УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

Документација пројекта

Студент: Невена Прокић
Број индекса: SW6/2019

Предмет: Паралелно Програмирање
Тема: Множење матрица

Ментор рада: проф. др Мирослав Поповић, Миа Степановић

Нови Сад, јун, 2021.

Садржај

1. Увод

1.1. Множење матрица

Нека су *m, n* природни бројеви. Под матрицом формата $m \times n$ над скупом R подразумевамо сваку уређену *m*-торку уређених *n*-торки елемената скупа R, коју ћемо представити на следећи начин:

$$A=\begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1j} & \cdots & \alpha_{1n} \\ \vdots & & & & \\ \alpha_{i1} & & \alpha_{ij} & & \alpha_{in} \\ \vdots & & & & \\ \alpha_{m1} & \cdots & \alpha_{mj} & \cdots & \alpha_{mn} \end{bmatrix}$$

Производ матрица A и B дефинишемо, ако је испуњен услов да су формати матрица редом $m \times p$ и $p \times n$ (тј. број колона матрице A је једнак броју врста матрице B), као матрицу $A \cdot B$ формата $m \times n$ над скупом R, чија је (i, j)

координата једнака скаларном производу *i* -те врсте матрице A и *j*-те колоне матрице B, тј.:

$$[A \cdot B]_{ij} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \cdots & \cdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mp} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \cdots & \cdots & \vdots \\ b_{p1} & b_{p2} & \cdots & b_{pn} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11} \cdot b_{11} + \cdots + a_{1p} \cdot b_{p1} & \cdots & a_{11} \cdot b_{1n} + \cdots + a_{1p} \cdot b_{pn} \\ \vdots & \cdots & \vdots \\ \vdots & \cdots & \vdots \\ a_{m1} \cdot b_{11} + \cdots + a_{mp} b_{p1} & \cdots & a_{m1} \cdot b_{p1} + \cdots + a_{mp} \cdot b_{pn} \end{bmatrix}$$

Производ матрица A и B није дефинисан ако број колона матрице A није једнак броју врста матрице B. Ако постоји матрица A · B, тада она има врста колико и матрица A, а колона колико и матрица B.

1.2. Задатак

Учитавање матрице треба да учита садржај улазне датотеке и формира две матрице које касније треба помножити. Такође треба проверити да ли су улазне матрице добро формиране, то значи да свака колона има исти број елемената (аналогно треба проверити и за редове).

Множење матрица треба најпре да провери да ли је услов дефинисаности производа две матрице испуњен. Уколико је овај услов испуњен извршава се множење матрица.

Упис резултујуће матрице треба да упише у текстуални фајл матрицу која је производ улазних матрица.

Тестирање самог програма ће бити извршено уз помоћ функција које проверавају да ли се добијено решење поклапа са тачним решењем који ће се налазити у засебној датотеци (correctResultsmxn).

2. Анализа проблема

Проблеми које уочавамо приликом анализе јесу:

1. Имплементација алгоритма серијски
2. Паралелизација серијског кода
3. Верификација серијске и паралелне имплементације

Најпре треба приступити имплементацији серијског алгоритма који треба да буде са јасно одвојеним блоковима обраде. Овакав начин имплементације ће нам касније олакшати паралелизацију алгоритма.

За имплементацију паралелног алгоритма потребно је видети која је одговарајућа врста паралелизације (декомпозиција задатака, података ...). Такође, треба уочити блокове који се могу паралелизовати. Затим је потребно одредити начин на који ћемо поделити задатке, ту долазимо и до проблема синхронизације и комуникације. Ове проблеме треба решити неком од могућих метода. Приступ свим дељеним променљивама треба да буде искључив, овога се треба придржавати и приликом паралелизације података и између задатака.

3. Концепт решења

Приликом реализације пралелизма потребно је уочити различите верзије.

