Қазақстан Республикасының Білім және ғылым министрлігі

Әл-Фараби атындағы Қазақ ұлттық университеті

Назынбек Нарқыз Асқатқызы

СЫЗЫҚТЫҚ ТЕҢДЕУЛЕР ЖҮЙЕСІН ШЕШУДІҢ ЖОҒАРЫӨНІМДІ АЛГОРИТМДЕРІН ӘЗІРЛЕУ ЖӘНЕ ТАЛДАУ

ДИПЛОМДЫҚ ЖҰМЫС

5В070400-«Есептеу техникасы және бағдарламалық қамтамасыз ету»

Алматы, 2022

Қазақстан Республикасының Білім және ғылым министрлігі

Әл-Фараби атындағы Қазақ ұлттық университеті

Ақпараттық технологиялар факультеті

Информатика кафедрасы

ДИПЛОМДЫҚ ЖҰМЫС

Тақырыбы: «СЫЗЫҚТЫҚ ТЕҢДЕУЛЕР ЖҮЙЕСІН ШЕШУДІҢ ЖОҒАРЫӨНІМДІ АЛГОРИТМДЕРІН ӘЗІРЛЕУ ЖӘНЕ ТАЛДАУ»

5В070400-«Есептеу техникасы және бағдарламалық қамтамасыз ету»

Орындаған

Ғылыми жетекші,

PhD, доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(қолы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(қолы)

Назынбек Н.А.

Бекбаева М.К.

Қорғауға жіберілді:

Хаттама № , « » \_\_\_\_\_\_\_\_\_ 2022 ж.

Кафедра меңгерушісі \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дарибаев Б.С.

(қолы және мөрі)

Норма бақылаушы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Түркен Г.

(қолы)

Алматы, 2022

**АҢДАТПА**

Дипломдық жұмыс кіріспе, негізгі бөлім, қорытынды, пайдаланылған әдебиеттер тізімі мен қосымшалардан тұрады. Дипломдық жұмыста 53 бет, 14 сурет, 3 кесте, 8 әдебиет, 1 қосымша бар.

Кілттік сөздер: СЫЗЫҚТЫҚ АЛГЕБРАЛЫҚ ТЕҢДЕУЛЕР ЖҮЙЕСІ, ГАУСС ӘДІСІ, БІРІКТІРІЛГЕН ГРАДИЕНТТЕР ӘДІСІ, OPENMP ПАРАЛЛЕЛЬДЕУ, САНДЫҚ ТАЛДАУ, ФАЙЛМЕН ЖҰМЫС.

Зерттеу нысаны – сызықтық алгебралық теңдеулер жүйесін шешудің жоғарыөнімді алгоритмдерін С++ тілінде әзірлеу, кодты талдау.

Жұмыстың мақсаты: теңдеулер жүйесін шешу алгоритмдерін әзірлеп, оның сызықты және параллель орындалуын есептеу уақыты бойынша талдау.

Дипломдық жұмысты орындау барсында теңдеулер жүйесін шешудің Гаусс және біріктірілген градиенттер әдістері қарастырылды. Алгоритмді әзірлеу үшін Visual Studio бағдарламалау ортасы таңдалды. Параллельдеу үшін OpenMP кітапхана мүмкіндіктері қолданылды. Есептеу эксперименттерінде сызықтық алгебралық теңдеулер жүйесінен құралған коэффициенттер матрицасы диагональды, үшдиагональды және қарапайым болған жағдайлары қарастырылған. Эксперимент нәтижелері талданды.

**РЕФЕРАТ**

Дипломная работа состоит из введения, основной части, заключения, списка использованной литературы и приложения. Дипломная работа содержит 53 страницу, 14 рисунок, 3 таблицы, 8 литературу, 1 приложение.

Ключевые слова: СИСТЕМА ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ, МЕТОД ГАУССА, МЕТОД ИНТЕГРАЛЬНЫХ ГРАДИЕНТОВ, OPENMP PARALLELIZATION, ЦИФРОВОЙ АНАЛИЗ, РАБОТА С ФАЙЛАМИ.

Объект исследования - разработка высокопроизводительных алгоритмов решения систем линейных алгебраических уравнений на языке C++, анализ кода.

Цель работы: разработать алгоритмы решения систем уравнений и проанализировать их линейное и параллельное выполнение во времени.

В ходе работы были рассмотрены гауссовские и комбинированные градиентные методы решения систем уравнений. Для разработки алгоритма была выбрана среда программирования Visual Studio. Для параллелизма использовались возможности библиотеки OpenMP. В вычислительных экспериментах матрица коэффициентов, строящаяся из системы линейных алгебраических уравнений, считается диагональной, трехдиагональной и простой. Результаты эксперимента были проанализированы.

**ABSTRACT**

The thesis consists of an introduction, a main part, a conclusion, a list of references and an appendix. The thesis contains 53 pages, 14 drawing, 3 tables, 8 literatures, 1 appendix.

Keywords: SYSTEM OF LINEAR ALGEBRAIC EQUATIONS, GAUSSIAN METHOD, INTEGRATED GRADIENT METHOD, OPENMP PARALLELIZATION, DIGITAL ANALYSIS, FILE WORK.

Object of research - development of high-performance algorithms for solving systems of linear algebraic equations in C ++, code analysis.

The purpose of the work: to develop algorithms for solving systems of equations and to analyze their linear and parallel execution over time.

Gauss and combined gradient methods for solving systems of equations were considered in the course of the thesis. The Visual Studio programming environment was chosen to develop the algorithm. OpenMP library features were used for parallelism. In computational experiments, the matrix of coefficients consisting of a systems of linear algebraic equations is considered to be diagonal, three-diagonal and simple. The results of the experiment were analyzed.

**МАЗМҰНЫ**

[**БЕЛГІЛЕУЛЕР МЕН ҚЫСҚАРТЫЛҒАН СӨЗДЕР** 6](#_Toc104161084)

[**КІРІСПЕ** 7](#_Toc104161085)

[**1** **ГАУСС ӘДІСІ** 10](#_Toc104161086)

[**1.1** **Сипаттамасы және модификациялары** 10](#_Toc104161087)

[**1.2** **Жетекші элементті таңдау алгоритмін бағдарламалау** 20](#_Toc104161088)

[**1.3** **Жетекші элементті таңдау алгоритмін параллелдеу** 22](#_Toc104161089)

[**2** **БІРІКТІРІЛГЕН ГРАДИЕНТТЕР ӘДІСІ** 26](#_Toc104161090)

[**2.1** **Сипаттамасы және модификациялары** 26](#_Toc104161091)

[**2.2** **CG әдісінің алгоритмін бағдарламалау** 32](#_Toc104161092)

[**2.3** **CG алгоритмін параллелдеу** 35](#_Toc104161093)

[**3** **ЕСЕПТЕУ ЭКСПЕРИМЕНТТЕРІ ЖӘНЕ ТАЛДАУ** 37](#_Toc104161094)

[**3.1** **Эксперименттің жүргізілу шарттары** 37](#_Toc104161095)

[**3.2** **Эксперименттер қоюға қажетті қосымша класстар жайлы** 38](#_Toc104161096)

[**3.3** **Эксперимент нәтижелерін алу және талдау** 44](#_Toc104161097)

[**ҚОРЫТЫНДЫ** 52](#_Toc104161098)

[**ҚОЛДАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ** 53](#_Toc104161099)

[**ҚОСЫМША** 54](#_Toc104161100)

# **БЕЛГІЛЕУЛЕР МЕН ҚЫСҚАРТЫЛҒАН СӨЗДЕР**

САТЖ – сызықтық алгебралық теңдеулер жүйесі.

 – өлшемді коэффициенттер матрицасы.

 – бос мүшелер векторы.

 – Айнымалылар векторы.

 – теңдеулер жүйесінің матрицалық түрде жазылуы.

 –  және  векторларын скалярлы көбейту, басқаша .

OpenMP – Open Multi-Processing – C, C++ және Fortran тілдеріндегі бағдарламаларды параллелдеуге арналған ашық стандарт.

CG – conjugate-gradient – біріктірілген градиент әдісі.

# **КІРІСПЕ**

Көптеген қолданбалы, оның ішінде экономикалық , табиғат құбылыстары, өндірістік инженерия, кескінді тану және т.б. сызықтық теңдеулер жүйесіне әкелінеді.

 айнымалысы бар және  сызықтық теңдеулерден тұратын жүйе келесідей жазылады:

 (1)

мұндағы  - сәйкесінше айнымалылардың коэффициенттері және теңдеулердің бос мүшелері деп аталатын ерікті сандар.[1]

Қысқаша белгілеуде, жинақтау белгілерін пайдаланып, жүйені келесідей жазуға болады:

 (2)

1. *жүйенің шешімі* деп жүйенің әр теңдеуіндегі белгісіз айнымалы орнына қойғанда  шындыққа айналатын  сандар жиынын айтамыз.[2]

Сызықтық алгебралық теңдеулер жүйесінің дым аз дегенде бір шешімі бар болса, онда оны *үйлесетін* (совместной), ал шешімдері жоқ болса *үйлеспейтін* (несовместной) деп аталады.

Үйлесетін теңдеулер жүйесінің жалғыз шешімі болса, *анықталған*, ал бірнеше шешімі болса, *анықталмаған* деп аталады. Мысалы,  - теңдеулер жүйесі үйлесімді және анықталған, себебі жүйенің бір-ақ қана шешімі бар, және ол ;  - теңдеулер жүйесі үйлеспейтін болып табылады, себебі жүйені қанағаттандыратын ешқандай сандар жиынын таба алмаймыз;  - теңдеулер жүйесі үйлесімді және анықталмаған, себебі жүйенің бірден көп, басқаша айтқанда шексіз көп шешімі бар, және ол  мұндағы  - кез – келген сан.[3-4]

Егер екі жүйенің шешімдер жиыны бірдей болса, онда оларды *эквивалентті* деп атайды. (1) – жүйені элементар түрлендірулерінің көмегімен (мысалы, теңдеулердің екі бөлігін де нөлге тең емес сандарға көбейту; жүйенің теңдеулерін бір – біріне қосу) берілген (1) жүйеге эквивалентті теңдеулер жүйелерін аламыз.

1. Жүйені матрицалық формада жазайық.



мұндағы  - айнымалы алдындағы коэффициенттер матрицасы, немесе жүйе матрицасы;  - айнымалылар баған – матрицасы;  - бос мүшелер баған – матрицасы.

 матрицасының бағандар саны мен  матрицасының жолдар саны тең болғандықтан, оларды көбейтуге болады. Нәтижесінде:



 баған-матрицаны аламыз. Және ол (1) жүйенің сол жағы болып табылады. Матрицалардың теңдік ережесі бойынша (1) жүйені келесі түрде жазсақ болады:

 (3)

Теңдеулер жүйесін шешудің барлық әдістерін шартты түрде дәл (нақты) және жуық деп бөлуге болады. Дәл алгоритмдерге Крамер, Гаусс, Джордан-Гаусс, прогонка және т.б. әдістер жатады. Жуықтап есептеу әдістеріне итерациялық әдістер (Якоби, Зейдель, релаксация, біріктірілген градиенттер және т.б.), квадрат түбірлер әдісін және т.б. жатады.[5]

**Дипломдық жұмыстың өзектілігі.** Тәжірибеде өте үлкен өлшемдегі сызықтық теңдеулер жүйесін шешуге тура келеді. Мысалыға, экономикада кіріс-шығыс балансын құрастыру кезінде белгісіздер саны жүзден асатын жүйелер кездеседі. Бұл тақырыптың өзектілігі осындай сияқты жүйелерді шешуде қолданылатын әдістерді зерттеуге және талдауға арналған

**Дипломдық жұмыстың мақсаты**:

* САТЖ шешудің тура және итерациялық әдістерін зерттеу
* САТЖ шешудің алгоритмін программалау тілінде құру
* Құрылған алгоритм коды бойынша есептеу эксперименттерін жүргізіп, алынған нәтижелер бойынша анализ жасау

**Дипломдық жұмыстың тапсырмалары:**

* САТЖ шешудің тікелей және итерациялық әдістеріне шолу
* Гаусс және CG әдісінің сызықты алгоритмін С++ тілінде жазу
* Гаусс және CG әдісінің параллельді алгоритмін С++ тілінде жазу
* Эксперименттер өткізіп, нәтижесі бойынша талдау жүргізу

# **ГАУСС ӘДІСІ**

## **Сипаттамасы және модификациялары**

Сызықтық алгебралық теңдеулер жүйесінің(САТЖ) бір шешімі де шексіз көп шешімі де шешімдері жоқ болуы да мүмкін. САТЖ шешудің барлық әдістері екінші жағдайды, яғни жүйенің шексіз көп шешімдері болған жағдайда шешімнің біреуін де таба алмайды. Мысалы, Крамер әдісі мен матрицалық әдіс қолданылмайды, алайда Гаусс әдісімен шешуге болады.

Бұл әдістің тарихына шолу жасайтын болсақ, бұл әдіс Карл Фридрих Гауссқа дейін де белгілі болғанын байқаймыз. Әдістің алғашқы белгілі сипаттамасы біздің дәуірімізге дейінгі I ғасыр және II ғасыр арасында құрастырылған қытайлық «Тоғыз кітаптағы математика» трактатында көрсетілген[6]. САТЖ шешудің Гаусс әдісін кей оқулықтарды Гаусстық жою әдісі деп те атайды.

Гаусс әдісі – сызықтық алгебралық теңдеулер жүйесін шешудің классикалық әдісі. Бұл элементар түрлендірулерді қолдана отырып, теңдеулер жүйесі сатылы (немесе үшбұрышты) түрдегі эквивалентті жүйеге келтірілгенде, айнымалы мәндерді дәйекті жою әдісі, оның ішінен барлық басқа айнымалылар соңғысынан бастап дәйекті түрде табылды.

Жүйедегі (1)  айнымалысының алдында бірінші теңдеуде нөлге тең емес  деп есептейік. Егер нөлге тең болса, онда теңдеулердің орнын ауыстырып нөлге тең емес жағдайына келтіреміз.

*1 – қадам.* САТЖ – нің бірінші теңдеуін сәйкес сандарға көбейтіп (атап айтқанда ) және алынған теңдеулерді (1) жүйенің сәйкесінше екінші, үшінші, …, - ші теңдеулеріне қоссақ, бірінші теңдеуден басқа, яғни екінші теңдеуден бастап  айнымалысынан құтыламыз.

 (4)

мұндағы үстіндегі  белгісі бірінші қадамнан кейінгі пайда болған жаңа коэффициентті білдіреді.

*2 – қадам.* Жүйеде (4)  деп есептейік. Егер нөлге тең болса, онда теңдеулердің орнын ауыстырып нөлге тең емес жағдайына келтіреміз. (4) жүйенің екінші теңдеуін сәйкес сандарға көбейтіп (атап айтқанда ) және алынған теңдеулерді (4) жүйенің сәйкесінше үшінші, төртінші, …, - ші теңдеулеріне қоссақ, үшінші теңдеуден бастап  айнымалысынан құтыламыз.



мұндағы үстіндегі  белгісі екінші қадамнан кейінгі пайда болған жаңа коэффициентті білдіреді.

Осылайша қадам жалғастыра берсек, әр қадамда  айнымалыларынан құтыламыз. Және соңғы  қадамда келесі жүйені аламыз

 (5)

Соңғы  теңдеуіндегі нөл саны олардың сол жақ бөліктері  түрінде болатынын білдіреді. Егер де (5) жүйенің  бос мүшелерінің ең болмағанда біреуі нөлге тең болмаса, онда осы теңдік қарама-қайшы болады да, (1) жүйе үйлеспейтін болып саналады, яғни (1) жүйенің шешімі жоқ болады.

