

ARHITECTURA SISTEMELOR DE CALCUL SEMINAR 0x05

NOTIȚE SUPT 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

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

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

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

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

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

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

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

c) $1 \text{ ns} + (0.1 \times (5 \text{ ns} + (0.01 \times (10 \text{ ns} + (A \times 50 \text{ ns}))))) = t_{\text{RAM}}$

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

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

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

c) $1 \text{ ns} + (0.1 \times (5 \text{ ns} + (0.01 \times (10 \text{ ns} + (A \times 50 \text{ ns}))))) = t_{\text{RAM}}$

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 5\$

$$1 \times 0.9^A + (0.1 \times (5 \times 0.9^B + (0.01 \times (10 \times 0.9^C + (0.002 \times 50)))))) = t_{\text{RAM}} / 1000$$

vrem: minimize $100 \times A + 25 \times B + 5 \times C$

rezolvați pentru A, B și C

ÎNTREBĂRI SCURTE, EX. 2

a)

ÎNTREBĂRI SCURTE, EX. 2

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

ÎNTREBĂRI SCURTE, EX. 2

- a) utilizare CPU (eventual sisteme multi-core), media aritmetică
- b) wall-clock time, media aritmetică
- c)

ÎNTREBĂRI SCURTE, EX. 2

- a) utilizare CPU (eventual sisteme multi-core), media aritmetică
- b) wall-clock time, media aritmetică
- c) memoria RAM, maximum
- d)

ÎNTREBĂRI SCURTE, EX. 2

- a) utilizare CPU (eventual sisteme multi-core), media aritmetică
- b) wall-clock time, media aritmetică
- c) memoria RAM, maximum
- d) performanță per Watt, media aritmetică sau maximum
- e)

ÎNTREBĂRI SCURTE, EX. 2

- a) utilizare CPU (eventual sisteme multi-core), media aritmetică
- b) wall-clock time, media aritmetică
- c) memoria RAM, maximum
- d) performanță per Watt, media aritmetică sau maximum
- e) wall-clock time, 50/90/99th percentile mediana
- f)

ÎNTREBĂRI SCURTE, EX. 2

- a) utilizare CPU (eventual sisteme multi-core), media aritmetică
- b) wall-clock time, media aritmetică
- c) memoria RAM, maximum
- d) performanță per Watt, media aritmetică 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 aritmetică
- b) wall-clock time, media aritmetică
- c) memoria RAM, maximum
- d) performanță per Watt, media aritmetică 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 aritmetică
- b) wall-clock time, media aritmetică
- c) memoria RAM, maximum
- d) performanță per Watt, media aritmetică 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

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

b) 2 adunări, 4 înmulțiri

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

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

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:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$$

$$C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$$

$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$$

$$C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$$

$$M_1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$

$$M_2 := (A_{2,1} + A_{2,2})B_{1,1}$$

$$M_3 := A_{1,1}(B_{1,2} - B_{2,2})$$

$$M_4 := A_{2,2}(B_{2,1} - B_{1,1})$$

$$M_5 := (A_{1,1} + A_{1,2})B_{2,2}$$

$$M_6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$

$$M_7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

$$C_{1,1} = M_1 + M_4 - M_5 + M_7$$

$$C_{1,2} = M_3 + M_5$$

$$C_{2,1} = M_2 + M_4$$

$$C_{2,2} = M_1 - M_2 + M_3 + M_6$$

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:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$$

$$C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$$

$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$$

$$C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$$

$$O(n^3)$$

$$M_1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$

$$M_2 := (A_{2,1} + A_{2,2})B_{1,1}$$

$$M_3 := A_{1,1}(B_{1,2} - B_{2,2})$$

$$M_4 := A_{2,2}(B_{2,1} - B_{1,1})$$

$$M_5 := (A_{1,1} + A_{1,2})B_{2,2}$$

$$M_6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$

$$M_7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

$$C_{1,1} = M_1 + M_4 - M_5 + M_7$$

$$C_{1,2} = M_3 + M_5$$

$$C_{2,1} = M_2 + M_4$$

$$C_{2,2} = M_1 - M_2 + M_3 + M_6$$

$$O(n^{2.8})$$

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?
 - $c += x_i y_i$
 - 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?

