2a.). Zato ju "dvignemo" tako, da so komponente $a_{j,i}, b_{i,k}, 0 \le i \le n-1$ paroma v istih procesorjih (Slika 2b.). To storimo sočasno za vse pare vektorjev $a_{j,*}, b_{*,k}, 0 \le j,k \le n-1$. Istoležne komponente sočasno pomnožimo v vseh n procesorjih, ter v vsakem stolpcu seštejemo produkte tako, da se njihove vsote "sesedejo" na dno v $C_{[0,j,k]}$.

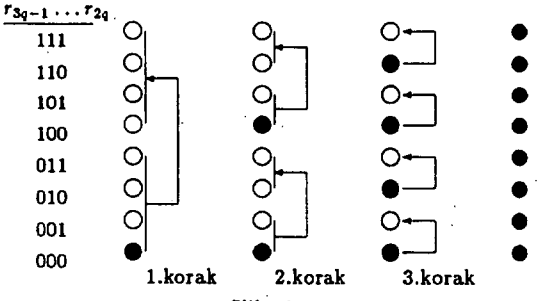


Slika 2a



Slika 2b

Sedaj pa podrobneje opišimo algoritem. V prvem delu preslikamo obe matriki z dna še na vse ostale nivoje tridimenzionalnega polja, tako da na koncu dobimo $A_{[i,j,k]} = a_{j,k}$ ter $B_{[i,j,k]} = b_{j,k}$, za vse nivoje i, $0 \le i \le n-1$. Na prvi pogled kaže, da je za to potrebnih $\Theta(n)$ korakov, saj je vsak procesor $P_{[0,j,k]}$ navzgor neposredno povezan le s $q-1 = \log(n)-1$ procesorji $P_{[2^m,j,k]}, 1 \le m \le q-1$, poleg tega pa lahko v enem koraku procesor pošlje podatek le enemu sosedu. Toda v SIMD-CC so procesorji med seboj tako povezani, da je možno v vsakem koraku podvojiti število tistih procesorjev v stolpcu, ki že imajo začetni podatek. Zato preslikavanje zahteva $q = \Theta(\log n)$ korakov. Za $n = 8$ je preslikavanje v enem stolpcu ilustrirano na sliki 3.



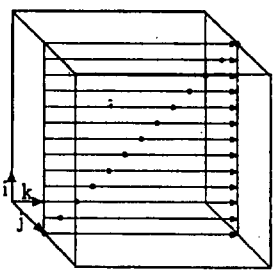
Slika 3

V splošnem pa za to poskrbijo stavki 1...8:

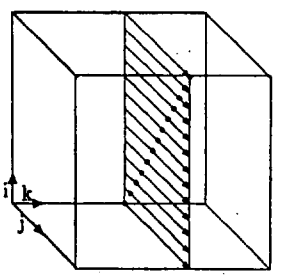
```
1  for m = 3q − 1 downto 2q do
2    forall P_r, 0 ≤ r ≤ p − 1 do
3      if r_m = 0 then A_{r(m)} ← A_r
4    endforall;
5    forall P_r, 0 ≤ r ≤ p − 1 do
6      if r_m = 0 then B_{r(m)} ← B_r
7    endforall
8  endfor;
```

Na koncu velja $A_{[i,j,k]} = a_{j,k}$ ter $B_{[i,j,k]} = b_{j,k}, 0 \le i,j,k \le n-1$.

Pri $k = i$ dobimo $A_{[i,j,i]} = a_{j,i}$; ti elementi se nahajajo na diagonali, kot kaže Slika 4a. Podobno dobimo $B_{[i,i,k]} = b_{i,k}$, če vzamemo $j = i$ (Slika 4b.). Vsebino registra $A_{[i,j,i]}$ prenesemo v vse ostale registre $A_{[i,j,k]}$, $0 \le k \le n-1$ (Slika 4a.). Vsebino registra $B_{[i,i,k]}$ pa prenesemo v vse registre $B_{[i,j,k]}$, $0 \le j \le n-1$ (Slika 4b.).