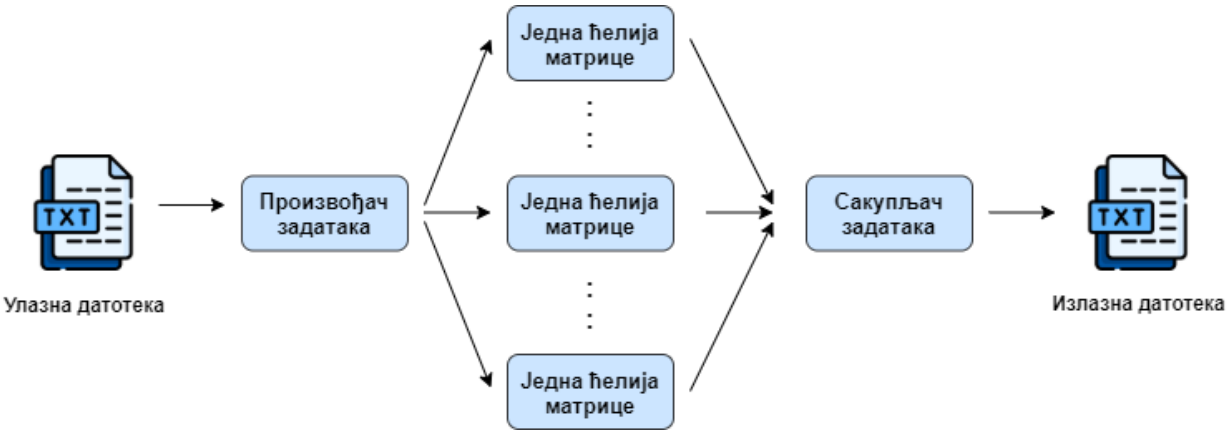
3.1. `parallel_for`

Уз помоћ овог шаблона можемо поделити матрицу на делове над којима ће се множење извршавати у одвојеним независним нитима. Потребно је преклапање *operatora ( )*, које мора бити const због спречавања завистности између итерација. Такође треба искористити *blocked_range2d*, template класа у TBB библиотеци. Она преставља дводимензиони итерациони простор са доњим и горњим границама.

3.2. `task_group`

Случај имплементације паралелизма уз помоћ TBB задатака представља распоређивање множење матрице на мање задатке који се засебно извршавају. Задатаке можемо поделити на следеће начине:

1. Једна ћелија матрице је један задатак
2. Једна колона (врста) је један задатак
3. Број језгара на рачунару представља број задатака



Произвођач задатака ће формирати број затака који је одређен из услова који начин поделе задатака се тренутно извршава. Сваки задатак се посебно извршава. Након извршења свих задатака сакупљач може извршити синхронизацију и покупити решења појединачних задатака. Сва решења ће формирати коначно решење које, у нашем случају, треба да се поклапа са серијским решењем проблема.

4. Опис решења

4.1. Секвенцијалан програм - модули и основне методе

4.1.1. Модул главног програма(MainProgram)

```
int mainSerial(int argc, char* argv[])
```

Почетна функција програма. Покреће све функције програма, приказује информације и време извршавање програма.

4.1.2. Модул за рад са датотекама (WorkWithFile)

Модул у коме се налази рад са датотекама.

4.1.2.1. Метода за читање улазне датотеке

```
MatrixDTO readFile(char* fileNameInput)
```

Метода учитава улазни фајл и формира две матрице.
Параметри:

- fileNameInput - име улазног фајла

Повратна вредност:

- DTO објекат који преноси учитане матрице из фајла

```
void readMatrix::makeMatrix(string line, int number,
MatrixDTO& m)
```

Метода која формира матрице обрадом сваке линије која јој је прослеђена.

