



# УНИВЕРЗИТЕТ У НОВОМ САДУ ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА  
НОВИ САД

## ПРОЈЕКТНИ ЗАДАТАК

Кандидат: Сара Стојков  
Број индекса: SV 38 / 2023

Предмет: Паралелно програмирање  
Назив пројекта: Паралелни *PageRank* алгоритам

Ментор рада: др Душан Кењић

Нови Сад, септембар 2025. године

**САДРЖАЈ**

1. Увод .....	1
2. Анализа проблема.....	4
3. Концепт решења.....	6
4. Опис решења.....	7
5. Тестирање и евалуација решења.....	12
6. Закључак.....	15

## СПИСАК СЛИКА

Слика 1. Визуелизација резултата PageRank алгоритма као графа.....	2
Слика 2. Приказ примера усмереног графа, његове матрице повезаности и листе повезаности.....	3
Слика 3. Пример покретања програма.....	12
Слика 4. Испис статистике о графу.....	13
Слика 5. Приказ сумираних убрзања у табели.....	13
Слика 6. Поређење времена извршавања различитих оптимизација.....	14

# 1. Увод

## 1.1 Индексирање веб страница и рад претраживача

Индексирање је темељни процес који омогућава претраживачима да брзо и ефикасно пронађу релевантне информације на огромном пространству светске мреже. Претраживачи користе аутоматизоване програме, познате као веб роботи или *crawlers*, који константно претражују интернет, откривају нове странице и прате постојеће хиперлинкове. Свака страница коју робот посети се анализира, а садржај (текст, слике, видео) се обрађује и уноси у огромну базу података, познату као индекс.

Овај процес је сличан прављењу индексне картице у библиотеци – уместо да се прегледа свака књига, довољно је потражити књигу по кључној речи у индексу. Међутим, с обзиром на експоненцијални раст интернета, индексирање постаје све већи изазов. Проблем није само у величини података, већ и у њиховој динамичној природи – странице се константно ажурирају, бришу или замењују. Зато је рангирање кључно; оно одређује које ће се странице прво приказати кориснику, а тај редослед се не заснива само на кључним речима, већ и на процењеној "ауторитативности" и релевантности странице.

## 1.2 PageRank алгоритам

PageRank је алгоритам који је осмислио Лари Пејџ, а његов циљ је да додели нумеричку вредност свакој веб страници како би се одредила њена релативна важност. Алгоритам се заснива на представљању страница као чворова графа, а линкова између страница као грана графа (честа визуелизација може се видети на Слици 1). Основна идеја алгоритма заснива се на концепту гласања: сваки линк са странице А ка страници Б представља "глас" за страницу Б. Међутим, нису сви гласови једнаки. Гласови са страница које су и саме важне имају већу тежину и преносе више "ауторитета" нпр. Један линк са

популарне странице као што је “National Geographic” може имати већу тежину од 5 линкова са странице просечне основне школе у Србији.

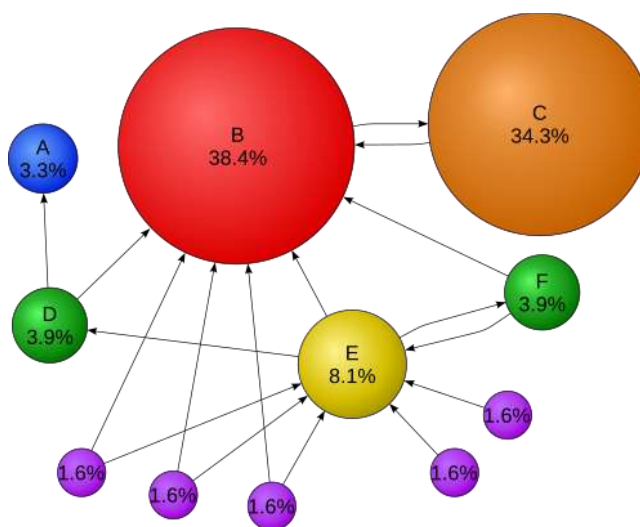
Процес израчунавања PageRank вредности је итеративан и може се описати следећом формулом:

$$PR(A) = (1 - d) + d \sum_{i=1}^n \frac{PR(T_i)}{C(T_i)}$$

где је:

- $PR(A)$  PageRank вредност странице  $A$ .
- $d$  фактор пригушења (damping factor), обично у литератури постављен на 0.85, који представља вероватноћу да ће корисник наставити да кликће на линкове уместо да насумично скочи на другу страницу.
- $T_i$  су све странице које линкују ка страници  $A$ .
- $C(T_i)$  је укупан број излазних линкова са странице  $T_i$ .

