ARHITECTURA SISTEMELOR DE CALCUL SEMINAR 0x05

NOTIȚE SUPORT SEMINAR

Cristian Rusu

TIMPI CACHING, EX. 1

a) timpul total de acces memorie este (din curs)

TIMPI CACHING, EX. 1

a) **timpul total de acces memorie este (din curs)**

$$1 \text{ ns} + (0.1 \times (5 \text{ ns} + (0.01 \times (10 \text{ ns} + (0.002 \times 50 \text{ ns})))))$$

TIMPI CACHING, EX. 1

a) **timpul total de acces memorie este (din curs)**

$1 \text{ ns} + (0.1 \times (5 \text{ ns} + (0.01 \times (10 \text{ ns} + (0.002 \times 50 \text{ ns})))))$

în cazul nostru

$1 \text{ ns} + (A \times (5 \text{ ns} + (0.01 \times (10 \text{ ns} + (0.002 \times 50 \text{ ns}))))) = t_{RAM} / 2$

TIMPI CACHING, EX. 1

a) **timpul total de acces memorie este (din curs)**

$1 \text{ ns} + (0.1 \times (5 \text{ ns} + (0.01 \times (10 \text{ ns} + (0.002 \times 50 \text{ ns})))))$

în cazul nostru

$1 \text{ ns} + (A \times (5 \text{ ns} + (0.01 \times (10 \text{ ns} + (0.002 \times 50 \text{ ns}))))) = t_{RAM} / 2$

b) $\mathbf{1 \text{ ns} + (0.1 \times (A \text{ ns} + (0.01 \times (10 \text{ ns} + (0.002 \times 50 \text{ ns}))))) = t_{RAM} / 10}$

TIMPI CACHING, EX. 1

a) **timpul total de acces memorie este (din curs)**

   1 ns + (0.1 x (5 ns + (0.01 x (10 ns + (0.002 x 50 ns)))))

   în cazul nostru

   1 ns + (A x (5 ns + (0.01 x (10 ns + (0.002 x 50 ns))))) = t_{RAM} / 2

b) **1 ns + (0.1 x (A ns + (0.01 x (10 ns + (0.002 x 50 ns))))) = t_{RAM} / 10**

c) **1 ns + (0.1 x (5 ns + (0.01 x (10 ns + (A x 50 ns))))) = t_{RAM}**

TIMPI CACHING, EX. 1

a) **timpul total de acces memorie este (din curs)**

1 ns + (0.1 x (5 ns + (0.01 x (10 ns + (0.002 x 50 ns)))))

în cazul nostru

1 ns + (A x (5 ns + (0.01 x (10 ns + (0.002 x 50 ns))))) = t_{RAM} / 2

b) **1 ns + (0.1 x (A ns + (0.01 x (10 ns + (0.002 x 50 ns))))) = t_{RAM} / 10**

c) **1 ns + (0.1 x (5 ns + (0.01 x (10 ns + (A x 50 ns))))) = t_{RAM}**

d) pentru L1, să trecem de la 1ns la 0.9ns ne costă 100$

pentru L2, să trecem de la 5ns la 4.5ns ne costă 25$

pentru L3, să trecem de la 10ns la 9ns ne costă 10$

A ns + (0.1 x (B ns + (0.01 x (C ns + (0.002 x 50 ns))))) = t_{RAM}/1000

vrem: minimize 10 A + 2.5 B + C

rezolvați pentru A, B și C

ÎNTREBĂRI SCURTE, EX. 2

a)

ÎNTREBĂRI SCURTE, EX. 2

a) utilizare CPU (eventual sisteme multi-core), media artimetică

b)

ÎNTREBĂRI SCURTE, EX. 2

a) utilizare CPU (eventual sisteme multi-core), media artimetică

b) wall-clock time, media aritmetică

c)

.

ÎNTREBĂRI SCURTE, EX. 2

a) utilizare CPU (eventual sisteme multi-core), media artimetică

b) wall-clock time, media aritmetică

c) memoria RAM, maximum

d)

ÎNTREBĂRI SCURTE, EX. 2

a) utilizare CPU (eventual sisteme multi-core), media artimetică

b) wall-clock time, media aritmetică

c) memoria RAM, maximum

d) performanță per Watt, media artimetică sau maximum

e)

ÎNTREBĂRI SCURTE, EX. 2

a) utilizare CPU (eventual sisteme multi-core), media artimetică

b) wall-clock time, media aritmetică

c) memoria RAM, maximum

d) performanță per Watt, media artimetică sau maximum

e) wall-clock time, 50/90/99th percentile mediana

f)