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?
 - $c += x_i y_i$
 - 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 $\leq 50\%$

LUCRU CU VECTOR/MATRICE, EX. 5

- produs matrice-matrice

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 3 & 2 \end{pmatrix}_{3 \times 3} \begin{pmatrix} 10 & 11 \\ 7 & 5 \\ 2 & 4 \end{pmatrix}_{3 \times 2} = \begin{pmatrix} 1*10+2*7+3*2 & 1*11+2*5+3*4 \\ 4*10+5*7+6*2 & 4*11+5*5+6*4 \\ 1*10+3*7+2*2 & 1*11+3*5+2*4 \end{pmatrix}_{3 \times 2}$$

- 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];
```


LUCRU CU VECTOR/MATRICE, EX. 5

- produs matrice-matrice

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 3 & 2 \end{pmatrix}_{3 \times 3} \begin{pmatrix} 10 & 11 \\ 7 & 5 \\ 2 & 4 \end{pmatrix}_{3 \times 2} = \begin{pmatrix} 1*10+2*7+3*2 & 1*11+2*5+3*4 \\ 4*10+5*7+6*2 & 4*11+5*5+6*4 \\ 1*10+3*7+2*2 & 1*11+3*5+2*4 \end{pmatrix}_{3 \times 2}$$

- 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];
```

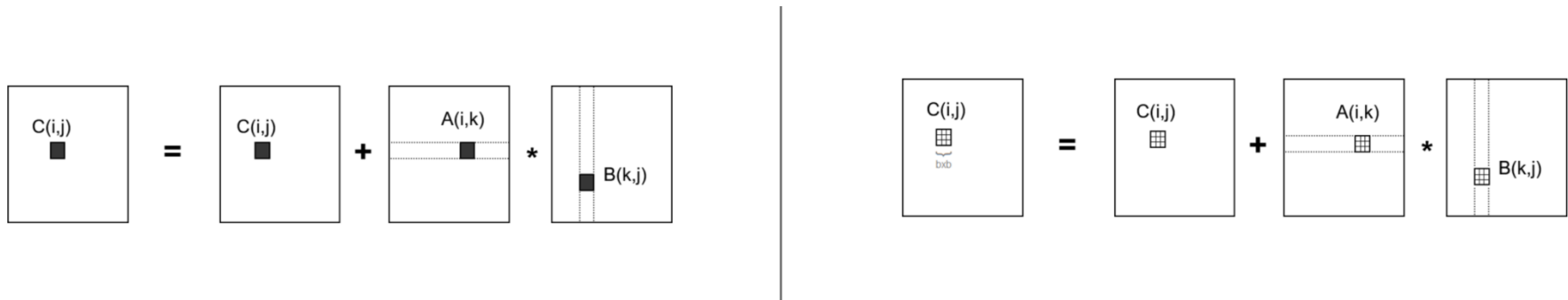
```
# varianta C
for (int k = 0; k < n; ++k)
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            C[i][j] += A[i][k] * B[k][j];
```

LUCRU CU VECTOR/MATRICE, EX. 5

- produs matrice-matrice

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 3 & 2 \end{pmatrix}_{3 \times 3} \begin{pmatrix} 10 & 11 \\ 7 & 5 \\ 2 & 4 \end{pmatrix}_{3 \times 2} = \begin{pmatrix} 1*10+2*7+3*2 & 1*11+2*5+3*4 \\ 4*10+5*7+6*2 & 4*11+5*5+6*4 \\ 1*10+3*7+2*2 & 1*11+3*5+2*4 \end{pmatrix}_{3 \times 2}$$

- 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



LUCRU CU VECTOR/MATRICE, EX. 5

- produs matrice-matrice

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 3 & 2 \end{pmatrix}_{3 \times 3} \begin{pmatrix} 10 & 11 \\ 7 & 5 \\ 2 & 4 \end{pmatrix}_{3 \times 2} = \begin{pmatrix} 1*10+2*7+3*2 & 1*11+2*5+3*4 \\ 4*10+5*7+6*2 & 4*11+5*5+6*4 \\ 1*10+3*7+2*2 & 1*11+3*5+2*4 \end{pmatrix}_{3 \times 2}$$