Итерације се понављају све док се *PageRank* вредности за све странице не стабилизују, односно док не конвергирају. Овај итеративни приступ је идеалан за паралелну обраду, јер се вредност сваког чвора у одређеној итерацији може рачунати независно од других чворова.



Слика 1. Визуелизација резултата PageRank алгоритма као графа

### 1.3 Задатак

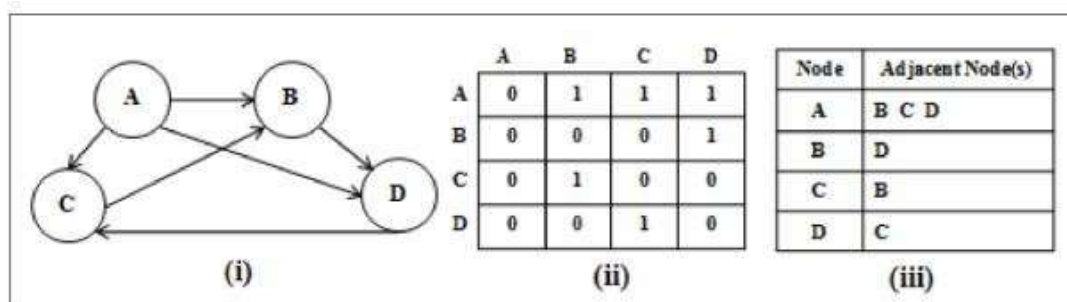
Главни задатак овог пројекта јесте да се имплементира и анализира паралелна верзија PageRank алгоритма, користећи предности *Intel Threading Building Blocks* (TBB) библиотеке. Пре самог рачунања, потребно је обезбедити граф који ће се обрађивати. Треба обезбедити генерисање графа, при чему се као модел за генерисање може користити *Erdős-Rényi* – где се иницијализују само чворови и посматра се свака грана из комплетног графа

и има шансу  $p$  да буде укључена у финални граф. Граф ће бити представљен или помоћу листе повезаних чворова (*adjacency list*), с тим што имплементација мора да обради графове који немају изоловане чворове, односно чворове без излазних грана – они представљају посебан проблем овог алгоритма.

Поред саме имплементације, значајан део задатка јесте и евалуација перформанси. Потребно је извршити поређење времена извршавања секвенцијалног и паралелног алгоритма. Као део резултата, треба приказати број итерација до конвергенције, укупно време извршавања и фактор убрзања (*speedup*). Коначно, ради боље анализе, потребно је исписати првих 10 чворова са највећом PageRank вредношћу и анализирати понашање програма у зависности од броја процесорских језгара

## 2. Анализа проблема

Проблем паралелизације PageRank алгоритма на великим скуповима података, као што је интернет, није тривијалан. Иако је алгоритам итеративан и самим тим очигледан кандидат за паралелизацију, у пракси се јавља неколико кључних изазова. Један од њих јесте избор структуре података за представљање графа. Иако је матрица повезаности (*adjacency matrix*) савршена за математичку формулу PageRank-а, њена меморијска захтевност је експоненцијална, што је чини неприкладном за реалне, обимне веб графове. С друге стране, листа повезаних чворова (*adjacency list*) је меморијски ефикаснија и због тога је изабрана у овом пројекту. Ипак, пажљиво треба радити са њом јер може да унесе додатне изазове у имплементацији, посебно када више нити приступа истој структури.



Слика 2. Приказ усмереног графа (i), његове матрице повезаности (ii) и листе повезаности (iii)

Још један значајан изазов је балансирање оптерећења. У ацикличним графовима, неки чворови могу имати знатно већи број улазних линкова, што захтева више рачунских операција. Ако се задаци једноставно поделе на нити, неке нити могу завршити свој посао много брже од других, што доводи до неискоришћености процесорских језгара. ТБВ библиотека нуди механизме за динамичку поделу посла, што помаже да се овај проблем ублажи. Такође, треба обратити пажњу на руковање "висећим" чворовима (*dangling nodes*).

Ови чворови, који немају излазне линкове, могу да доведу до тога да се укупна сума *PageRank* вредности смањи, што нарушава принцип да се укупна важност свих страница мора очувати.

