Универзитет у Новом Саду

Факултет техничких наука

Документација предметног пројекта

Студенти: Милан Јовкић, SV8/2021

Василије Марковић, SV15/2021

Предмет: Паралелно програмирање

Тема предметног пројекта: Слагалица – игра са словима

Садржај

[Анализа проблема 2](#_Toc135560549)

[ О игри 2](#_Toc135560550)

[ О проблему 2](#_Toc135560551)

[Концепт решења 2](#_Toc135560552)

[ О коришћеним технологијама 3](#_Toc135560553)

[ О подацима 3](#_Toc135560554)

[ Структура игре 3](#_Toc135560555)

[Програмско решење 4](#_Toc135560556)

[ Структура main програма 4](#_Toc135560557)

[ Структура серијских и паралелних функција 5](#_Toc135560558)

[Испитивање 6](#_Toc135560559)

[ Карактеристике рачунара 6](#_Toc135560560)

[ Серијско извршавање 6](#_Toc135560561)

[ Паралелно извршавање 6](#_Toc135560562)

[ Параметри паралелизације 7](#_Toc135560563)

[Анализа резултата 7](#_Toc135560564)

[ Резултати паралелизације 7](#_Toc135560565)

[ Закључак 9](#_Toc135560566)

# Анализа проблема

## О игри:

Прва игра ТВ квиза Слагалица представља надметање такмичара у тражењу најдуже речи која се може направити од понуђених 12 слова. У свом пројекту избацили смо аспект такмичења, али задржали механике игре. Програм прима и уважава искључиво речи написане словима ћирилице и које се налазе у неформалном речнику доступном на интернету.

## О проблему:

Пројекат смо осмислили уочивши да у алгоритму симулирања Слагалице морају постојати делови погодни за паралелизацију. Наиме, листа прихватљивих речи коју програм константно користи садржи готово речи, што се може сматрати прилично обимном структуром података.

Кроз њу се итерира сваки пут кад:

* компјутер тражи речи које се могу саставити од датих генерисаних слова партије и које имају највећу могућу дужину
* систем проверава да ли је реч коју је корисник унео присутна у понуђеном речнику

Одлучили смо да паралелизујемо кôд у ове две функционалности.

# Концепт решења

## О коришћеним технологијама:

Пројекат је писан у програмском језику C++, у окружењу Visual Studio 2022.

За паралелни рад апликације коришћене су групе задатака модула TBB (Thread Building Blocks – task group). Оне омогућују да се било која функција, која би се иначе извршавала серијски, подели на више задатака. Задаци се извршавају истовремено, те се по завршетку свих њих добијени резултат споји назад у једну линију и кôд наставља да се обавља у серијском режиму.

## О подацима:

Речник од речи нашли смо на интернету, на насумичном форуму о граматици и језику. Садржи велик број полусложеница, неформалних израза и сложених облика речи који се не могу наћи на ТВ Слагалици. Ова разлика утиче на правила игре као и саму величину итерационог простора, али такође чини извођење игре знатно интересантнијим. Речник се налази у фолдеру data под именом Recnik.txt.

У истом фолдеру може се наћи азбука Slova.txt која садржи свих 30 слова, али са измењеним фреквенцијама појављивања (за детаље погледати поглавље Програмско решење).

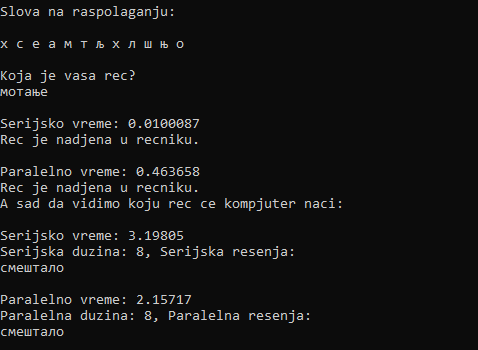
## Структура игре:

Свака рунда се састоји од неколико ставки. На самом почетку компјутер генерише 12 насумичних слова и приказује их на екран играчу. Затим је на играча ред да унесе своју реч, која улази у проверу: прво се проверава да ли је реч формирана од задатих слова помоћу методe isSubset, чији је рад објашњен у наредном поглављу.

