Algorytmy numeryczne

Wprowadzenie

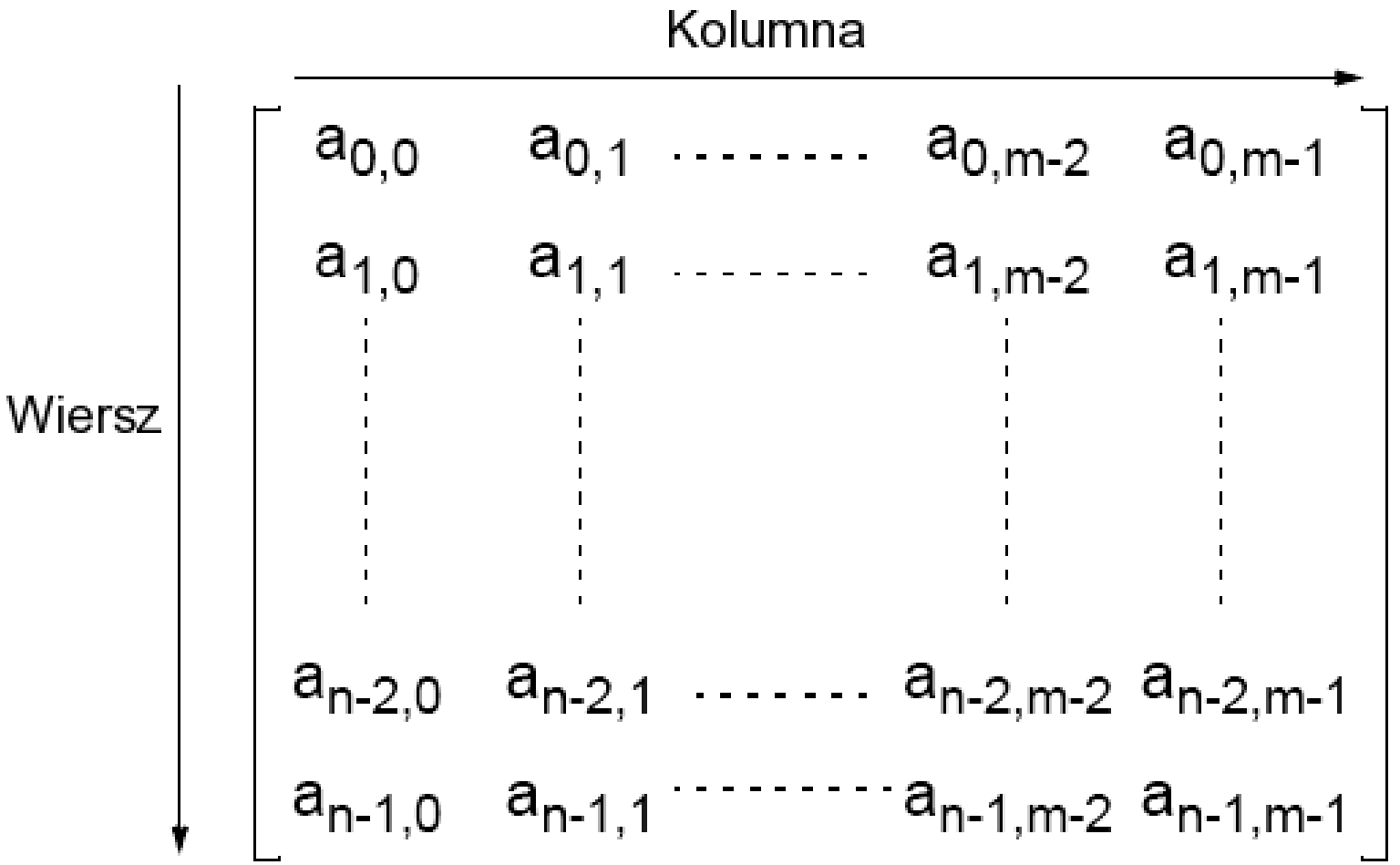
- Obliczenie numeryczne są najważniejszym zastosowaniem komputerów równoległych.

- Przykładem są symulacje zjawisk fizycznych, których przeprowadzenie sprowadza się do rozwiązania układu równań różniczkowych cząstkowych.

- Z kolei przybliżone rozwiązanie układu równań różniczkowych cząstkowych jest znajdywane poprzez rozwiązanie układu równań liniowych.

- Na dzisiejszym wykładzie przedstawimy
  - Algorytmy macierzowe - głównie mnożenie dwóch macierzy
  - Rozwiązywanie układu równań liniowych.

Macierze - przypomnienie



- Macierz nxm (o n wierszach i m kolumnach).

Macierze - dodawanie

- Dodawanie dwóch macierzy (o identycznych wymiarach) polega na zsumowaniu odpowiadających sobie elementów.

C=A+B

- Jeżeli elementy macierzy A oznaczymy jako $a_{i,j}$, a elementy macierzy B jako $b_{i,j}$, to element macierzy wynikowej C $c_{i,j}$ jest określony zależnością:

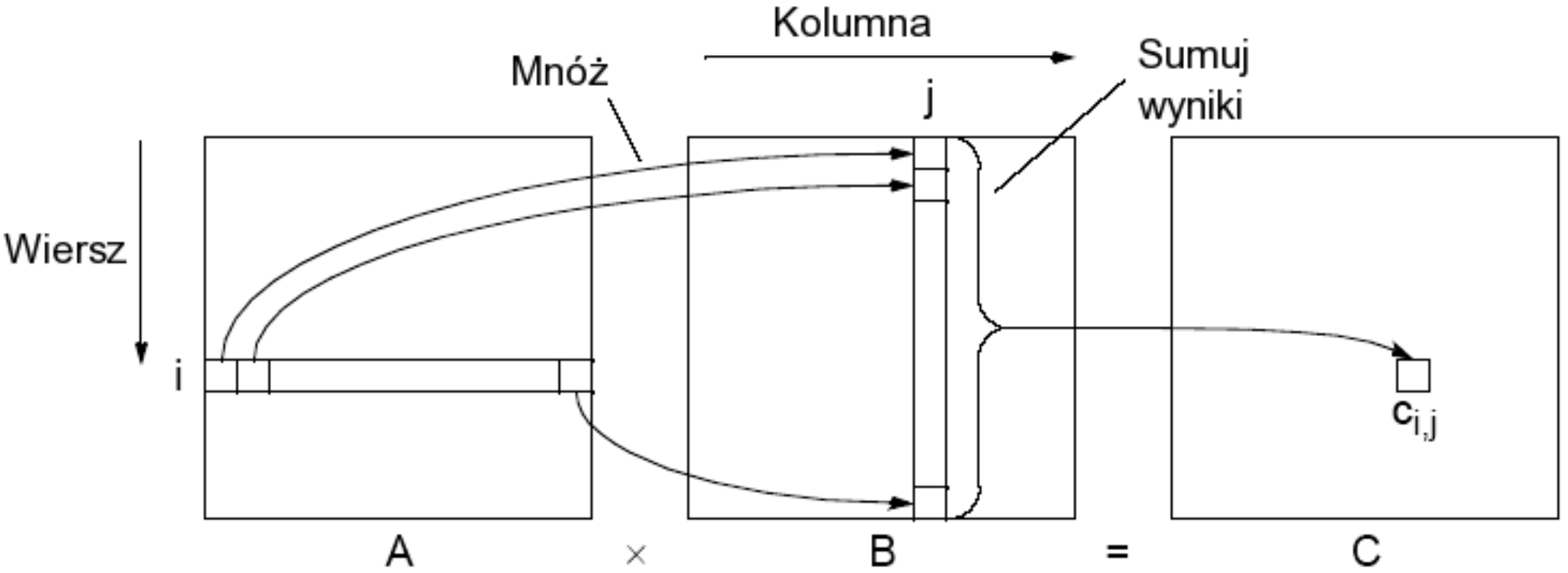
$$c_{i,j} = a_{i,j} + b_{i,j}$$

- Zrównoleglenie. Nie jest to problem tak interesujący jak mnożenie macierzy. Oczywista idea polega na jednoczesnej dekompozycji trzech macierzy A,B,C. Możliwe są dwa podejścia:
  - każdy procesor otrzymuje podzbiór wierszy (lub kolumn) każdej macierzy (dekompozycja wierszami - ang. row oriented decomposition).
  - każdy procesor otrzymuje prostokątny blok elementów macierzy. Mówimy wtedy o dystrybucji blokami (ang. block decomposition).