Коначно, иако паралелна обрада обећава брзину, синхронизација између нити може да постане уско грло. Итеративни алгоритам захтева да све нити заврше своје рачунање пре него што се крене у следећу итерацију, а затим се резултати морају безбедно објединити. Коришћење механизма као што су конкурентни редови за комуникацију између задатака може да унесе значајне перформансне губитке због трошкова синхронизације. Због тога је оптимизација ових синхронизационих тачака кључна за постизање правог убрзања.



### 3. Концепт решења

Концепт решења за паралелну имплементацију PageRank алгоритма заснива се на експлоатацији особина итеративног рачунања, где се вредности за сваки чвор могу израчунавати независно у свакој итерацији. Уместо традиционалног секвенцијалног приступа, који служи као контролна група за мерење перформанси, главна имплементација користи паралелни приступ заснован на задацима (*task-based parallelism*) који омогућава да се рачунање распореди на више процесорских језгара.

За саму паралелизацију рачунања, користи се *tbb::parallel\_for* из ТВВ библиотеке. Он омогућава да се итерација кроз све чворове графа подели на мање, независне блокове рада. На овај начин, свака нит може да рачуна нову *PageRank* вредност за свој део чворова, без директне интеракције са другим нитима. Овај приступ обезбеђује добро балансирање оптерећења, јер ТВВ динамички распоређује задатке на расположива језгра. Поред тога, за проверу конвергенције, која захтева обједињавање резултата из свих нити, користи се *tbb::parallel\_reduce*. Ова функција ефикасно сабира парцијалне грешке израчунате од стране сваке нити, чиме се избегавају скуп механизми синхронизације и глобална конкуренција.

Поред основне паралелне верзије, решење обухвата и напредне оптимизације како би се додатно побољшале перформансе. Једна од њих је оптимизација блокова, која се фокусира на побољшање искоришћености кеш меморије. Користећи величину кеш линије, алгоритам приступа подацима на начин који смањује број промашаја у кешу, што значајно убрзава извршавање. Такође, имплементирана је и адаптивна расподела задатака, која процењује обим посла за сваки чвор (на основу броја улазних линкова) и распоређује их на нити тако да све нити имају приближно једнако оптерећење.

## 4. Опис решења

### 4. Модули и основне методе решења

#### 4. 1. Main метода из main.cpp

```
int main();
```

Ово је главна функција програма. Увезује све остале елементе програма и покреће програм. Након што учита све друге класе, покреће *PageRankBenchmark* тестирања и анализе које ће бити дискутоване у даљем тексту.

#### 4.2 Класа графа (Graph)

Класа која представља граф. Има подразумевани празан конструктор и конструктор коме се може проследити број чворова, као и деструктор.

Атрибути графа:

- N (број чворова)
- adjListIn (*adjacency* листа улазних грана)
- outDegree (излазни степени чворова)
- names (структура мапе која служи за именовање чворова)

##### 4.2.1. Основне операције са графовима

```
void addEdge(int from, int to);  
const std::vector<int>& getInNeighbors(int v) const;  
int getOutDegree(int v) const;
```

Метода за додавање гране прима као параметре чвор из којег иде грана и чвор у који иде грана и нема повратну вредност већ мења сам граф.

Метода *getInNeighbors* за одабрани чвор (као параметар прима број чвора) враћа вектор његових комшија, ако посматрамо улазеће гране.

Метода за добављање степена чвора (на основу излазећих грана) враћа цео број који представља колико грана излази из одабраног чвора.

#### 4.2.2. Генерисање насумичног графа

```
void generateErdosRenyi(double p, unsigned int seed = 0);
void generateErdosRenyiWithEdges(int targetEdges, unsigned int seed=0);
```

Ове методе служе да попуне празан граф. ErdosRenyi модел за генерисање рандом графа подразумева да свака грана (уређени пар 2 чвора, јер је граф усмерен) има одређену шансу да буде укључена у граф. Прва метода то ради потпуно рандом, док се другој методи може задати жељени број грана што је итекако погодно за тестирање.

#### 4.2.3. Функције за анализу

```
void printGraphStats() const;
int getTotalEdges() const;
double getAverageDegree() const;
```

Класичне методе за проверу стања графа, обухватају испис особина графа попут броја чворова, броја грана, просечног степена, густине графа, највећег и најмањег степена графа. Остале методе враћају укупан број грана и просечан број степена чвора у датом графу, тим редом.