Осылайша, кез-келген үйлесімді САТЖ-үшін (5) жүйеде  сандары нөлге тең. Бұл жағдайда, (5) жүйенің соңғы  теңдеуі қарама-қайшылық көрсетпейді, және де (1) жүйені шешу кезінде елемеуге болады. Әлбетте, «артық» теңдеулерді алып тастағаннан кейін екі жағдай болуы мүмкін: а) (5) жүйесіндегі теңдеулер саны айнымалылар санына тең, яғни  (бұл жағдайда жүйе үшбұрыш пішінге ие); б) r < n (бұл жағдайда (5) жүйе сатылы пішінге ие).

(1) жүйенің оның эквивалентті жүйесіне (5) өтуі Гаусс әдісінің *тура жүріс*, ал (5) жүйесінен айнымалыларды табу *кері жүріс* деп аталады.

Гаусс түрлендірулерін теңдеулердің өздерімен емес, олардың коэффициенттерінің матрицасымен түрлендіруді орындау арқылы жүргізу ыңғайлы. Келесідей матрицаны қарастырайық:

 (6)

(1) жүйенің *кеңейтілген матрицасы* деп аталады, өйткені ол жүйенің А матрицасына қосымша бос мүшелер бағанын қосылған.

*Мысалы,* Берілген САТЖ Гаусс әдісімен шешімін тап.



*Шешімі.* САТЖ – ның кеңейтілген матрицасы келесі түрде болады:



*1 – қадам.*  болғандықтан, матрицаның бірінші жолын (-2), (-3), (-2) сандарына көбейтіп, және сәйкес екінші, үшінші және төртінші жолдарға қосамыз. Осының арқасында екінші жолдан бастап, бірінші баған элементтері нөлге тең болады, басқаша айтқанда екінші теңдеуден бастап  айнымалысын жоямыз.



Келесі қадам бастамас бұрын нөлге тең көріп тұрмыз, демек матрица жолдарын өзара орнын ауыстырып нөлге тең емес жағдайына келтіреміз.



*2 – қадам.*  матрицаның екінші жолын (-7/4) санына көбейтіп, төртінші жолға қосамыз. Осының арқасында үшінші жолдан бастап, екінші баған элементтері нөлге тең болады, басқаша айтқанда үшінші теңдеуден бастап  айнымалысын жоямыз.



*3 – қадам.*  болғандықтан, матрицаның үшінші жолын (13,5 / 8 = 27 / 16) санына көбейтіп, және сәйкес төртінші жолға қосамыз. Сонда келесі матрица шығады:



Матрица түрін жүйе түріне аударып жазсақ. Келесі жүйені аламыз:



Соңғы жүйені шешу үшін астынан үстіге қарай айнымалы мәндерін есептеп отырамыз. Осы кезеңін Гаусс әдісінің кері жүрісі деп атайды. Төртінші теңдеуден ; осы шешімді алдындағы теңдеуге (атап айтқанда үшінші теңдеуге) қойып есептейміз ; ал екінші теңдеуге қойып ; ал бірінші теңдеуге қойып . Яғни, теңдеу шешімі .

Есептеу сынақтарын жасау кезінде, және талдау кезінде де (1) жүйенің үйлесімді болу мүмкіндігін көбейту үшін біз айнымалылар саны  мен теңдеулер санын  тең қыламыз, басқаша айтқанда  матрицасы квадратты түрде болады.

Гаусс әдісін бағдарламалауды бастамас бұрын, әдістің келесі қадамын есептеудің жалпы математикалық теңдеуін есептеп алайық.

Тура жүріс кезеңінде, келесі қадам үшін таңдалған коэффициент астындағы коэффициенттердің мәнін есептеу формуласы:

 (7)

Ал кері жүріс кезеңінде, коэффициенттерді және де есептелінген айнымалыларды ескере отырып, келесі айнымалыны есептеу формуласы

 (8)

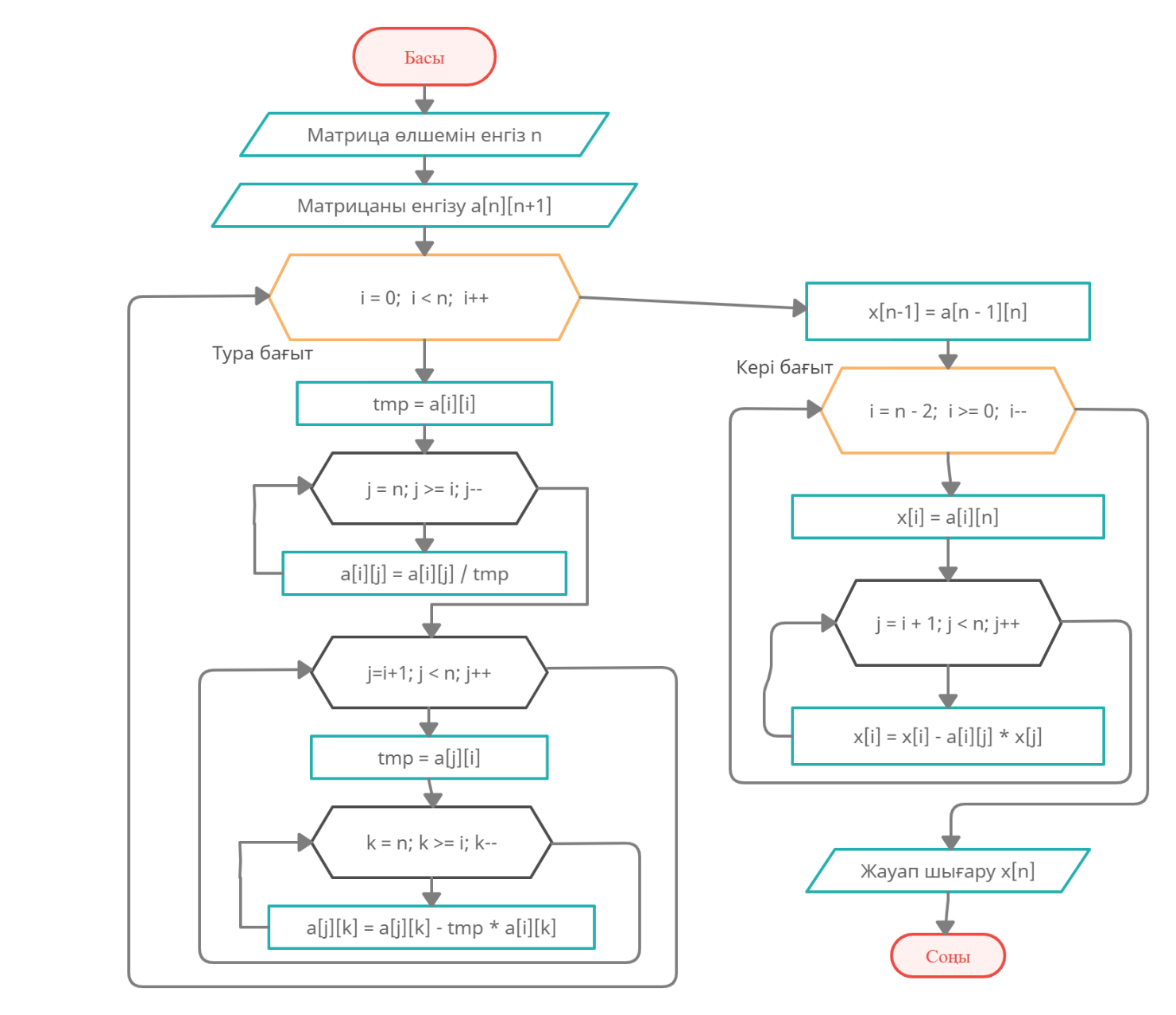
Жүйені (1) және (7) формуланы біріктіріп тура жүріс кезеңін есептеудің жалпы формуласын аламыз:



Ал кері жүріс кезеңіндегі есептеудің жалпы формула



Енді екі кезеңді қосып, Гаусс әдісінің алгоритмін блок-схемасын құрастырайық. (1 – сурет)



1 – сурет. Гаусс әдісінің алгоритмі блок – схема түрінде

Алайда, классикалық алгоритмі бағдарламалап, есептеу кезінде диагональ элементінің кіші болған кезде, қателіктердің жиналуы байқалды, ол есептеуіш бағдарламаның тұрақсыздығына әкеледі.

Бұл мәселені болдырмаудың ықтимал жолы – бағдарламалауға Гаусс әдісінің модификацияланған алгоритмін қолдану. Модификацияланған әдістің классикалық әдістен айырмашылығы тура жүріс кезінде баған элементтері ішіндегі абсолютті мәні бойынша ең үлкен элемент тұрған жолды жетекші қылып, осы бағандағы басқа элементтерді жоямыз. Сондай – ақ есептеу жылдамдығын тағы да өсіру үшін, жолдардың орнын ауыстыру операциясын орындамаймыз. Жою барысында егер де қандай да бір жол осыған дейінгі қадамдарда жетекші болған болса, сол жолдың элементтерін секіріп өтіп кетеміз.

*Мысалы*, Берілген САТЖ – ны Гаусс әдісінің баған бойынша жетекші элементті таңдауымен шешімін тап.



*Шешімі.* САТЖ – ның кеңейтілген матрицасы келесі түрде болады:



*1 – қадам.*  1 – ші бағанда, абсолютті мәні бойынша ең үлкен элемент 4 – ші жолда орналасқан болғандықтан, матрицаның төртінші жолын  сандарына көбейтіп, сәйкес бірінші, екінші және үшінші жолдарға қосамыз. Осының арқасында бірінші, екінші және үшінші жолдардың, бірінші бағанында элементтері нөлге тең болады, басқаша айтқанда төртінші жолдан басқа теңдеулерде  айнымалысын жоямыз.



*2 – қадам.* 2 – ші бағанда, абсолютті мәні бойынша ең үлкен элемент 4 – ші жолда орналасқан. Алайда, 4 – ші жол бізде алдыңғы қадамдарда жетекші жол болғандықтан, оны ала алмаймыз, осы себептен 2 – ші бағанда ең үлкен элемент ретінде 2 – ші жолда тұрған элементті айтамыз. Екінші жолды  сандарына көбейтіп, сәйкесінше бірінші және үшінші жолға қосамыз. Осы арифметикалық түрлендірулердің арқасында бірінші және үшінші жолдың екінші баған элементтері нөлге тең болады, басқаша айтқанда бірінші және үшінші теңдеуде  айнымалысын жоямыз.



*3 – қадам.* Бірінші және үшінші жолдар ішінен 3 – ші бағанында, абсолютті мәні бойынша ең үлкен элемент 1 – ші жолда орналасқан. Бірінші жолды  сандарына көбейтіп, үшінші жолға қосамыз. Осы арифметикалық түрлендірулердің арқасында үшінші жолдың үшінші баған элементі нөлге тең болады, басқаша айтқанда үшінші теңдеуде  айнымалысын жоямыз.

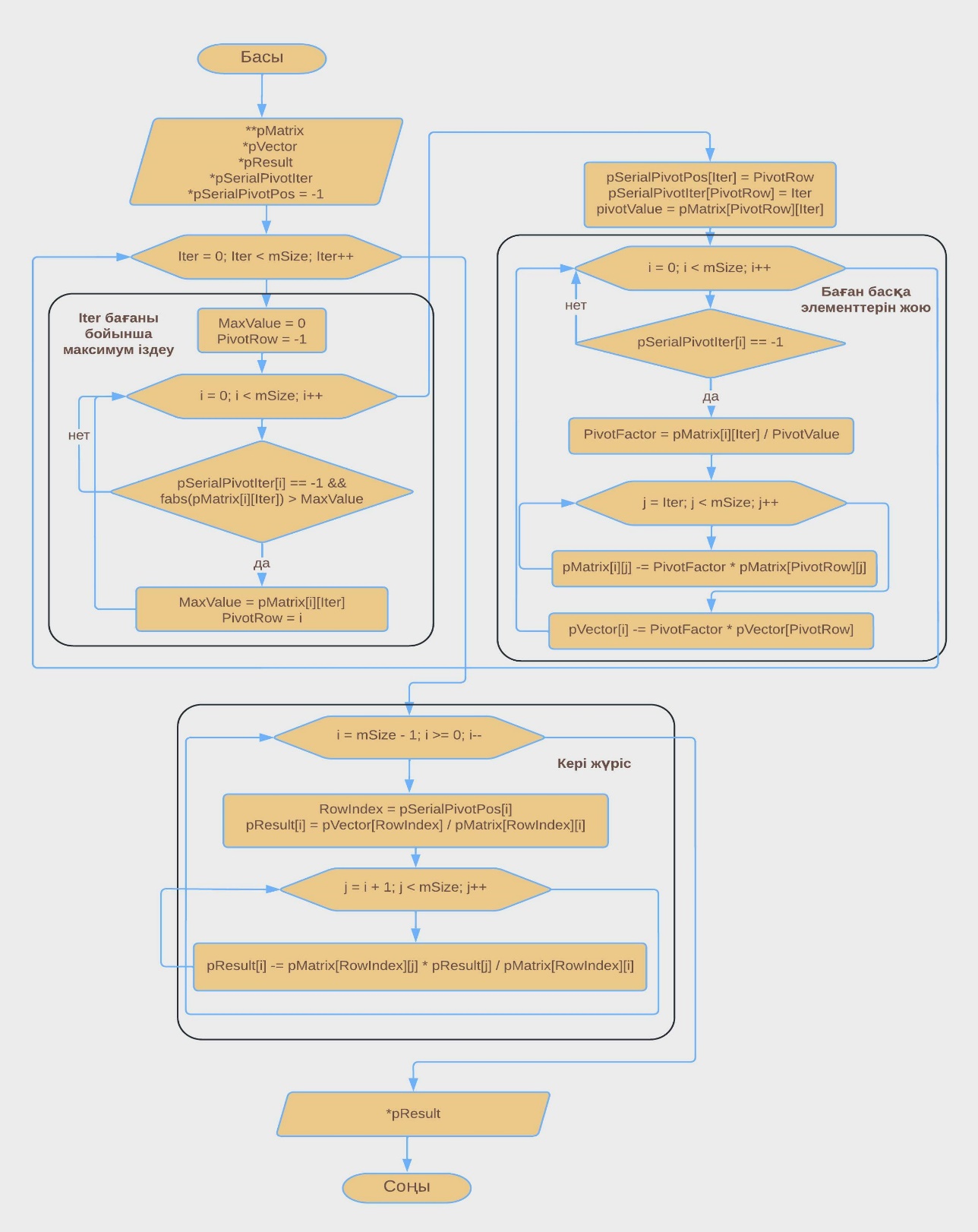


Осымен тура жүріс кезеңі аяқталады. Соңғы шыққан матрица түрін жүйе түріне аударып жазсақ. Келесі жүйені аламыз:



Соңғы жүйені шешу үшін тура жүріске қарсы жүріс жасаймыз. Яғни бірінші үшінші теңдеуді шешеміз, сол кезде біз  шешімін аламыз. Осы шешімді бірінші теңдеуге қойып,  есептеп аламыз; ал екінші теңдеуге қойып ; ал төртінші теңдеуге қойып . Яғни, теңдеу шешімі .

Енді Гаусс әдісінің баған бойынша жетекші элементті таңдау алгоритмін блок-схемасын құрастырайық. (2 – сурет)



2 – сурет. Гаусс әдісінің жетекші элементті таңдауы алгоритмі блок – схемасы

## **Жетекші элементті таңдау алгоритмін бағдарламалау**

Енді 2 – суретте көрсетілген блок – схема бойынша С++ тілінде бағдарлама жазамыз. Бағдарламамызды GaussSerial класс құрудан бастаймыз. Бұл класс бірнеше әдістен тұрады. Әдістер:

* Класстың конструкторы;
* Максимумды іздеу әдісі;
* Баған басқа элементтерін жою әдісі;
* Кері жүріс – айнымалыларды есептеу әдісі;
* Көрсетілген әдістерді ретімен орындау әдісі.