Mnożenie macierzy

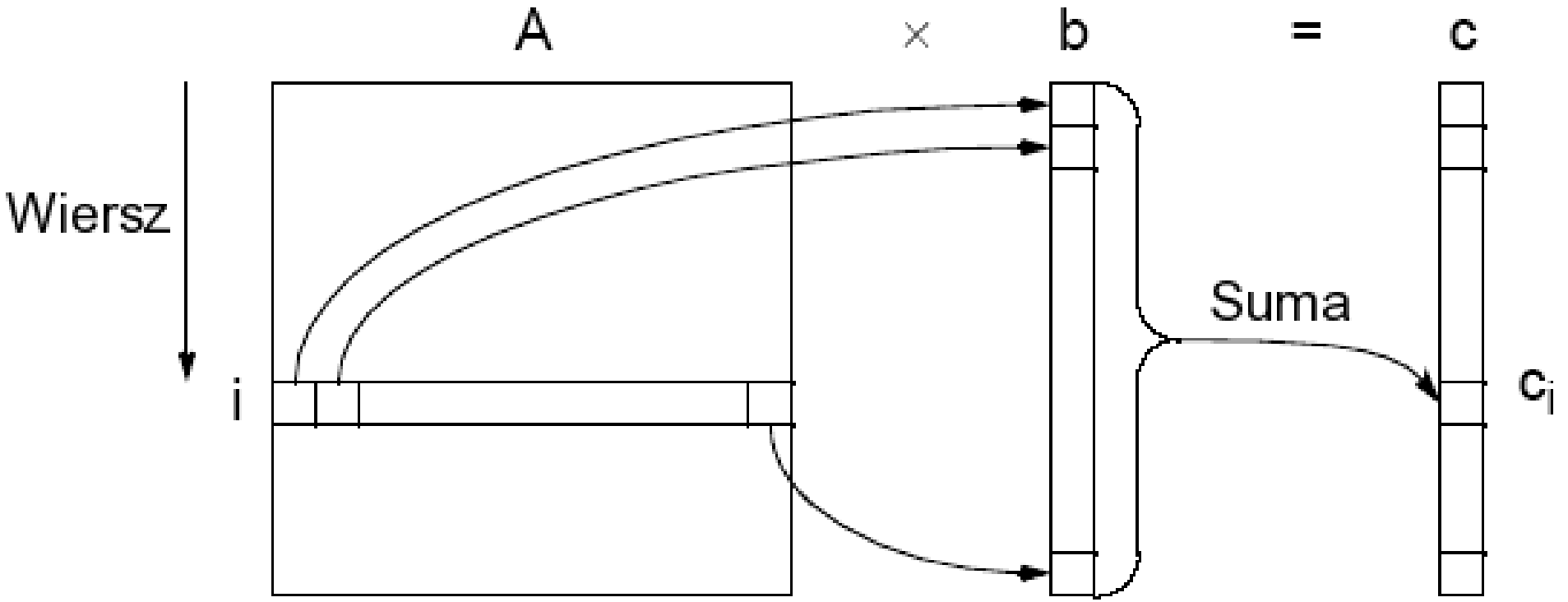
- Mnożenie dwóch macierzy A (o wymiarze nxl) i B (o wymiarze lxm) daje nam w wyniku macierz C, której elementy obliczane $c_{i,j}$ (i<=0<n oraz j<=0<m) są jako:

$$c_{i,j} = \sum_{k=0}^{l-1} a_{i,k} b_{k,j}$$



Mnożenie macierzy i wektora

- Jest to szczególny przypadek mnożenia macierzy w którym wektor potraktowany jest jako macierz nx1 (o n wierszach i jednej kolumnie).



Implementacja mnożenia macierzy

- Zakładamy że macierze A,B,C mają wymiar nxn (kwadratowe). Mnożenie macierzy implementuje następujący kod:

```
for (i = 0; i < n; i++)
   for (j = 0; j < n; j++) {
      c[i][j] = 0;
      for (k = 0; k < n; k++)
         c[i][j] = c[i][j] + a[i][k] * b[k][j];
   }
```

- Algorytm wykonuje n^3 mnożeń i n^3 dodawań, ma zatem złożoność $O(n^3)$.
- Powyższy kod jest stosunkowo łatwy do zrównoleglenia.

Równoległa implementacja mnożenia macierzy

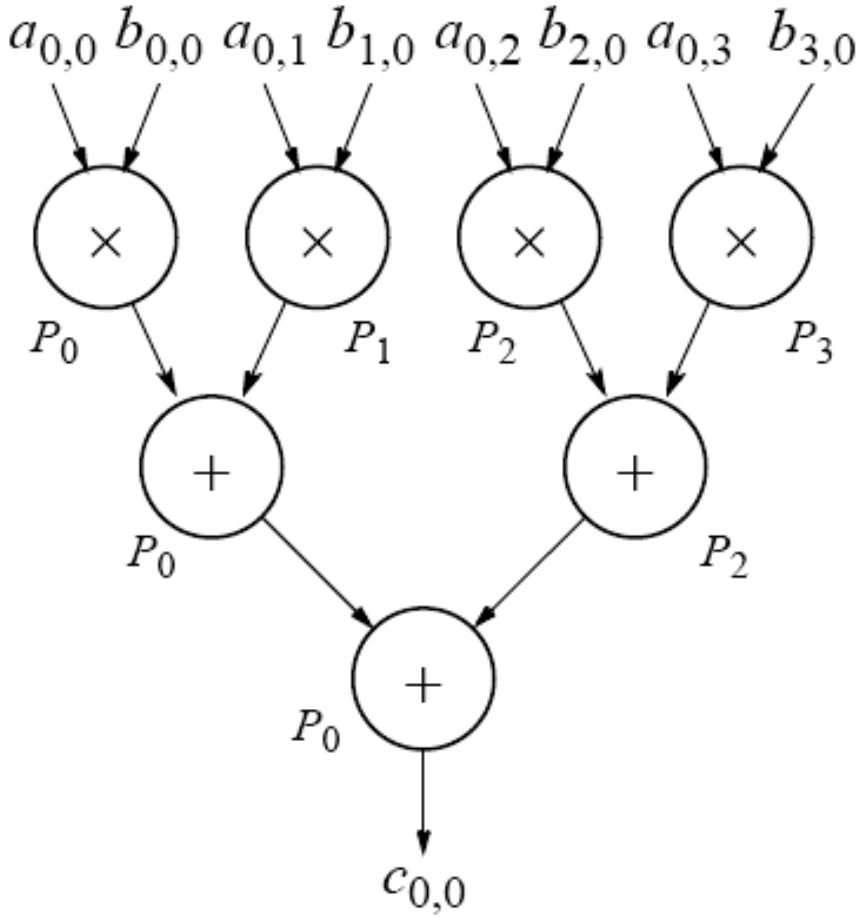
- Złożoność $O(n^2)$ z wykorzystaniem n procesorów. Każdy z procesorów wykonuje inną iterację zewnętrznej pętli for. Wykorzystujemy fakt, że Iteracje pętli for są niezależne.

- Złożoność O(n) z wykorzystaniem n^2 procesorów. Każdy procesor oblicza jeden element macierzy C.

- Złożoność O(log(n)) z wykorzystaniem n^3 procesorów. n procesorów wykonuje najbardziej wewnętrzną pętlę for. W odróżnieniu od dwóch poprzednich metoda nie jest optymalna ze względu na koszt, gdyż $O(n^3) \neq n^3 * O(\log(n))$.

- Matematycznie wykazano, że O(log(n)) jest dolnym ograniczeniem na złożoność równoległego algorytmu mnożenia macierzy.

Wewnętrzna pętla for w czasie O(log(n))



- Przykład dla macierzy 4x4. i czterech procesorów.

- Przy pomocy konstrukcji drzewiastej n liczb może być dodanych z wykorzystaniem n procesorów w czasie O(log(n)). Prowadzi to do złożoności O(log(n)) całego algorytmu mnożenia macierzy z wykorzystaniem n^3 procesorów.

