Introduction to Parallel Programming

Section 8.

Parallel Methods for Matrix Multiplication

Gergel V.P., Professor, D.Sc.,
Software Department
Contents

- Problem Statement
- Sequential Algorithm
- Algorithm 1 – Block-Striped Decomposition
- Algorithm 2 – Fox’s method
- Algorithm 3 – Cannon’s method
- Summary
Problem Statement

Matrix multiplication:

\[ C = A \cdot B \]

or

\[
\begin{pmatrix}
  c_{0,0}, & c_{0,1}, & \ldots, & c_{0,l-1} \\
  \vdots & & \ddots & \vdots \\
  c_{m-1,0}, & c_{m-1,1}, & \ldots, & c_{m-1,l-1}
\end{pmatrix}
\begin{pmatrix}
  a_{0,0}, & a_{0,1}, & \ldots, & a_{0,n-1} \\
  \vdots & & \ddots & \vdots \\
  a_{m-1,0}, & a_{m-1,1}, & \ldots, & a_{m-1,n-1}
\end{pmatrix}
\begin{pmatrix}
  b_{0,0}, & b_{0,1}, & \ldots, & b_{0,l-1} \\
  \vdots & & \ddots & \vdots \\
  b_{n-1,0}, & b_{n-1,1}, & \ldots, & b_{n-1,l-1}
\end{pmatrix}
\]

The matrix multiplication problem can be reduced to the execution of \(m \cdot l\) independent operations of matrix \(A\) rows and matrix \(B\) columns inner product calculation

\[
c_{ij} = (a_i, b_j^T) = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}, \quad 0 \leq i < m, \quad 0 \leq j < l
\]

\textbf{Data parallelism can be exploited to design parallel computations}
Sequential Algorithm...

```c
// Algorithm 8.1
// Sequential algorithm of matrix multiplication
double MatrixA[Size][Size];
double MatrixB[Size][Size];
double MatrixC[Size][Size];
int i, j, k;
...
for (i=0; i<Size; i++){
    for (j=0; j<Size; j++){
        MatrixC[i][j] = 0;
        for (k=0; k<Size; k++){
        }
    }
}
```
Sequential Algorithm

- Algorithm performs the matrix $C$ rows calculation sequentially.
- At every iteration of the outer loop on $i$ variable a single row of matrix $A$ and all columns of matrix $B$ are processed.

$m \cdot l$ inner products are calculated to perform the matrix multiplication.

The complexity of the matrix multiplication is $O(mnl)$. 
Algorithm 1: Block-Striped Decomposition…

- **A fine-grained approach** – *the basic subtask* is calculation of one element of matrix \( C \)

\[
c_{ij} = (a_i, b^T_j), \quad a_i = (a_{i0}, a_{i1}, ..., a_{in-1}), \quad b^T_j = (b_{0j}, b_{1j}, ..., b_{n-1j})^T
\]

- Number of basic subtasks is equal to \( n^2 \). Achieved parallelism level is redundant!
- As a rule, the number of available processors is less than \( n^2 \) (\( p < n^2 \)), so it will be necessary to perform the subtask scaling
Algorithm 1: Block-Striped Decomposition...

- **The aggregated subtask** – the calculation of one row of matrix \( C \) (the number of subtasks is \( n \))

- **Data distribution** – *rowwise block-striped* decomposition for matrix \( A \) and *columnwise block-striped* decomposition for matrix \( B \)
Algorithm 1: Block-Striped Decomposition…

- Analysis of Information Dependencies…
  - Each subtask hold one row of matrix $A$ and one column of matrix $B$,
  - At every iteration each subtask performs the inner product calculation of its row and column, as a result the corresponding element of matrix $C$ is obtained
  - Then every subtask $i$, $0 \leq i < n$, transmits its column of matrix $B$ for the subtask with the number $(i+1) \mod n$.

After all algorithm iterations all the columns of matrix $B$ were come within each subtask one after another
Algorithm 1: based on block-striped decomposition

Scheme of Information Dependences

[Diagram of information dependences]
Algorithm 1: Block-Striped Decomposition...