Осы әдістерге тоқталсақ.

*Класс конструкторы*. Есептеу орындалу барысында екі ақпаратты, әр жолдың жетекші болғаны және қай итерацияда қай жол жетекші болғанын сақтайтын екі массив қажет. pSerialPivotIter массиві тура жүріс кезінде алдыңғы итерацияларда жетекші болған жолдарды арналған. Басында бұл массивтің барлық элементі -1 тең болады. pSerialPivotPos массиві кері жүріс кезінде теңдеулерді шешу ретін анықтауға қажет. Бұл әдістің коды келесідей болады:

pSerialPivotIter = new int [mSize];

pSerialPivotPos = new int [mSize];

for (int j = 0; j < mSize; j++) {

pSerialPivotIter[j] = -1;

}

*Максимумды іздеу әдісі.* Бұл әдісте біз баған бойынша барлық жолды циклмен өтіп модулі бойынша үлкен және pSerialPivotIter массивіне жол индексі -1 – ге тең жолдарды іздейді. Әдіс ішінде екі айнымалы қолданылады. MaxVal – цикл бағандағы модулі бойынша үлкен элементтің мәні, PivotRow – цикл бағандағы модулі бойынша үлкен элемент орналасқан жолдың индексі. Бұл әдістің коды келесідей болады:

double MaxVal = 0;

int PivotRow = -1;

for (int i = 0; i < mSize; i++) {

if ((pSerialPivotIter[i] == -1) && (fabs(pMatrix[i][Iteration]) > MaxVal)) {

PivotRow = i;

MaxVal = fabs(pMatrix[i][Iteration]);

}

}

return PivotRow;

*Баған басқа элементтерін жою әдісі*. Бұл әдісте біз алдыңғы итерацияларда жетекші болмаған жолдардың баған элементін жоямыз, нөлге айныдырамыз. Сондай – ақ, жойылатын бағаннан кейінгі тұрған элементтерге де және бос мүшелер векторына да арифметикалық түрлендіруді орындаймыз. PivotValue – баған бойынша үлкен мәні, PivotFactor алгебралық түрлендіру үшін көбейту мәні. Жоюды орындау кезінде, алдыңғы жоюлар орындалған кездегі жетекші болған жолдарға алгебралық түрлендірулер орындалмауы керек. Оны қадағалау үшін pSerialPivotIter массивіндегі сәйкес жолдың мәні -1 – ге тең болмауы керек. Бұл әдістің коды келесідей болады:

Int i, j;

Double; PivotValue = pMatrix[PivotRow][Iteration];

for (i = 0; i < mSize; i++) {

if (pSerialPivotIter[i] == -1) {

double PivotFactor = pMatrix[i][Iteration] / PivotValue;

for (j = Iteration; j < mSize; j++) {

pMatrix[i][j] -= PivotFactor \* pMatrix[PivotRow][j];

}

pVector[i] -= PivotFactor \* pVector[PivotRow];

}

}

*Кері жүріс – айнымалыларды есептеу әдісі.* Бұл әдісте жетекші болу ретімен жазылған pSerialPivotPos массивіне соңынан басына қарай жүре отырып, айнымалыларды есептеп pResult массивіне жазып отырамыз. Бұл әдістің коды келесідей болады:

Int j, k, Row;

for (j = mSize - 1; j >= 0; j--) {

Row = pSerialPivotPos[j];

pResult[j] = pVector[Row] / pMatrix[Row][j];

for (k = j + 1; k < mSize; k++) {

pResult[j] - = pMatrix[Row][k] \* pResult[k] / pMatrix[Row][j];

}

}

*Көрсетілген әдістерді ретімен орындау әдісі.* Бұл әдіс – есептеуді бастаушы әдіс. Мұнда әдістерді орындау реті көрсетіледі. Бағандарды кезегімен жою үшін итерациямен орындаймыз. Бұл әдістің коды келесідей болады:

for (int Iteration = 0; Iteration < mSize; Iteration ++) {

int PivotRow = findPivotRow(pMatrix, Iteration);

pSerialPivotIter[PivotRow] = Iteration;

pSerialPivotPos[Iteration] = PivotRow;

serialColumnElimination(pMatrix, pVector, PivotRow, Iteration);

}

Осы кодтардың бәрін орындау үшін екі өлшемді \*\*pMatrix және бос мүшелер векторы \*pVector енгізіледі. Шешімдер векторы \*pResult - ке жазылады.

## **Жетекші элементті таңдау алгоритмін параллелдеу**

OpenMP кітапханасы математикалық есептеулерде жиі пайдаланылады, өйткені бағдарламаның жеке процедуралар мен алгоритмдердің параллелизациясы өте жылдам және көп қиындықсыз параллелдеуге мүмкіндік береді. Оларға параллель сұрыптау, матрицаны көбейту және сызықтық теңдеулер жүйесін шешу алгоритмдері жатады. Абстракцияның бұл деңгейінде OpenMP сияқты параллельді бағдарламалау технологиясын пайдалану ыңғайлы.

OpenMP (Open Multi-Processing) – көп процессорлы жүйелерде көп ағынды қосымшаларды бағдарламалауға арналған компилятор директивалары, кітапхана процедуралары және орта айнымалы мәндерінің жиынтығы. OpenMP тармақтарды біріктіру параллельді орындау үлгісін пайдаланады. OpenMP бағдарламасы бастапқы ағын деп аталатын орындаудың жалғыз ағыны ретінде басталады. Жіп параллельді құрылымды кездестіргенде, ол өзінен және бірқатар қосымша ағындардан тұратын жаңа ағындар тобын жасайды және жаңа топтың көшбасшысы болады. Жаңа топтың барлық мүшелері (негізгі ағынды қоса) параллельді құрылымда кодты орындайды. Параллельді конструкцияның соңында жасырын тосқауыл бар. Параллельді құрастырудан кейін пайдаланушы кодының орындалуы тек негізгі ағында жалғасады. Басқа параллель аймақтарды параллель аймаққа салуға болады.

«Үлкен параллелизация» идеясының арқасында OpenMP үлкен параллельді циклдері бар (матрицаны матрицаға көбейту, матрицаны векторға көбейту және т.с.с.) есептеу бағдарламаларын қиындықсыз параллельдеудегісі келетін бағдарламашылар үшін өте ыңғайлы. Бағдарламашы жаңа параллельді бағдарламаны құрастырмайды, тек қана құрастырылған бағдарлама алгоритміне OpenMP директиваларын дәйекті түрде қосылады.

Параллельді алгоритмдерді енгізу міндеті өте күрделі, сондықтан деректерді параллель өңдеуді жүзеге асыру құрылғысына кірмей текшелерден бағдарламаларды құруға мүмкіндік беретін параллель алгоритмдердің жеткілікті үлкен саны бар.

Көрсетілген артықшылықтарды ескере отырып, алгоритмді параллелдеуге OpenMP кітапханасы қолданылады. Сондай-ақ, сынақтарды бірнеше ағындар санымен қойылатынын ескеріп, параллелдеу барысында ағын санын оңай басқаратын кілтсөздер қосуымыз керек.

Бағдарламамызды GaussParallel класс құрудан бастаймыз. Бұл класс бірнеше әдістен тұрады. Әдістер:

* Класстың конструкторы;
* Максимумды іздеу әдісі;
* Баған басқа элементтерін жою әдісі;
* Кері жүріс – айнымалыларды есептеу әдісі;
* Көрсетілген әдістерді ретімен орындау әдісі.

Көрсетілген әдістердің бәрін алдыңғы бөлімде жазылған GaussSerial классынан аламыз. Осыдан кейін параллелдеуге келетін бөліктерге параллелдеу кілттерін қойып параллелдеу алгоритмін бастаймыз. Енді қай әдісте, қай бөлікті параллелдейтінімізді нақтырақ көрсетейік.

*Класс конструкторы*. Бұл әдісте pSerialPivotIter -1 – ге меншіктеу қайталау операциясы бар. Осы циклды параллелдеу үшін OpenMP – дің #pragma omp parallel for кілтсөзін циклдің алдына қосып жазамыз. Сынақтарды бірнеше ағындар санымен қойылатынын ескеріп, параллелдеу барысында ағын санын оңай басқаратын кілтсөзді осы классқа қосу керек. OpenMP ағындар санын басқарудың оңай әдісін ұсынады. omp\_set\_num\_threads(threads\_count) функциясы жақша ішіне натурал сан енгізіледі. Біздің жағдайда бұл айнымалылар {2, 4, 6, 8, 10, 12}. Бұл әдістің коды келесідей болады:

omp\_set\_num\_threads(threads\_count);

pSerialPivotIter = new int [mSize];

pSerialPivotPos = new int [mSize];

#pragma omp parallel for

for (int j = 0; j < mSize; j++) {

pSerialPivotIter[j] = -1;

}

*Максимумды іздеу әдісі.* Бұл әдісте максимумды іздеу процессін бірнеше ағынға бөлсек болады. Әр ағын өзі алған бөліктен максимумды тауып, басқа ағындар да тауып болғаннан кейін бір-бірімен салыстырады. Енді осы ойымызды іске асыру үшін әр ағын ішінде MaxVal және PivotRow айнымалыларын құрамыз. Осы екі айнымалыны TThreadPivotRow деген тип құрастырып, осының ішіне салып қояйық. Содан кейін әр ағын басталғанда өздері үшін осы типтегі ThreadPivotRow айнымалысын жасап алсын, OpenMP – дің бөліктерді параллелдеу #pragma omp parallel кілтсөздерін қолданамыз. Осы кілтсөзден кейін біз жүйелік жақша ашып ішінде ThreadPivotRow айнымалысын жасаймыз. Ары қарай максимумды іздеу циклін параллелдейік, сол кезде бір жол бір ағында ғана қаралады. Циклді параллелдеу үшін #pragma omp for кілтсөзін қолданамыз. Осы цикл аяқталған соң, біз әр ағыннан шыққан максимумды басқа ағындардан табылған максимумдер мен салыстырып, барлық ағын арасынан максимумды табамыз. Оны орындау үшін #pragma omp critical кілтсөзін қолданамыз. Бұл әдістің коды келесідей болады:

double MaxVal = 0;

int PivotRow = -1;

int i;

#pragma omp parallel

{

TThreadPivotRow ThreadPivotRow;

ThreadPivotRow. MaxVal = 0;

ThreadPivotRow. PivotRow = -1;

#pragma omp for

for (i = 0; i < mSize; i++) {

if ((pSerialPivotIter[i] == -1) && (fabs(pMatrix[i][Iteration]) > ThreadPivotRow. MaxVal)) {

ThreadPivotRow. PivotRow = i;

ThreadPivotRow. MaxVal = fabs(pMatrix[i][Iteration]);

}

}

#pragma omp critical

{

if (ThreadPivotRow. MaxVal > MaxVal) {

MaxVal = ThreadPivotRow. MaxVal;

PivotRow = ThreadPivotRow. PivotRow;

}

}

}

return PivotRow;

*Баған басқа элементтерін жою әдісі*. Бұл әдісте жолдарға алгебралық түрлендіруді орындау процессін параллелдеуге болады. Егер біз әр жолды әр ағында орындайтын болсақ, есептеуге кететін уақыт азаяды. Демек әр жол үшін өзінің көбейтілу мәні бар болғандықтан, әр ағында да тек өзіне ғана тиісті мәні болу керек. Мұндай мүмкіндікті іске асыруға бізге OpenMP – дің private (PivotFactor) кілтсөзі көмектеседі. Кілтсөзді әр ағында PivotFactor айнымалысы әртүрлі мәнге ие болатындықтан, сол айнымалыны жақшаға ішіне жаздық. Келесі ескеретін жайт, алдыңғы итерацияларда жетекші болған жолдарға алгебралық түрлендірулер орындауға болмайды. Осы жайтты параллелдеу кезінде тиімді пайдалану үшін OpenMP – дің schedule (dynamic, 1) кілтсөзін қолданамыз. Кілтсөз циклдің келесі орындалуын бос ағынға салу үшін, сондай – ақ тек қана бір ғана циклді салу үшін жазылды. Негізгі мақсат – егер ағынға берілген жол алдыңғы итерациялардың бірінде жетекші болған болса, онда ағын осы циклда есептеуін жалғастыра алмайды, басқаша айтқанда босайды сол кезде келесі циклді есептеуді береміз. Бұл әдістің коды келесідей болады:

Int i, j;

Double PivotFactor, PivotValue = pMatrix[PivotRow][Iteration];

#pragma omp parallel for private (PivotFactor) schedule(dynamic, 1)

for (i = 0; i < mSize; i++) {

if (pSerialPivotIter[i] == -1) {

PivotFactor = pMatrix[i][Iteration] / PivotValue;

for (j = Iteration; j < mSize; j++) {

pMatrix[i][j] -= PivotFactor \* pMatrix[PivotRow][j];

}

pVector[i] -= PivotFactor \* pVector[PivotRow];

}

}

*Кері жүріс – айнымалыларды есептеу әдісі.* Бұл әдісте екі цикл бар, алайда сыртқы цикл параллелдеу тиімсіз. Себебі сыртқы цикл алдыңғы цикл есептеулері бітпей басталса онда қате шешім аламыз. Сондықтан біз тек ішкі циклді ғана параллелдейміз. Параллелдеу кезінде бізде tmp айнымалысы ағындар есептеуін аяқтағаннан кейін бір айнымалыға біріктірілуі керек, басқаша айтқанда жойылуы керек. Мұндай мүмкіндікті іске асыруға OpenMP – дің reduction (-:tmp) кілтсөзі көмектеседі. Reduction кілтсөзінің жақша ішінде бірінші параметрі қалай жойылу керектігін көрсетеді, біздің жағдайда азайтылу керек. Ал екінші параметрі ретінде қай айнымалы бойынша жойылу орындалу керектігі көрсетіледі. Бұл әдістің коды келесідей болады:

Int j, k, Row;

for (j = mSize - 1; j >= 0; j--) {

Row = pSerialPivotPos[j];

Double tmp = pVector [Row] / pMatrix [Row][j];

#pragma omp parallel for reduction (-:tmp)

for (k = j + 1; k < mSize; k++) {

tmp - = pMatrix[Row][k] \* pResult[k] / pMatrix[Row][j];

}

pResult[j] = tmp;

}

*Көрсетілген әдістерді ретімен орындау әдісі.* Бұл әдіс – есептеуді бастаушы әдіс. Мұнда әдістерді орындау реті көрсетіледі, сондықтан мұндай ағындарға бөлу тиімсіз болу табылады. Демек, бұл әдіс коды өзгертілмейді.

Енді бұл класста жаңа айнымалы типін құрастыруымыз керек. Типті TThreadPivotRow атаумен болсын. Ал бұл типтің ішінде екі айнымалы болады, біріншісі double типті MaxVal айнымалысы, ал екіншісі int типті PivotRow айнымалысы. Типті құрастыру коды келесідей болады:

typedef struct {

int PivotRow;

double MaxVal;

} TThreadPivotRow;

Осы кодтардың бәрін орындау үшін екі өлшемді \*\*pMatrix және бос мүшелер векторы \*pVector енгізіледі. Шешімдер векторы \*pResult – ке жазылады.