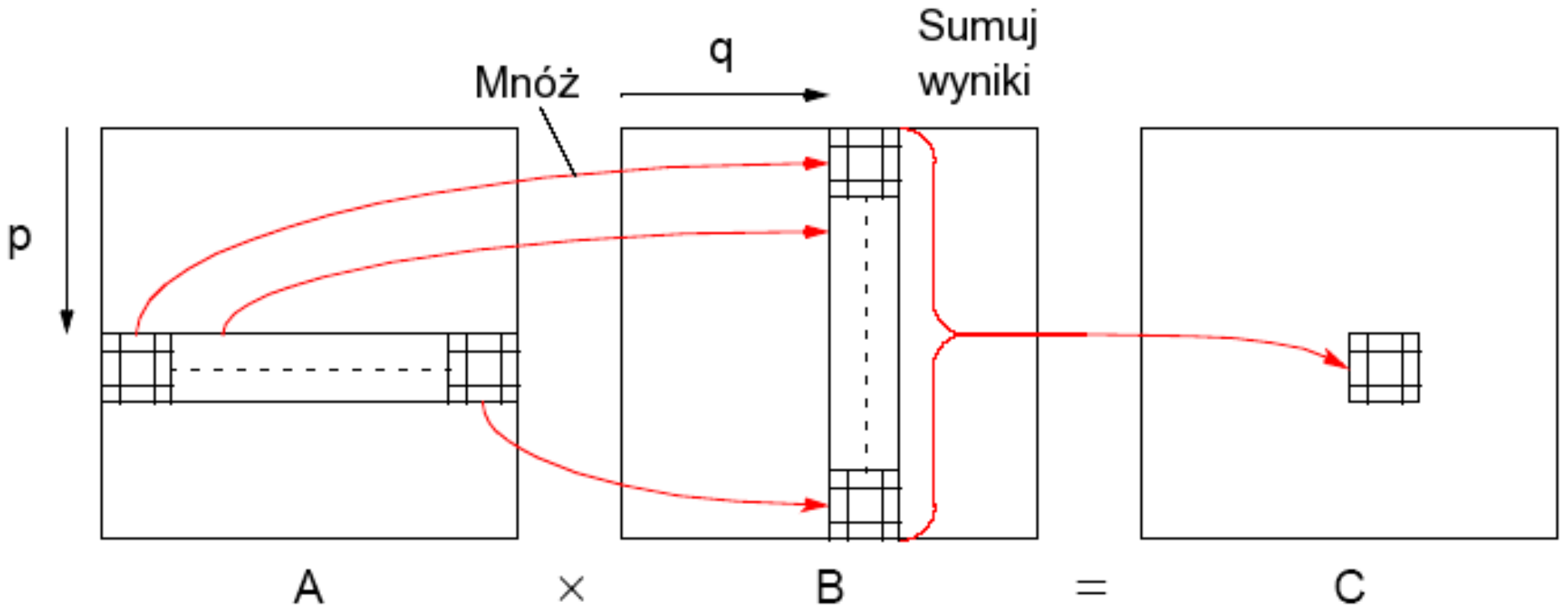
Podział na podmacierze

- W praktycznych zastosowaniach n jest na tyle duże, że mamy do dyspozycji o wiele mniej niż n procesorów dla macierzy nxn.
- W takiej sytuacji musimy wykonać partycjonowanie danych. Każda macierz jest dzielona na bloki elementów zwane podmacierzami (ang. submatrix). Submacierzami możemy manipulować tak, jakby były pojedynczymi elementami macierzy.
- Przypuśćmy, że macierze A,B,C podzieliiśmy na s^2 podmacierzy. (s wszerz oraz s w dół). Każda podmacierz ma wymiar mxm, gdzie m=n/s. Używając notacji, w której $A_{p,q}$ oznacza podmacierz w wierszy p i kolumnie q, algorytm mnożenia macierzy możemy zapisać jako:

```
for (p = 0; p < s; p++)
   for (q = 0; q < s; q++) {
      C[p,q] = 0;
      for (r = 0; r < m; r++)
         C[p,q] = C[p,q] + A[p,r] * B[r,q];
}
```

- Należy pamiętać, że wiersz $C_{p,q} = C_{p,q} + A_{p,r} * B_{r,q}$ oznacza mnożenie podmacierzy $A_{p,r}$ oraz $B_{r,q}$ używając algorytmu mnożenia macierzy.
- Algorytm nazywany jest blokowym mnożeniem macierzy (ang. block matrix multiplication)

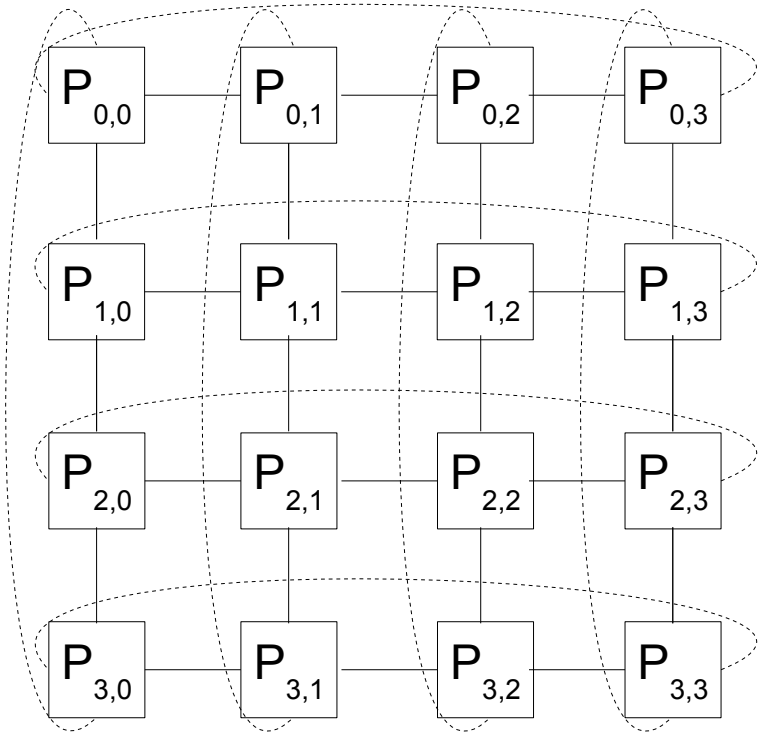
Blokowe mnożenie macierzy - ilustracja



- Podstawowa technika wykorzystywana w mnożeniu macierzy nxn, gdy mamy do dyspozycji mniej niż n procesorów.

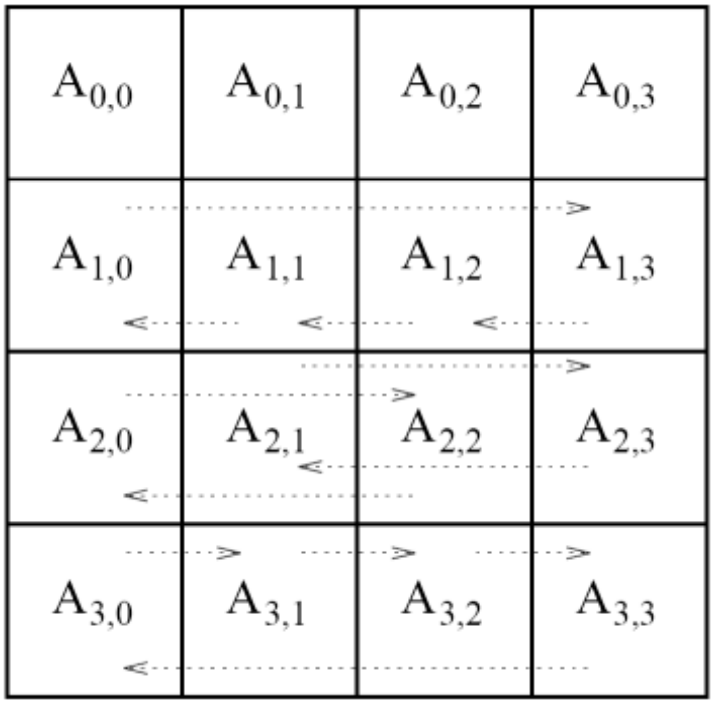
Algorytm Cannona

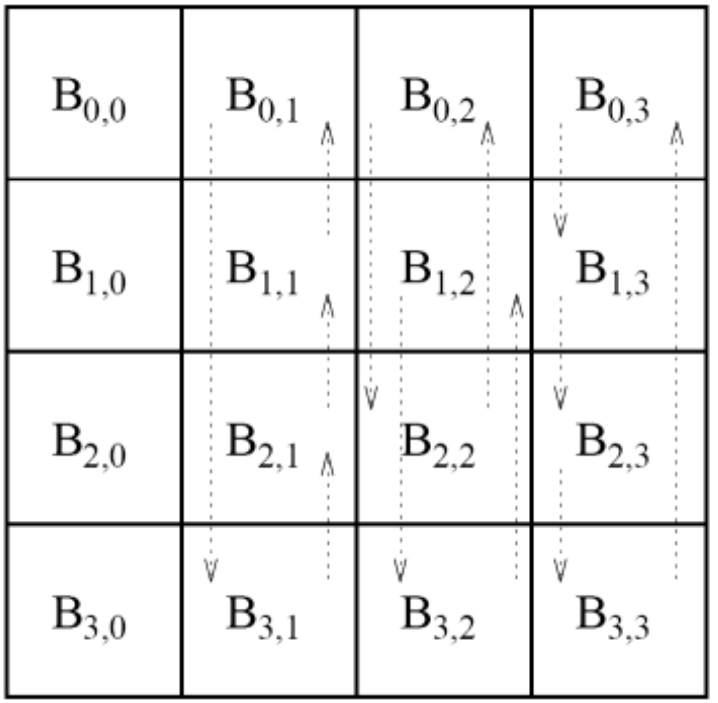
- Algorytm w którym procesory tworzą logiczną strukturę typu 2D-torus.