Уколико јесте, програм улази у другу проверу, која представља прво место где смо одлучили да паралелизујемо кôд. Серијска провера се састоји од листања свих речи у речнику, и проверавања да ли се било која од њих поклапа са унетом. Кôд смо паралелизовали тако што смо увели задатке који поделе итерациони простор речи на више мањих, те сваки задатак обрађује одговарајући подинтервал.

Након целокупне провере уноса, игра улази у друго место погодно за паралелизацију. Програм опет пролази кроз све речи речника, али овај пут памти оне које имају највећу нађену дужину и које се могу саставити од задатих слова. Паралелизам је још једном остварен делећи итерациони регион и тиме смањујући рад сваког од задатака. Након извршења, на екран се приказују сва најбоља решења која је компјутер нашао.

Уколико је корисникова унета реч била kraj, написано на латиници, програм се прекида, а у супротном прелази у наредну рунду, где врши све кораке изнова.



# Програмско решење

## Структура main програма:

Пре него што крене извршавање рунди, потребно је прикупити податке са којима се рукује. На почетку програма неопходним подацима се пуне 2 вектора стрингова: randomLetters и allWords.

Као што је већ поменуто, у датотеци data/Slova.txt налазе се сва слова азбуке. Међутим, самогласници А, Е, И, О и У јављају се по три пута, како би, приликом генерисања задатих слова једне партије, имали већу вероватноћу појављивања, а самим тим олакшали састављање дугачких речи. Дакле, у randomLetters смештају се свих 30 различитих слова азбуке, са додатним двоструким инстанцама самогласника.

Напомена: слова су типа string ради избегавања потенцијалних проблема при конверзији, јер је руковање ћирилицом представљало повремене компликације у писању кода.

Након прикупљања слова, из data/Recnik.txt се прикупљају све речи и складиште у вектор allWords.

Свака итерација игре представља по једну рунду. Ефекат је реазлизован преко бесконачне while петље која се прекида ако и само ако је играч као реч унео „kraj“. Рад појединачне рунде је описан у одељку изнад, с тим да је битно нагласити да се за мерење времена извршења паралелних и серијских функција користило заглавље <tbb/tick\_count.h>, а за генерисање насумичних бројева заглавље <random>.

Такође, испис и унос ћириличних слова остварено је махом помоћу позива следећих функција на самом почетку програма:

SetConsoleOutputCP(CP\_UTF8);  
SetConsoleCP(CP\_UTF8);

из заглавља <Windows.h>.

## Структура серијских и паралелних функција:

Помоћним функцијама се приступа из једног .cpp фајла и једног header фајла.

Slagalica.h – датотека у којој су декларисане главне функције коришћене у програму:

* bool isSubset(string subset, string set);
* bool findMatchSerial(string userWord, const vector<string>& allWords);
* bool findMatchParallel(string userWord, const vector<string>& allWords);
* int findBestSolutionsSerial(string randomWord, const vector<string>& allWords, vector<string>& solutions);
* int findBestSolutionsParallel(string randomWord, const vector<string>& allWords, vector<string>& solutions);

Slagalica.cpp – датотека у којој су дефинисане функције које садрже серијски и паралелни кôд, као и једна потфункција:

* isSubset – потфункција која узима 2 стринга и проверава да ли су сва слова у првом стрингу (subset) садржана у другом (set); користи се у налажењу најбољих речи, као и провери да ли је корисникова реч састављена искључиво од датих слова.
* findBestSolutionsSerial – серијски вид фунцкије која пролази кроз вектор стрингова који представља речник (allWords) и помоћу методе isSubset налази речи највеће дужине које се могу формирати од понуђених слова (randomWord), затим их смешта у нови вектор (solutions) и враћа њихову дужину (int bestLength).
* findBestSolutionsParallel – паралелни вид претходне функције, имплементиран рекурзијом, који у базном случају позива претходну функцију findBestSolutionsSerial, док у главном делу распоређује рад по TBB задацима.
* findMatchSerial – серијски вид функције која пролази кроз речник (allWords) и пореди сваку реч са оном коју је корисник унео (userWord) која обуставља рад кад је нађе или пређе кроз све речи.
* findMatchParallel – паралелни вид претходне функције, имплементиран рекурзијом, који у базном случају позива findMatchSerial, док у главном делу распоређује рад по TBB задацима.