Параметри:

- line - текућа линија из улазног фајла
- number - представља број матрице која се тренутно чита
- m - DTO објекат у ком ће бити смештена учитана матрица

4.1.2.2. Метода за упис излазне датотеке

```
void writeFile(char* fileNameOutput,
vector<vector<int>> matrix)
```

Метода која уписује резултат у излазни фајл.
Параметри:

- fileNameOutput - име излазног фајла
- vector<vector<int>> matrix - резултујућа матрица

4.1.3. Модул за рад са матрицама (Matrix)

Модул садржи функције које раде са матрицама.

4.1.3.1. Функција провере формата

```
void Matrix::checkCorrectness()
```

Функција проверава да ли су учитане матрице коректног формата, тј. да ли свака колона има исти број елемената (аналогно треба проверити и за редове).

4.1.3.2. Функција за множење матрица

```
matrixMultiplication(Matrix& matrix1,Matrix& matrix2,
Matrix& matrix3)
```

Функција која извршава серијски алгоритам множења матрица. Ово представља кључан део програма.
Параметри:

- параметри ове функције су три матрице од којих прве две су улазне матрице, док је трећа резултујућа матрица.

4.1.4. Функција за исписивање грешака (error)

```
void error(const string& s)
```

Ова функција служи за исписивање било које грешке и завршавање програма након ње.
Параметри:

- s - текст грешке

4.2. Паралелни програм - модули и основне методе

4.1.1. Модул главног програма(MainProgram)

```
int mainParallel(int argc, char* argv[])
```

Почетна функција програма. Покреће све функције програма, приказује информације и време извршавање програма.

4.1.2. Модул за рад са датотекама (WorkWithFile)

Модул у коме се налази рад са датотекама.

4.1.2.1. Метода за читање улазне датотеке

```
MatrixDTO readFile(char* fileNameInput)
```

Метода учитава улазни фајл и формира две матрице.
Параметри:

- fileNameInput - име улазног фајла

Повратна вредност:

- DTO објекат који преноси учитане матрице из фајла

```
void readMatrix::makeMatrix(string line, int number,
MatrixDTO& m)
```

Метода која формира матрице обрадом сваке линије која јој је прослеђена.

Параметри:

- line - текућа линија из улазног фајла
- number - представља број матрице која се тренутно чита
- m - DTO објекат у ком ће бити смештена учитана матрица

4.1.2.2. Метода за упис излазне датотеке

```
void writeFile(char* fileNameOutput,
vector<vector<int>> matrix)
```

Метода која уписује резултат у излазни фајл.
Параметри:

- fileNameOutput - име излазног фајла
- vector<vector<int>> matrix - резултујућа матрица

4.1.3. Модул за рад са матрицама (Matrix)

Модул садржи функције које раде са матрицама.

4.1.3.1. Функција провере формата

```
void Matrix::checkCorrectness()
```

Функција проверава да ли су учитане матрице коректног формата, тј. да ли свака колона има исти број елемената (аналогно треба проверити и за редове).

4.1.4. Функција за исписивање грешака (error)

```
void error(const string& s)
```

Ова функција служи за исписивање било које грешке и завршавање програма након ње.
Параметри:

- s - текст грешке

4.1.5. Модул који имплементира parallel for (ParallelFor)

```
void matrixMultiplicationParallelFor(MatrixDTO& m,
Matrix& matrix)
```

Ова функција позива *parallel for* и иницијализује један објекат класе `class MultiplyMatrixClass.` Она садржи прекопљени ***const operator ( )***. Самом прекопљеном оператору прослеђујемо template класу `blocked_range2d<int, int>.` Коришћен је *blocked_range2d* јер радимо са дво-димензионалним низовима.
Параметри:

- m - ДТО објекат за пренос улазних матрица
- matrix - резултујућа матрица

4.1.6. Модул који имплементира поделу задатака по ћелијама матрице (TaskGroupOneCell)

```
void initCounters(vector<atomic<int>>& c, int row, int
col)
```

Функција која иницијализује вектор атомских променљива који нам показује колико задаци морају да сачекају пре него што крену са изршавањем.