- Procesor $P_{i,j}$ po zakończeniu obliczeń będzie przechowywał podmacierz $C_{i,j}$. Na początku podmacierz $C_{i,j}$ jest równa zeru.
- Procesor $P_{i,j}$ mnoży aktualnie przechowywaną podmacierz macierzy A z podmacierzą macierzy B, dodając wynik do $C_{i,j}$.
- Następnie przesyła podmacierz A sąsiadowi z lewej, a podmacierz B sąsiadowi z góry.

Algorytm Cannona - wstępne ustawienie podmacierzy

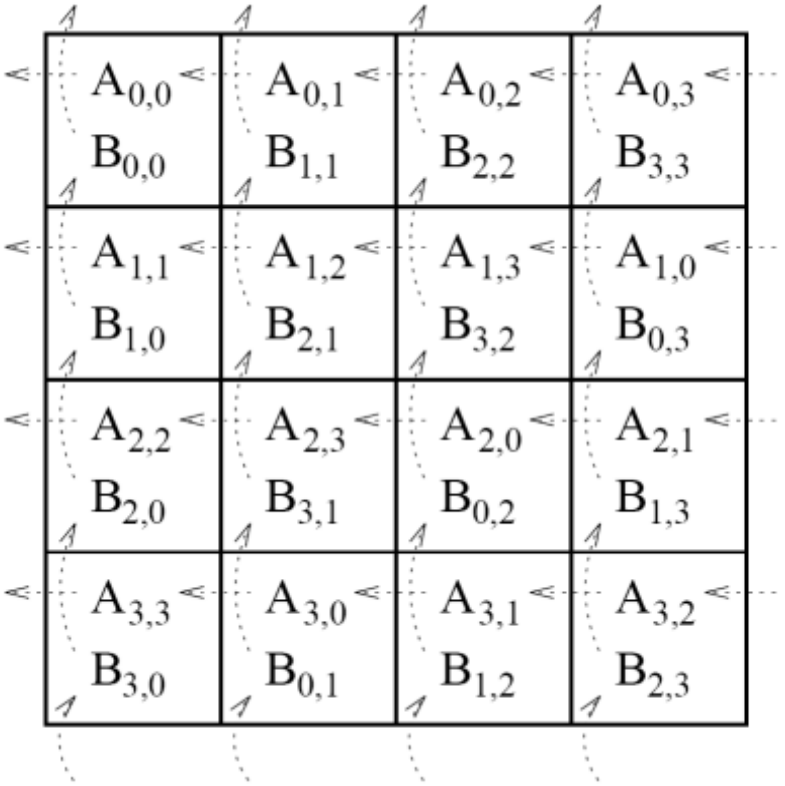




- Na początku procesor $P_{i,j}$ przechowuje podmacierze $A_{i,j}$ oraz $B_{i,j}$.
- We wstępnym kroku algorytmu i-ta kolumna A jest przesyłana i-wierszy w lewo. Z kolei j-ty wiersz B jest przesyłany j kroków w górę.

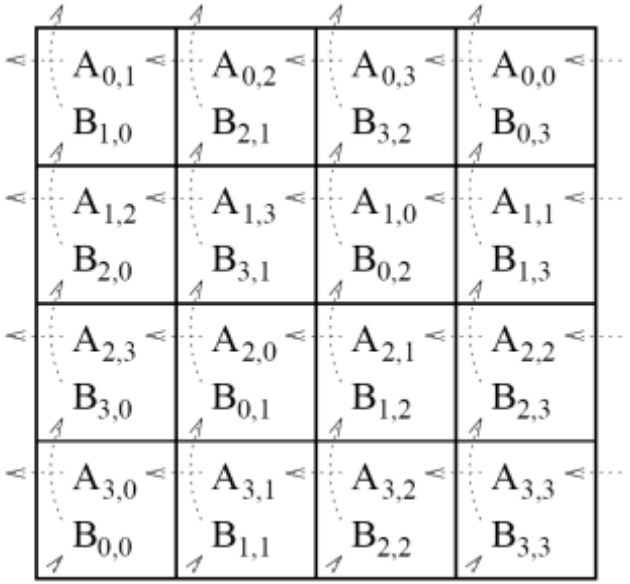
Algorytm Cannona (2)

- Po wstępnym przesunięciu położenie podmacierzy jest następujące:

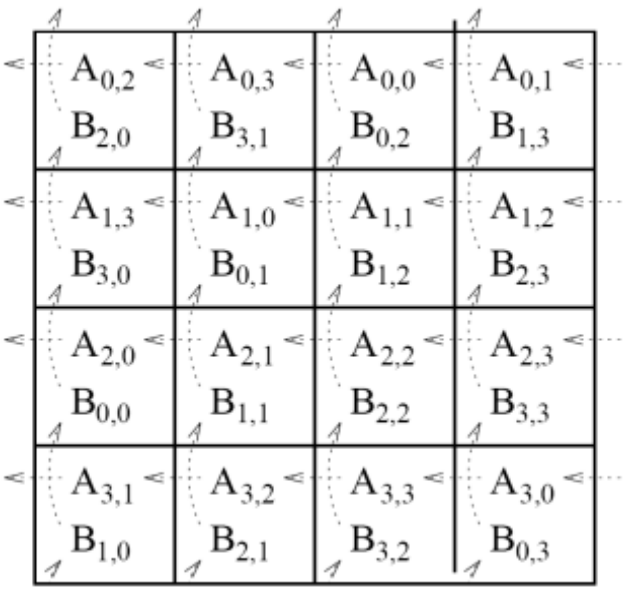


- W jednym kroku procesor $P_{i,j}$ mnoży aktualne podmacierze A oraz B, dodaje wynik do przechowywanej wielkości $C_{i,j}$.
- Następnie przesyła podmacierz A w lewo a podmacierz B w górę.

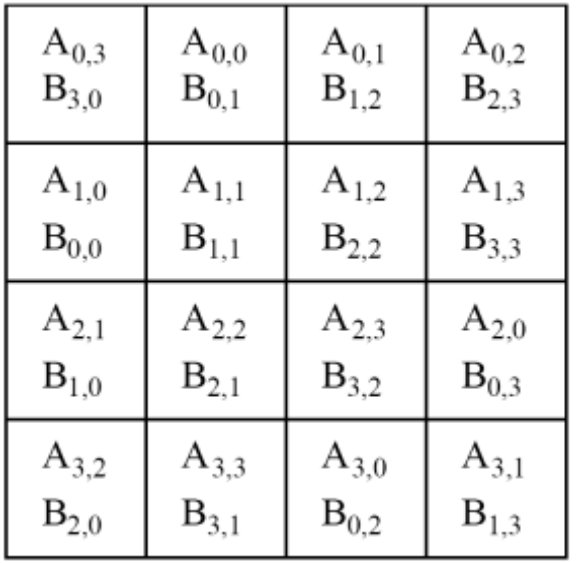
Algorytm Cannona (3)



Po pierwszym przesunięciu



Po drugim przesunięciu



Po trzecim przesunięciu

- Dzięki sprytnej manipulacji podmacierzami złożoność pamięciowa algorytmu Cannona jest $O(n^2)$. Nie jest wymagana dodatkowa pamięć w porównaniu z algorytmem szeregowym.

Macierze a układy równań liniowych

$$
\begin{aligned}
a_{0,0}x_0 + a_{0,1}x_1 + \ldots + a_{0,n-1}x_{n-1} &= b_0 \\
a_{1,0}x_0 + a_{1,1}x_1 + \ldots + a_{1,n-1}x_{n-1} &= b_1 \\
&\ldots \\
a_{n-1,0}x_0 + a_{n-1,1}x_1 + \ldots + a_{n-1,n-1}x_{n-1} &= b_{n-1}
\end{aligned}
$$

- Powyższy układ n równań liniowych może być zapisany w postaci macierzowej jako:

$$Ax = b$$

gdzie

A jest macierzą nxn stałych a.

b jest wektorem stałych b

x jest wektorem niewiadomych

Eliminacja Gaussa

- Polega na redukcji do układu równań z macierzą trójkątną postaci:

$$\begin{aligned} x_0 + u_{0,1}x_1 + u_{0,2}x_2 + \cdots + u_{0,n-1}x_{n-1} &= y_0, \\ x_1 + u_{1,2}x_2 + \cdots + u_{1,n-1}x_{n-1} &= y_1, \\ \vdots & \quad \vdots \\ x_{n-1} &= y_{n-1}. \end{aligned}$$

- Macierzowo możemy to zapisać:

$$Ux=y,$$

gdzie macierz U jest macierzą trójkątną górną (z elementami poniżej przekątnej równymi zero, i elementami na przekątnej równymi jeden).