# **БІРІКТІРІЛГЕН ГРАДИЕНТТЕР ӘДІСІ**

## **2.1 Сипаттамасы және модификациялары**

Біріктірілген градиенттер (CG) әдісі САТЖ шешімін алуға арналған итерациялық әдіс болып табылады. Әдістің негізгі артықшылығы - ол қадамдардың шектеулі санымен квадраттық оңтайландыру есебін шешеді. Сондықтан алдымен квадраттық функцияны оңтайландырудың біріктірілген градиент әдісі сипатталады, итерациялық формулалар шығарылады және жинақтылық жылдамдығының бағалаулары беріледі. Осыдан кейін ерікті функционалдылықты оңтайландыру үшін біріктірілген градиенттер әдісі қалай жалпыланғаны көрсетіледі, әдістің әртүрлі нұсқалары қарастырылады және конвергенция талқыланады.[7-8]

«Біріктірілген градиент әдісі» термині мағынасыз тіркестердің үйреншікті болып қабылдануының және ешқандай таң қалдырмайтынының бір мысалы болып табылады. Мәселе мынада, практикалық қызығушылық тудырмайтын нақты жағдайды қоспағанда, градиенттер біріктірілмейді және конъюгаттық бағыттар градиенттермен ешқандай байланысы жоқ. Әдістің атауы шартсыз экстремумды табудың бұл әдісі мақсат функциясының градиенті және конъюгаттық бағыттар ұғымдарын біріктіретінін көрсетеді.

Төменде қолданылатын белгілер туралы бірнеше сөз.

Екі вектордың скаляр көбейтіндісі , деп жазылады және скалярлардың қосындысын көрсетеді: . Және  екенін ескеруіміз керек. Егер  пен  ортогональды болса, онда . Яғни,  және  сияқты 1x1 матрицасына түрлендіретін өрнектер скалярлар ретінде қарастырылады.

Бастапқыда біріктірілген градиент әдісі (3) түрдегі сызықтық алгебралық теңдеулер жүйесін шешу үшін жасалды.  матрицасы берілген, квадраттық, симметриялы, оң анықталған матрица болып табылсын. Бұл жүйені шешу сәйкес квадраттық форманың минимумын табуға тең.

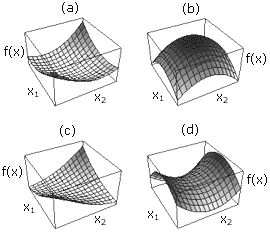
Квадраттық пішін жай ғана скаляр, келесі түрдегі кейбір  векторының квадраттық функциясы:

 (9)

 сызықты түрлендіру матрицасы мен  скаляр функциясы арасында мұндай байланыстың болуы сызықтық алгебраның кейбір формулаларын интуитивті сызбалармен суреттеуге мүмкіндік береді. Мысалы, кез келген нөлге тең емес  векторы үшін мыналар дұрыс болса,  матрицасы оң-анықталған деп аталады:

 (10)

3-суретте матрицалар үшін квадраттық пішіндердің сәйкесінше қалай көрінетіні көрсетілген.оң анықталған матрица (а), теріс анықталған матрица (b), оң анықталмаған матрица (c), анықталмаған матрица (d)

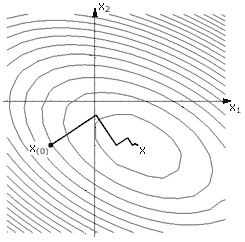


3 – сурет. Әртүрлі анықталған матрицалар үшін квадраттық пішіндері

Яғни, егер  матрицасы оң-анықталған болса, онда (3) теңдеулер жүйесін шешудің орнына оның квадраттық функциясының минимумын табуға болады. Сонымен қатар, біріктірілген градиент әдісі мұны  немесе одан аз қадамдармен орындайды, мұндағы  - белгісіз  векторының өлшемі. Оның минимум нүктесіне жақын орналасқан кез келген тегіс функция квадратпен жақсы жуықталатындықтан, квадраттық емес функцияларды да минимизациялау үшін дәл осындай әдісті қолдануға болады. Бұл жағдайда әдіс ақырлы болуды тоқтатады да итерациялы болады.

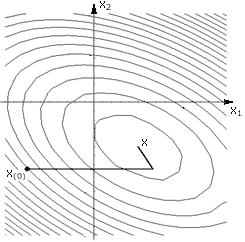
Біріктірілген градиент әдісін қарастыра отырып, функцияның экстремумын табудың қарапайым әдісінен – ең тік түсу әдісінен бастаған жөн. 4-суретте ең тік түсу әдісін қолдану арқылы минималды нүктеге дейінгі қозғалыс траекториясы көрсетілген. Бұл әдістің мәні:

* бастапқы  нүктесінде градиент есептеледі, ал қозғалыс мақсат функциясы төмендегенше антиградиент бағытында жүргізіледі;
* функцияның кемуі тоқтаған жерде градиент қайтадан есептеледі және төмендеу жаңа бағытта жалғасады;
* процесс ең төменгі нүктеге жеткенше қайталанады.



4 – сурет. Ең тік түсу әдісімен минималды нүктеге дейінгі қозғалыс траекториясы.

Бұл жағдайда қозғалыстың әрбір жаңа бағыты алдыңғысына ортогональды болады. Қозғалыстың жаңа бағытын таңдаудың ақылды жолы бар емес пе? Бар және ол біріктірілген бағыттар әдістері деп аталады. Ал біріктірілген градиенттер әдісі тек біріктірілген бағыттар әдістерінің тобына жатады. 5-суретте біріктірілген градиент әдісін қолдану кезінде минималды нүктеге дейінгі қозғалыс траекториясы көрсетілген.



5 – сурет. Біріктірілген градиенттер әдісін қолдану кезінде минималды нүктеге дейінгі қозғалыс траекториясы

Біріктірілген анықтамасы келесідей тұжырымдалған: екі  және  векторлары  - біріктірілген (немесе  матрицасына қатысты біріктірілген) немесе  - ортогональды деп аталады, егер  пен  скаляр көбейтіндісі нөлге тең болады, яғни:

 (11)

Конъюгацияны ортогоналдылық ұғымының жалпылауы деп санауға болады. Шынында да,  матрицасы сәйкестік матрицасы болғанда, (11) теңдігіне сәйкес  және  векторлары ортогональ болады. Ортогоналдылық пен конъюгация ұғымдарының арақатынасын басқа жолмен де көрсетуге болады: 5 – суретті ойша созамыз, сонда тең деңгейлі сызықтар эллипстен шеңберге айналады, ал конъюгацияланған бағыттар жай ортогональ болады.

Конъюгаттық бағыттарды қалай есептеу керектігін анықтау қалады. Мүмкін болатын әдістердің бірі - сызықтық алгебра әдістерін, атап айтқанда, Грам-Шмидт ортогонализация процесін қолдану. Бірақ бұл үшін сіз  матрицасын білуіңіз керек, сондықтан көптеген тапсырмалар үшін (мысалы, көп қабатты нейрондық желілерді оқыту) бұл әдіс жарамайды. Бақытымызға орай, конъюгаттық бағытты есептеудің басқа, итеративті жолдары бар, олардың ең танымалы Флетчер-Ривз формуласы:

 (12)

мұндағы,

 (13)

(12) формулада жаңа конъюгаттық бағыт бұрылыс нүктесіндегі антиградиент пен қозғалыстың алдыңғы бағытын (13) формуламен есептелген коэффициентке көбейту арқылы алынғанын білдіреді. (12) формула бойынша есептелген бағыттар конъюгаттық болып шығады, егер функция минимизацияланған 2 түрінде берілген. Яғни, квадраттық функциялар үшін конъюгаттық градиент әдісі минимумды n қадаммен табады (n - іздеу кеңістігінің өлшемі). Жалпы түрдегі функциялар үшін алгоритм ақырлы болуды тоқтатады және итеративті болады. Сонымен бірге, Флетчер мен Ривз алгоритмдік процедураны әрбір n+1 қадам сайын қайта бастауды ұсынады.

Конъюгаттық бағытты анықтаудың басқа формуласын, Полак-Рибер формуласын (Polak-Ribiere) беруге болады:

 (14)

Флетчер-Ривс әдісі, егер бастапқы нүкте қажетті минимумға жеткілікті жақын болса, жинақталады, ал Полак-Райбер әдісі сирек жағдайларда шексіз цикл жасай алады. Дегенмен, соңғысы көбінесе бұрынғы әдіске қарағанда тезірек біріктіріледі. Бақытымызға орай, Полак-Райбер әдісінің конвергенциясына  таңдау арқылы кепілдік беруге болады. Бұл  шарты бойынша алгоритмді қайта іске қосуға тең. Алгоритмдік процедураны қайта бастау іздеудің соңғы бағытын ұмытып, алгоритмді ең жылдам түсу бағытында қайтадан бастау үшін қажет.

Төменде жалпы (квадраттық емес) функцияларды азайтуға арналған конъюгаттық градиент алгоритмі берілген.

1. Антиградиент ерікті нүктеде  есептеледі:



1. Функция төмендеген кезде есептелетін бағытта кему, басқаша айтқанда, кішірейтетін  мәнін іздейміз:



1. Алдыңғы операцияда табылған нүктеге өтеміз:



1. Осы кездегі антиградиентті есептеу:



1. (13) немесе (14) формула бойынша есептеулер. Алгоритмді қайта іске қосу, яғни іздеудің соңғы бағытын ұмытып, алгоритмді ең жылдам түсу бағытында қайта бастау үшін Флетчер-Ривз формуласы үшін әрбір n+1 қадам сайын 0 тағайындалады. , Полак-Райбер формуласы үшін -



1. Жаңа конъюгаттық бағытты есептейміз:



1. Екінші пунктқа көшеміз.

Жоғарыда келтірілген алгоритмнен 2-қадамда функцияны бір өлшемді минимизациялау жүзеге асырылатыны шығады. Бұл үшін, атап айтқанда, Фибоначчи әдісін, алтын қима әдісін немесе екіге бөлу әдісін қолдануға болады. Тезірек конвергенция Ньютон-Рафсон әдісімен қамтамасыз етілген, бірақ ол үшін Гессиан матрицасын есептей білу қажет. Соңғы жағдайда оңтайландыру жүргізілетін айнымалы келесі формула бойынша әрбір итерация қадамында есептеледі:



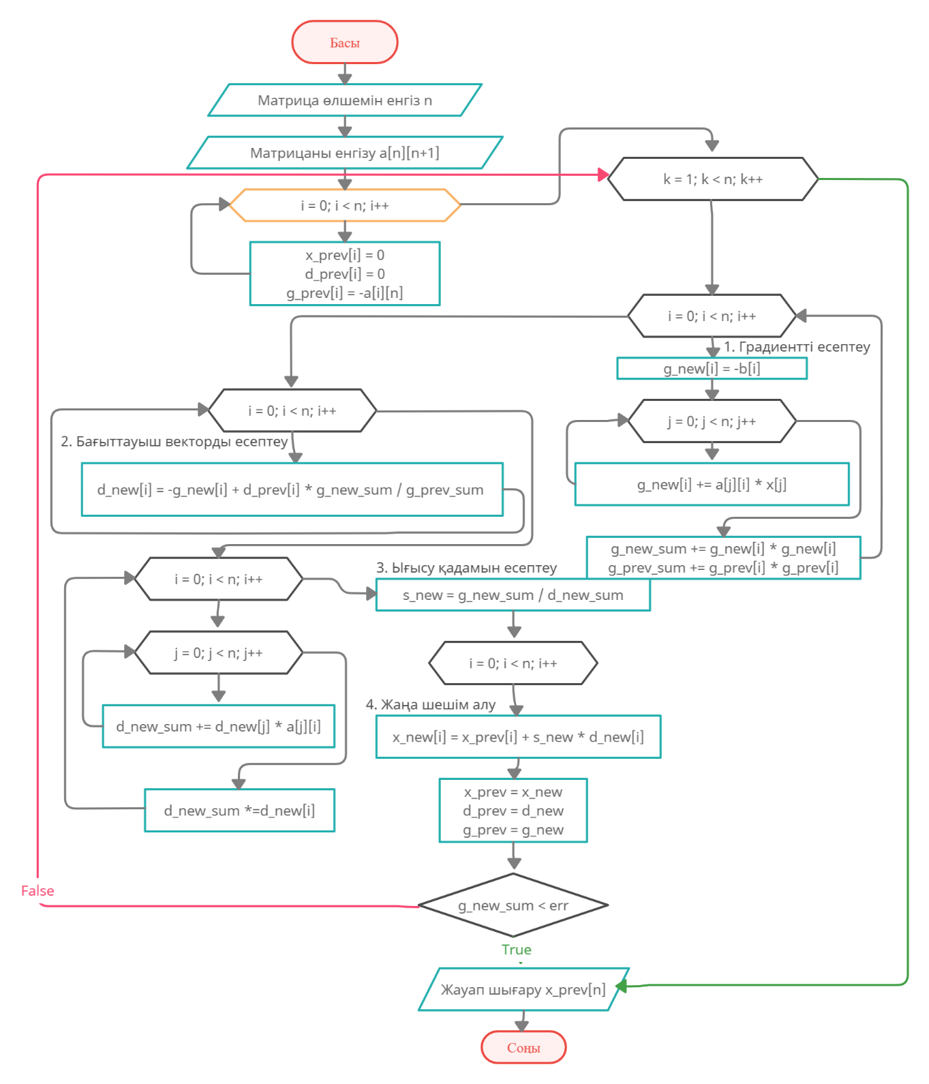
Мұндағы, Гессе матрицасы



Бұл кейбір авторларға конъюгаттық градиент әдісін екінші ретті әдіс ретінде жіктеуге негіз береді, дегенмен әдістің мәні екінші туындыларды есептеуді мүлдем қажет етпейді.

Нейрондық желілерді оқытуда конъюгаттық бағыттар әдісін қолдану туралы бірнеше сөз. Бұл жағдайда дәуірлер бойынша оқыту пайдаланылады, яғни мақсаттық функцияны есептеу кезінде оқу жиынының барлық үлгілері ұсынылады және қателік функциясының орташа квадраты (немесе оның кейбір модификациясы) есептеледі. Градиентті есептеу кезінде де дәл солай болады, яғни бүкіл жаттығу жиыны бойынша жалпы градиент пайдаланылады.

Бұл әдіс алгоритмі 6 – суретте көрсетілгендей болады.



6 – сурет. CG әдісінің блок – схемасы.

## **2.2 CG әдісінің алгоритмін бағдарламалау**

Енді 6 – суретте көрсетілген блок – схема бойынша С++ тілінде бағдарлама жазамыз. Бағдарламамызды CGSerial класс құрудан бастаймыз.

Класста маңызды бірнеше айнымалылар бар, олар:  – градиент векторы,  – бағыт векторы,  –  матрицасы мен  векторының көбейтіндісінің векторы,  – бағыт қадам ұзындығы,  – конюгатталған бағыт,  –  скалярлы көбейтіндісі,  –  скалярлы көбейтіндісі,  – градиенттің өлшемінің дәлдігі.

Бастапқы мәндерді цикл арқылы меншіктеп шығайық:

for (i = 0; i < Size; i++) {

pResult[i] = pVector[i];

g[i] = pVector[i];

for (j = 0; j < Size; j++) {

g[i] -= pMatrix[i][j] \* pVector[j];

}

d[i] = g[i];

sum\_new\_g += g[i] \* g[i];

}

Бастапқы мәндерді меншіктегеннен кейін біріктірілген градиенттер әдісі бойынша жаңа градиентті, жаңа бағытты, жаңа шешімді, жаңа қадамды цикл көмегімен есептейміз. Біздер цикл неше рет айналатындығын білмейтін болғандықтан C++ тіліндегі итерация саны анықталмаған do { операциялар } while (шарт) циклді қолданамыз.