Slika 4a



Slika 4b

Za to poskrbijo stavki 9...12 oziroma 13...16:

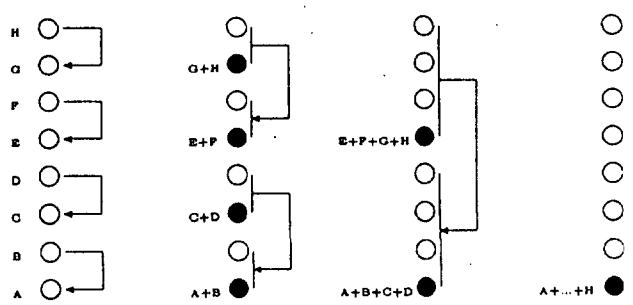
```
9  for m = q − 1 downto 0 do
10   forall P_r, 0 ≤ r ≤ p − 1 do
11     if r_m = r_{2q+m} then A_{r(m)} ←A_r
12   endforall;
13 for m = 2q − 1 downto q do
14   forall P_r, 0 ≤ r ≤ p − 1 do
15     if r_m = r_{q+m} then B_{r(m)} ←B_r
16   endforall;
```

Na koncu je $A_{[i,j,k]} = a_{j,i}$ ter $B_{[i,j,k]} = b_{i,k}$, $0 \le i,j,k \le n-1$, (Slika 2b).

Sedaj pa lahko sočasno pomnožimo vsebine registrov $A_{[i,j,k]}$ ter $B_{[i,j,k]}$ v vseh p procesorjih, kot to opisujejo stavki 17...19:

```
17 forall P_r, 0 ≤ r ≤ p − 1 do
18   C_r := A_r*B_r
19 endforall;
```

V zanjem koraku moramo sešteti produkte, ki se nahajajo v stolpcih, tako da se njihove vsote $c_{j,k}$ pojavijo na dnu tridimenzionalnega polja. Postopek seštevanja v enem stolpcu je za $n = 8$ ilustriran na Sliki 5, v splošnem pa je opisan s stavki 20...24.



Slika 5

```
20 for m = 2q to 3q − 1 do
21   forall P_r, 0 ≤ r ≤ p − 1 do
22     if r_m = 0 then C_r ← C_r+C_{r(m)}
23   endforall
24 endfor;
```

Časovno kompleksnost algoritma sestavlja $5q = 5\log(n)$ korakov sočasnega prenašanja podatkov med procesorji, eno sočasno množenje in $\log(n)$ sočasnih seštevanj. Opisani algoritem ima torej časovno kompleksnost reda $\Theta(\log n)$.

5. Množenje matrik na SIMD-PS

Naj bo dimenzija matrik enaka $n = 2^q$, za nek $q \in \mathbf{N}$. V modelu SIMD-PS naj bo $p = n^3 = 2^{3q}$ procesorjev $P_0, \ldots, P_{p-1}$. V vsakem procesorju P_r se nahajajo registri A_r, B_r in C_r. Vsak procesor P_r je povezan s tremi sosednjimi procesorji $P_{sh(r)}$, $P_{ex(r)}$ in $P_{un(r)}$, kjer so indeksi $sh(r)$, $ex(r)$ ter $un(r)$ števila, katerih binarni zapis dobimo iz binarno zapisanega indeksa

$$r = r_{3q-1}r_{3q-2}\ldots r_{2q}r_{2q-1}\ldots r_q r_{q-1}\ldots r_1 r_0$$

na naslednji način:

$$sh(r) = r_{3q-2}\ldots r_{2q}r_{2q-1}\ldots r_q r_{q-1}\ldots r_1 r_0 r_{3q-1}$$

$$ex(r) = r_{3q-1}r_{3q-2}\ldots r_{2q}r_{2q-1}\ldots r_q r_{q-1}\ldots r_1 \bar{r}_0$$

$$un(r) = r_0 r_{3q-1}r_{3q-2}\ldots r_{2q}r_{2q-1}\ldots r_q r_{q-1}\ldots r_1$$

Tu je $\bar{r}_0$ komplement bita r_0. Funkcije sh, un in ex se imenujejo "shuffle", "unshuffle" ter "exchange" in izvršijo ciklično rotacijo bitov argumenta v levo, desno oziroma komplementiranje zadnjega bita.