- Przy redukcji wykorzystujemy fakt, że do dowolnego równania układu możemy dodać inne równanie pomnożone przez stałą, otrzymując układ równań z takimi samymi rozwiązaniami.
- Układ równań z macierzą trójkątną może być łatwo rozwiązany przez podstawienie, poczynając od zmiennej x_{n-1}.

Eliminacja Gaussa

$$a_{0,0}x_0 + a_{0,1}x_1 + \ldots + a_{0,n-1}x_{n-1} = b_0$$
$$a_{1,0}x_0 + a_{1,1}x_1 + \ldots + a_{1,n-1}x_{n-1} = b_1$$

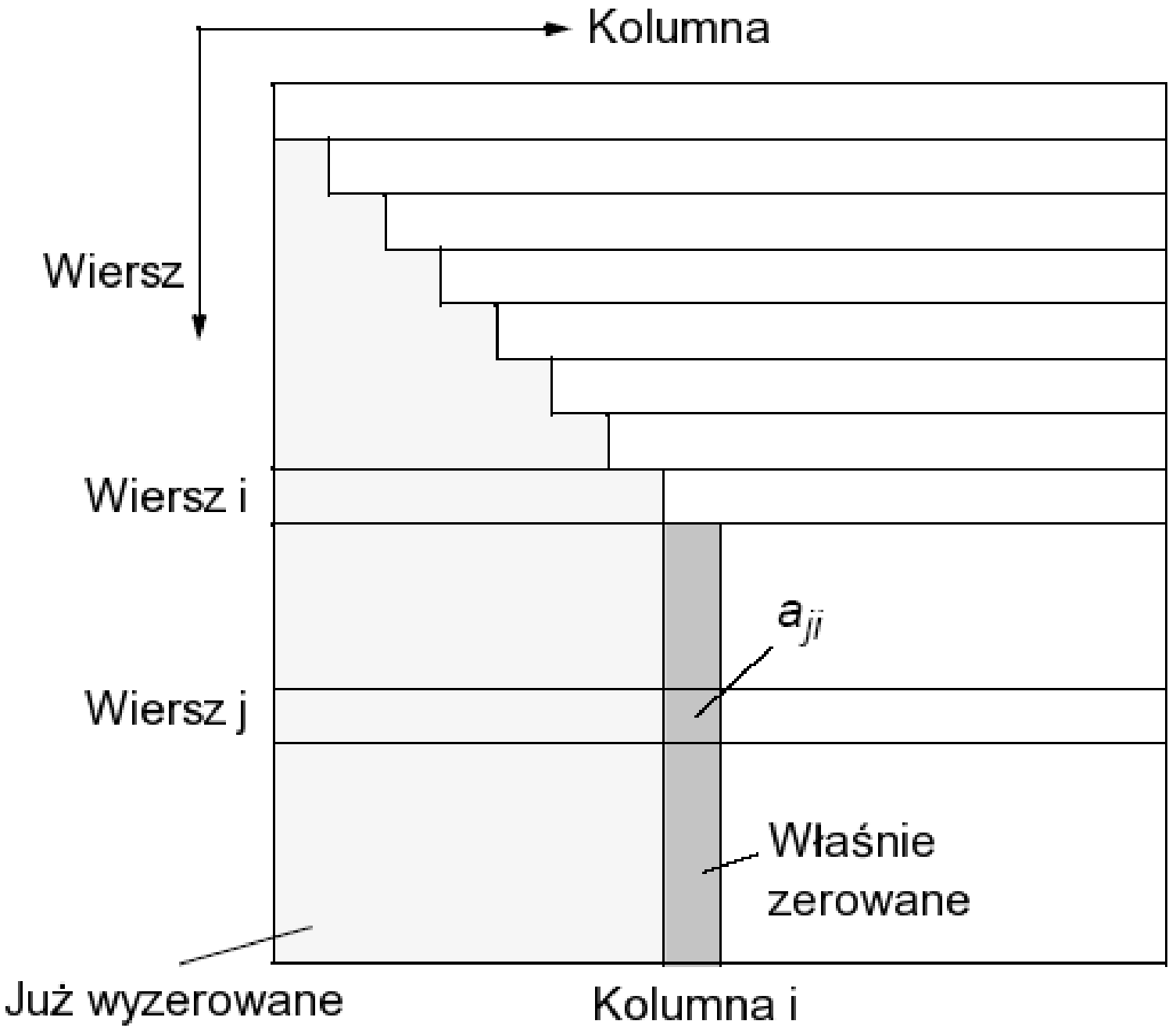
$$a_{n-1,0}x_0 + a_{n-1,1}x_1 + \ldots + a_{n-1,n-1}x_{n-1} = b_{n-1}$$

- Startuje z wiersza i i postępuje w stronę wierszy o rosnących numerach.
- W każdym wierszuj j poniżej i wiersz j jest zastępowany przez:

$$\text{wiersz}_j - \text{wiersz}_i * a_{j,i} / a_{i,i}$$

- Np. Rozważamy wiersz 0 . $a_{1,0}$ zostanie zastąpione przez $a_{1,0} - a_{0,0} * a_{1,0} / a_{0,0} = a_{1,0} - a_{1,0} = 0$! Podobnie wszystkie inne elementy w kolumnie 0.
- W efekcie dla iteracji dotyczącej wiersza i, wszystkie elementy w kolumnie i poniżej $a_{i,i}$ zostaną wyzerowane.

Eliminacja Gaussa - ilustracja



Eliminacja Gaussa - kod

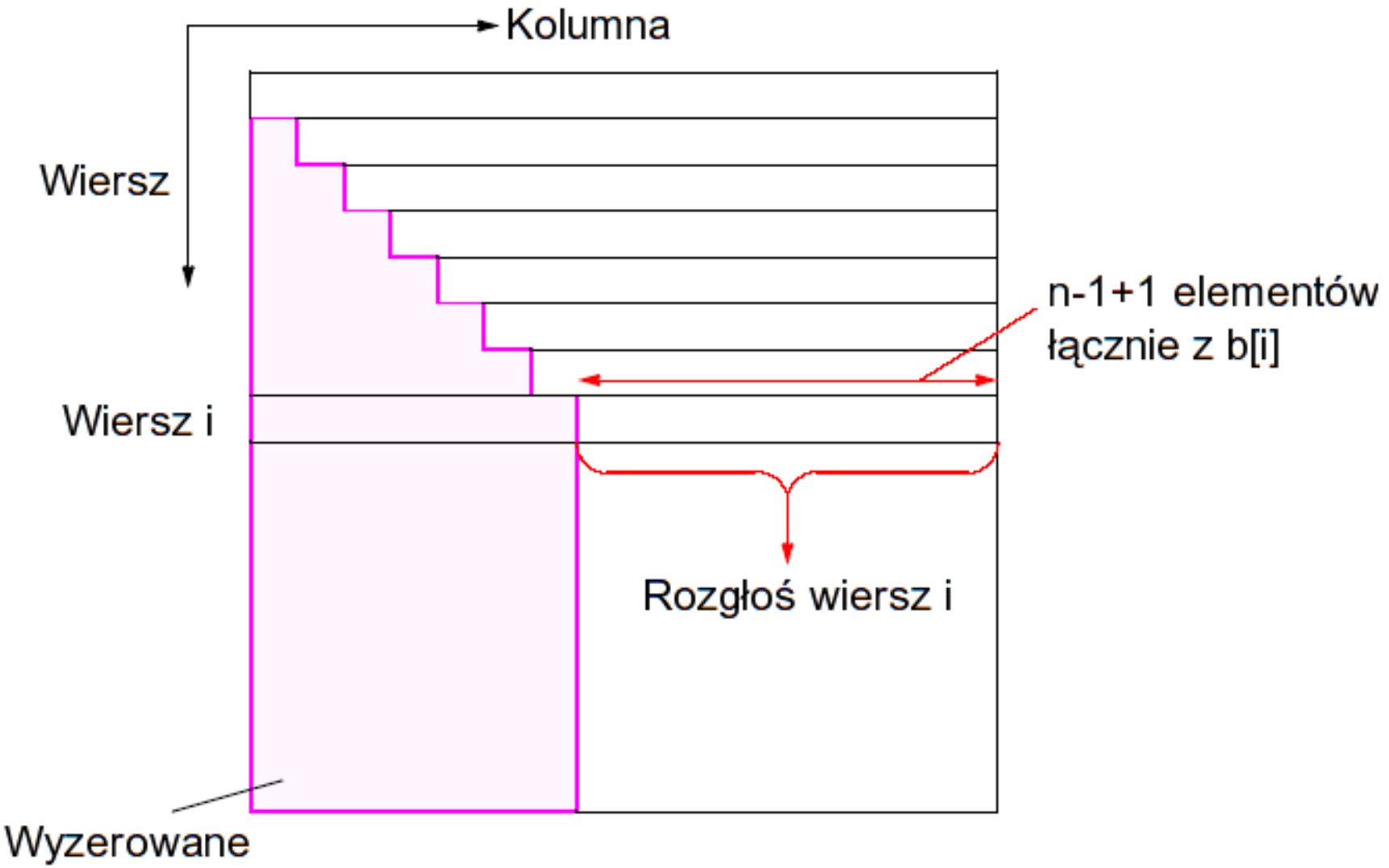
```
// Iteruj po wszystkich wierszach za wyjątkiem ostatniego
for (i = 0; i < n-1; i++)
// Postępuj dla wierszy poniżej wiersza i
          for (j = i+1; j < n; j++) {
               m = a[j][i]/a[i][i];
// Modyfikuj elementy wiersza na przekątnej i na prawo od niej
// Pozostałe zostały już wyzerowane
               for (k = i; k < n; k++)
                  a[j][k] = a[j][k] - a[i][k] * m;
// Zmodyfikuj prawą stronę równania (wektor stałych b)
               b[j] = b[j] - b[i] * m;
}
```