ÎNTREBĂRI SCURTE, EX. 2

a) utilizare CPU (eventual sisteme multi-core), media artimetică

b) wall-clock time, media aritmetică

c) memoria RAM, maximum

d) performanță per Watt, media artimetică sau maximum

e) wall-clock time, 50/90/99th percentile mediana

f) wall-clock time speedup, media aritmetică

g)

ÎNTREBĂRI SCURTE, EX. 2

a) utilizare CPU (eventual sisteme multi-core), media artimetică

b) wall-clock time, media aritmetică

c) memoria RAM, maximum

d) performanță per Watt, media artimetică sau maximum

e) wall-clock time, 50/90/99th percentile mediana

f) wall-clock time speedup, media aritmetică

g) minimum = când zgomotul/erorile din sistem sunt minime (*best-case behavior*), maximum = când zgomotul/erorile din sistem sunt maxime (*worst-case behavior*)

h)

ÎNTREBĂRI SCURTE, EX. 2

a) utilizare CPU (eventual sisteme multi-core), media artimetică

b) wall-clock time, media aritmetică

c) memoria RAM, maximum

d) performanță per Watt, media artimetică sau maximum

e) wall-clock time, 50/90/99th percentile mediana

f) wall-clock time speedup, media aritmetică

g) minimum = când zgomotul/erorile din sistem sunt minime (*best-case behavior*), maximum = când zgomotul/erorile din sistem sunt maxime (*worst-case behavior*)

h) dacă măsurăm cât mai bine și exact fiecare componentă, putem optimiza cât mai bine (semnificativ)

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	
1	9	3	
2	8	2	
3	2	20	
4	10	2	

.

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B
1	9	3	3
2	8	2	4
3	2	20	0.1
4	10	2	5
Media	7.25	6.75	3.025

Concluzia:

.

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B
1	9	3	3
2	8	2	4
3	2	20	0.1
4	10	2	5
Media	7.25	6.75	3.025

Concluzia: Program B este de 3 ori mai rapid decât Program A

.

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B	B/A
1	9	3	3	
2	8	2	4	
3	2	20	0.1	
4	10	2	5	
Media	7.25	6.75	3.025	

Concluzia: Program B este de 3 ori mai rapid decât Program A

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B	B/A
1	9	3	3	0.33
2	8	2	4	0.25
3	2	20	0.1	10
4	10	2	5	0.2
Media	7.25	6.75	3.025	2.7

Concluzia: Program B este de 3 ori mai rapid decât Program A
Concluzia:

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B	B/A
1	9	3	3	0.33
2	8	2	4	0.25
3	2	20	0.1	10
4	10	2	5	0.2
Media	7.25	6.75	3.025	2.7

Concluzia: Program B este de 3 ori mai rapid decât Program A
Concluzia: Program A este de 2.7 ori mai rapid decât Program B

Care e problema?

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B	B/A
1	9	3	3	0.33
2	8	2	4	0.25
3	2	20	0.1	10
4	10	2	5	0.2
Media	7.25	6.75	3.025	2.7

Concluzia: Program B este de 3 ori mai rapid decât Program A
Concluzia: Program A este de 2.7 ori mai rapid decât Program B

Nu luați media aritmetică a rapoartelor A/B sau B/A

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B	B/A
1	9	3	3	0.33
2	8	2	4	0.25
3	2	20	0.1	10
4	10	2	5	0.2
Media	(a) 7.25	(a) 6.75	(g) 1.57	(g) 0.64

Concluzia: Program B este de 3 ori mai rapid decât Program A
Concluzia: Program A este de 2.7 ori mai rapid decât Program B

Nu luați media aritmetică a rapoartelor A/B sau B/A
Luați media geometrică a rapoartelor A/B sau B/A

- în acest caz, media rapoartelor este raportul medilor

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B	B/A
1	9	3	3	0.33
2	8	2	4	0.25
3	2	20	0.1	10
4	10	2	5	0.2
Media	(a) 7.25	(a) 6.75	(g) 1.57	(g) 0.64

Vrem să comparăm Program A vs. Program B: cine este mai rapid? A sau B?

TIMPI DE RULARE, EX. 3

Test	Program A	Program B	A/B	B/A
1	9	3	3	0.33
2	8	2	4	0.25
3	2	20	0.1	10
4	10	2	5	0.2
Media	(a) 7.25	(a) 6.75	(g) 1.57	(g) 0.64

Vrem să comparăm Program A vs. Program B: cine este mai rapid? A sau B?