# Испитивање

## Карактеристике рачунара:

* Оперативни систем – Windows (x64)
* Процесор – AMD Ryzen 5 4500U
* RAM меморија – 8 GB
* Број језгара на процесору – 6
* Број нити на процесору – 6

## Серијско извршавање:

findMatchSerial – алгоритам ради на принципу једне for петље која пролази кроз речник, пореди речи са унешеном речју и престаје с радом ако наиђе на тражену реч или дође до краја.

findBestSolutionsSerial – алгоритам ради на принципу for петље која такође пролази кроз речник, али, за сваку итерацију, у функцији isSubset сортира реч из речника, као и понуђена слова, те их пореди карактер по карактер.

## Паралелно извршавање:

findMatchParallel – алгоритам ради на принципу TBB задатака, који се деле на делова, и сваки од њих рекурзивно обрађује део речника дужине , где је дужина дела речника на дубини стабла задатака где се задаци позивјуа; сваки задатак пређе у серијско изврашавање кад је дужина података на којима он ради мања од вредности MATCH\_CUTOFF, дефинисане на почетку програма; враћа решење (bool вредност) тако што врши дисјункцију свих добијених решења (враћа тачно ако и само ако је реч нађена у бар једном од интервала).

findBestSolutionsParallel – алгоритам ради на принципу TBB задатака, који се деле на делова, и сваки од њих рекурзивно обрађује део речника дужине , где је дужина дела речника на дубини стабла задатака где се задаци позивају; сваки задатак пређе у серијско изврашавање кад је дужина података на којима ради мања од вредности SOLUTION\_CUTOFF, дефинисане на почетку програма; повратна дужина пронађених речи, као и садржај вектора који складишти решења, зависи од односа између дужина речи враћених завршетком свих појединачних задатака (узима се унија свих вектора решења чије су речи највеће дужине у поређењу са осталима).

## Параметри паралелизације:

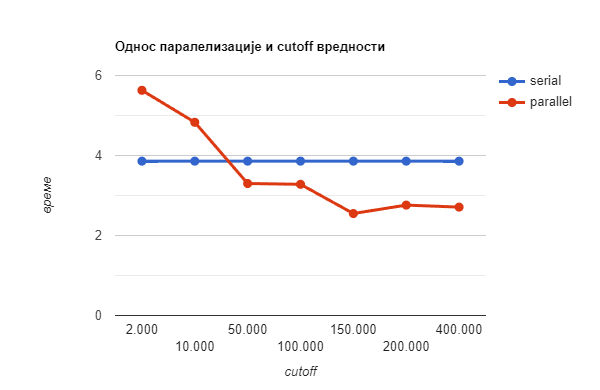
Током испитивања, обе CUTOFF вредности смо мењали кроз многобројне константе у опсегу од до (било шта изнад нема смисла, јер се рад тада своди на серијско извршавање, како се речник састоји од речи), те бележили просечна времена извршења за сваку од константи.

Други параметар који смо варирали током тестирања био је сам број задатака у свакој од паралелних функција. Како смо схватили да се кôд којим се рукује добијање излазних вредности из једног позива паралелне функције знатно компликује због додавања различитих услова при увођењу већег броја задатака, оптимално смо се одлучили да тестирамо функције поделом њиховог рада на 2, 3 и 4 појединична TBB задатка.

# Анализа резултата

## Резултати паралелизације:

findBestSolutions – на графику испод приказан је однос између серијског и паралелног рада алгоритма за налажење најбољих речи. Рад је описан на случају када се група задатака дели на 3 појединачна задатка, што се емпиријски показало као опција која доноси оптимална времена извршавања.