- Złożoność obliczeniowa wersji szeregowej O(n3).
- Co się zdarzy gdy a[i][i] jest równe zeru (lub bliskie zeru). Rozwiązanie: zamiana wierszy (tak aby w nowym wierszu a[i][i] było największe co do wartości bezwzględnej; zapewnia to najlepszą stabilność numeryczną).
  - Zamiana wierszy nie wpływa na rozwiązanie układu.

Eliminacja Gaussa - zrównoleglenie

- Idea zrównoleglenia: procesor przechowuje elementy jednego wiersza i operuje na tym wierszu. Wymagane n procesorów dla n równań.
- Zanim procesor P_j przeprowadzi operację na wierszu j musi o trzymać wiersz i od procesora P_i.
  - Najpierw procesor P_0 rozgłasza (broadcast) wiersz zerowy wszystkim procesorom, które modyfikują swoje wiersze.
  - Następnie procedura jest powtarzana z procesorami P_1, P_2,.., P_{n-2} rozgłaszającymi elementy swojego wiersza. Procesor Pi rozsyła elementy a[i][i+1], a[i][i+2], ..., a[i][n-2], b[i] (elementy od a[i][0] do a[i] zostały wyzerowane) procesorom o większych numerach.
- Wydajność tej metody nie jest największa, ponieważ procesor Pi bierze udział tylko w elimincaji dla wierszy 0,1,...,i.

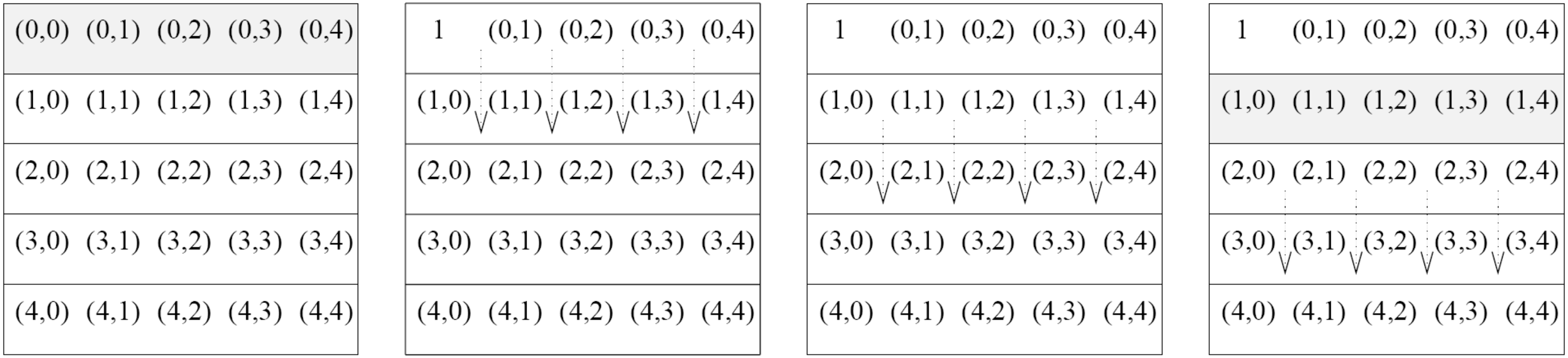
Eliminacja Gaussa - rozgłaszanie wiersza



- W praktycznych zastosowaniach liczba procesorów << liczba wierszy macierzy

Wersja potokowa

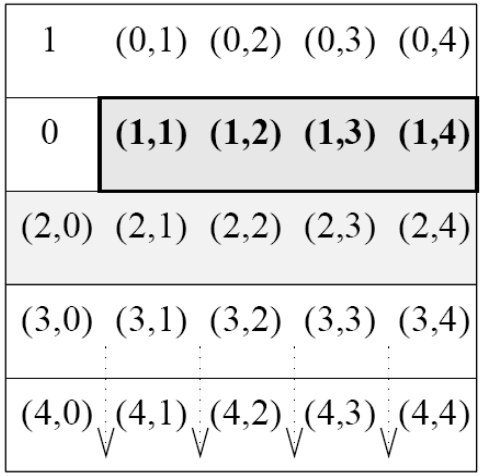
- W poprzedniej wersji k-ta iteracja zewnętrznej pętli for rozpoczyna się, dopiero po zakończeniu k+1 iteracji.
- W wersji potokowej procesor P_i czeka na odebranie i wyeliminowanie wszystkich wierszy o numerach mniejszych od i.
- Po tej chwili wysyła swój wiersz procesorowi P_{i+1}.
- Przykład dla macierzy 5x5 i pięciu procesów:



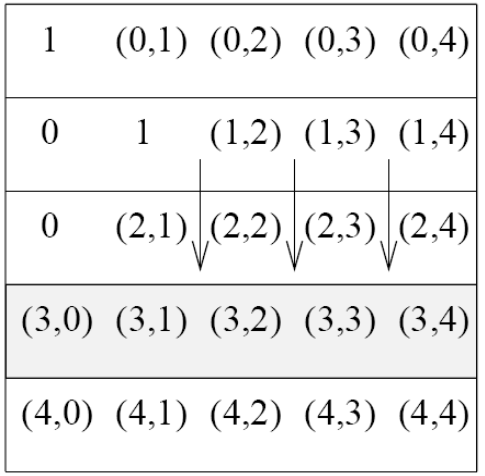
(a) Iteracja k=0 rozpoczyna się (b) (c) (d)

- (a) P_0 dzeili wiersz przez a[0][0] (b) P_0 wysyła wiersz 0 do P_1 (c) P_1 wysyła wiersz 0 do P_2. (d) P_1 przeprowadza eliminację używając wiersza 0 i jednocześnie P_2 wysyła wiersz 0 do P_3.

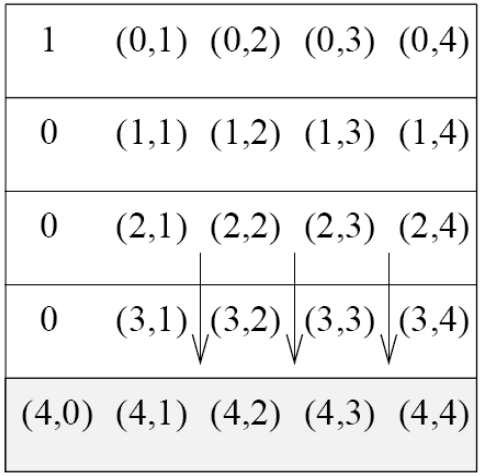
Potokowa wersja eliminacji Gaussa - c.d.