Циклде *бірінші пунктте*, алгоритм бойынша қадам ұзындығы есептеп алу. Оның математикалық формуласы келесідей:

 (15)

Бөлшектің алымындағы скалярлық көбейтінді алдыңғы градиентті есептеген кезде есептелінді, яғни біз онда  айнымалсын қоя саламыз. Ал бөліміндегі екінші көбейтінді  болады. Бізге оны көбейту аса қиындыққа түспейді. Бірінші операцияның коды келесідей болады:

for (i = 0; i < Size; i++) {

A\_prev\_d[i] = 0;

for (j = 0; j < Size; j++) {

A\_prev\_d[i] += pMatrix[i][j] \* d[j];

}

ip += A\_prev\_d[i] \* d[i];

}

step = sum\_prev\_g / ip;

*Екінші пунктте* бағыт және жаңа қадам (15) көмегімен жаңа шешімді есептеп аламыз. Оның математикалық формуласы келесідей:

 (16)

(16) формуладағы бүкіл айнымалылар белгілі болғандықтан кодын жазу аса қиындық тудырмайды:

for (i = 0; i < Size; i++) {

pResult[i] += step \* d[i];

}

*Үшінші пунктте* жаңа қадам (15) және  векторының көмегімен жаңа градиентті есептеп аламыз. Оның математикалық формуласы келесідей:

 (17)

(17) формуладағы бүкіл айнымалылар белгілі болғандықтан кодын жазу аса қиындық тудырмайды. Сондай – ақ есептеу жылдамдығын арттыру үшін екінші және үшінші пункт кодтарын біріктіріп бір циклге жазайық:

for (i = 0; i < Size; i++) {

pResult[i] += step \* d[i];

g[i] -= step \* A\_prev\_d[i];

sum\_new\_g += g[i] \* g[i];

}

Келесі пунктке өтер алдында жаңа градиенттің дәлдігін тексеруіміз қажет. Егер дәлдігіміз қажетті дәлдіктен асса, итерацияны тоқтатсақ болады, егер әлі дәлдігі әлсіз болса, онда есептеуімізді жалғастыруға тура келеді.

if (sqrt(sum\_new\_g) <= Accuracy) break;

*Төртінші пунктте* біз біріктірілгендік параметрін есептеу. Басқаша айтқанда екі ескі және жаңа градиенттерді қатынасы. Оны математикалық формуласы келесідей:

 (18)

(18) формуладағы бөлшектің алымы мен бөліміндегі скаляр көбейтінді алдыңғы есептеулерімізде алынды, олар сәйкесінше  және  болып табылады. Оның коды келесідей болады:

beta = fmax(sum\_new\_g / sum\_prev\_g, 0);

*Бесінші пунктте* шешімді іздеудің жаңа бағыты есептелінеді. Формуласы:

 (19)

(19) формула есептеу бір циклмен орындалады. Коды келесідей:

for (i = 0; i < Size; i++) {

d[i] = g[i] + beta \* d[i];

}

Осы бесінші пунктті есептегеннен кейін, келесі итерацияға өтеміз. Жаңа итерацияда алдыңғы итерациядан тек қана  мәнін ғана жаңарту керек:

sum\_prev\_g = sum\_new\_g;

double ip = 0;

Итерациялар градиент өлшемінің дәлдігі  өлшемінен үлкен болғанға дейін қайталана береді. Ал жауап болса pResult векторы ретінде есептелінеді.

## **2.3 CG алгоритмін параллелдеу**

Алгоритмді параллелдеуге OpenMP кітапханасы қолданылады. Сондай-ақ, сынақтарды бірнеше ағындар санымен қойылатынын ескеріп, параллелдеу барысында ағын санын оңай басқаратын кілтсөздер қосуымыз керек.

Бағдарламамызды CGParallel класс құрудан бастаймыз. CGSerial классындағы кодты осы классқа көшіріп қоямыз. Содан кейін OpenMP кітапханасының кілтсөздерін пайдаланамыз.

Алдымен, бастапқы мәндерді меншіктеу циклын параллельдесек болады. Ол үшін #pragma omp parallel for reduction(+:sum\_new\_g) private (i) кілтсөзін қолданамыз. Мұндағы reduction кілтсөзінің жақша ішінде бірінші параметрі қалай жойылу керектігін көрсетеді, біздің жағдайда қосылу керек. Ал екінші параметрі ретінде қай айнымалы бойынша жойылу орындалу керектігі көрсетіледі. Private кілтсөзде, әр ағында i айнымалысы әртүрлі мәнге ие болатындықтан, сол айнымалыны жақшаға ішіне жаздық. Соңында бастапқы мәнді меншіктеу коды келесідей өзгереді:

#pragma omp parallel for reduction(+:sum\_new\_g) private (i)

for (i = 0; i < Size; i++) {

pResult[i] = 0;

g[i] = pVector[i];

d[i] = -pVector[i];

sum\_new\_g += g[i] \* g[i];

}

Бастапқы мәндерді меншіктегеннен кейін итерациялар басталады. Итерацияда бірінші пункт қадам ұзындығы  есептеу. Бұл есептеуде матрицаны векторға көбейту операциясы бар. Әр ағынға векторды және бір жолды көбейтуді орындасақ, параллельдеу ұтымды болады. private (i,j) осы кілтсөз көмегімен осы мақсатымызға жетеміз. Келесі кілтсөз reduction(+:ip), бұл кілтсөздің қандай қызмет атқаратынын алдыңғы пунктте айтып өттік. Сонда қадам ұзындығын есептеу коды келесідей болады:

#pragma omp parallel for reduction(+:ip) private (i,j)

for (i = 0; i < Size; i++) {

A\_prev\_d[i] = 0;

for (j = 0; j < Size; j++){

A\_prev\_d[i] += pMatrix[i][j] \* d[j];

}

ip += A\_prev\_d[i] \* d[i];

}

step = sum\_prev\_g / ip;

Екінші және үшінші пунктте де осындай есептеулер бар. reduction(+:sum\_new\_g) private (i) осы кілтсөзді қосып параллельдесек тиімді болады.

#pragma omp parallel for reduction(+:sum\_new\_g) private (i)

for (i = 0; i < Size; i++) {

pResult[i] = pResult[i] - step \* d[i];

g[i] = g[i] + step \* A\_prev\_d[i];

sum\_new\_g += g[i] \* g[i];

}

Төртінші пункт ешқандай параллельдеуді талап етпейді.

Бесінші пункт жаңа бағытты есептеу пунктісінде цикл бар. Біз соны private (i) кілтсөзімен параллельдейік.

#pragma omp parallel for private (i)

for (i = 0; i < Size; i++) {

d[i] = -g[i] + beta \* d[i];

}

Дайын болған кодта енді ешқандай параллельдеу қажет емес. Осымен біріктірілген градиенттер әдісінің сызықты алгоритмін параллельдеу аяқталды.

# **ЕСЕПТЕУ ЭКСПЕРИМЕНТТЕРІ ЖӘНЕ ТАЛДАУ**

## **Эксперименттің жүргізілу шарттары**

Экспериментте матрицаларды және бос мүше векторларын кездейсоқ сандармен толтырамыз. Есептеудің дұрыстығын автоматты түрде тексеруіміз керек. Шешімнің дұрыстығы кезінде, шешімнің қателігі  кіші болуы қадағаланады.

Кіріс ақпараттары үшін бірнеше параметрлерді анықтап аламыз. Есептеу эксперименттерінің параметрлері келесідей:

* Матрица өлшемі;
* Матрица типтері;
* Ағындар саны.

*Матрица өлшемі*. Есептеу эксперименттерді үлкен өлшемді матрицалар үшін жүргізіледі. Себебі кіші өлшемдегі матрица үшін параллелдеу технологиясы онша көп өсім бермейді, керісінше жадыдан ақпаратты оқып жазу уақытының кесірінен параллелденген алгоритм шешім алуға көп уақыт жұмсауы мүмкін. Эксперимент кезінде матрица өлшемдері 100х100, 500х500, 1000х1000, 1500х1500, 2000х2000, 2500х2500, 3000х3000 болады.

*Матрица типтері*. Практикада, өндірісте есептеулерде қолданылатын матрица түрлері көп. Оның ішінде танымал үш типті матрицаны қолданамыз.

Біріншісі диагоналды матрица деп аталады. Берілген матрицаның диагоналынан басқа элементтері нөлге тең болады. Мысалы,



Екіншісі үш диагоналды симметриялы матрица деп аталады. Берілген матрицаның үш диагоналынан басқа элементтері нөлге тең болады. Мысалы,



Үшіншісі симметриялы толық матрица деп аталады. Берілген матрицаның ешқандай элементі нөлге тең емес болады. Мысалы,



Эксперименттердің бәрі жалғыз ноутбукта орындалады. Ноутбуктың сипаттамасы келесідей:

* Microsoft Windows 10 Professional;
* Компилятор Microsoft Visual C++;
* HexaCore Intel Core i7-9750H, 3000 MHz, 12 Multi CPU (ағын);
* 15,9 GB RAM.

Біздің ноутбук 12 ағынды қолдайтын болғандықтан, эксперимент кезінде максимум ағын саны 12 болады. Сондай-ақ, аралық ағындар санын да экспериментте қолданамыз. Нақтырақ айтсақ, экспериментте берілген жүйені алдымен сызықты алгоритммен, содан кейін 2 ағынмен параллелді шешеміз, содан кейін 4, 6, 8, 10, ал ең соңында 12 ағынмен шешеміз. Есептеуге кеткен уақыттарды талдау оңай болу үшін барлық кеткен уақыттарды \*.csv типте сақтаймыз.

## **Эксперименттер қоюға қажетті қосымша класстар жайлы**

Эксперимент ыңғайлы, әрі қатесіз өткізу үшін бағдарламамызға бірнеше класстар жасау керек. Олар:

* Кіріс ақпараттарын кездейсоқ генерациялау классы;
* Шешімді тексеру, берілген матрицаны тексеру және т.б., көмекші матрица классы;
* \*.csv файлмен жұмыс істеу классы;
* Және соңғысы, басқарушы класс.

*Ақпараттарды генерациялау классы.* Класс атын dataGen деп атайық. Класс ішінде үш типтегі матрицаны, бос мүше векторын кездейсоқ сандармен толтыратын үш әдіс болсын. Енді осы үш әдіске ретімен тоқталсақ.

Бірінші экспериментте, диагональ типтегі матрица болады. Оның қандай болатын алдыңғы бөлімде айтып өттік. Алдымен кездейсоқ сандарды қайтаратын srand(unsigned(clock())) функциясын шақырамыз. Кездейсоқ сандарды алу функциясын инициализация жасағаннан кейін, екі цикл көмегімен \*\*pMatrix, \*pVector кездейсоқ мәндерді rand() көмегімен толтырамыз. Матрица толтыру кезінде тек қана диагональ элементтеріне (i == j) ғана мән береміз, басқа кезде нөлге меншіктейміз. Бұл әдіс коды келесідей:

srand(unsigned(clock()));

for (int i = 0; i < Size; i++) {

pVector[i] = (double)rand() / double(Size);

for (int j = 0; j < Size; j++) {

if (i == j) {

pMatrix[i][j] = (double) rand() / double(Size);

}

else {

pMatrix[i][j] = 0;

}

}

}

Екінші экспериментте, үш диагональді типтегі матрица болады. Оның қандай болатын алдыңғы бөлімде айтып өттік. Алдымен кездейсоқ сандарды қайтаратын srand(unsigned(clock())) функциясын шақырамыз. Кездейсоқ сандарды алу функциясын инициализация жасағаннан кейін, екі цикл көмегімен \*\*pMatrix, \*pVector кездейсоқ мәндерді rand() көмегімен толтырамыз. Матрица толтыру кезінде тек қана диагональден 1 қадам ары жатқан элементтерге (fabs(j - i) < 2) ғана мән береміз, басқа кезде нөлге меншіктейміз. Бұл әдіс коды келесідей:

srand(unsigned(clock()));

for (int i = 0; i < Size; i++) {

pVector[i] = (double)rand() / double(Size);

for (int j = 0; j < Size; j++) {

if (fabs(j - i) < 2){

pMatrix[i][j] = pMatrix[j][i] = (double) rand() / double(Size);

}

else {

pMatrix[i][j] = 0;

}

}

}

Үшінші экспериментте, симметриялы матрица болады. Оның қандай болатын алдыңғы бөлімде айтып өттік. Алдымен кездейсоқ сандарды қайтаратын srand(unsigned(clock())) функциясын шақырамыз. Кездейсоқ сандарды алу функциясын инициализация жасағаннан кейін, екі цикл көмегімен \*\*pMatrix, \*pVector кездейсоқ мәндерді rand() көмегімен толтырамыз. Мұнда алдыңғы эксперементтердегідей шарт болмайды. Бұл әдіс коды келесідей:

srand(unsigned(clock()));

for (int i = 0; i < Size; i++) {

pVector[i] = (double)rand() / double(Size);

for (int j = 0; j < Size; j++) {

pMatrix[i][j] = pMatrix[j][i] = (double) rand() / double(Size);

}

}

*Көмекші матрица классы.* Класс атын matrixHelpers деп атайық. Класс ішінде ең қолданбалы шешімді тексеру, матрица мен векторларды бастапқы қалыпқа әкелу, және матрицаны не векторды экранға шығару әдістері бар. Енді осы әдістерге ретімен тоқталсақ.

Шешімді тексеру әдісінде \*\*OriginalA матрицасын \*pResult векторын көбейтеміз. Көбейтуді жылдамдату үшін OpenMP көмегімен параллелдейміз. Параллелдеу кезінде reduction(+:tmp) кілтсөзін қолданамыз, себебі бізде tmp мәні барлық ағын бойынша қосылып шығу керек. Көбейтіндіні \*pRightPartVector векторына меншіктейміз. Көбейтіп біткеннен кейін \*pRightPartVector векторынан \*pVector бос мүше векторынан азайтамыз, ол айырма абсолютті мәні бойынша 0,01 дәлдігінен кіші болмау керек. Басқаша айтқанда fabs(pRightPartVector[i] - pVector[i]) > Accuracy осы шарт орындалмағаны жөн. Егер шарт орындалса equal мәнін өсіреміз. Осы азайту операциясын да OpenMP кітапханасының reduction(+:equal) кілтсөзінің көмегімен параллелдеп есептейміз. Осы азайту операциясы аяқталғаннан кейін equal мәні 0 – ден үлкен болса, шешім біз қалаған дәлдікпен шешілмеді дегенді білдіреді. Енді осы әдістің кодын келтірейік:

double\* pRightPartVector;

int equal = 0, i, j;

double Accuracy = 0.01f;

pRightPartVector = new double[Size];

for (i = 0; i < Size; i++) {

double tmp = 0;

#pragma omp parallel for reduction(+:tmp)

for (j = 0; j < Size; j++) {

tmp += pMatrix[i][j] \* pResult[j];

}

pRightPartVector[i] = tmp;

}

#pragma omp parallel for reduction(+:equal)

for (i = 0; i < Size; i++) {

if (fabs(pRightPartVector[i] - pVector[i]) > Accuracy) {

equal++;

}

}

if (equal > 0) {

printf(" Wrong.");

}

else {

//printf(" Сorrect.");

}

Матрицаны және векторларды бастапқы қалыпқа алып келу әдісі. Біздің бір экспериментте бір матрицаны және бос мүше векторын бірнеше рет шешетін болғандықтан, матрица мен векторларды әр есептеу біте бастапқы мәндерге алып келіп отыру керек. Бұл операцияны екі циклда орындаймыз. Параллелдеу операцияны жылдамдатуға мүмкіндік береді. Әдіс коды келесідей:

for (int i = 0; i < Size; i++) {

B[i] = OriginalB[i];

X[i] = 0;

#pragma omp parallel for

for (int j = 0; j < Size; j++)

A[i][j] = OriginalA[i][j];

}

Матрицаны немесе векторды экранға шығару екі немесе бір цикл арқылы, printf("%.4f ", matrix[i]) көмекші функциясы көмегімен іске асырамыз. Бұл әдістерде параллелдеуді қолданбаймыз. Егер қолданар болсақ, ағындар бір – бірімен экранға шығару бойынша жарысуы мүмкін. Ал бұл бізге қате тұжырым жасатуы мүмкін.

*Файлмен жұмыс классы.* Баған аттарын және алгоритмдердің есептеуге кеткен уақыттарын жазатын екі әдіс құрастырамыз. Файлдармен fstream кітапханасының көмегімен жұмыс істейміз.