### 4.3 Класа PageRank алгоритма (PageRank)

Класа која представља сам алгоритам који налази чворове са највећом шансом да буду одабрани. У себи има сву главну логику пројекта која се ослања на граф, помоћну структуру.

Има следеће атрибуте:

- graph
- damping фактор који представља вероватноћу да ће просечан насумични корисник пратити линкове на некој страници уместо да скочи на потпуно нову
- prOld стари резултат алгоритма да би се проверило конвергирање

#### 4.3.1. Помоћне методе

```
bool hasConverged(double epsilon) const;
bool hasConvergedRelative(double epsilon) const;
void initialize();
```

Прве две функције као повратну вредност враћају булову променљиву, а као параметар узимају реалан број епсилон који означава колику „епсилон околину“ ћемо сматрати довољно блиском да кажемо да су резултати конвергирани и да алгоритам може да се заустави.

Последња метода служи за иницијализацију објекта. На основу броја чворова у графу се иницијализују стара PageRank вредност `prOld` и нова `prNew`.

У наставку, све функције које извршавају сам алгоритам узимају исте параметре:

- `int maxIter` – максималан број итерација, понављања алгоритма
- `double epsilon` – као код оптимизационих проблема, ако су резултати у епсилон околини један другог, значи да је резултат довољно близак и да је конвергирао

Повратне вредности нема јер мењају саму структуру.

Оба параметра за циљ имају задавање краја алгоритма.

#### 4.3.2. Секвенцијално извршавање (`runSequential`)

```
void runSequential(int maxIter, double epsilon);
```

Стандардна, једнонитна имплементација PageRank алгоритма. Она рачуна нове вредности PageRank-а унутар једне петље, без употребе паралелизације. Служи као основа за поређење перформанси.

#### 4.3.3. Паралелно извршавање (`runParallel`)

```
void runParallel(int maxIter, double epsilon, int numThreads);
```

Основна паралелна имплементација PageRank алгоритма. Користи `tbb::parallel_for` да би расподелила рачунање PageRank вредности за сваки чвор на расположиве нити, чиме се постиже паралелизација. `tbb::blocked_range` дефинише опсег чворова који се обрађују у свакој нити.

#### 4.3.4. Оптимизација: блок-оптимизован (`runParallelBlockOptimized`)

```
void runParallelBlockOptimized(int maxIter, double epsilon, int numThreads);
```

Оптимизована верзија која побољшава локалност података (`cache locality`). Уместо да нити обрађују произвољне чворове, ова верзија их групише у блокове величине прилагођене величини кеш линије. Ово смањује промашаје кеша (`cache misses`) и убрзава приступ подацима.

#### 4.3.5. Оптимизација: векторизовано (`runParallelVectorized`)

```
void runParallelVectorized(int maxIter, double epsilon, int numThreads);
```

Ова функција покушава да искористи векторске инструкције процесора (poput SIMD-а) тако што унутар петље обрађује више података истовремено. Кроз **unrolling** петље (тј. разматавање петље за 4 елемента унапред), смањује се број итерација петље и омогућава се боља искоришћеност процесорске архитектуре.

#### 4.3.6. Оптимизација: NUMA свестан (`runParallelNUMAAware`)

```
void runParallelNUMAAware(int maxIter, double epsilon, int numThreads);
```

Дизајнирана да оптимизује перформансе на **NUMA** (Non-Uniform Memory Access) архитектурама. Она грубо распоређује чворове међу нитима тако да свака нит обрађује блокове чворова који се налазе у близини меморије коју та нит користи, чиме се смањује кашњење приступу подацима.

#### 4.3.7. Оптимизација: адаптивни grain size (`runParallelAdaptiveGrain`)

```
void runParallelAdaptiveGrain(int maxIter, double epsilon, int numThreads);
```

Користи приступ са **адаптивном величином зрна** (`adaptive grain size`). Ова функција прво израчунава процењену количину посла за сваки чвор (на основу броја улазних веза), а затим распоређује чворове међу нитима тако да свака нит добије отприлике исту укупну количину посла. Ово помаже у бољем балансирању оптерећења, посебно код графова са неравномерном дистрибуцијом веза.

### 4.4 Сабрани резултати и тестирање (`PageRankBenchmark`)

Класа која увезује све претходно наведене елементе и покреће тестове на предефинисаним „димензијама“ графова, тестирајући и оптимизације алгоритма. При сваком покретању алгоритма мери се време и на основу односа времена за секвенцијално извршавање рачуна се фактор убрзања.