- Aggregating and Distributing the Subtasks among the Processors:
  - In case when the number of processors \( p \) is less than the number of basic subtasks \( n \), calculations can be aggregated in such a way that each processor would execute several inner products of matrix \( A \) rows and matrix \( B \) columns. In this case after the completion of computation, each aggregated basic subtask determines several rows of the result matrix \( C \),
  - Under such conditions the initial matrix \( A \) is decomposed into \( p \) horizontal stripes and matrix \( B \) is decomposed into \( p \) vertical stripes,
  - Subtasks distribution among the processors have to meet the requirements of effective representation of the ring structure of subtask information dependencies
**Algorithm 1: Block-Striped Decomposition…**

- **Efficiency Analysis…**
  - Speed-up and Efficiency generalized estimates

\[
S_p = \frac{n^3}{(n^3/p)} = p \quad E_p = \frac{n^3}{p \cdot (n^3/p)} = 1
\]

*Developed method of parallel computations allows to achieve ideal speed-up and efficiency characteristics*
Efficiency Analysis (detailed estimates):
- Time of parallel algorithm execution, that corresponds to the processor calculations:
  \[ T_p^{(calc)} = \frac{n^2}{p} \cdot (2n - 1) \cdot \tau \]
- Time of the data transmission operations can be obtained by means of the Hockney model:
  \[ T_p^{(comm)} = (p - 1) \cdot (\alpha + w \cdot n \cdot \frac{n}{p}) / \beta \]
  (we assume that all the data transmission operations between the processors during one iteration can be executed in parallel)

Total time of parallel algorithm execution is:
\[ T_p = \frac{n^2}{p} (2n - 1) \cdot \tau + (p - 1) \cdot (\alpha + w \cdot n \cdot \frac{n}{p}) / \beta \]
**Algorithm 1: Block-Striped Decomposition…**

- **Results of computational experiments…**
  - Comparison of theoretical estimations and results of computational experiments

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>2 processors</th>
<th>4 processors</th>
<th>8 processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model</td>
<td>Experiment</td>
<td>Model</td>
</tr>
<tr>
<td>500</td>
<td>0,8243</td>
<td>0,3758</td>
<td>0,4313</td>
</tr>
<tr>
<td>1000</td>
<td>6,51822</td>
<td>5,4427</td>
<td>3,3349</td>
</tr>
<tr>
<td>1500</td>
<td>21,9137</td>
<td>20,9503</td>
<td>11,1270</td>
</tr>
<tr>
<td>2000</td>
<td>51,8429</td>
<td>45,7436</td>
<td>26,2236</td>
</tr>
<tr>
<td>2500</td>
<td>101,1377</td>
<td>99,5097</td>
<td>51,0408</td>
</tr>
<tr>
<td>3000</td>
<td>174,6301</td>
<td>171,9232</td>
<td>87,9946</td>
</tr>
</tbody>
</table>

![Graph showing computational time vs. matrix size](image-url)
Algorithm 1: Block-Striped Decomposition...

- Results of computational experiments:
  - Speedup

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Serial Algorithm</th>
<th>2 processors</th>
<th>4 processors</th>
<th>8 processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Speed Up</td>
<td>Time</td>
</tr>
<tr>
<td>500</td>
<td>0.8752</td>
<td>0.3758</td>
<td>2.3287</td>
<td>0.1535</td>
</tr>
<tr>
<td>1000</td>
<td>12.8787</td>
<td>5.4427</td>
<td>2.3662</td>
<td>2.2628</td>
</tr>
<tr>
<td>1500</td>
<td>43.4731</td>
<td>20.9503</td>
<td>2.0750</td>
<td>11.0804</td>
</tr>
<tr>
<td>2500</td>
<td>201.2915</td>
<td>99.5097</td>
<td>2.0228</td>
<td>56.9203</td>
</tr>
<tr>
<td>3000</td>
<td>347.8434</td>
<td>171.9232</td>
<td>2.0232</td>
<td>111.9642</td>
</tr>
</tbody>
</table>

![Graph showing speedup vs. number of processors]

- Algorithm 1: Block-Striped Decomposition
Algorithm 1': Block-Striped Decomposition…

- Another possible approach for the data distribution is the rowwise block-striped decomposition for matrices \( A \) and \( B \)
Algorithm 1': Block-Striped Decomposition…

- Analysis of Information Dependencies
  - Each subtask holds one row of matrix $A$ and one row of matrix $B$,
  - At every iteration the subtasks perform the element-to-element multiplications of the rows; as a result the row of partial results for matrix $C$ is obtained,
  - Then every subtask $i$, $0 \leq i < n$, transmits its row of matrix $B$ for the subtask with the number $(i+1) \mod n$.