- rulăm programele de mai multe ori
- comparăm linie cu linie în tabelul de mai sus
- pentru fiecare linie decidem cine câștigă (A sau B)
- apoi calculăm: care este probabilitatea ca A să fie mai rapid decât B dacă am observat că în n cazuri (din totalul de N) A este mai rapid decât B
- p-value

https://en.wikipedia.org/wiki/P-value

ÎNMULȚIRE COMPLEXĂ, EX. 4

a)

ÎNMULȚIRE COMPLEXĂ, EX. 4

a) $z = (a+bi)x(c+di) = ac - bd + i(ad + bc)$

b)

.

ÎNMULȚIRE COMPLEXĂ, EX. 4

a) $z = (a+bi)x(c+di) = ac - bd + i(ad + bc)$

b) 2 adunări, 4 înmulțiri

c)

.

ÎNMULȚIRE COMPLEXĂ, EX. 4

a) $z = (a+bi)x(c+di) = ac - bd + i(ad + bc)$

b) 2 adunări, 4 înmulțiri

c) calculăm $S1 = ac$, $S2 = bd$ și $S3 = (a+b)x(c+d)$

$z = S1-S2 + i(S3-S1-S2)$

ÎNMULȚIRE COMPLEXĂ, EX. 4

a) $z = (a+bi)x(c+di) = ac - bd + i(ad + bc)$

b) 2 adunări, 4 înmulțiri

c) calculăm $S1 = ac$, $S2 = bd$ și $S3 = (a+b)x(c+d)$

$z = S1-S2 + i(S3-S1-S2)$

5 adunări, 3 înmulțiri

d)

ÎNMULȚIRE COMPLEXĂ, EX. 4

a) $z = (a+bi)x(c+di) = ac - bd + i(ad + bc)$

b) 2 adunări, 4 înmulțiri

c) calculăm $S1 = ac$, $S2 = bd$ și $S3 = (a+b)x(c+d)$

$z = S1-S2 + i(S3-S1-S2)$

5 adunări, 3 înmulțiri

d) C1 – costul unei adunări

C2 – costul unei înmulțiri

ÎNMULȚIRE COMPLEXĂ, EX. 4

a) z = $(a+bi)x(c+di) = ac - bd + i(ad + bc)$

b) 2 adunări, 4 înmulțiri

c) calculăm $S1 = ac$, $S2 = bd$ și $S3 = (a+b)x(c+d)$

$z = S1-S2 + i(S3-S1-S2)$

5 adunări, 3 înmulțiri

d) C1 – costul unei adunări

C2 – costul unei înmulțiri

$2C1 + 4C2 > 5C1 + 3C2$

$C2/C1 > 3$

ÎNMULȚIRE MATRICE, EX. 4

e) algoritmul lui Strassen

- vrem să calculăm C = AB (unde A și B sunt matrice)

```
for (i = 0; i < row_length_A; i++)
{
    for (k = 0; k < column_length_B; k++)
    {
        sum = 0;
        for (j = 0; j < column_length_A; j++)
        {
            sum += A[i][j] * B[j][k];
        }
        C[i][k] = sum;
    }
}
```

- pe blocuri:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\
\mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\
\mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\
\mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}
\end{aligned}$$

$$\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}$$

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}$$

ÎNMULȚIRE MATRICE, EX. 4

e) algoritmul lui Strassen

- vrem să calculăm C = AB (unde A și B sunt matrice)

```
for (i = 0; i < row_length_A; i++)
{
    for (k = 0; k < column_length_B; k++)
    {
        sum = 0;
        for (j = 0; j < column_length_A; j++)
        {
            sum += A[i][j] * B[j][k];
        }
        C[i][k] = sum;
    }
}
```

- pe blocuri:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\
\mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\
\mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\
\mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}
\end{aligned}$$

$O(n^3)$

$$\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}$$

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}$$

$O(n^{2.8})$

LUCRU CU VECTOR/MATRICE, EX. 5

- produs scalar

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	…	x_9

y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	…	y_9

- cum calculăm eficient?

LUCRU CU VECTOR/MATRICE, EX. 5

- **produs scalar**

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	…	x_9
y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	…	y_9

- **cum calculăm eficient?**
  - $c \mathrel{+}= x_i\, y_i$
  - instrucțiune Fast Multiply-Add (fma)
  - aceeași operație pe date diferite (SIMD)
  - foarte ușor de paralelizat
    - atenție, rezultatul va fi diferit (adunarea nu mai e asociativă)
    - cu *p* procesoare ne așteptăm să fim de *p* ori mai rapizi
  - exploatează cache la maxim: datele sunt continue în memorie