Podobno kot pri modelu SIMD-CC bomo tudi tu uporabljali za indekse izmenično oznaki r in $[x,y,z]$, kjer je $0 \le r \le p-1$, $0 \le x,y,z \le n-1$ ter $r = 2^{2q}x + 2^q y + z$. Prednost drugega zapisa je, da si lahko SIMD-PS predstavljamo kot tridimenzionalno polje procesorjev; števila x,y,z povedo, kje v polju se nahaja $P_{[x,y,z]}$. Zopet pa moramo opozoriti, da procesorja, ki sta v tem polju sosednja, v splošnem nista povezana, saj so medprocesorske povezave vzpostavljene tako, kot je bilo opisano zgoraj.

Sedaj pa opišimo algoritem za množenje matrik na modelu SIMD-PS. Najprej preslikamo matriki A in B v registre A oziroma B na dno polja: $A_{[0,i,t]} := a_{i,t}$ in $B_{[0,i,t]} := b_{i,t}$, $0 \le i,t \le n-1$. Od tu matriki preslikamo še na vseh ostalih $n-1$ vzporednih plasti. Slednje opisujejo stavki 1...8:

```
1 for b = 0 to q − 1 do
2   forall P_r, 0 ≤ r ≤ p − 1 do
3     A_{sh(r)} ←A_r;
4     B_{sh(r)} ←B_r;
5     if r_0 = 0 then A_{ex(r)} ←A_r;
6     if r_0 = 0 then B_{ex(r)} ←B_r;
7   endforall
8 endfor;
```

Trditev: *Po izvršitvi stavkov* 1...8 *velja* $A_{[i,t,s]} = a_{i,t}$ *in* $B_{[i,t,s]} = b_{i,t}$, $0 \le i,t,s \le n-1$.

Dokaz: Register $A_{[0,i,t]}$ z vsebino $a_{i,t}$ se nahaja v procesorju P_r, kjer je $r = \underbrace{00\ldots0}_{0}\underbrace{r_{2q-1}\ldots r_q}_{i}\underbrace{r_{q-1}\ldots r_0}_{t}$.

V prvi izvedbi zanke 1 se vsebina $a_{i,t}$ registra $A_{[0,i,t]}$ z ukazom 3 preslika v $A_{sh(r)}$, kjer $sh(r) = 0\ldots0r_{2q-1}\ldots r_q r_{q-1}\ldots r_0 0$, z ukazom 5 pa iz tega registra še v register $A_{ex(sh(r))}$, kjer je $ex(sh(r)) = 0\ldots0r_{2q-1}\ldots r_q r_{q-1}\ldots r_0 1$. Predpostavimo, da so po b ponovitvah zanke 1 vrednosti $a_{i,t}$ v vseh $s = 2^b$ registrih A_r, kjer je $r = \underbrace{00\ldots0}_{0}\underbrace{r_{2q-1}\ldots r_q}_{i}\underbrace{r_{q-1}\ldots r_0}_{t}\underbrace{r_{3q-1}\ldots r_{3q-1-(b-1)}}_{j}$ in $0 \le j \le 2^b - 1$. Po naslednji ponovitvi zanke je $a_{i,t}$ zaradi 3 in 5 v vseh $s = 2^{b+1}$ registrih A_r, kjer je $r = \underbrace{00\ldots0}_{0}\underbrace{r_{2q-1}\ldots r_q}_{i}\underbrace{r_{q-1}\ldots r_0}_{t}\underbrace{r_{3q-1}\ldots r_{3q-1-b}}_{j}$ in $0 \le j \le 2^{b+1} - 1$. Ker je $n = 2^q$, se po q ponovitvah zanke 1 $a_{i,t}$ nahaja v vseh n registrih $A_{[i,t,s]}$, kjer $0 \le s \le n-1$. Ker je bil par i,t poljuben, je s tem prvi del trditve dokazan. Dokaz za registre B je podoben. □

V naslednjem koraku premestimo vsebine registrov A:

```
9  for b = 0 to q − 1 do
10   forall P_r, 0 ≤ r ≤ p − 1 do
11     A_{sh(r)} ←A_r
12   endforall
13 endfor;
```

Trditev: *Po izvršitvi vrstic* 9...13 *je* $A_{[i,t,s]} = a_{s,i}, 0 \le i,t,s \le n-1$.