*After all algorithm iterations all rows of matrix $B$ were come within every subtask one after another*
Algorithm 1': Block-Striped Decomposition

Scheme of Information Dependences
Algorithm 2: Fox’s method…

- **Data distribution** – *checkerboard scheme*

- **Basic subtask** is a procedure, that calculates all elements of one block of matrix \( C \)

\[
\begin{pmatrix}
A_{00}A_{01}\ldots A_{0q-1} \\
\ldots \\
A_{q-10}A_{q-11}\ldots A_{q-1q-1}
\end{pmatrix}
\times
\begin{pmatrix}
B_{00}B_{01}\ldots B_{0q-1} \\
\ldots \\
B_{q-10}B_{q-11}\ldots B_{q-1q-1}
\end{pmatrix}
= \begin{pmatrix}
C_{00}C_{01}\ldots C_{0q-1} \\
\ldots \\
c_{q-10}C_{q-11}\ldots C_{q-1q-1}
\end{pmatrix},
\]

\[C_{ij} = \sum_{s=0}^{q-1} A_{is}B_{sj}\]
Algorithm 2: Fox’s method…

Analysis of Information Dependencies

- Subtask with $(i,j)$ number calculates the block $C_{ij}$ of the result matrix $C$. As a result, the subtasks form the $q \times q$ two-dimensional grid,
- Each subtask holds 4 matrix blocks:
  - block $C_{ij}$ of the result matrix $C$, which is calculated in the subtask,
  - block $A_{ij}$ of matrix $A$, which was placed in the subtask before the calculation starts,
  - blocks $A_{ij}'$ and $B_{ij}'$ of matrix $A$ and matrix $B$, that are received by the subtask during calculations.
Algorithm 2: Fox’s method…

- **Analysis of Information Dependencies** — during iteration \( l \), \( 0 \leq l < q \), algorithm performs:
  - The subtask \((i, j)\) transmits its block \( A_{ij} \) of matrix \( A \) to all subtasks of the same horizontal row \( i \) of the grid; the \( j \) index, which determines the position of the subtask in the row, can be obtained using equation:
    \[
    j = (i + l) \mod q,
    \]
    where \( \mod \) operation is the procedure of calculating the remainder of integer-valued division,
  - Every subtask performs the multiplication of received blocks \( A'_{ij} \) and \( B'_{ij} \) and adds the result to the block \( C_{ij} \)
    \[
    C'_{ij} = C_{ij} + A'_{ij} \times B'_{ij}
    \]
  - Every subtask \((i,j)\) transmits its block \( B'_{ij} \) to the neighbor, which is previous in the same vertical line (the blocks of subtasks of the first row are transmitted to the subtasks of the last row of the grid).
Algorithm 2: Fox’s method…

Scheme of Information Dependences

First Iteration

Second Iteration
Algorithm 2: Fox’s method..

- Scaling and Distributing the Subtasks among the Processors
  - The sizes of the matrices blocks can be selected so that the number of subtasks will coincides the number of available processors $p$,
  - The most efficient execution of the parallel the Fox’s algorithm can be provided when the communication network topology is a two-dimensional grid,
  - In this case the subtasks can be distributed among the processors in a natural way: the subtask $(i,j)$ has to be placed to the $p_{i,j}$ processor
Algorithm 2: Fox’s method...

- Efficiency Analysis...

  Speed-up and Efficiency generalized estimates

\[
S_p = \frac{n^2}{n^2 / p} = p \\
E_p = \frac{n^2}{p \cdot (n^2 / p)} = 1
\]

Developed method of parallel computations allows to achieve ideal speed-up and efficiency characteristics.
Algorithm 2: Fox’s method

Efficiency Analysis (detailed estimates):

- Time of parallel algorithm execution, that corresponds to the processor calculations:
  \[ T_p^{(calc)} = q \left[ \frac{n^2}{p} \cdot \left(2n/q - 1\right) + \frac{n^2}{p} \right] \cdot \tau \]

- At every iteration one of the processors in each row of the processors grid transmits its block of matrix \( A \) to the rest processors in the row:
  \[ T_p^{(comm)} = \log_2 q \left( \alpha + w \frac{n^2}{p} / \beta \right) \]

- After the matrix block multiplication each processor transmits its blocks of matrix \( B \) to its neighbor in the vertical column:
  \[ T_p^{(comm)} = \alpha + w \frac{n^2}{p} / \beta \]