LUCRU CU VECTOR/MATRICE, EX. 5

- **produs matrice-vector**

$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \\ -1 & 6 \end{pmatrix} \times \begin{pmatrix} 4 \\ 7 \end{pmatrix} = \begin{pmatrix} (2 * 4) + (3 * 7) \\ (4 * 4) + (5 * 7) \\ (-1 * 4) + (6 * 7) \end{pmatrix} = \begin{pmatrix} 29 \\ 51 \\ 38 \end{pmatrix}$$

- **cum calculăm eficient?**

https://mercris.wordpress.com/2012/06/29/linear-algebra-cheat-sheet-for-machine-learning/

LUCRU CU VECTOR/MATRICE, EX. 5

- **produs matrice-vector**

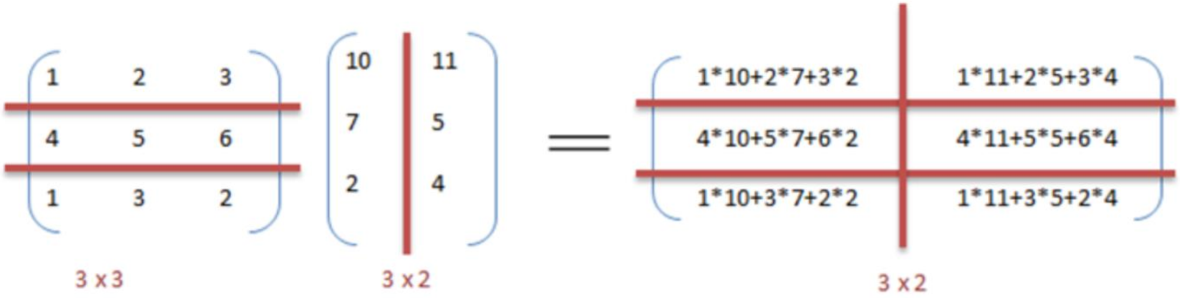
$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \\ -1 & 6 \end{pmatrix} \times \begin{pmatrix} 4 \\ 7 \end{pmatrix} = \begin{pmatrix} (2 * 4) + (3 * 7) \\ (4 * 4) + (5 * 7) \\ (-1 * 4) + (6 * 7) \end{pmatrix} = \begin{pmatrix} 29 \\ 51 \\ 38 \end{pmatrix}$$

- **cum calculăm eficient?**
  - $c \mathrel{+}= x_i \, y_i$
  - *n* produse scalare (se pot realiza în paralel)
  - cum exploatăm eficient cache-ul?
    - dacă putem forța cache-ul să țină vectorul atunci cache miss rate va fi cu siguranță <= 50%

https://mercris.wordpress.com/2012/06/29/linear-algebra-cheat-sheet-for-machine-learning/

LUCRU CU VECTOR/MATRICE, EX. 5

- **produs matrice-matrice**



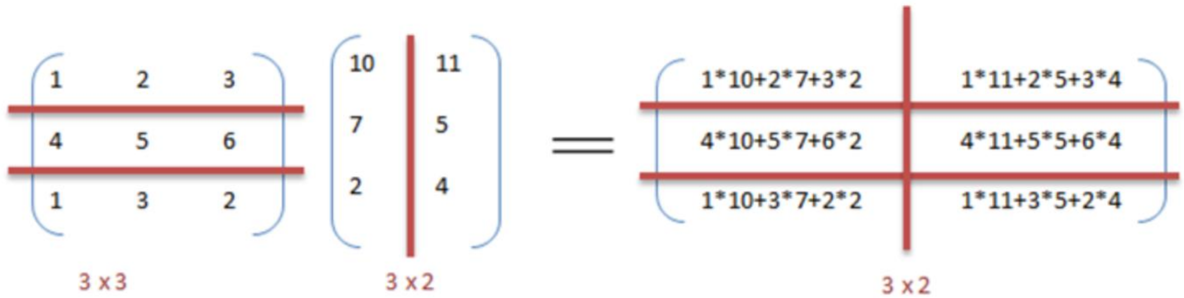
- **cum calculăm eficient?**

```
# varianta A
for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
     for (int k = 0; k < n; ++k)
        C[i][j] += A[i][k] * B[k][j];

# varianta B
for (int j = 0; j < n; ++j)
  for (int i = 0; i < n; ++i)
     for (int k = 0; k < n; ++k)
        C[i][j] += A[i][k] * B[k][j];

# varianta C
for (int k = 0; k < n; ++k)
  for (int i = 0; i < n; ++i)
     for (int j = 0; j < n; ++j)
        C[i][j] += A[i][k] * B[k][j];
```

https://mercris.wordpress.com/2012/06/29/linear-algebra-cheat-sheet-for-machine-learning/