#### 4.4.1. Покретање тестова (`runBenchmarkSuite`)

```
void runBenchmarkSuite();
```

Ова метода је главни покретач свих тестова. Она пролази кроз унапред дефинисане величине графа и густине и тако генерише различите тест случајеве. За сваки тест се позива функција `runSingleBenchmark`, а на крају се приказује сумиран извештај и уписује се у CSV фајл.

#### 4.4.2. Појединачан тест (runSingleBenchmark)

```
void runSingleBenchmark(int nodes, double edge_probability)
```

Улазни параметри су број чворова и вероватноћа да грана буде укључена и на основу њих се генерише насумични граф (методом из класе графа). Након тога мери се време извршавања секвенцијалне и паралелне верзије алгоритма. На крају извршавања се рачуна фактор убрзања и пореде се резултати између секвенцијалног и паралелног начина рачунања.

#### 4.4.3. Методе за тестирање оптимизација (testOptimizedVersions)

```
void testOptimizedVersions(int nodes, double p)
```

Метода намењена поређењу оптимизованих верзија паралелног алгоритма – тестира *Block Optimized*, *Vectorized*, *NUMA Aware* и *Adaptive Grain*. За сваки од ових приступа проблему се мери време извршавања које се онда пореди са секвенцијалним начином и тиме даје увид у резултате.

Исто као горепоменута функција, параметри су управо параметри за попуњавање графа насумично – број чворова *nodes* и вероватноћа да нека грана постоји *p*.

#### 4.4.4. Помоћне методе за испис и чување (printSummaryReport, exportResultsToCSV)

```
void printSummaryReport()
```

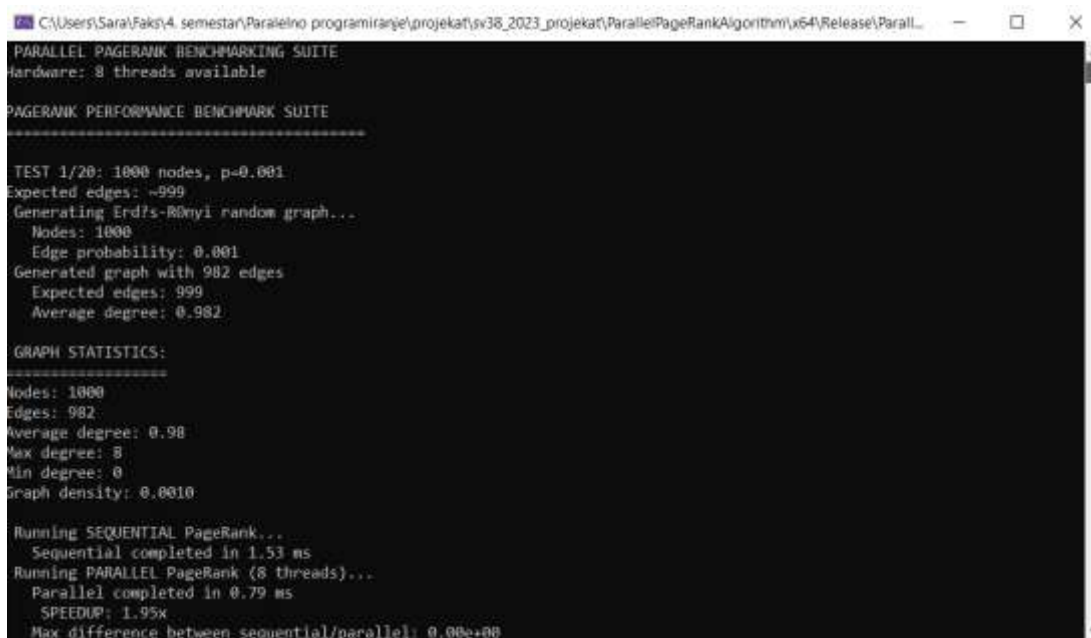
```
void exportResultsToCSV(const std::string& filename)
```

Функције које олакшавају преглед и праћење рада самог програма и алгоритма.