Total time of parallel algorithm execution is:

\[ T_p = q \left[ \frac{n^2}{p} \cdot \left(2n/q - 1\right) + \frac{n^2}{p} \right] \cdot \tau + \left( q \log_2 q + (q - 1) \right) \frac{(\alpha + w \frac{n^2}{p})}{\beta} \]
Algorithm 2: Fox’s method…

- Description of the parallel program sample…
  - The main function
    - implements the computational method scheme by sequential calling out the necessary subprograms

Code
Algorithm 2: Fox’s method…

- Description of the parallel program sample…
  - The function `CreateGridCommunicators`:
    - creates a communicator as a two-dimensional square grid,
    - determines the coordinates of each process in the grid,
    - creates communicators for each row and each column separately.

Code
Algorithm 2: Fox’s method...

- Description of the parallel program sample...
  - The function \textit{ProcessInitialization}:
    - sets the matrix sizes,
    - allocates memory for storing the initial matrices and their blocks,
    - initializes all the original problem data (in order to determine the elements of the initial matrices we will use the functions \textit{DummyDataInitialization} and \textit{RandomDataInitialization})

\textbf{Code}
Algorithm 2: Fox’s method…

- Description of the parallel program sample…
  - The function *ParallelResultCalculation*:
    - executes the parallel Fox algorithm of matrix multiplication (the matrix blocks and their sizes must be given to the function as its arguments)

Code
Algorithm 2: Fox’s method...

Description of the parallel program sample...

- **Iteration execution**: transmission of matrix \( A \) blocks to the processors in the same row of the processors grid (the function *AblockCommunication*):
  
  - The number of sending processor *Pivot* is determined, then the data transmission is performed,
  
  - For the transmission the block \( pA\text{block} \) of matrix \( A \) is used. This block was placed on the processor before the calculation started,
  
  - The block transmission is performed by means the function *MPI_Bcast* (the communicator *RowComm* is used).

Code
Algorithm 2: Fox’s method...

- **Description of the parallel program sample...**
  
  - **Iteration execution**: Matrix block multiplication is executed (the function *BlockMultiplication*):
    
    - This function multiplies the block of matrix $A$ ($pAblock$) and the block of matrix $B$ ($pBblock$) and adds the result to the block of matrix $C$

**Code**
Algorithm 2: Fox’s method…

- **Description of the parallel program sample**
  - **Iteration execution**: Cycle shift of the matrix B blocks along the columns of the processor grid is implemented (the function *BblockCommunication*):
    - Every processor transmits its block to the processor with the number *NextProc* in the grid column,
    - Every processor receives the block, which was sent by the processor with the number *PrevProc* from the same grid column,
    - To perform such actions the function *MPI_SendRecv_replace* is used, which provides all necessary block transmissions and uses only single memory buffer *pBblock*.

**Code**
Algorithm 2: Fox’s method…

- Results of computational experiments…
  - Comparison of theoretical estimations and results of computational experiments

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>4 processors</th>
<th>9 processors</th>
<th>4 processors</th>
<th>9 processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model</td>
<td>Experiment</td>
<td>Model</td>
<td>Experiment</td>
</tr>
<tr>
<td>500</td>
<td>0,4217</td>
<td>0,2190</td>
<td>0,2200</td>
<td>0,1468</td>
</tr>
<tr>
<td>1000</td>
<td>3,2970</td>
<td>3,0910</td>
<td>1,5924</td>
<td>2,1565</td>
</tr>
<tr>
<td>1500</td>
<td>11,0419</td>
<td>10,8678</td>
<td>5,1920</td>
<td>7,2502</td>
</tr>
<tr>
<td>2000</td>
<td>26,0726</td>
<td>24,1421</td>
<td>12,0927</td>
<td>21,4157</td>
</tr>
<tr>
<td>2500</td>
<td>50,8049</td>
<td>51,4735</td>
<td>23,3682</td>
<td>41,2159</td>
</tr>
<tr>
<td>3000</td>
<td>87,6548</td>
<td>87,0538</td>
<td>40,0923</td>
<td>58,2022</td>
</tr>
</tbody>
</table>

![Graph: Comparison of theoretical estimations and results of computational experiments]
Algorithm 2: Fox’s method

Results of computational experiments:

- Speedup

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Serial Algorithm</th>
<th>Parallel Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Speed Up</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>Speed Up</td>
</tr>
<tr>
<td>500</td>
<td>0,8527</td>
<td>0,2190</td>
</tr>
<tr>
<td>1000</td>
<td>12,8787</td>
<td>3,0910</td>
</tr>
<tr>
<td>1500</td>
<td>43,4731</td>
<td>10,8678</td>
</tr>
<tr>
<td>2000</td>
<td>103,0561</td>
<td>24,1421</td>
</tr>
<tr>
<td>2500</td>
<td>201,2915</td>
<td>51,4735</td>
</tr>
<tr>
<td>3000</td>
<td>347,8434</td>
<td>87,0538</td>
</tr>
</tbody>
</table>
Algorithm 3: Cannon’s Method…

- **Data distribution** – _Checkerboard scheme_

\[
\begin{array}{c}
\text{A} \\
\begin{array}{c|c|c|c}
\hline
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\hline
\end{array}
\end{array}
\quad \times \quad
\begin{array}{c}
\text{B} \\
\begin{array}{c|c|c|c}
\hline
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\hline
\end{array}
\end{array}
= \begin{array}{c}
\text{C} \\
\begin{array}{c|c|c|c}
\hline
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\hline
\end{array}
\end{array}
\]

- **Basic subtask** is a procedure, that calculates all elements of one block of matrix _C_

\[
\begin{pmatrix}
A_{00}A_{01}\ldots A_{0q-1} \\
\vdots \\
A_{q-10}A_{q-11}\ldots A_{q-1q-1}
\end{pmatrix}
\times
\begin{pmatrix}
B_{00}B_{01}\ldots B_{0q-1} \\
\vdots \\
B_{q-10}B_{q-11}\ldots B_{q-1q-1}
\end{pmatrix}
= \begin{pmatrix}
C_{00}C_{01}\ldots C_{0q-1} \\
\vdots \\
c_{q-10}C_{q-11}\ldots C_{q-1q-1}
\end{pmatrix}, \quad C_{ij} = \sum_{s=0}^{q-1} A_{is} B_{sj}
\]
Algorithm 3: Cannon’s Method…

- **Analysis of Information Dependencies:**
  - The subtask with the number \((i,j)\) calculates the block \(C_{ij}\), of the result matrix \(C\). As a result, the subtasks form the \(q \times q\) two-dimensional grid,
  - The initial distribution of matrix blocks in Cannon’s algorithm is selected in such a way that the first block multiplication can be performed without additional data transmission:
    - At the beginning each subtask \((i,j)\) holds the blocks \(A_{ij}\) and \(B_{ij}\),
    - For the \(i\)-th row of the subtasks grid the matrix \(A\) blocks are shifted for \((i-1)\) positions to the left,
    - For the \(j\) –th column of the subtasks grid the matrix \(B\) blocks are shifted for \((j-1)\) positions upward,
  - Data transmission operations are the example of the *circular shift* communication
Algorithm 3: Cannon’s Method…

- Redistribution of matrix blocks on the first stage of the algorithm

\[
\begin{array}{c}
A_{0,0} \\
B_{0,0}
\end{array}
\quad \begin{array}{c}
A_{0,1} \\
B_{0,1}
\end{array}
\quad \begin{array}{c}
A_{0,2} \\
B_{0,2}
\end{array}
\quad \begin{array}{c}
A_{0,0} \\
C_{0,0}=0
\end{array}

\begin{array}{c}
A_{0,1} \\
C_{0,1}=0
\end{array}
\quad \begin{array}{c}
A_{0,2} \\
C_{0,2}=0
\end{array}
\quad \begin{array}{c}
A_{0,0} \\
C_{0,0}=0
\end{array}
\quad \begin{array}{c}
A_{0,1} \\
C_{0,1}=0
\end{array}
\quad \begin{array}{c}
A_{0,2} \\
C_{0,2}=0
\end{array}

\begin{array}{c}
A_{1,0} \\
B_{1,0}
\end{array}
\quad \begin{array}{c}
A_{1,1} \\
B_{1,1}
\end{array}
\quad \begin{array}{c}
A_{1,2} \\
B_{1,2}
\end{array}
\quad \begin{array}{c}
A_{1,1} \\
C_{1,1}=0
\end{array}