LUCRU CU VECTOR/MATRICE, EX. 5

- **produs matrice-matrice**



- **cum calculăm eficient?**
  - c += x_i y_i
  - n^2 produse scalare (se pot realiza în paralel)
  - dintre cele 3 variate din dreapta, care

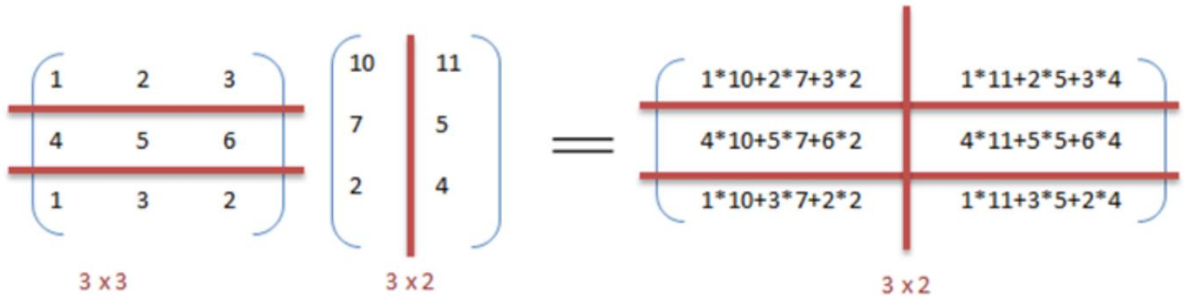
  este mai rapidă în C? (testați și explicați)
  - cum exploatăm eficient cache-ul?

```
# varianta A
for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
     for (int k = 0; k < n; ++k)
        C[i][j] += A[i][k] * B[k][j];

# varianta B
for (int j = 0; j < n; ++j)
  for (int i = 0; i < n; ++i)
     for (int k = 0; k < n; ++k)
        C[i][j] += A[i][k] * B[k][j];

# varianta C
for (int k = 0; k < n; ++k)
  for (int i = 0; i < n; ++i)
     for (int j = 0; j < n; ++j)
        C[i][j] += A[i][k] * B[k][j];
```

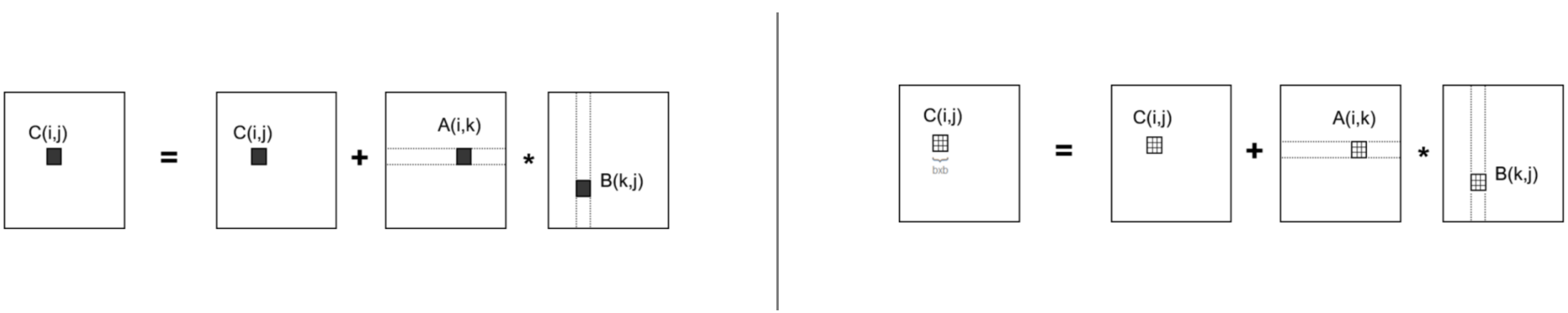
https://mercris.wordpress.com/2012/06/29/linear-algebra-cheat-sheet-for-machine-learning/