Баған тақырыбын жазу әдісінде көрсетілген бағанда есепті қандай әдіспен шешілгені жазылады. Тақырыптар динамикалық түрде болу үшін экспериментте жүргізілетін ағындар саны массивін осы әдіске жібереміз. Содан кейін осы массивті цикл арқылы өтіп, файлға жазып шығамыз. Ал файл аты result қосылған матрица типінің нөмірі болады. Бұл әдіс коды келесідей болады:

ofstream myfile;

string filename = "result";

filename += to\_string(experiment\_number);

filename +=".csv";

myfile.open(filename);

myfile << "Размер;Гаусс линейно;";

for (int i = 0; i < size; i++) {

myfile << "Гаусс параллель(" << threads\_array[i] << ");";

}

myfile << "CG линейно;";

for (int i = 0; i < size; i++) {

myfile << "CG параллель(" << threads\_array[i] << ");";

}

myfile.close();

Есептеуге кету уақытын жазу әдісінде тақырыптар бойынша керек бағанға жазады. Файлдағы ақпаратты жоймай астына жалғастырып жаза беру үшін файлды ашқан кезде ios::app атрибутын енгіземіз. Бұл әдіс коды келесідей болады:

ofstream myfile;

string filename = "result";

filename += to\_string(experiment\_number);

filename += ".csv";

myfile.open(filename,ios::app);

myfile << "\n" <<Size << "x" << Size << ";" << times;

myfile.close();

Және осы csv файлды Microsoft Excel бағдарламасында ашқан кезде уақытты айнымалысы double типте болу үшін, уақыттағы нүкте символын үтір символына ауыстыру керек. Оның коды келесідей:

size\_t start\_pos = 0;

while ((start\_pos = str.find(from, start\_pos)) != std::string::npos) {

str.replace(start\_pos, from.length(), to);

start\_pos += to.length();

}

*Басқарушы класс.* Экспериментті бастар алдында матрица типтер саны, ағындар санының массиві көрсетіледі. Файлдағы кестенің тақырыбын жазып шығамыз. Содан кейін матрица көлемінің өлшемін цикл арқылы көрсетіп отырамыз. Экспериментте массив өлшемі 100, 500, 1000, 1500, … болады.

Енді циклға тоқталсақ. Цикл ең алғашқы болып коэффициенттер матрицасын және бос мүшелер векторын dataGen классының көмегімен кездейсоқ сандармен толтырамыз.

dataGen::randomDataInitialization(OriginalA, originalB, mSize);

Әдіспен есептеуді бастамас бұрын pMatrix, pVector, pResult айнымалыларына бастапқы мәндерді меншіктейміз. Оны matrixHelpers классының setDefault әдісімен орындаймыз.

matrixHelpers::setDefault(OriginalA, originalB, mSize, pMatrix, pVector, pResult);

Айнымалыларды бастапқы қалыпқа келтіргеннен кейін, есептеудің басталу уақытын startTime айнымалысына жазамыз.

startTime = omp\_get\_wtime();

Таймерді жібергеннен кейін алгоритммен есепті шешуге кірісеміз. Алдымен есепті Гаусс әдісінің сызықты алгоритмімен, содан кейін параллель алгоритмімен шешеміз.

gaussSerialSolver = new GaussSerial(mSize);

gaussSerialSolver->resultCalculation(pMatrix, pVector, pResult);

Содан кейін біріктірілген градиенттер әдісінің сызықты алгоритмімен содан кейін ең соңында біріктірілген градиент әдісінің параллель алгоритмімен шешеміз.

CGSerialSolver = new CGSerial();

CGSerialSolver->resultCalculation(pMatrix, pVector, pResult, mSize);

Әр алгоритм есептеуін аяқтай таймерді тоқтатып отырамыз.

finishTime = omp\_get\_wtime();

Таймерді тоқтатқаннан кейін шешімді matrixHelpers классының көмегімен тексереміз.

matrixHelpers::testSolvingResult(OriginalA, originalB, pResult, mSize);

Осыдан кейін есептеуге кеткен уақытты файлға жазамыз.

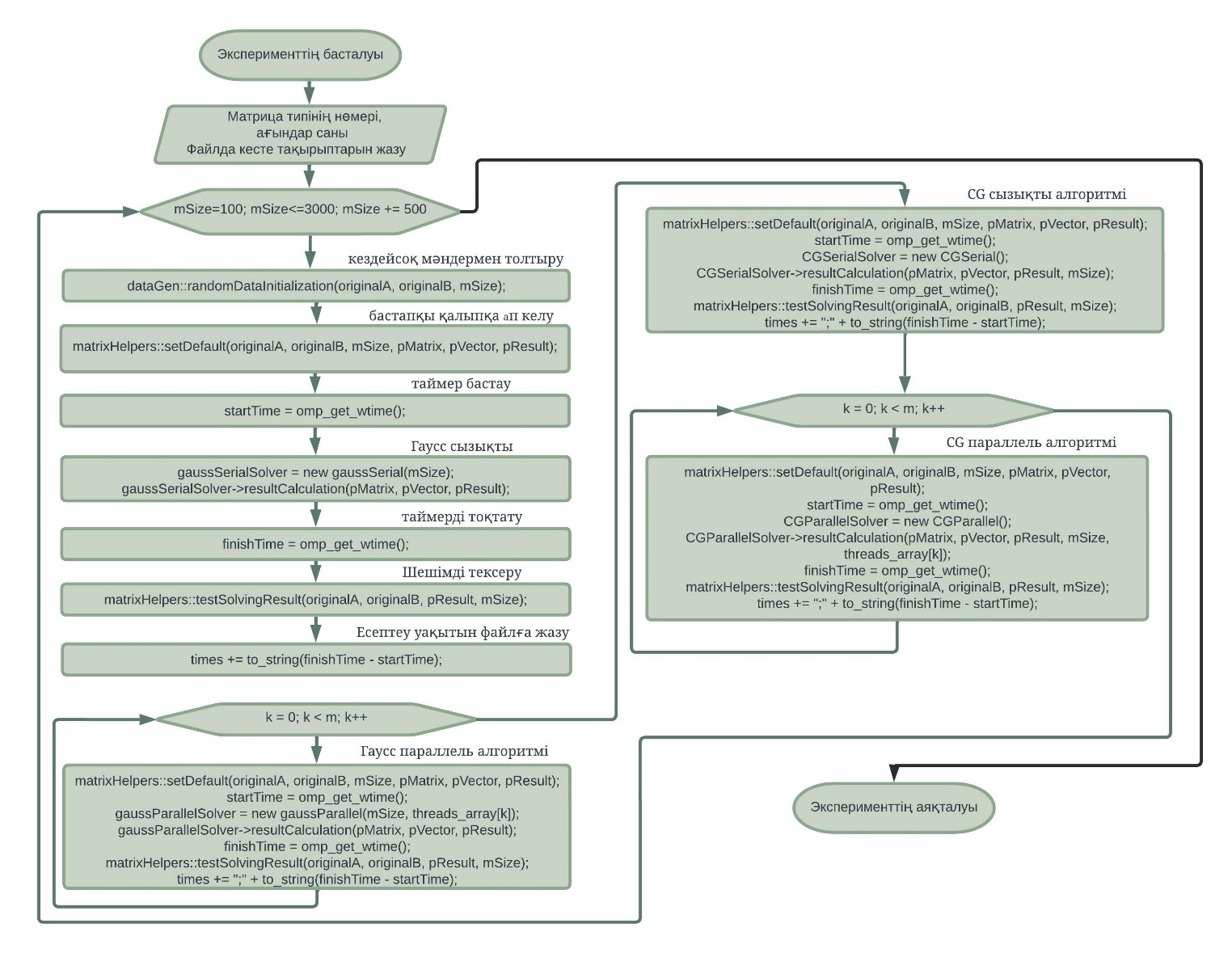
csvExport::addTimes(mSize, experiment\_number, times);

Эксперименттің өту процессі 7 – суретте көрсетілген блок – схема бойынша жүреді.

Экспериментті жүргізу алдында experiment\_number айнымалысын 1 мәнін меншіктейміз. Ол дегеніміз коэффициенттер матрицасы диагональ матрица болады дегенді білдіреді. Яғни dataGen классының, DiagonalDataInitialization әдісімен матрицаны толтырамыз. Бұл типті матрицаның шешуге кеткен уақыттары result1.csv файлында сақталады.

Келесі экспериментте experiment\_number айнымалысын 2 мәнін меншіктейміз. Ол дегеніміз коэффициенттер матрицасы үш диагональді матрица болады дегенді білдіреді. Яғни dataGen классының, ThreeDiagonalDataInitialization әдісімен матрицаны толтырамыз. Бұл типті матрицаның шешуге кеткен уақыттары result2.csv файлында сақталады.

Және соңғы үшінші экспериментте experiment\_number айнымалысын 3 мәнін меншіктейміз. Ол дегеніміз коэффициенттер матрицасы қарапайым симметриялы матрица болады дегенді білдіреді. Яғни dataGen классының, randomDataInitialization әдісімен матрицаны толтырамыз. Бұл типті матрицаның шешуге кеткен уақыттары result3.csv файлында сақталады.



7 – сурет. Эксперимент алгоритмінің блок-схемасы

## **Эксперимент нәтижелерін алу және талдау**

Есептеу эксперименті алдыңғы бөлімде көрсетілген алгоритмдер мен блок схемалар бойынша C++ тілінде код жазудан басталады. Кодты Visual Studio 2022 бағдарламалау ортасында жазамыз. OpenMP кітапханасының мүмкіндіктерін бағдарламада қолдана алу үшін келесі жолмен жүріп, операцияны істейміз:

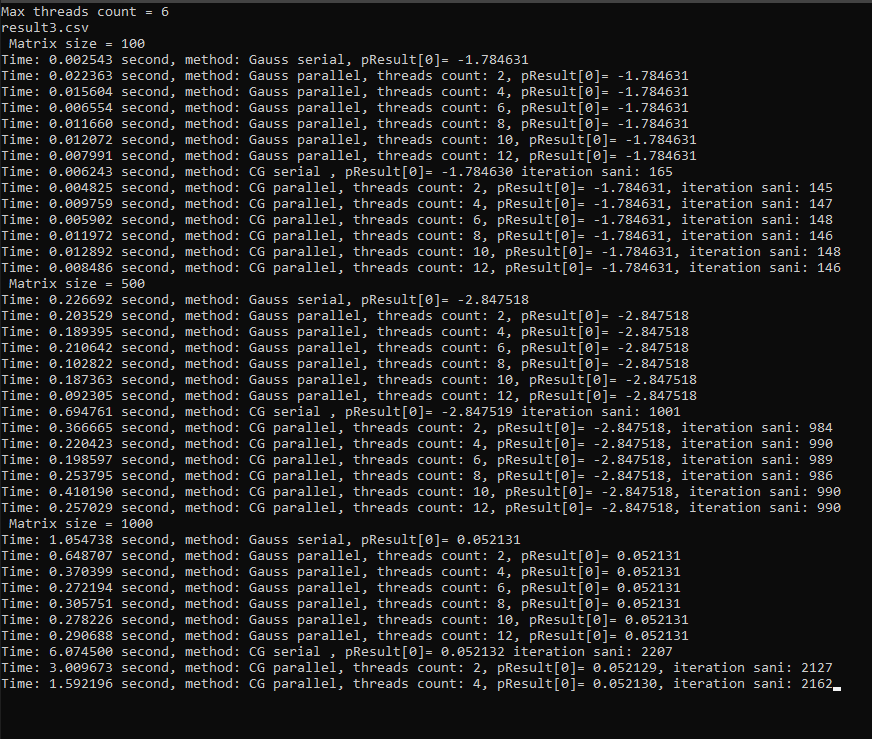
Проект –> Свойства: satj –> Свойства конфигурации –> С/С++ –> Язык –> Поддержка OpenMP –> Да(/openmp)

Одан басқа есептеу жылдамдығын арттырып, кодты Visual Studio – ның алдыңғы версияларында да орындалуы үшін келесі атрибуттармен жинаймыз:

/JMC /permissive- /MP /ifcOutput "x64\Debug\" /GS /W3 /Zc:wchar\_t /ZI /Gm- /Od /sdl /Fd"x64\Debug\vc143.pdb" /Zc:inline /fp:fast /D "\_DEBUG" /D "\_CONSOLE" /D "\_UNICODE" /D "UNICODE" /errorReport:prompt /WX- /Zc:forScope /RTC1 /std:c17 /Gd /MDd /openmp /std:c++20 /FC /Fa"x64\Debug\" /EHsc /nologo /Fo"x64\Debug\" /Ot /Fp"x64\Debug\satj.pch" /diagnostics:column

Мұндағы кейбір атрибуттарды Visual Studio бағдарламасы өзі қосады.

Орындау басталғанда 8 – сурет экранға бірінші максималды ағындар саны шығады. Екінші жолда есептеуге кететін уақыттар жазылатын файлдың аты шығады. Содан кейін әр жаңа жолда матрица өлшемі, содан кейін сол өлшемдегі жүйені көрсетілген әдістің аты, алгоритм типі (сызықты, параллель), егер параллель болса ағындар саны, содан кейін есептеуге кеткен уақыт, және соңғы элемент pResult векторының бірінші (индекс бойынша 0) элементінің мәні шығады. Егер әдістің алгоритмі қате шешім берсе экранда «Wrong» сөзі шығады.



8 – сурет. Бағдарлама орындалуы

Бағдарлама орындалуы аяқталғаннан кейін сәйкес файлды ашып есептеу уақыттарын талдаймыз.

Бірінші экспериментте біз коэффициенттер матрицасы диагональды болады, жүйені шешеміз. Есептеу уақыттарының кестесі келесідей болды (1 – кесте)

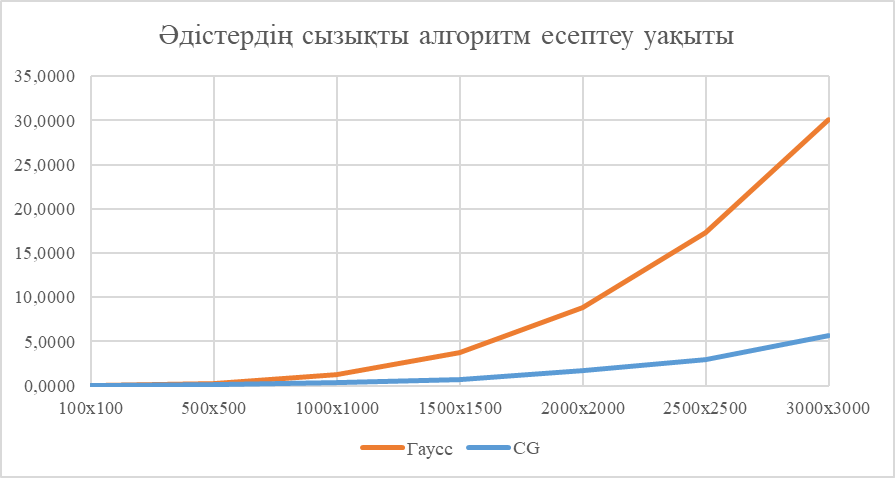
1 – кесте.