- 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];
```

PERFORMANȚA MULTI-CORE, EX. 6

a) $S_{\text{Amdahl}} = 1,78$ și $S_{\text{Gustafson}} = 4,5$

b) pentru Amdahl

dacă un program are timpul de execuție T atunci $T = (1-p)T + pT$ (facem distincția între partea paralelizabilă și cea 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 $S = T / ((1-p)T + p/sT) = 1 / (1-p + p/s)$

pentru Gustafson

sistemul este capabil să execute $E = (1-p)E + pE$ iar partea care se poate îmbunătăți este doar pE , care devine δpE , deci execuția este îmbunătățită $((1-p)E + \delta pE) / E = 1 - p + \delta p$

c) $S_{\text{Amdahl}} = 1$ și $S_{\text{Gustafson}} = 1$ (nicio îmbunătățire)

d) $S_{\text{Amdahl}} = 1$ și $S_{\text{Gustafson}} = 1$ (nicio îmbunătățire)

e) $L_{\text{Amdahl}} = 1/(1-p)$, $S_{\text{Amdahl}} < 1/(1-p)$

f) $L_{\text{Gustafson}} = \infty$

g) verificați explicațiile 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 $N \times 2.1 / f$, după $N \times 1.7 / f$
- d) 1.48

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 = $145 \times 2.24 / f$

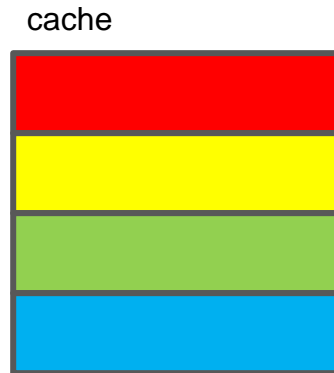
cpu time pe sistemul Y = $140 \times 2.86 / (1.2 \times f)$

accelerarea este raportul valorilor: $Y / X \approx 1.03$ (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

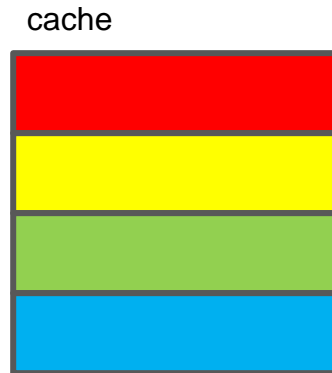


- 1) trebuie să știm ce culoare suntem
- 2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)
- 3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

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



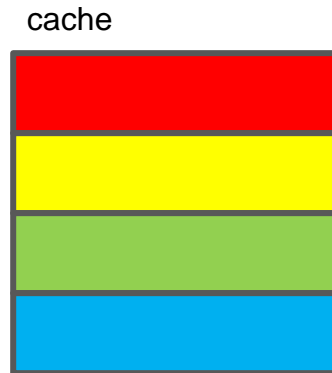
- 1) trebuie să știm ce culoare suntem
- 2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)
- 3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

(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



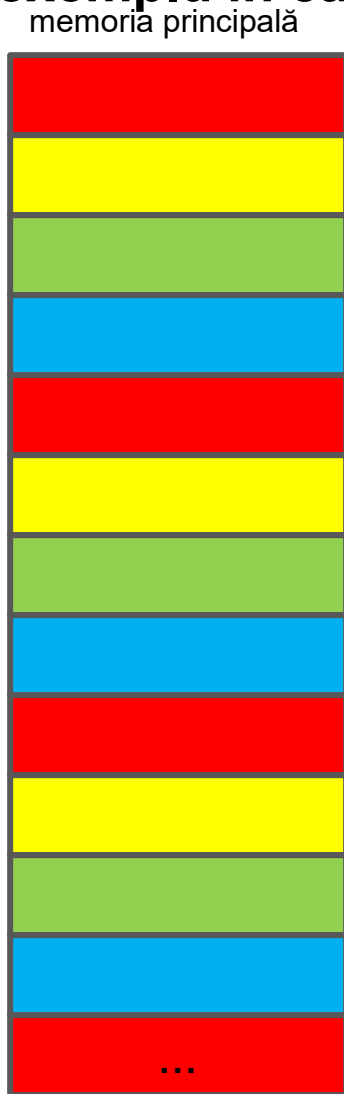
- 1) trebuie să știm ce culoare suntem
- 2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)
- 3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

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



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)

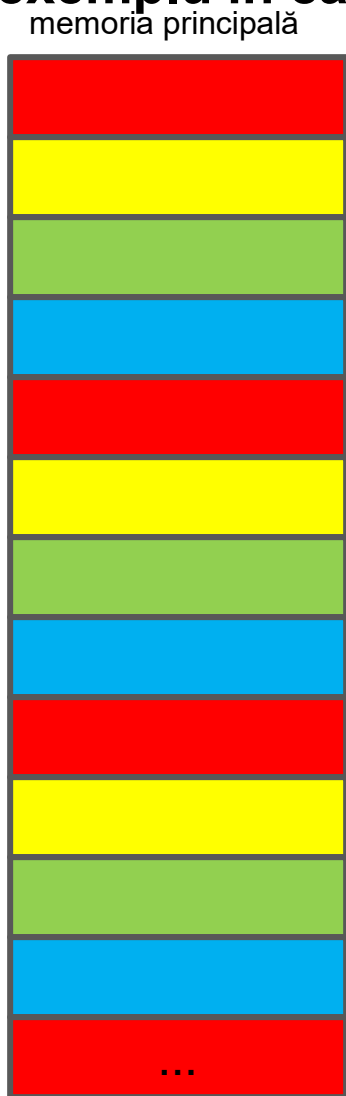
9 biți (pentru că sunt $2^9 = 512$ blocuri/culori în cache)

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



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 în 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)