LUCRU CU VECTOR/MATRICE, EX. 5

- **produs matrice-matrice**



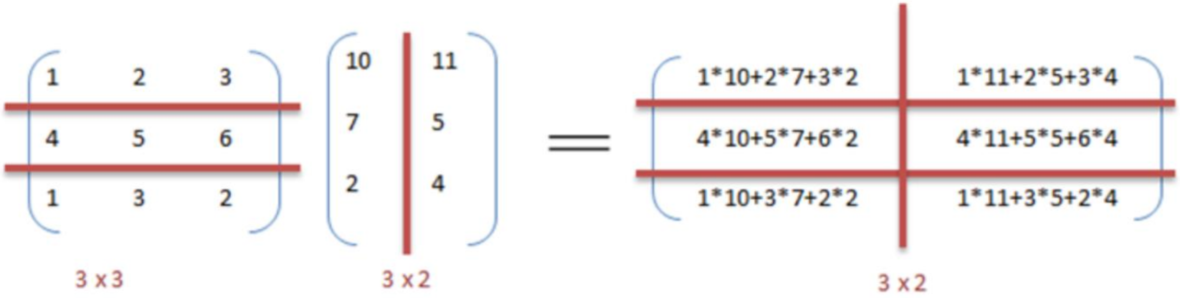
- **cum calculăm eficient?**
  - $c \mathrel{+}= x_i\, y_i$
  - n^2 produse scalare (se pot realiza în paralel)
  - cum exploatăm eficient cache-ul?
    - calcul pe blocuri, nu pe linii sau coloane