\begin{array}{c}
A_{1,2} \\
C_{1,2}=0
\end{array}
\quad \begin{array}{c}
A_{1,0} \\
C_{1,0}=0
\end{array}
\quad \begin{array}{c}
A_{1,1} \\
C_{1,1}=0
\end{array}
\quad \begin{array}{c}
A_{1,2} \\
C_{1,2}=0
\end{array}
\quad \begin{array}{c}
A_{1,0} \\
C_{1,0}=0
\end{array}
\quad \begin{array}{c}
A_{1,1} \\
C_{1,1}=0
\end{array}
\quad \begin{array}{c}
A_{1,2} \\
C_{1,2}=0
\end{array}

\begin{array}{c}
A_{2,0} \\
B_{2,0}
\end{array}
\quad \begin{array}{c}
A_{2,1} \\
B_{2,1}
\end{array}
\quad \begin{array}{c}
A_{2,2} \\
B_{2,2}
\end{array}
\quad \begin{array}{c}
A_{2,0} \\
B_{2,0}
\end{array}

\begin{array}{c}
A_{2,2} \\
B_{2,2}
\end{array}
\quad \begin{array}{c}
A_{2,0} \\
B_{0,1}
\end{array}
\quad \begin{array}{c}
A_{2,1} \\
B_{0,2}
\end{array}
\quad \begin{array}{c}
A_{2,1} \\
B_{1,2}
\end{array}
\quad \begin{array}{c}
A_{2,2} \\
C_{2,0}=0
\end{array}
\quad \begin{array}{c}
A_{2,0} \\
C_{2,0}=0
\end{array}
\quad \begin{array}{c}
A_{2,1} \\
C_{2,1}=0
\end{array}
\quad \begin{array}{c}
A_{2,2} \\
C_{2,2}=0
\end{array}
\quad \begin{array}{c}
A_{2,0} \\
C_{2,0}=0
\end{array}
\quad \begin{array}{c}
A_{2,1} \\
C_{2,1}=0
\end{array}
\quad \begin{array}{c}
A_{2,2} \\
C_{2,2}=0
\end{array}
\end{array}
 Algorithm 3: Cannon’s Method…

- Analysis of Information Dependencies:
  - After the redistribution, which was performed at the first stage, the matrix blocks can be multiplied without additional data transmission operations,
  - To obtain all of the rest blocks after the operation of blocks multiplication:
    - Matrix $A$ blocks are shifted for one position left along the grid row,
    - Matrix $B$ blocks are shifted for one position upward along the grid column.
Algorithm 3: Cannon’s Method…

- Scaling and Distributing the Subtasks among the Processors:
  - The sizes of the matrices blocks can be selected so that the number of subtasks will coincides the number of available processors $p$,
  - The most efficient execution of the parallel Canon’s algorithm can be provided when the communication network topology is a two-dimensional grid,
  - In this case the subtasks can be distributed among the processors in a natural way: the subtask $(i,j)$ has to be placed to the $p_{i,j}$ processor
Algorithm 3: Cannon’s Method...

- Efficiency Analysis...

- Speed-up and Efficiency generalized estimates

\[ S_p = \frac{n^2}{n^2 / p} = p \]

\[ E_p = \frac{n^2}{p \cdot (n^2 / p)} = 1 \]

Developed method of parallel computations allows to achieve ideal speed-up and efficiency characteristics
Algorithm 3: Cannon’s Method…

Efficiency Analysis (detailed estimates):
- The Cannon’s algorithm differs from the Fox’s algorithm only in the types of communication operations, that is why:
  \[ T_p^{(calc)} = (\frac{n^2}{p}) \cdot (2n - 1) \cdot \tau \]
- Time of the initial redistribution of matrices blocks:
  \[ T_p^{(comm)} = 2 \cdot \left( \frac{\alpha + w \cdot (\frac{n^2}{p}) / \beta}{\alpha} \right) \]
- After every multiplication operation matrix blocks are shifted:
  \[ T_p^{(comm)} = 2 \cdot \left( \frac{\alpha + w \cdot (\frac{n^2}{p}) / \beta}{\alpha} \right) \]

Total time of parallel algorithm execution is:

\[ T_p = q \left[ (\frac{n^2}{p}) \cdot \left( \frac{2n}{q} - 1 \right) + (\frac{n^2}{p}) \right] \cdot \tau + (2q + 2) \left( \frac{\alpha + w (\frac{n^2}{p}) / \beta}{\alpha} \right) \]
Algorithm 3: Cannon’s Method…

- Results of computational experiments…
  - Comparison of theoretical estimations and results of computational experiments