Праметри:

- c - вектор атомских промељивих
- row - број редова
- col - број колона

```
int sum(int i, int j, MatrixDTO& m, Matrix& matrix3)
```

Функција која рачуна резултат једне ћелије матрице.

Патаметри:

- i - број реда
- j - број колоне
- m - ДТО објекат за пренос улазних матрица
- matrix3 - резултујућа матрица

```
void matrixMultiplicationOneCells
(vector<atomic<int>>& c, Matrix& matrix3, int i, int j,
MatrixDTO& m)
```

Функција која одређује задатаке за извршење множења матрица, тј. појединачних ћелија.

Параметри:

- c - вектор атомских променљивих
- matrix3 - резултујућа матрица
- i - број реда
- j - број колоне
- m - ДТО објекат за пренос улазних матрица

4.1.7. Модул који имплементира поделу задатака по колонама матрице (TaskGroupCol)

```
void initCountersCol(vector<atomic<int>>& c, int col,
int row)
```

Функција која иницијализује вектор атомских променљива који нам показује колико задаци морају да сачекају пре него што крену са изршавањем.

Праметри:

- c - вектор атомских промељивих
- col - број колона
- row - број редова

```
vector<int> sumCol(int j, vector<vector<int>> matrix1,
vector<vector<int>> matrix2, vector<vector<int>>& matrix3)
```

Функција која рачуна резултат једне колоне матрице.
Параметри:

- j - број колоне
- matrix1 - прва улазна матрица
- matrix2 - друга улазна матрица
- matrix3 - резултујућа матрица

```
void matrixMultiplicationCol(vector<atomic<int>>& c,
Matrix& matrix3, int i, vector<vector<int>> matrix1,
vector<vector<int>> matrix2)
```

Функција која одређује задатке за извршење множења матрица, тј. колоне матрице.
Параметри:

- c - вектор атомских променљивих
- matrix3 - резултујућа матрица
- i - број колоне
- matrix1 - прва улазна матрица
- matrix2 - друга улазна матрица

4.1.8. Модул који имплементира поделу задатака по редовима матрице(TaskGroupRow)

```
void initCountersRow(vector<atomic<int>>& c, int col)
```

Функција која иницијализује вектор атомских променљива који нам показује колико задаци морају да сачекају пре него што крену са изршавањем.
Праметри:

- c - вектор атомских промељивих
- col - број колона

```
vector<int> sumRow(int i, Matrix matrix1, Matrix
matrix2, Matrix& matrix3)
```

Функција која рачуна резултат једног реда матрице.
Параметри:

- i - број реда
- matrix1 - прва улазна матрица
- matrix2 - друга улазна матрица
- matrix3 - резултујућа матрица

```
    void matrixMultiplicationCol(vector<atomic<int>>& c,
Matrix& matrix3, int i, Matrix matrix1, Matrix matrix2)
```

Функција која одређује задатке за извршење множења матрица, тј. реда матрице.
Параметри:

- c - вектор атомских променљивих
- matrix3 - резултујућа матрица
- i - број реда
- matrix1 - прва улазна матрица
- matrix2 - друга улазна матрица

4.1.9. Модул који имплементира поделу матрице по броју језгара рачунара (TaskGroupNumberOfCores)

```
    void sumCores(Matrix matrix1, Matrix matrix2, int
start, int finish, map<int,vector<int>>& mapOfCordinates,
Matrix& matrix)
```

Функција која рачуна резултат дела матрице. Матрица је подељена на онолико делова колико рачунар има језгара, у овом случају на 4.
Параметри:

- matrix1 - прва улазна матрица
- matrix2 - друга улазна матрица
- - индекс почетка
- - индекс краја
- - мапа индекса и елемената матрице
- matrix - резултујућа матрица