Прва функција приказује скраћене резултате свих поменутих тестова. Форматира тре

## 5. Тестирање и евалуација решења

Програм је покретан на процесору 12th Gen Intel(R) Core(TM) i3-1215U 1.20 GHz, а за број нити се брине сам програм тако што на почетку провери колико нити паралелно може да се извршава. Функције за анализу времена извршавања олакшавају преглед рада алгорита и пореде резултате паралелног извршавања са секвенцијалним ради валидације. Све исписе генерише класа за тестирање и анализу (*PageRankBenchmark*) (видети слике 3, 4 и 5 за примере).



```
PARALLEL PAGERANK BENCHMARKING SUITE
Hardware: 8 threads available

PAGERANK PERFORMANCE BENCHMARK SUITE
=====

TEST 1/20: 1000 nodes, p=0.001
Expected edges: ~999
Generating Erdős-Rényi random graph...
Nodes: 1000
Edge probability: 0.001
Generated graph with 982 edges
Expected edges: 999
Average degree: 0.982

GRAPH STATISTICS:
=====
Nodes: 1000
Edges: 982
Average degree: 0.98
Max degree: 8
Min degree: 0
Graph density: 0.0010

Running SEQUENTIAL PageRank...
Sequential completed in 1.53 ms
Running PARALLEL PageRank (8 threads)...
Parallel completed in 0.79 ms
SPEEDUP: 1.95x
Max difference between sequential/parallel: 0.00e+00
```

Слика 3. Пример извршавања програма – конкретно испис функције за тестирање, почетак рада програма уз испис да има 8 нити

```

GRAPH STATISTICS:
=====
Nodes: 100000
Edges: 99996130
Average degree: 999.96
Max degree: 2195
Min degree: 1806
Graph density: 0.0100

Running SEQUENTIAL PageRank...
Sequential completed in 275.98 ms
Running PARALLEL PageRank (8 threads)...
Parallel completed in 97.18 ms
SPEEDUP: 2.84x
Max difference between sequential/parallel: 0.00e+00
Results are numerically identical!
Test completed

```

Слика 4. Пример исписа функције статистике графа, уз приказ осталих параметара попут убрзања и времена извршавања

Nodes	Edges	Seq(ms)	Par(ms)	Speedup	Efficiency	Max Diff
1000	982	1.8	0.6	1.64	28.5	30.0e+00
1000	5006	8.3	0.4	8.82	18.3	30.0e+00
1000	10919	8.2	0.2	1.84	13.8	30.0e+00
1000	10911	8.4	0.2	1.86	23.3	30.0e+00
5000	24081	1.3	0.7	1.86	23.2	30.0e+00
5000	124783	1.7	0.6	2.92	36.4	30.0e+00
5000	249582	2.8	0.9	3.16	39.5	30.0e+00
5000	499489	3.2	0.6	5.36	67.8	30.0e+00
10000	99886	1.7	0.6	2.98	36.3	30.0e+00
10000	499582	2.7	1.8	2.77	38.6	30.0e+00
10000	999742	4.5	1.3	3.88	48.8	30.0e+00
10000	2000188	8.8	2.8	3.18	39.8	30.0e+00
50000	2500441	22.2	7.2	3.08	38.5	30.0e+00
50000	1249554789	3	16.1	2.58	11.3	30.0e+00
50000	249958878	8	24.8	2.87	15.9	30.0e+00
50000	49997563113	7	48.8	2.28	28.5	30.0e+00
100000	99996130	48.2	16.8	2.87	15.9	30.0e+00
100000	49996420149	2	58.8	2.88	17.3	30.0e+00
100000	99996130276	8	97.2	2.84	15.5	30.0e+00
100000	200000417247	9	98.7	2.58	13.8	30.0e+00

Average speedup: 2.84x  
Hardware threads: 8  
Results exported to pagerank\_benchmark\_results.csv

Слика 5. Пример сумираних резултата за све бројеве чворова и грана коришћене при тестирању (ово се уписује у CSV)

## 5.1 Поређење секвенцијалног и паралелног извршавања

Секвенцијални модел, иако је функционалан и служи као основа за поређење, показао се као неефикасан за обраду великих скупова података. С друге стране, паралелна верзија PageRank-а, користећи Intel TBB библиотеку, искористила је моћ вишејезгарних процесора и постигла значајно убрзање.

Кључна разлика лежи у начину обраде. Док секвенцијални програм обрађује чворове један по један, паралелна верзија расподељује задатке (обраду различитих чворова) на више радних нити. То доводи до мањег времена извршавања, што је посебно важно за велике графове који би у секвенцијалној верзији захтевали дуготрајно рачунање.