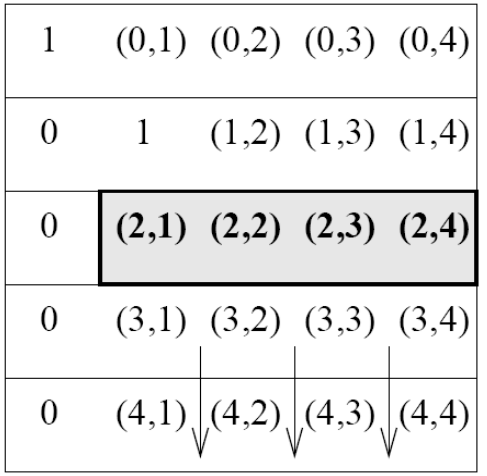
(e) Rozpoczyna się iteracja k=1



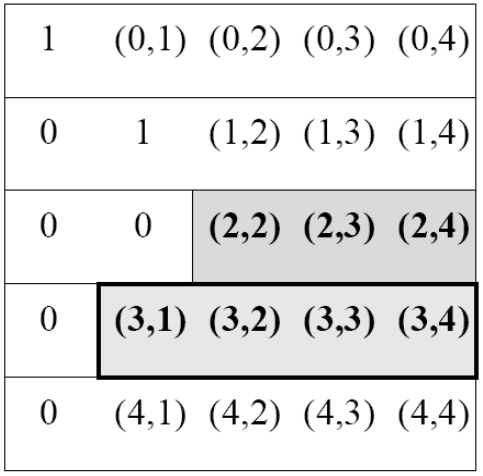
(f)



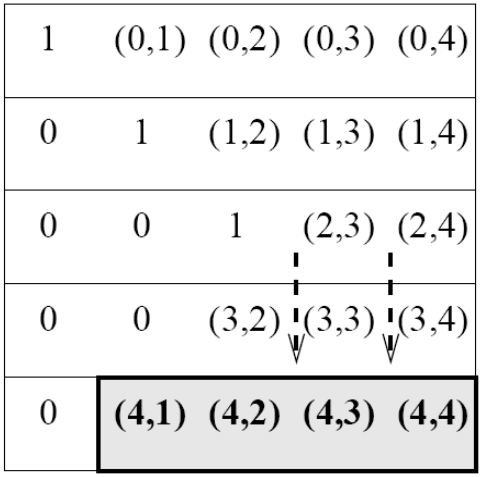
(g) Iteracja k=0 zakończona



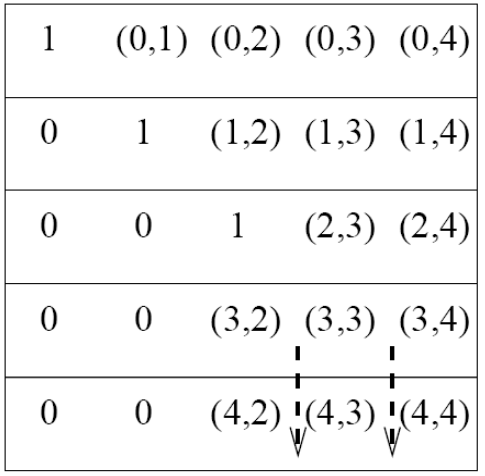
(h)



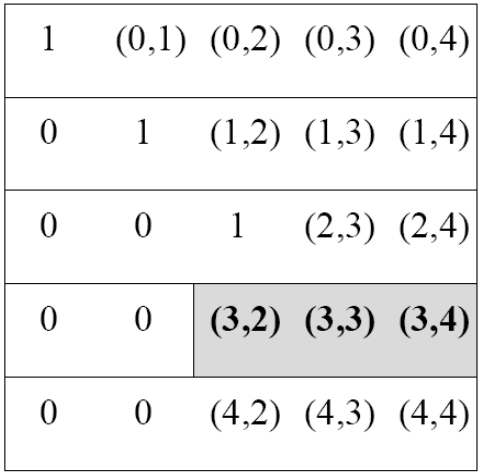
(i) Start iteracji k=2



(j) Koniec iteracji k=1



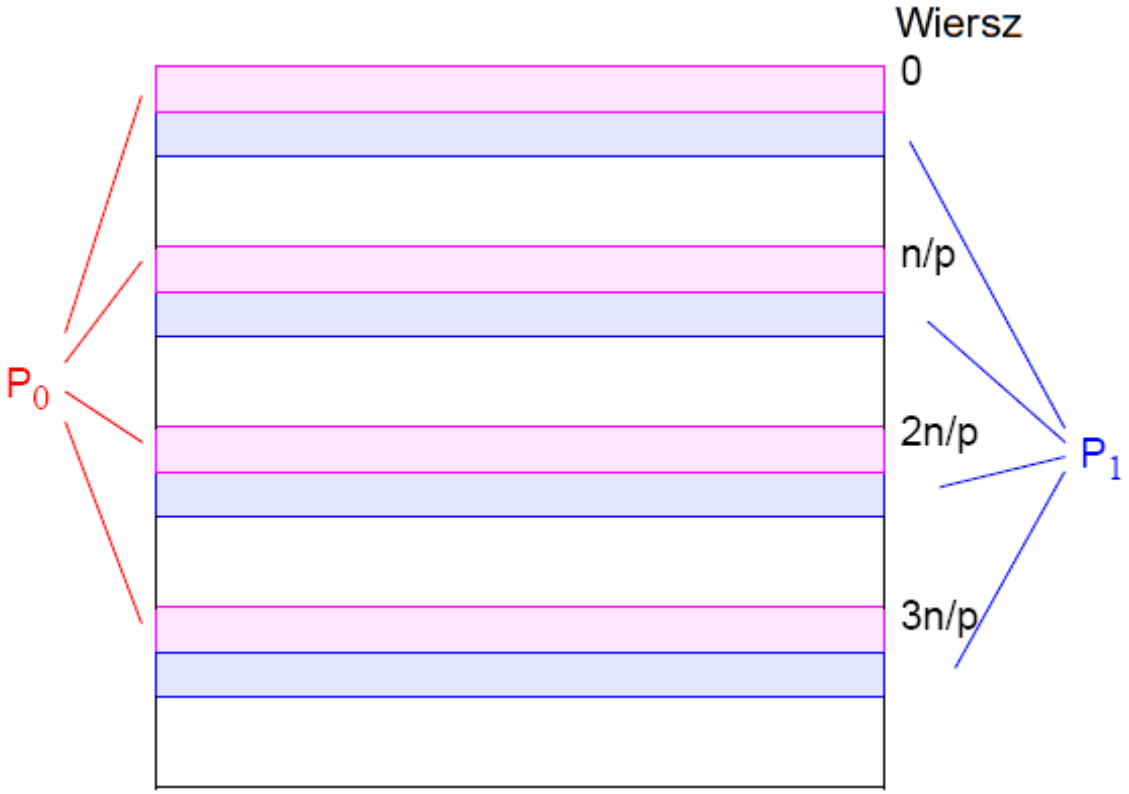
(k)



(l)

- W tym samym czasie na różnych procesorach mogą być aktywne różne iteracje.

Partycjonowanie danych



- n wierszy i p procesorów, n=4*p
- P0 - wiersze 0,n/p/2n/p,3n/p.
- P1 - wiersze 1,n/p+1,2n/p+1,3n/p+1.

- Taki podział danych równoważy obciążenie pomiędzy procesory.

Rozwiązanie układu równań trójkątnych

$$
\begin{aligned}
x_0 + u_{0,1}x_1 + u_{0,2}x_2 + \cdots + u_{0,n-1}x_{n-1} &= y_0, \\
x_1 + u_{1,2}x_2 + \cdots + u_{1,n-1}x_{n-1} &= y_1, \\
&\vdots \qquad\qquad\qquad\quad \vdots \\
x_{n-1} &= y_{n-1}.
\end{aligned}
$$

```
1.  procedure BACK_SUBSTITUTION (U, x, y)
2.  begin
3.      for k := n − 1 downto 0 do   /* Main loop */
4.          begin
5.              x[k] := y[k];
6.              for i := k − 1 downto 0 do
7.                  y[i] := y[i] − x[k] × U[i, k];
8.          endfor;
9.  end BACK_SUBSTITUTION
```

- Kosz $O(n^2)$ w porównaniu z $O(n^3)$ dla eliminacji Gaussa.
- Możliwa implementacja potokowa z czasem O(n) dla n procesorów.

Gotowe procedury numeryczne dla języków C i Fortran

- BLAS (Basic Linear Algebra Subprograms) - operacje wektor-wektor (level 1), macierz-wektor (Level 2) i macierz-macierz (Level 3). Producenci sprzętu dostarczają wersję zoptymalizowane (np. AMD- ACML; Intel - MKL). Istnieje wersja auto-dostrajająca się do procesora (ATLAS).
- LAPACK. Pakiet bibliotek do rozwiązywania układów równań liniowych, zagadnień własnych i innych zagadnień numerycznych. Wykorzystuje operacje BLAS.
- BLACS - dostarcza operacje BLAS dla maszyn z przesyłaniem komunikatów opartych na MPI. Macierze i wektory są rozproszone pomiędzy procesory systemu równoległego.
- ScaLAPACK - oparty na BLACS podzbiór LAPACK dla maszyn z przesyłaniem komunikatów opartych na MPI.