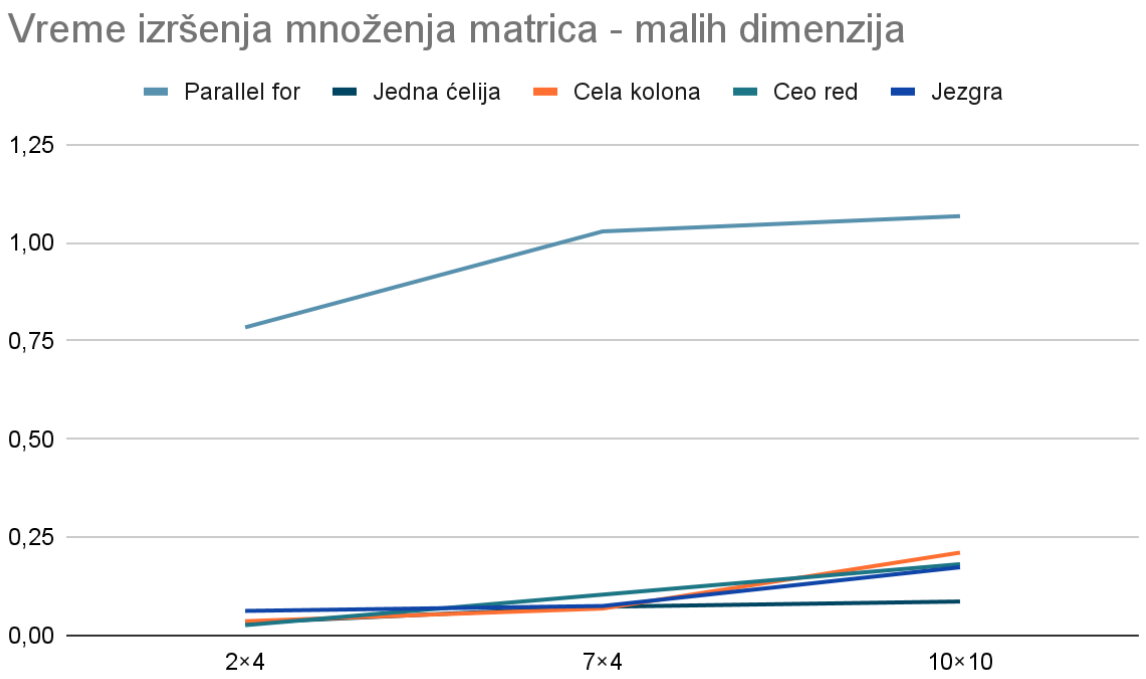
```
    void matrixMultiplicationNumberOfCores(Matrix matrix1,
Matrix matrix2, Matrix& matrix)
```

Функција која одређује задатке за извршење множења матрица, тј. дела матрице. У самој функцији се прави мапа која као кључеве има индексе од 0 до броја елемената матрице, а вредности су уређени пар који представља позицију елемента у матрици.
Параметри:

- matrix1 - прва улазна матрица
- matrix2 - друга улазна матрица
- matrix - резултујућа матрица