На -оси приказане су различите вредности дефинисане константе SOLUTION\_CUTOFF са којима смо мерили време, док су на -оси забележена просечна времена извршења паралелног и серијског кода.

Закључак који се може извући јесте да је са премалом CUTOFF вредношћу програм форсиран да дели задатке рекурзивно до превелике дубине, процес који захтева значајну количину времена. Ова чињеница се огледа на левом делу графика, где је паралелно време убедљиво спорије од серијског.

Приметимо такође да је најниже паралелно време постигнуто кад је CUTOFF вредност постављена на око , што је приближно трећини укупне дужине речника, те сваки од 3 позвана задатка обрађује трећину укупног посла, и не дели се даље на нове нивое подзадатака.

Оптимално жељено убрзање налажења оптималних речи је дакле остварено са следећим параметрима: 3 задатка у једном позиву паралелне функције и за CUTOFF вредност; оно у просеку износи приближно секунде (са просечног серијског времена од секунди на просечно паралелно време од секунди).

findMatch – на графику испод приказан је однос између паралелног и серијског и паралелног рада алгоритма за проверу исправности кориснички унете речи. Број задатака у једном позиву паралелне функције је такође 3, с тим да се овај пут резултат знатно мање разликовао у односу на различите вредности броја задатака. Штавише, пустили смо алгоритам и са опцијом да прави 10 задатака, и време паралелног извршавања се у просеку разликовало за мање од једне стотинке.

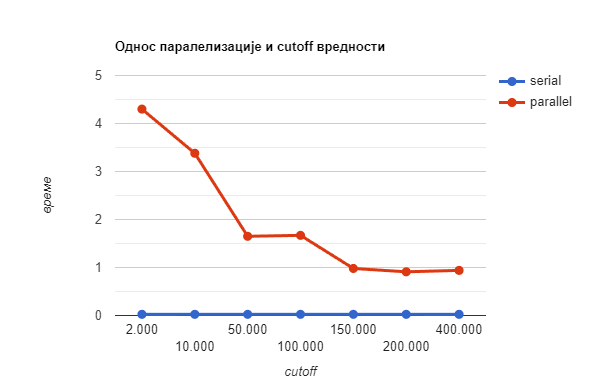


График је форматиран на исти начин као и први, с тим да овај пут је за различите CUTOFF вредности мењана константа MATCH\_CUTOFF. Битна напомена о тестирању ове функције је то да смо најчешће уносили непостојеће речи, како бисмо посматрали време за „најгоре могуће случајеве“, односно када је алгоритму неопходно да прође кроз цео речник.

Са слике се одмах може извући закључак за ову паралелизацију, а то је да је она беспотребна. Наиме, иако монотоност графика који представља паралелно време подсећа на ону са претходног мерења, разлика се уочава у самом раду серијског кода, који је показао неочекивано високе перформансе, вршећи целокупан свој рад у року од просечно секунде.

Иако се перформансе паралелног извршавања и овај пут очевидно побољшавају за веће CUTOFF вредности, сам процес додељивања задатака и и чекања на њихово извршење заузима време неколико редова величине веће од оног које је потребно једноставном пролазу кроз for петљу реализованом у серијском коду. Ипак, истаћи ћемо најбољу просечну брзину паралелног рада, која износи секунди, и то када је CUTOFF подешен на .

## Закључак:

Циљ који је остварен у овом пројекту је тај да се показао ефекат паралелизације серијског кода у приметном убрзању сопственог извршења. На местима где се рукује великим скупом података уочили смо простор за унапређење и, познавањем TBB технологије и концепата паралелног програмирања, успешно смо спровели исто у реализацију.

Такође смо увидели и мане и ограничења овог алата, у томе да су се одређене операције показале временски захтевним за коришћење у посматраним условима. Међутим, оставили смо и те резултате у извештају, зато што у свету инжењерства и неуспех може представљати неку врсту успеха, у виду емпиријског увиђаја о датој теми, корисног за будући рад.