Утврђено је да се убрзање највише постиже у деловима који су чисто рачунског карактера, док се оно смањује у фазама које захтевају синхронизацију. О томе говоре и времена извршавања (нпр. Слика 4).

## 5.2 Поређење различитих паралелних извршавања

Пројекат је тестирао неколико напредних оптимизација паралелне имплементације како би се додатно побољшале перформансе (али видети слику 6).

- **Block Optimized** (Оптимизација по блоковима): Овај приступ групише чворове у блокове ради боље искоришћености кеш меморије. То смањује број промашаја кеша и убрзава приступ подацима, што је посебно корисно код великих графова.
- **Vectorized** (Векторизована верзија): Користећи технику "размотавања петље" (loop unrolling), ова верзија омогућава компајлеру да искористи векторске инструкције процесора (поруч SIMD-а) и обради више података истовремено.
- **NUMA-Aware** (Свесна NUMA архитектуре): Ова оптимизација је намењена системима са NUMA архитектуром (неуниформни приступ меморији). Функција распоређује задатке тако да нити приступају подацима који су физички ближи њиховој меморији, смањујући тиме кашњење.
- **Adaptive Grain** (Адаптиван grain size): Уместо да додељује исти број чворова свакој нити, ова верзија процењује укупну количину посла и додељује нитима делове посла једнаке по величини, чиме се постиже боље балансирање оптерећења.

Резултати тестова су ипак показали да је најбржи приступ класични паралелни, додуше то би се морало тестирати на различитим типовима и величинама графова (улаза).

OPTIMIZATION COMPARISON SUMMARY:			
Method	Time (ms)	Speedup	Efficiency
Sequential	0.81	1.00	baseline
Parallel Basic	0.38	2.14	26.8%
Block Optimized	0.54	1.49	18.7%
Vectorized	0.44	1.82	22.8%
NUMA Aware	0.41	1.96	24.5%
Adaptive Grain	0.53	1.53	19.2%

All benchmarks completed successfully!

Слика 6. Поређење перформанси различитих оптимизација паралелног алгоритма које циљају различите мане

## 6. Закључак

Развој пројекта „Паралелни PageRank алгоритам“ омогућио је имплементацију и верификацију паралелне обраде великих графова уз коришћење Intel Threading Building Blocks (TBB) библиотеке. Пројекат је започет анализом перформанси секвенцијалног PageRank алгоритма и идентификацијом потенцијалних уских грла при обради масивних скупова података. На основу те анализе, реализован је паралелни приступ заснован на итеративном рачунању PageRank вредности, уз примену *tbb::parallel\_for* за расподелу задатака и *tbb::parallel\_reduce* за комбиновање резултата. Испитивање исправности решења спроведено је поређењем добијених резултата са секвенцијалном верзијом алгоритма, при чему је постигнута функционална идентичност и прецизност унутар дефинисане толеранције.

Паралелна имплементација показала је значајно побољшање скалабилности, посебно на великим графовима, док су оптимизације попут адаптивне величине зрна и локализације података омогућиле ефикасније коришћење ресурса процесора. Ипак, идентификовани су сценарији у којима паралелизација није доносила очекивано убрзање, нарочито када је дошло до додатних захтева као што су одређивање јединствених решења и упис резултата. Главни изазов представља висок трошак синхронизације, нарочито при коришћењу конкурентних редова за комуникацију између задатака, што је указало на значај пажљивог одабира паралелних примитива.

Пројекат пружа солидну основу за даље истраживање и побољшање алгоритма. Могуће надоградње укључују испитивање алтернативних метода синхронизације, као што су атомске операције или *lock-free* структуре, како би се смањили режијски трошкови и повећала ефикасност. Такође, проширење на дистрибуиране системе омогућило би обраду

граfoва који не могу стати у меморију једног рачунара, уз примену модела као што је MapReduce или MPI. Поред тога, имплементација тежинских веза и персонализованих варијанти PageRank-а могла би да омогући прецизније рангирање релевантности страница на основу специфичних захтева корисника. Детаљнија анализа перформанси, уз коришћење алата за профилисање попут *Intel VTune*-а, могла би да идентификује уска грла, кашњења и боље искоришћеност кеша, што би омогућило финије оптимизације.

Све у свему, пројекат успешно демонстрира примену паралелизације на PageRank алгоритам и пружа функционалну и прошириву основу за будућа истраживања, оптимизације и експерименте са различитим моделима обраде граfoва.