Диагональдық матрицалы жүйені шешуге кеткен уақыттар

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Өлшемі** | **Гаусс** | | | | | | | **CG** | | | | | | |
| **сызықты** | **2 ағын** | **4 ағын** | **6 ағын** | **8 ағын** | **10 ағын** | **12 ағын** | **сызықты** | **2 ағын** | **4 ағын** | **6 ағын** | **8 ағын** | **10 ағын** | **12 ағын** |
| 100x100 | 0,001475 | 0,005078 | 0,003821 | 0,007045 | 0,007895 | 0,008382 | 0,009772 | 0,00239 | 0,003334 | 0,001677 | 0,002314 | 0,003328 | 0,004815 | 0,005476 |
| 500x500 | 0,210065 | 0,120529 | 0,141728 | 0,087563 | 0,127587 | 0,101513 | 0,093654 | 0,087086 | 0,042553 | 0,020737 | 0,020865 | 0,019022 | 0,027014 | 0,025828 |
| 1000x1000 | 1,205933 | 0,634424 | 0,415214 | 0,373776 | 0,332691 | 0,308849 | 0,333401 | 0,34353 | 0,176102 | 0,136266 | 0,092999 | 0,085127 | 0,080062 | 0,084754 |
| 1500x1500 | 3,718928 | 2,073793 | 1,247801 | 1,050612 | 1,012978 | 0,944539 | 0,951437 | 0,64616 | 0,335772 | 0,220024 | 0,180578 | 0,165404 | 0,134419 | 0,182435 |
| 2000x2000 | 8,85843 | 4,814027 | 2,912545 | 2,387678 | 2,363687 | 2,236935 | 2,250397 | 1,745131 | 0,983004 | 0,577645 | 0,50262 | 0,489846 | 0,450299 | 0,52868 |
| 2500x2500 | 17,326756 | 9,272688 | 5,695793 | 4,713355 | 4,729326 | 4,341794 | 4,405794 | 2,965649 | 1,643066 | 0,992603 | 0,851874 | 0,805909 | 0,69997 | 0,873732 |
| 3000x3000 | 30,104294 | 16,1902 | 9,795955 | 8,159727 | 8,040493 | 7,762699 | 7,686993 | 5,615587 | 3,092781 | 2,022119 | 1,720142 | 1,610638 | 1,34502 | 1,685817 |

Талдауды жеңілдету үшін әр өлшемдегі жүйелерді шешуге ең аз уақыттар шешкен әдісті жасыл түспен, ал ең көп уақыт жұмсағанды қызғылт сары түспен боялды. Кестені алдын – ала талдау кезінде, біріктірілген градиенттер әдісінің бөлігінде жасыл түсті ұяшықтар көп екенін байқаймыз.

Әдістердің сызықты алгоритмдерінің есептеу жылдамдығын салыстырайық (9 – сурет). Егер график төмен болса, есептеуге аз уақыт жұмсалады.



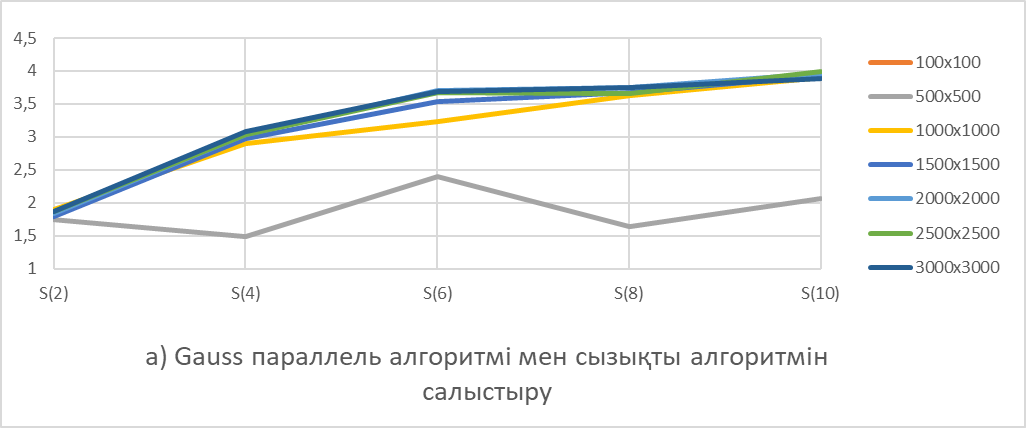
9 – сурет. Диагональдық матрицалы жүйені әдістердің сызықты алгоритмдерімен шешу уақытын салыстыру

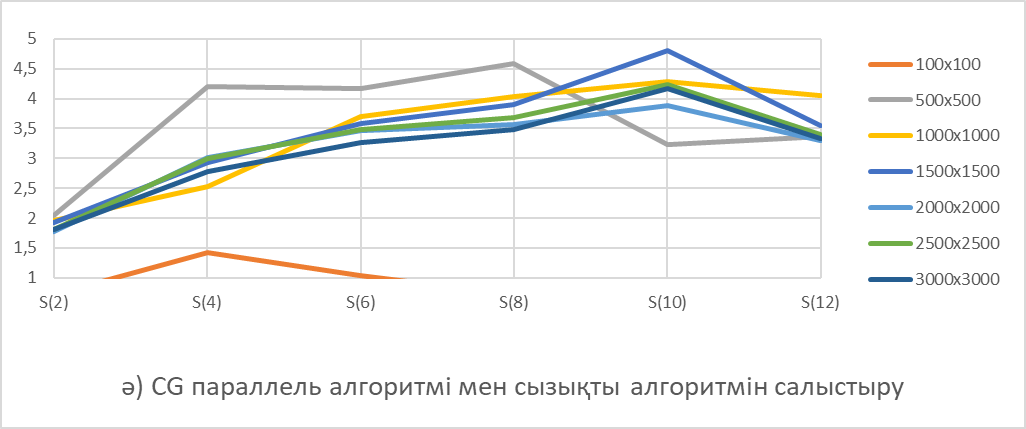
Диагональдық матрицалы жүйені шешкенде сызықты алгоритмдер арасында біріктірілген градиенттер әдісі жақсы нәтиже көрсетеді.

Әдістердің параллельді алгоритмдерінің есептеу жылдамдығын салыстырайық. Салыстыруды параллельдеудің тиімділік коэффициентін келесі формуламен есептейміз:



Мұндағы,  -  ағынды алгоритмнің сызықты алгоритмге қарағанда тиімділік коэффициенті,  -  ағынды алгоритмнің есептеу уақыты,  - сызықты алгоритмнің есептеу уақыты.





10 – сурет. Диагональдық матрицалы жүйені әдістердің параллель алгоритмінің тиімділігі: а) Гаусс әдісі; ә) біріктірілген градиенттер әдісі

100х100 өлшемді диагоналдық матрицалы жүйені шешкенде ешқандай әдістің параллель алгоритмі тиімсіз. 500х500 өлшемнен бастап сызықты алгоритмді параллелдеу 1,5 коэффициенттен жоғары тиімділікті көрсетті.

Екінші экспериментте біз коэффициенттер матрицасы үш диагональды болатын жүйені шешеміз. Есептеу уақыттарыны келесідей болды (2 – кесте)

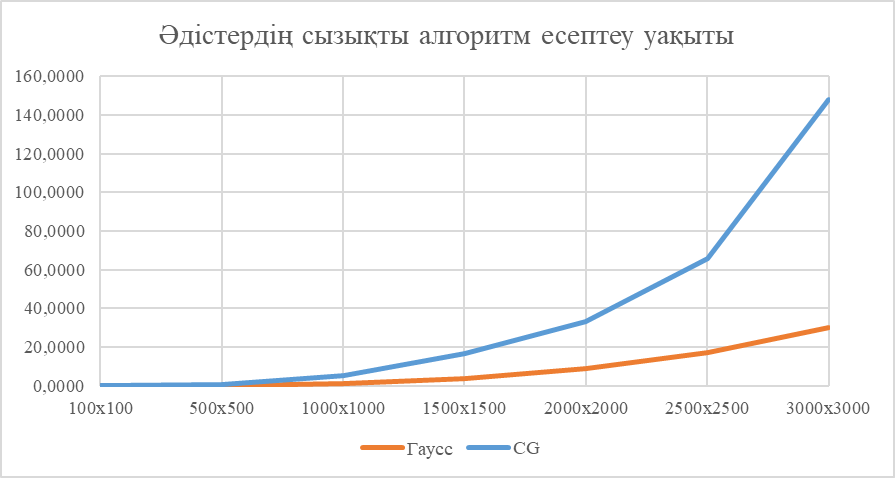
2 – кесте.

Үш диагональдық матрицалы жүйені шешуге кеткен уақыттар

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Өлшемі** | **Гаусс** | | | | | | | **CG** | | | | | | |
| **сызықты** | **2 ағын** | **4 ағын** | **6 ағын** | **8 ағын** | **10 ағын** | **12 ағын** | **сызықты** | **2 ағын** | **4 ағын** | **6 ағын** | **8 ағын** | **10 ағын** | **12 ағын** |
| 100x100 | 0,0014 | 0,0038 | 0,0042 | 0,0063 | 0,0081 | 0,0086 | 0,0106 | 0,0064 | 0,0050 | 0,0028 | 0,0040 | 0,0049 | 0,0061 | 0,0077 |
| 500x500 | 0,2237 | 0,1153 | 0,1518 | 0,1547 | 0,0993 | 0,0876 | 0,0809 | 0,6367 | 0,2682 | 0,1391 | 0,1537 | 0,1370 | 0,1152 | 0,1412 |
| 1000x1000 | 1,1593 | 0,5908 | 0,3721 | 0,3187 | 0,2970 | 0,2940 | 0,3009 | 5,4127 | 2,9006 | 1,7506 | 1,6137 | 1,4601 | 1,2480 | 1,5623 |
| 1500x1500 | 3,7581 | 1,9607 | 1,2118 | 0,9854 | 0,9756 | 0,9199 | 0,9142 | 16,4219 | 9,2029 | 5,3292 | 4,8664 | 4,4451 | 3,8997 | 4,7049 |
| 2000x2000 | 8,9061 | 4,7210 | 2,8291 | 2,4495 | 2,2688 | 2,1745 | 2,1640 | 33,0573 | 18,3599 | 10,9516 | 9,8376 | 8,9875 | 8,0709 | 9,6320 |
| 2500x2500 | 17,3675 | 9,1811 | 5,5308 | 4,5842 | 4,4492 | 4,2953 | 4,2103 | 65,6621 | 37,8197 | 21,8656 | 19,3624 | 18,4953 | 16,5479 | 20,3550 |
| 3000x3000 | 30,2802 | 16,0541 | 9,2066 | 7,5975 | 7,4348 | 7,2228 | 7,1637 | 147,7174 | 82,5579 | 49,2515 | 44,2349 | 42,9052 | 36,5436 | 48,2805 |

Талдауды жеңілдету үшін әр өлшемдегі жүйелерді шешуге ең аз уақыттар шешкен әдісті жасыл түспен, ал ең көп уақыт жұмсағанды қызғылт сары түспен боялды. Кестені алдын – ала талдау кезінде, Гаусс әдісінің бөлігінде жасыл түсті ұяшықтар көп екенін байқаймыз.

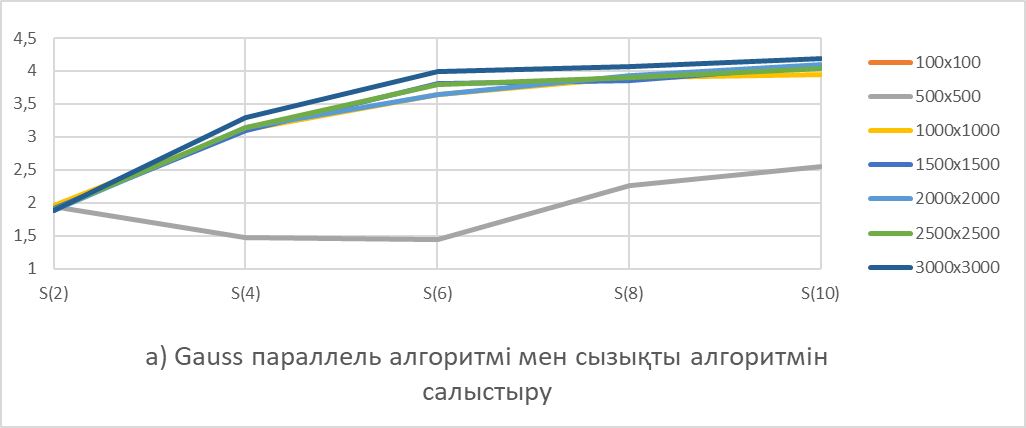
Әдістердің сызықты алгоритмдерінің есептеу жылдамдығын салыстырайық (11 – сурет). Неғұрлым төмен болса, соншалықты жақсы.



11 – сурет. Үш диагоналдық матрицалы жүйені әдістердің сызықты алгоритмдерімен шешу уақытын салыстыру

Үш диагональдық матрицалы жүйені шешкенде сызықты алгоритмдер арасында Гаусс әдісінің жетекші элементті таңдау алгоритмі жақсы нәтиже көрсетеді. Себебі біріктірілген градиенттер әдісінде матрицаны векторға көбейту операциясы көп уақыт алады.

Әдістердің параллельді алгоритмдерінің есептеу жылдамдығын салыстырайық. Салыстыруды параллельдеудің тиімділік коэффициенті келесідей:



12 – сурет. Үш диагоналдық матрицалы жүйені әдістердің параллель алгоритмінің тиімділігі: а) Гаусс әдісі; ә) біріктірілген градиенттер әдісі

100х100 өлшемді үш диагоналдық матрицалы жүйені шешкенде Гаусс әдісінің параллель алгоритмі тиімсіз, ал біріктірілген градиенттер әдісі аз да болса да тиімділік көрсетті. 500х500 өлшемнен бастап сызықты алгоритмді параллелдеу 2 коэффициенттен жоғары тиімділікті көрсетті.

Үшінші экспериментте біз коэффициенттер матрицасы қарапайым әрі симметриялы болатын жүйені шешеміз. Есептеу уақыты келесідей болды (3 – кесте)

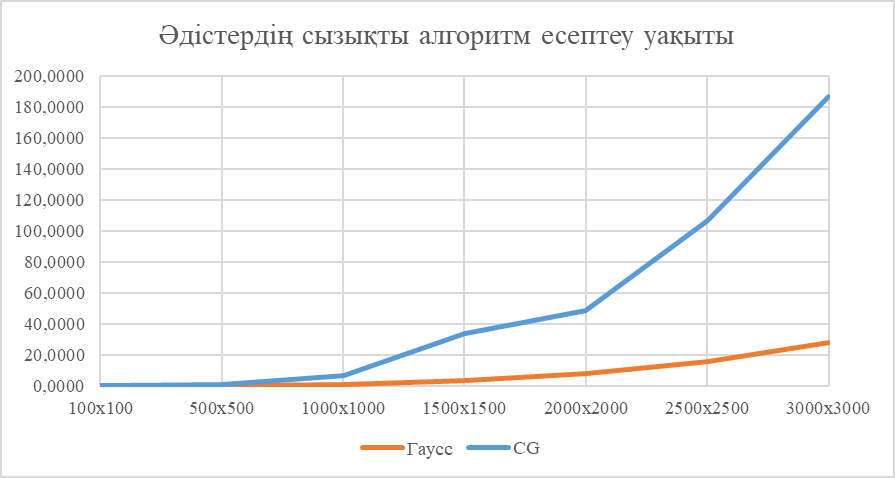
3 – кесте.