Dokaz: Pred izvršitvijo teh vrstic je $A_{[s,i,t]} = a_{s,i}, 0 \le s,i,t \le n-1$. Po q izvršitvah stavka 11 se $a_{s,i}$ iz $A_{[s,i,t]}$ prenese v $A_{[i,t,s]}$, saj velja $sh^q([s,i,t]) = [i,t,s]$. □

Po izvršitvi stavkov 1...13 sta v registrih $A_{[i,t,s]}$ in $B_{[i,t,s]}$ procesorja $P_{[i,t,s]}$ vrednosti $a_{s,i}$ in $b_{i,t}$. Njun produkt $a_{s,i}b_{i,t}$ je eden od n členov v končnem $c_{s,t}$, zato ju je smiselno pomnožiti. To seveda velja za vse procesorje P_r, $0 \le r \le p-1$, saj lahko vsak r zapišemo v obliki $r = [i,t,s]$, kjer $0 \le i,t,s \le n-1$. Sočasno

množenje v vseh procesorjih opisujejo stavki 14 ... 16:

```
14 forall P_r, 0 ≤ r ≤ p − 1 do
15   C_r := A_r*B_r
16 endforall;
```

Rezultat stavkov 14 ... 16 je $C_{[i,t,s]} = a_{s,i}b_{i,t}$. Ker želimo na koncu dobiti $c_{s,t} = \sum_{i=0}^{n-1} a_{s,i}b_{i,t}$, moramo za vsak par s,t sešteti vsebine vseh registrov $C_{[i,t,s]}, 0 \le i \le n-1$:

```
17 for b = 0 to q − 1 do
18   forall P_r, 0 ≤ r ≤ p − 1 do
19     C_sh(r) ← C_r;
20     C_r ← C_r + C_ez(r)
21   endforall
22 endfor;
```

Trditev: *Po izvršitvi stavkov* 17...22 *velja* $C_{[t,s,i]} = c_{s,t} = \sum_{k=0}^{n-1} a_{s,k}b_{k,t}$ *za vse* t,s,i, $0 \le t,s,i \le n-1$.

Dokaz: Izberimo poljuben par števil t,s, $0 \le t,s \le n-1$ in pišimo $f_i = a_{s,i}b_{i,t}$. Vsebina registra $C_{[i,t,s]}$ je f_i. Naj bosta τ in σ binarna zapisa števil t in s, torej $\tau,\sigma \in \{0,1\}^q$. Naj pomeni $\sum_{j_1\dots j_b}$ seštevanje po vseh $j_1\dots j_b \in \{0,1\}^b$. Naprimer, $\sum_{j_1 j_2}$ pomeni seštevanje, kjer indeks preteče binarna zaporedja 00, 01, 10 in 11.

Tedaj opazimo, da po prvi izvršitvi zanke 17 velja za vse $i_{q-1} \in \{0,1\}$:

$$C_{i_{q-2}\dots i_0\tau\sigma i_{q-1}} = \sum_{j_1} f_{j_1 i_{q-2}\dots i_0}.$$

Naj po b ponovitvah zanke 17 velja za vse $i_{q-1}\dots i_{q-1-(b-1)} \in \{0,1\}^b$:

$$C_{i_{q-1-b}\dots i_0\tau\sigma i_{q-1}\dots i_{q-1-(b-1)}} = \sum_{j_1\dots j_b} f_{j_1\dots j_b i_{q-1-b}\dots i_0}. \quad (*)$$