https://en.wikipedia.org/wiki/Communication-avoiding_algorithm

LUCRU CU VECTOR/MATRICE, EX. 5

- **produs matrice-matrice**



- **cum calculăm eficient?**
  - c += x_i y_i
  - n^2 produse scalare (se pot realiza în paralel)
  - cum exploatăm eficient cache-ul?
    - calcul pe blocuri, nu pe linii sau coloane

```
for (ii = 0; ii < SIZE; ii += BLOCK_SIZE)
  for (kk = 0; kk < SIZE; kk += BLOCK_SIZE)
    for (jj = 0; jj < SIZE; jj += BLOCK_SIZE)
      maxi = min(ii + BLOCK_SIZE, SIZE);
      for (i = ii; i < maxi; i++)
        maxk = min(kk + BLOCK_SIZE, SIZE);
        for (k = kk; k < maxk; k++)
          maxj = min(jj + BLOCK_SIZE, SIZE);
          for (j = jj; j < maxj; j++)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

https://en.wikipedia.org/wiki/Communication-avoiding_algorithm

PERFORMANȚA MULTI-CORE, EX. 6

a) **S_{Amdahl} = 1,78 și $S_{gustafson}$ = 4,5**

b) **pentru Amdahl**

dacă un program are timpul de execuție T atunci T = (1-p)T + pT (facem distincția între partea paralelizabilă și cel secvențială), dacă avem s core-uri atunci pT devine p/sT, deci accelerarea (raportul dintre timpul inițial și nou timp cu s core-uri) este S = T/((1-p)T + p/s T) = 1/ (1-p + p/s)

pentru Gustafson

sistemul este capabil să execute E = (1-p)E + pE iar partea care se poate îmbunătății este doar pE, care devine δpE, deci execuția este îmbunătățită ((1-p)E + δpE)/E = 1 – p + δp

c) **S_{Amdahl} = 1 și $S_{gustafson}$ = 1 (nicio îmbunătățire)**

d) **S_{Amdahl} = 1 și $S_{gustafson}$ = 1 (nicio îmbunătățire)**

e) **L_{Amdahl} = 1/(1-p), S_{Amdahl} < 1/(1-p)**

f) **$L_{gustafson} = \infty$**

g) **verificați explicațile de la punct b) și țineți cont de rezultatele la limitele pentru p, s și δ**

PERFORMANȚA CICLII, EX. 7

a) 2.1

b) 1.7

c) înainte de modificare N x 2.1 / f, după 0.8 x N x 1.7 / f

d) 1.85

PERFORMANȚA SISTEME, EX. 8

a) p = 80/145, p = 40/145 și p = 25/145 pentru sistemul X

p = 50/140, p = 50/140 și p = 40/140 pentru sistemul Y

b) 2.24 pentru sistemul X și 2.86 pentru sistemul Y

c) cpu time pe sistemul X = 145 x 2.24 / f

cpu time pe sistemul Y = 140 x 2.86 / (1.2 x f)

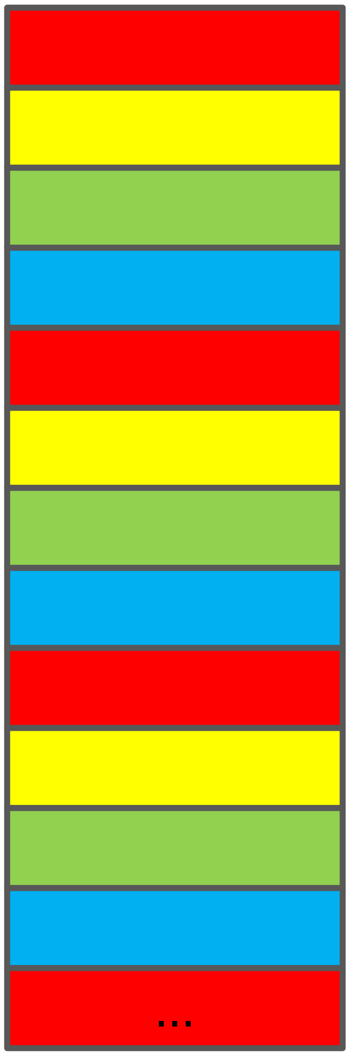
accelerarea este raportul valorilor: Y / X ≈ 1.03 (sistemul X este cu 3% mai rapid decât Y)

CACHE, EX. 9

a) $2^{32} / 2^{5} = 2^{27}$, $(2^{4} \times 2^{10}) / 2^{5} = 2^{9} = 512$ blocuri

b) un exemplu în care cache are doar 4 blocuri, cea mai simplă idee: un bloc din memoria principală dacă e în cache poate să fie doar într-o poziție din cache cu aceeași culoare

memoria principală



cache

1) trebuie să știm ce culoare suntem

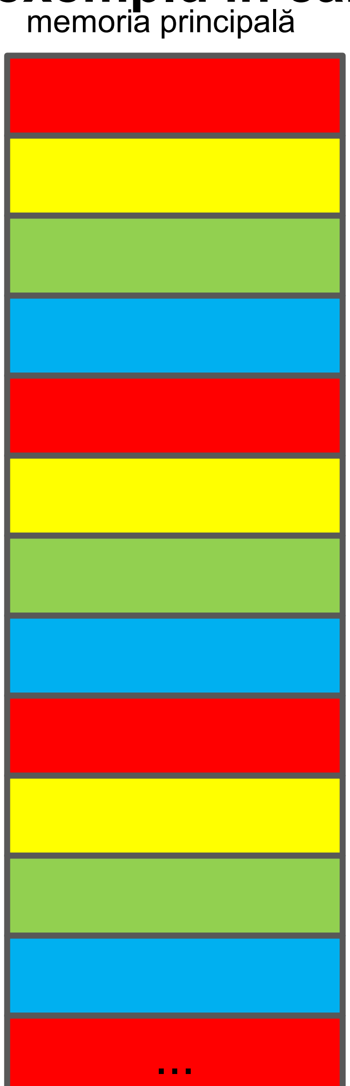
2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)

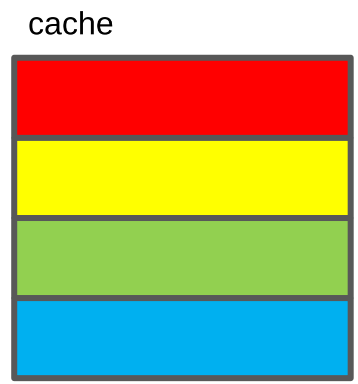