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>4 processors</th>
<th>9 processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model</td>
<td>Experiment</td>
</tr>
<tr>
<td>1000</td>
<td>3,4485</td>
<td>3,0806</td>
</tr>
<tr>
<td>1500</td>
<td>11,3821</td>
<td>11,1716</td>
</tr>
<tr>
<td>2000</td>
<td>26,6769</td>
<td>24,0502</td>
</tr>
<tr>
<td>2500</td>
<td>51,7488</td>
<td>53,1444</td>
</tr>
<tr>
<td>3000</td>
<td>89,0138</td>
<td>88,2979</td>
</tr>
</tbody>
</table>

![Comparison of theoretical estimations and results of computational experiments](chart.png)
Algorithm 3: Cannon’s Method

Results of computational experiments:

- Speedup

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Serial Algorithm</th>
<th>Parallel Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>4 processors</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>Speed Up</td>
</tr>
<tr>
<td>1000</td>
<td>12,8787</td>
<td>3,0806</td>
</tr>
<tr>
<td>1500</td>
<td>43,4731</td>
<td>11,1716</td>
</tr>
<tr>
<td>2000</td>
<td>103,0561</td>
<td>24,0502</td>
</tr>
<tr>
<td>2500</td>
<td>201,2915</td>
<td>53,1444</td>
</tr>
<tr>
<td>3000</td>
<td>347,8434</td>
<td>88,2979</td>
</tr>
</tbody>
</table>

Graph showing speedup vs. number of processors for different matrix sizes.
Summary…

- Three parallel algorithms for matrix multiplication are discussed:
  - Algorithm 1 - Block-Striped Decomposition
  - Algorithm 2 – Fox’s method (checkerboard decomposition),
  - Algorithm 3 – Cannon’s method (checkerboard decomposition),
- The parallel program sample for the Fox’s algorithm is described
- Theoretical analysis allows to predict the speed-up and efficiency characteristics of parallel computations with sufficiently high accuracy
Summary

- All presented algorithms have nearly the same theoretical estimations for speed-up and efficiency characteristics.
Discussions

- What sequential algorithms for matrix multiplication are known? What is the complexity of these algorithms?
- What basic approaches can be used for developing the parallel algorithms for matrix multiplication?
- What algorithm possesses the best speedup and efficiency characteristics?
- Which one of the described algorithms requires the maximal and minimal size of memory?
- What data communication operations are necessary for the parallel algorithms of matrix multiplication?
Exercises

- Develop the parallel programs for two algorithms of matrix multiplication based on block-striped decomposition. Compare the time of their execution.
- Develop the parallel program for the Cannon’s algorithm.
- Formulate the theoretical estimations for the execution time of these algorithms.
- Execute programs. Compare the time of computational experiments and the theoretical estimations being obtained.
References


Next Section

- Parallel Methods for Solving Linear Systems
Author’s Team

Gergel V.P., Professor, Doctor of Science in Engineering, Course Author
Grishagin V.A., Associate Professor, Candidate of Science in Mathematics
Abrosimova O.N., Assistant Professor (chapter 10)
Kurylev A.L., Assistant Professor (learning labs 4,5)
Labutin D.Y., Assistant Professor (ParaLab system)
Sysoev A.V., Assistant Professor (chapter 1)
Gergel A.V., Post-Graduate Student (chapter 12, learning lab 6)
Labutina A.A., Post-Graduate Student (chapters 7,8,9, learning labs 1,2,3, ParaLab system)
Senin A.V., Post-Graduate Student (chapter 11, learning labs on Microsoft Compute Cluster)
Liverko S.V., Student (ParaLab system)
The purpose of the project is to develop the set of educational materials for the teaching course “Multiprocessor computational systems and parallel programming”. This course is designed for the consideration of the parallel computation problems, which are stipulated in the recommendations of IEEE-CS and ACM Computing Curricula 2001. The educational materials can be used for teaching/training specialists in the fields of informatics, computer engineering and information technologies. The curriculum consists of the training course “Introduction in the methods of parallel programming” and the computer laboratory training “The methods and technologies of parallel program development”. Such educational materials makes possible to seamlessly combine both the fundamental education in computer science and the practical training in the methods of developing the software for solving complicated time-consuming computational problems using the high performance computational systems.

The project was carried out in Nizhny Novgorod State University, the Software Department of the Computing Mathematics and Cybernetics Faculty (http://www.software.unn.ac.ru). The project was implemented with the support of Microsoft.