Po vnovični izvršitvi stavka 19 je za vse $i_{q-1}\dots i_{q-1-(b-1)} \in \{0,1\}^b$:

$$C_{i_{q-1-(b+1)}\dots i_0\tau\sigma i_{q-1}\dots i_{q-1-(b-1)}0} = \sum_{j_1\dots j_b} f_{j_1\dots j_b 0 i_{q-1-(b+1)}\dots i_0}$$

in

$$C_{i_{q-1-(b+1)}\dots i_0\tau\sigma i_{q-1}\dots i_{q-1-(b-1)}1} = \sum_{j_1\dots j_b} f_{j_1\dots j_b 1 i_{q-1-(b+1)}\dots i_0}.$$

S stavkom 20 se obe vsoti seštejeta in priredita obema registroma, zato velja za vse $i_{q-1}\dots i_{q-1-b} \in \{0,1\}^{b+1}$:

$$C_{i_{q-1-(b+1)}\dots i_0\tau\sigma i_{q-1}\dots i_{q-1-b}} = \sum_{j_1\dots j_{b+1}} f_{j_1\dots j_{b+1} i_{q-1-(b+1)}\dots i_0}.$$

Torej velja enakost $(*)$ za vse $b, 0 \le b \le q-1$. Posebej za $b = q-1$ je

$$C_{\tau\sigma i_{q-1}\dots i_0} = \sum_{j_1\dots j_q} f_{j_1\dots j_q} = \sum_{j=0}^{n-1} f_j$$

za vse $i_{q-1}\dots i_0 \in \{0,1\}^q$, kar je isto kot $C_{[t,s,i]} = \sum_{j=0}^{n-1} a_{s,j}b_{j,t}$, $0 \le i \le n-1$. □

S tem je produkt matrik izračunan: element $c_{s,t}$ se nahaja v vseh n registrih $C_{[t,s,i]}$, $0 \le i \le n-1$.

Navadno pa želimo, da se rezultati nahajajo v registrih C tistih procesorjev, v katere sta bili na začetku vpisani matriki A in B. Potemtakem želimo, da se elementi $c_{s,t}$ pojavijo "na dnu" tridimenzionalnega polja: $C_{[0,s,t]} = c_{s,t}$, $0 \le s,t \le n-1$. Ta popravek izvršimo v dveh korakih.

Najprej opazimo, da po izvršitvi stavkov 1 ... 22 v posebnem primeru velja $C_{[t,s,t]} = c_{s,t}, \le 0 \le s,t \le n-1$. Vsebine teh registrov prenesemo v registre $C_{[s,t,0]}$:

```
23 for b = 0 to q − 1 do
24   forall P_r, 0 ≤ r ≤ p − 1 do
25     C_sh(r) ←C_r;
26     if r_0 = r_q = 1 then C_ez(r) ←C_r
27   endforall
28 endfor;
```

Trditev: *Po izvršitvi* 23 ... 28 *velja* $C_{[s,t,0]} = c_{s,t}$, *za vse* $0 \le s,t \le n-1$.

Dokaz: Zapišimo indeks $[t,s,t]$ binarno: $t_{q-1}\dots t_0 s_{q-1}\dots$ $\dots s_0 t_{q-1}\dots t_0$. Po prvi izvršitvi stavka 25 se $c_{s,t}$ prenese v register C z binarno zapisanim indeksom $t_{q-2}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 t_{q-1}$. Pri stavku 26 je za ta indeks $r_0 = r_q = t_{q-1}$. Če je $t_{q-1} = 1$, se $c_{s,t}$ prenese še v register C z indeksom $t_{q-2}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0$. Torej se $c_{s,t}$ gotovo nahaja v registru C, katerega binarno zapisani indeks je $t_{q-2}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0$. Predpostavimo, da je po b izvršitvah zanke 23 vrednost $c_{s,t}$ v registru C z binarno zapisanim indeksom $t_{q-1-b}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0\dots 0$. Ob naslednji ponovitvi zanke se s stavkom 25 vrednost $c_{s,t}$ prenese v register C z binarnim indeksom $t_{q-1-(b+1)}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0\dots 0 t_{q-1-b}$. Za ta indeks je $r_0 = r_q = t_{q-1-b}$. Če je $t_{q-1-b} = 1$ se vrednost $c_{s,t}$ s stavkom 26 prenese še v register C z binarnim indeksom $t_{q-1-(b+1)}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0\dots 0\bar{t}_{q-1-b}$. Vidimo, da se po q ponovitvah zanke 23 vrednost $c_{s,t}$ nahaja v registru C z binarnim indeksom $s_{q-1}\dots s_0 t_{q-1}\dots t_0 0\dots 0$, torej v registru $C_{[s,t,0]}$. Opisano velja za vsak $0 \le s,t \le n-1$.□