3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

CACHE, EX. 9

a) $2^{32} / 2^5 = 2^{27}$, $(2^4 \times 2^{10}) / 2^5 = 2^9 = 512$ blocuri

b) un exemplu în care cache are doar 4 blocuri, cea mai simplă idee: un bloc din memoria principală dacă e în cache poate să fie doar într-o poziție din cache cu aceeași culoare





1) trebuie să știm ce culoare suntem

2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)

3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

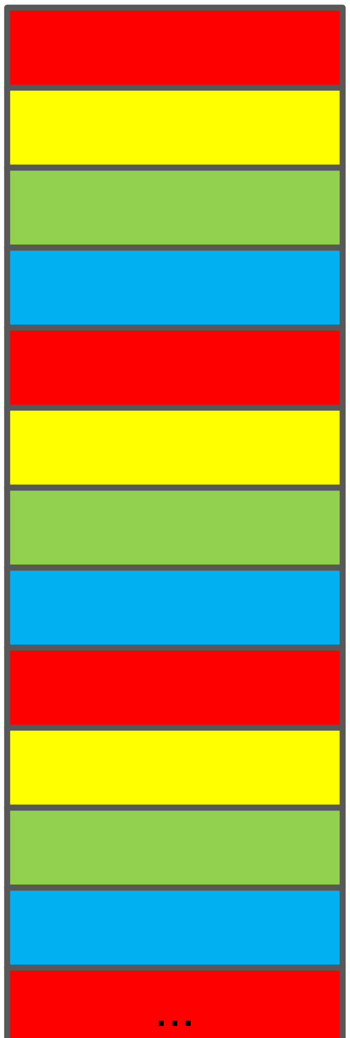
(2)	(1)	(3)

CACHE, EX. 9

a) $2^{32} / 2^5 = 2^{27}$, $(2^4 \times 2^{10}) / 2^5 = 2^9 = 512$ blocuri

b) un exemplu în care cache are doar 4 blocuri, cea mai simplă idee: un bloc din memoria principală dacă e în cache poate să fie doar într-o poziție din cache cu aceeași culoare

memoria principală



cache

1) trebuie să știm ce culoare suntem

2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)

3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

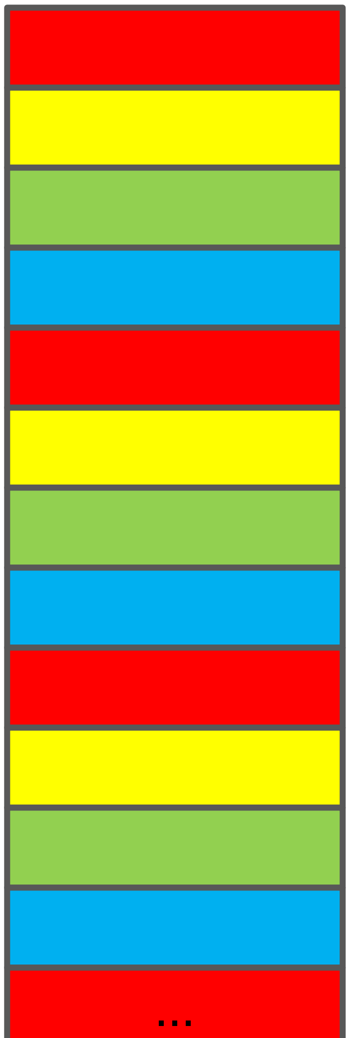
TAG	INDEX	OFFSET

CACHE, EX. 9

a) $2^{32} / 2^5 = 2^{27}$, $(2^4 \times 2^{10}) / 2^5 = 2^9 = 512$ blocuri

b) un exemplu în care cache are doar 4 blocuri, cea mai simplă idee: un bloc din memoria principală dacă e în cache poate să fie doar într-o poziție din cache cu aceeași culoare

memoria principală



cache

1) trebuie să știm ce culoare suntem

2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)

3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

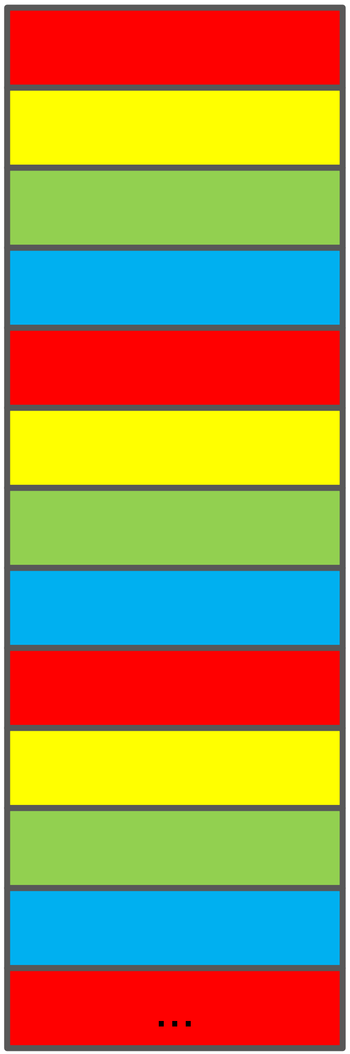
18 biți (ce rămâne pentru a identificare care bloc de aceeași culoare e cel corect)	9 biți (pentru că sunt 2^9 = 512 blocuri/culori în cache)	5 biți (pentru că sunt 2^5 = 32 bytes posibili în bloc)

CACHE, EX. 9

a) $2^{32} / 2^5 = 2^{27}$, $(2^4 \times 2^{10}) / 2^5 = 2^9 = 512$ blocuri

b) un exemplu în care cache are doar 4 blocuri, cea mai simplă idee: un bloc din memoria principală dacă e în cache poate să fie doar într-o poziție din cache cu aceeași culoare

memoria principală



tehnica aceasta de 1 la 1 se numește *direct mapping*
dacă un bloc din memoria principală poate să fie în mai multe blocuri din cache (nu doar unul singur) atunci tehnica se numește *N-set associative mapping* (unde N este numărul de blocuri din cache în care un bloc din memorie poate fi copiat (este fie acolo, fie nu e in cache)

această nouă tehnică oferă mai multa flexibilitate (1 la 1 este prea limitat)

1) trebuie să știm ce culoare suntem

2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)

3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

18 biți (ce rămâne pentru a identificare care bloc de aceeași culoare e cel corect)	9 biți (pentru că sunt 2^9 = 512 blocuri/culori în cache)	5 biți (pentru că sunt 2^5 = 32 bytes posibili în bloc)