Przykład z macierzą rzadką: cząstkowe równanie różniczkowe Laplace'a

$$\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2}=0$$

- Równanie modeluje między innymi rozkład temperatury w płytce, przy danej temepraturze na brzegach płytki (warunki brzegowe).
- Rozwiązaniem jest funkcja f(x,y).
- Podobnie równanie Poissona.
- Stosuje się metodę różnic skończonych. Definiujemy dwuwymiarową siatkę punktów. Jeżeli odległość Δ pomiędzy punktami jest niewielka to:

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{1}{\Delta^2}[f(x+\Delta, y)-2f(x, y)+f(x-\Delta, y)]$$

$$\frac{\partial^2 f}{\partial y^2} \approx \frac{1}{\Delta^2}[f(x, y+\Delta)-2f(x, y)+f(x, y-\Delta)]$$

Równanie Laplace'a

- Podstawiając do równania otrzymujemy:

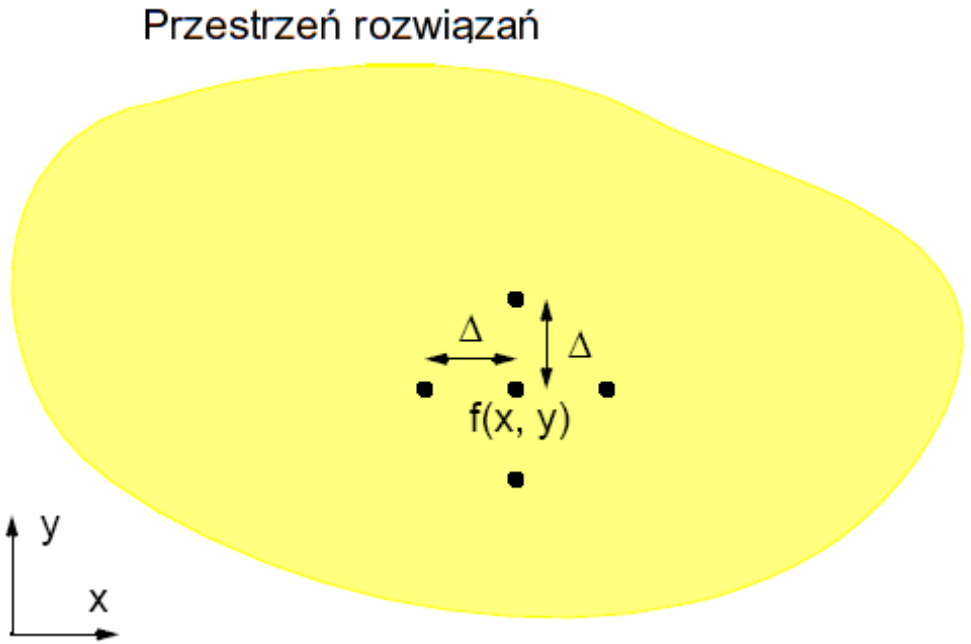
$$f(x,y) = \frac{[f(x-\Delta,y)+f(x,y-\Delta)+f(x+\Delta,y)+f(x,y+\Delta)]}{4}$$

- Możemy to zapisać jako formułę iteracyjną:

$$f^{k}(x,y) = \frac{[f^{k-1}(x-\Delta,y)+f^{k-1}(x,y-\Delta)+f^{k-1}(x+\Delta,y)+f^{k-1}(x,y+\Delta)]}{4}$$

gdzie $f^{k}(x,y)$ wartość funkcji w k-tej iteracji, $f^{k-1}(x,y)$ - wartość funkcji w k-1 iteracji.

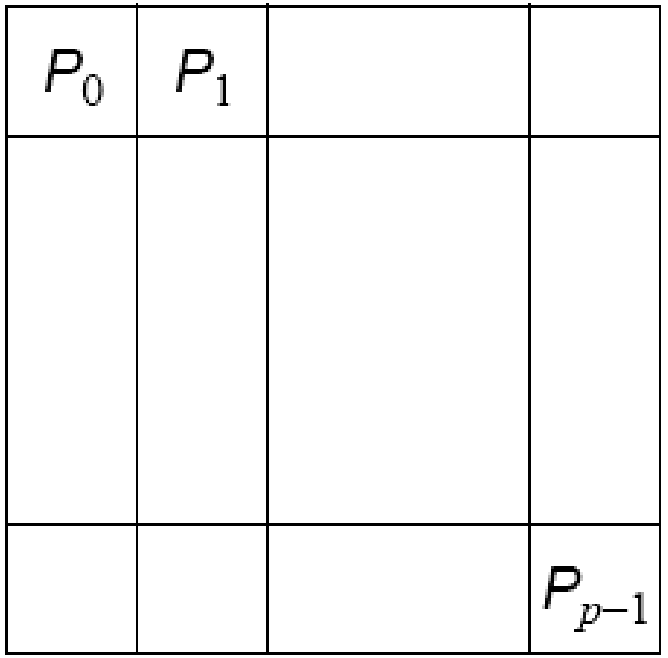
Metoda różnic skończonych



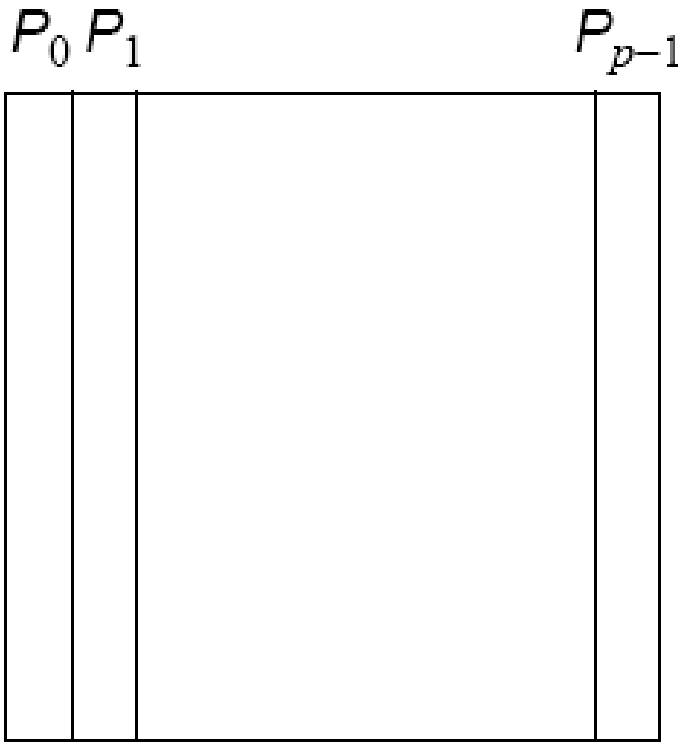
$$f^{k}(x,y) = \frac{[f^{k-1}(x-\Delta,y) + f^{k-1}(x,y-\Delta) + f^{k-1}(x+\Delta,y) + f^{k-1}(x,y+\Delta)]}{4}$$

- Na brzegach obszaru wartości funkcji są określone (warunki brzegowe).
- Załóżmy, że wartości funkcji w k-tej iteracji określone na siatce przechowujemy w macierzy.

Partycjonowanie macierzy pomiędzy procesory

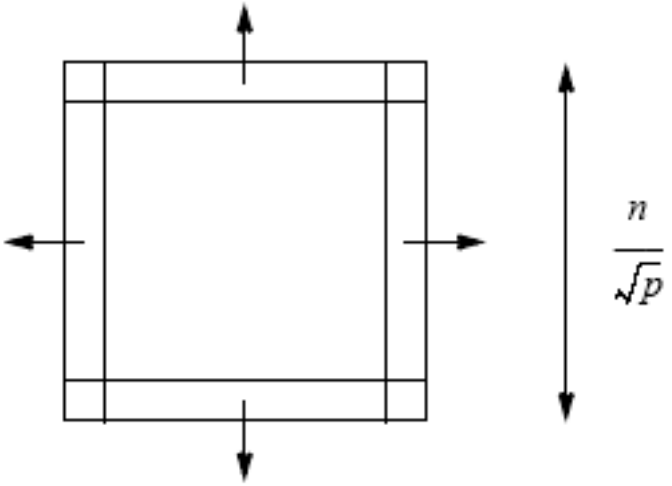


Bloki

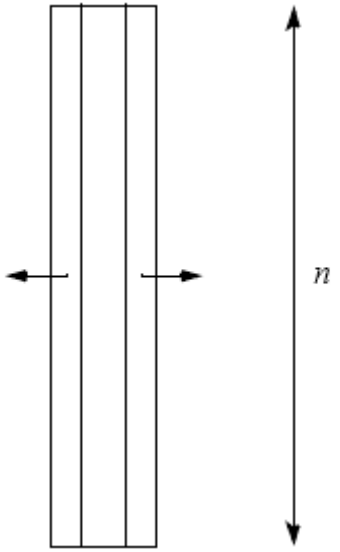


Paski (grupy kolumn)

Komunikacja przy dwóch typach partycjonowania



$$t_c = 8 * \left(t_s + \frac{n}{\sqrt{(p)}} t_W \right)$$



$$t_c = 4 * \left(t_s + n * t_W \right)$$

- Który typ dekompozycji jest lepszy ? Zależy od wartości n,p,t_s,t_w.