Končno vsebine registrov $C_{[s,t,0]}$ prenesemo v registre $C_{[0,s,t]}$:

```
29 for b := 0 to q − 1 do
30   forall P_r, 0 ≤ r ≤ p − 1 do
31     C_un(r) ← C_r
32   endforall
33 endfor.
```

Kakšna je časovna kompleksnost opisanega algoritma ? Vrstice 1 ... 33 zahtevajo $10q = 10\log(n)$ operacij sočasnega premeščanja vsebin registrov, eno sočasno množenje v vseh procesorjih ter $q = \log(n)$ sočasnih seštevanj. Opazimo, da je algoritem približno dvakrat počasnejši od algoritma na modelu SIMD-CC ter približno desetkrat počasnejši od teoretično najhitrejšega. Če pa se zadovoljimo z okrnjeno inačico algoritma (vrstice 1 ... 22), pa je potrebnih $7\log(n)$ sočasnih premeščanj ob enakem številu seštevanj in množenj. Asimptotična časovna kompleksnost je v obeh primerih $\Theta(\log n)$, torej je algoritem asimptotično časovno optimalen.

6. Zaključek

Opisani so bili nekateri algoritmi za vzporedno množenje kvadratnih matrik. Najprej je bil opisan postopek, katerega časovna kompleksnost je $\lceil \log n \rceil + 1$, kjer je n dimenzija matrik. Ta algoritem se ne ozira na nobenega od realnih modelov vzporednega računanja - njegov pomen je v določitvi spodnje meje za časovno kompleksnost vzporednega množenja matrik. Če pa izberemo nek računski model, se lahko časovna kompleksnost optimalnega algoritma (na izbranem modelu) poslabša. Pri modelu SIMD-MC^2 časovna kompleksnost optimalnega algoritma poskoči na $\Theta(n)$. Model, ki dovoljuje razvoj asimptotično absolutno optimalnega algoritma (s časovno kompleksnostjo $\Theta(\log n)$) je SIMD-CC (hiperkocka). Slabosti hiperkocke pa se pokažejo pri njeni VLSI implementaciji, saj število medprocesorskih povezav prehitro narašča. Teh slabosti nima model SIMD-PS, pri katerem je vsak procesor povezan vedno

le s tremi sosedi. Tudi ta model dovoljuje zasnovo algoritma za množenje matrik, ki je sicer približno dvakrat počasnejši, a seveda še vedno asimptotično časovno optimalen.

V prispevku nismo uporabljali pojma *cene* vzporednega algoritma, tj. produkta med številom procesorjev in časom izvrševanja, čeprav je cena primernejše merilo za primerjavo algoritmov. Iz opisanega je cene moč enostavno določiti.

Literatura

[1] S.G.Akl, *The Design and Analysis of Parallel Algorithms*, Prentice-Hall Int'l Editions, 1989.

[2] A.Borodin, I.Munro, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier Publishing Comp., 1975.

[3] D.Coppersmith, S.Winograd, *On the asymptotic complexity of matrix multiplication*, SIAM J.Comput., Vol.11, No.3, 1982, str.472-492.

[4] E.Dekel, D.Nassimi, S.Sahni, *Parallel matrix and graph algorithms*, SIAM J.Comput., Vol.10, No.4, 1981, str.657-675.

[5] W.M.Gentleman, *Some complexity results for matrix computations on parallel processors*, J.ACM, Vol.25, No.1, 1978, str.112-115.