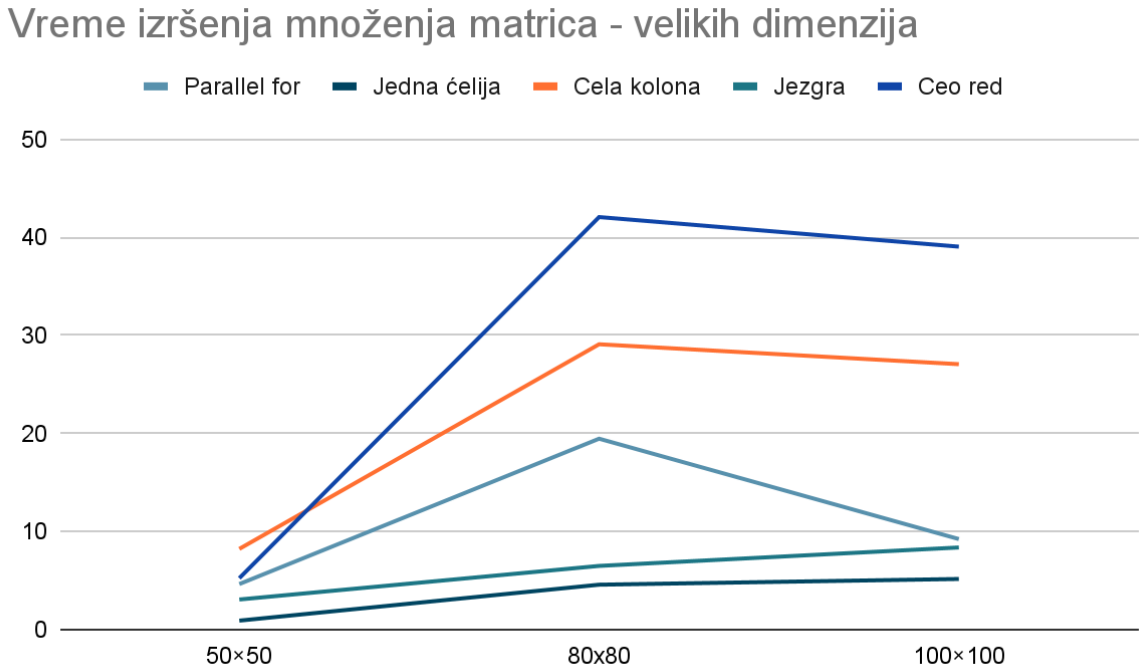
5. Тестирање

Приликом тестирања програма мерено је време извршења сваког имплементираног алгоритма. То значи да разликујемо време за серијско извршење алгоритма, паралелно извршење алгоритма као и за све његове начине имплементације (*parallel for*, подела задатака). У табели испод, као и на графицима се може уочити време извршења и који су алгоритми бољи у зависности од величине матрица.



Табеларни приказ графика на ком је укључено и серијско изрвшење.

	Serijsko	Parallel for	Jedna ćelija	Cela kolona	Ceo red	Jezgra
2×4	0,01954	0,78448	0,03254	0,0364	0,0263	0,06256
7×4	0,40192	1,02878	0,07316	0,0686	0,104	0,07508
10×10	2,4284	1,06764	0,08648	0,211	0,1816	0,17362



Табеларни приказ графика на ком је укључено и серијско изрвшење.

	Serijsko	Parallel for	Jedna ćelija	Cela kolona	Ceo red	Jezgra
50×50	1354,478	4,644	0,9131	8,2291	5,2448	3,069
80x80	80198,33	19,4659	4,57736	29,096	42,067	6,4938
100×100	32782,04	9,2312	5,1614	27,065	39,051	8,373

Приликом тестирања тачности самог програма коришћен је принцип учитавања тачног резултата из фајла. Тачан резултат је добијен уношењем улазних матрица у Matlab који је израчунао задато множење. Затим следи провера сваког елемента добијене матрице са елементом тачне матрице. Уколико је резултат тачан исписаће се порука о успешности, у супротном ће бити исписана порука о грешци.

```
bool checkResult(Matrix& correctMatrix, Matrix& resultMatrix)
```

Функција која проверава исправност множења матрица.

Параметри:

- correctMatrix - коректна матрица, учитана из фајла
- resultMatrix - добијена матрица извршењем програма

6. Закључак

Након имплементираног целог задатка, програм је извршаван и за серијску и за паралелну имплементацију. Покретање је извршено над истим величинама матрица, као и над истим елемнтима матрице. Изршено је мерење времена изршавања као што је приказано у секцији “Тестирање”. Као што можемо приметити серијско извршење је ефикасније користити једино у случају малих димензија матрица, док је паралелна имплементација далеко боља за матрице великих димензија. Наравно приликом мерења времена изршења мери се само извршење множења матрице, а не и учитавање из фајла, упис у фајл и провера тачности резултата.