Қарапайым матрицалы жүйені шешуге кеткен уақыттар

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Өлшемі** | **Гаусс** | | | | | | | **CG** | | | | | | |
| **сызықты** | **2 ағын** | **4 ағын** | **6 ағын** | **8 ағын** | **10 ағын** | **12 ағын** | **сызықты** | **2 ағын** | **4 ағын** | **6 ағын** | **8 ағын** | **10 ағын** | **12 ағын** |
| 100x100 | 0,0012 | 0,0032 | 0,0051 | 0,0030 | 0,0035 | 0,0044 | 0,0043 | 0,0083 | 0,0031 | 0,0025 | 0,0019 | 0,0029 | 0,0026 | 0,0033 |
| 500x500 | 0,1613 | 0,0800 | 0,0544 | 0,0486 | 0,0578 | 0,0551 | 0,0564 | 0,7457 | 0,3303 | 0,1758 | 0,1699 | 0,2013 | 0,1534 | 0,1284 |
| 1000x1000 | 0,9416 | 0,5484 | 0,3168 | 0,2644 | 0,2839 | 0,2701 | 0,2745 | 6,2958 | 3,1290 | 1,6022 | 1,5602 | 1,5372 | 1,3401 | 1,2963 |
| 1500x1500 | 3,1017 | 1,9067 | 1,0563 | 0,8047 | 0,8440 | 0,8263 | 0,7952 | 33,9771 | 14,7913 | 5,8394 | 4,4244 | 5,6024 | 7,9827 | 4,6976 |
| 2000x2000 | 8,1062 | 4,6043 | 2,5662 | 1,8662 | 1,8921 | 1,8555 | 1,8241 | 48,5242 | 27,7026 | 14,9344 | 12,3576 | 14,8356 | 12,4271 | 10,9662 |
| 2500x2500 | 15,2735 | 9,9721 | 5,4730 | 4,0538 | 4,1676 | 4,3115 | 3,9398 | 106,8089 | 56,1893 | 30,6255 | 39,7872 | 33,5271 | 30,2524 | 22,9884 |
| 3000x3000 | 27,7545 | 16,0334 | 8,9066 | 6,5830 | 6,7323 | 6,5213 | 6,2795 | 187,1467 | 102,7098 | 56,3456 | 70,7828 | 55,2318 | 50,8431 | 65,0902 |

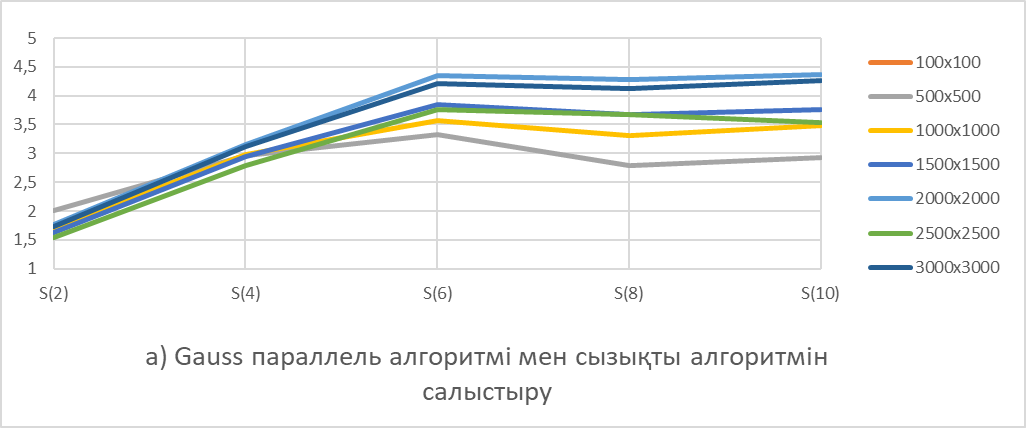
Талдауды жеңілдету үшін әр өлшемдегі жүйелерді шешуге ең аз уақыттар шешкен әдісті жасыл түспен, ал ең көп уақыт жұмсағанды қызғылт сары түспен боялды. Кестені алдын – ала талдау кезінде, Гаусс әдісінің бөлігінде жасыл түсті ұяшықтар көп екенін байқаймыз.

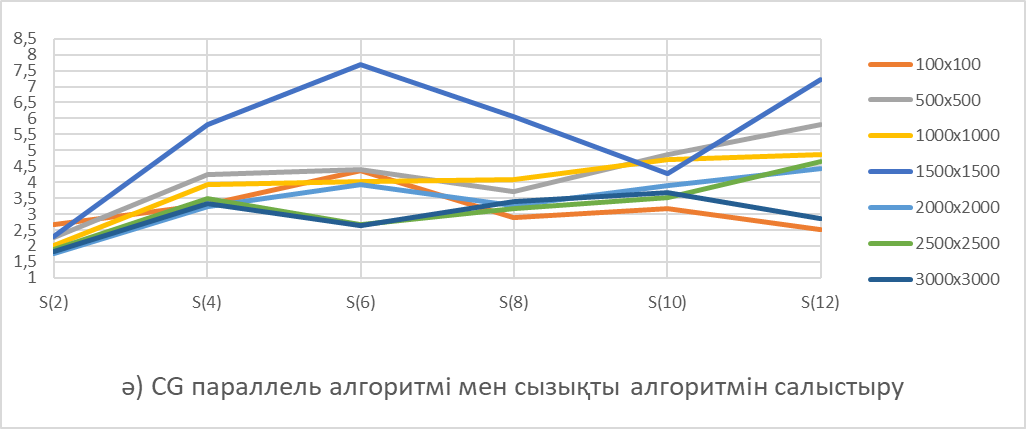
Әдістердің сызықты алгоритмдерінің есептеу жылдамдығын салыстырайық (13 – сурет). Неғұрлым төмен болса, соншалықты жақсы.



13 – сурет. Қарапайым матрицалы жүйені әдістердің сызықты алгоритмдерімен шешу уақытын салыстыру

Әдістердің параллельді алгоритмдерінің есептеу жылдамдығын салыстырайық. Салыстыруды параллельдеудің тиімділік коэффициенті келесідей:





14 – сурет. Қарапайым матрицалы жүйені әдістердің параллель алгоритмінің тиімділігі: а) Гаусс әдісі; ә) біріктірілген градиенттер әдісі

100х100 өлшемді қарапайым матрицалы жүйені шешкенде Гаусс әдісінің параллель алгоритмі тиімсіз, ал біріктірілген градиенттер әдісі 2,5 коэффициенттен жоғары тиімділікті көрсетті. 500х500 өлшемнен бастап сызықты алгоритмді параллелдеу 1,5 коэффициенттен жоғары тиімділікті көрсетті.

# **ҚОРЫТЫНДЫ**

Дипломдық жұмыс мақсаттары:

* САТЖ шешудің тура және итерациялық әдістерін зерттеу;
* САТЖ шешудің алгоритмін программалау тілінде құру;
* Құрылған алгоритм коды бойынша есептеу эксперименттерін жүргізіп, алынған нәтижелер бойынша анализ жасау.

Осы мақсатқа жету үшін жұмыс барысынды келесі міндеттер дәйекті түрде шешілді:

* САТЖ шешу әдістеріне қысқаша талдау жасау;
* Бағдарламалау алгоритмдерінің дәйекті блок – схемасын құру;
* Алгоритмді OpenMP функционалымен параллелдеу;
* Бірнеше типті коэффициенттер матрицасы үшін есептеу эксперименттерін іске асыру;
* Эксперимент нәтижелері бойынша талдау жасау;

Бірінші тарауда САТЖ шешудің ең танымал және қолданбалы әдісі, Гаусс әдісінің қысқаша тарихы, басқа дәл әдістермен салыстырғанда артықшылығы жазылған. Әдістің математикалық алгоритмінің егжей – тегжейлі талдауы, және осы алгоритмнің орындалуының көрнекті бір мысалдар көрсетілген. Осы есептеулердің жалпы формулаларын қолданып, бағдарламалау алгоритмінің блок-схемасы құрылды. Бағдарламалауға Гаусстың модификациялық әдісінің алгоритмі алынды. Сондай-ақ осы жетекші элементті таңдаумен жазылған алгоритм OpenMP көмегімен параллелдеудің нюанстары көрсетілді.

Екінші тарауда біріктірілген градиенттер әдісі қарастырылды. Әдістің алгоритмының блок – схемасы құрылған, және сол алгоритмның параллельдеу мүмкіндіктері қарастырылған.

Үшінші тарауда алдыңғы бөлімдерде құрылған блок-схема бойынша C++ тілінде бағдарламалардың қалай жазылғаны жайлы айтылған. Алгоритм дұрыс шешім беріп жатқанын анықтау үшін тексеруші бөлікке жазылды. Енді осы жазылған кодтың тиімділігі анықтау мақсатында үлкен өлшемді матрицалар есептеуге жіберілді. Ондай үлкен өлшемді матрицаларды қолмен толтырып отырмас үшін, кездейсоқ мәндермен толтыру көмекші бағдарлама жазылды. Осы қойылған эксперименттер нәтижесілеріне талдау жасалынды.

Егер коэффициенттер матрицасы диагональды болса біріктірілген градиенттер әдісі тез шешім береді. Ал басқа жағдайда Гаусс әдісі тиімдірек болады.

# **ҚОЛДАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ**

1. Ильин В. А., Позняк Э. Г. Линейная алгебра: Учебник для вузов. — T. 6. – М.: Физматлит, 2004. – 280 бет.
2. Кострикин А.И., Манин Ю.И. Линейная алгебра и геометрия. – М.: Наука, 1986. – 304 бет.
3. Стренг Г. Линейная алгебра и ее применения. – М.: Мир. – 1980. – 454 бет.
4. Ильин В.А., Ким Г.Д. Линейная алгебра и аналитическая геометрия. – М.: Проспект, 2007. – 400 бет.
5. Вержбицкий В. М. Основы численных методов. – М.: Высшая школа, 2009. – 840 бет.
6. Grcar Joseph F. How ordinary elimination became Gaussian elimination // Historia Mathematica. – 2011. – 38: Т. 2. – Б. 163-218.
7. Кремер Наум Шевелевич Высшая математика для экономистов. – М.: ЮНИТИ-ДАНА, 2010. – 3: Т. 1. – 479 бет.
8. Баландин М. Ю., Шурина Э.П. Методы решения СЛАУ большой размерности. – Новосибирск: НГТУ, 2000. – 70 б.

# **ҚОСЫМША**

Satj.cpp

#include <iostream>

#include <cstdlib>

#include <windows.h>

#include "stdio.h"

#include "omp.h"

#include "string.h"

#include <string>

using namespace std;

#include "dataGen.h" // кездейсоқ матрица құратын класс

#include "matrixHelpers.h" // шешімді тексеру класс

#include "gaussSerial.h" // Гаусс сызықты орындайтын коды

#include "gaussParallel.h" // Гаусс параллель орындайтын коды

#include "CGSerial.h" // CG сызықты орындайтын коды

#include "CGParallel.h" // CG параллель орындайтын коды

#include "csvExport.h" // \*.csv файлмен жұмыс

int main() {

printf("Max threads count = %d", omp\_get\_max\_threads());

int experiment\_number = 3; // номер эксперимента

int threads\_array[] = { 2, 4, 6, 8, 10, 12 }; // тут количество потоков

int m = sizeof(threads\_array) / sizeof(threads\_array[0]);

csvExport::write(threads\_array, experiment\_number, m);

for (int mSize = 100; mSize <= 3000; mSize += 500) { // тут размер матриц

cout << "\n Matrix size = " << mSize;

double\*\* originalA, \*\* pMatrix; //Коэффицент матрицасы (екі өлшемді)

double\* originalB, \* pVector; //Сызықтық жүйенің оң жағы

double\* pResult; //Нәтиже векторы

string times = ""; //строка для csv

originalB = new double[mSize];

pVector = new double[mSize];

pResult = new double[mSize];

originalA = new double\* [mSize];

pMatrix = new double\* [mSize];

for (int i = 0; i < mSize; i++) {

originalA[i] = new double[mSize];

pMatrix[i] = new double[mSize];

}

if (experiment\_number == 1) dataGen::DiagonalDataInitialization(originalA, originalB, mSize);//Деректерді генерациялау, pMatrix пен pVector кездейсоқ сандармен толтыру

else if (experiment\_number == 2) dataGen::ThreeDiagonalDataInitialization(originalA, originalB, mSize);//Деректерді генерациялау, pMatrix пен pVector кездейсоқ сандармен толтыру

else dataGen::randomDataInitialization(originalA, originalB, mSize);//Деректерді генерациялау, pMatrix пен pVector кездейсоқ сандармен толтыру

//Объекты бәрін құрастырамыз

gaussSerial\* gaussSerialSolver;

gaussParallel\* gaussParallelSolver;

CGSerial\* CGSerialSolver;

CGParallel\* CGParallelSolver;

matrixHelpers::setDefault(originalA, originalB, mSize, pMatrix, pVector, pResult);

double startTime = omp\_get\_wtime(); //запускаем таймер

gaussSerialSolver = new gaussSerial(mSize); // Гаусс сызықты алгоритмі

gaussSerialSolver->resultCalculation(pMatrix, pVector, pResult); // вызываем метод объекта - получаем услугу у друга

double finishTime = omp\_get\_wtime(); // останавливаем таймер, от полученного времени убавляем время старта, чтобы получить потраченнное время

printf("\nTime: %f second, method: %s, pResult[0]= %f", finishTime - startTime, "Gauss serial", pResult[0]);

times += to\_string(finishTime - startTime); //для сохранения в файле

matrixHelpers::testSolvingResult(originalA, originalB, pResult, mSize);//Нәтижені тексеру

// matrixHelpers::printVector(pResult, mSize);

for (int k = 0; k < m; k++) {

matrixHelpers::setDefault(originalA, originalB, mSize, pMatrix, pVector, pResult);

startTime = omp\_get\_wtime(); //запускаем таймер

gaussParallelSolver = new gaussParallel(mSize, threads\_array[k]);// создаем объект - добавляем в друзья

gaussParallelSolver->resultCalculation(pMatrix, pVector, pResult); // Гаусс параллель алгоритмі threads[i] поток

finishTime = omp\_get\_wtime(); // останавливаем таймер, от полученного времени убавляем время старта, чтобы получить потраченнное время

printf("\nTime: %f second, method: %s, threads count: %d, pResult[0]= %f", finishTime - startTime, "Gauss parallel", threads\_array[k], pResult[0]);

times += ";" + to\_string(finishTime - startTime); //для сохранения в файле

matrixHelpers::testSolvingResult(originalA, originalB, pResult, mSize);//Нәтижені тексеру

}

matrixHelpers::setDefault(originalA, originalB, mSize, pMatrix, pVector, pResult);

startTime = omp\_get\_wtime(); //запускаем таймер

CGSerialSolver = new CGSerial(); // CG сызықты алгоритмі

CGSerialSolver->resultCalculation(pMatrix, pVector, pResult, mSize);

finishTime = omp\_get\_wtime(); // останавливаем таймер, от полученного времени убавляем время старта, чтобы получить потраченнное время

printf("\nTime: %f second, method: %s, pResult[0]= %f", finishTime - startTime, "CG serial ", pResult[0]);

times += ";" + to\_string(finishTime - startTime); //для сохранения в файле

printf(" iteration sani: %d", CGSerialSolver->iterationsCount); //CG әдісінің бірі болса, онда қайталанулар санын көрсетеміз

matrixHelpers::testSolvingResult(originalA, originalB, pResult, mSize);//Нәтижені тексеру

for (int k = 0; k < m; k++) {

matrixHelpers::setDefault(originalA, originalB, mSize, pMatrix, pVector, pResult);

startTime = omp\_get\_wtime(); //запускаем таймер

CGParallelSolver = new CGParallel(); // CG параллель алгоритмі threads[i] поток

CGParallelSolver->resultCalculation(pMatrix, pVector, pResult, mSize, threads\_array[k]);

finishTime = omp\_get\_wtime(); // останавливаем таймер, от полученного времени убавляем время старта, чтобы получить потраченнное время

printf("\nTime: %f second, method: %s, threads count: %d, pResult[0]= %f", finishTime - startTime, "CG parallel", threads\_array[k], pResult[0]);

times += ";" + to\_string(finishTime - startTime); //для сохранения в файле

printf(", iteration sani: %d", CGParallelSolver->iterationsCount); //CG әдісінің бірі болса, онда қайталанулар санын көрсетеміз

matrixHelpers::testSolvingResult(originalA, originalB, pResult, mSize);//Нәтижені тексеру

}

csvExport::replaceAll(times,".", ",");

const char\* ti = times.c\_str(); // меняем стринг на чар, точку на запятую

csvExport::addTimes(mSize, experiment\_number, ti);

if (mSize == 100) mSize = 0;

}

}

dataGen.cpp

#include "dataGen.h"

#include "ctime"

#include "cstdlib"

void dataGen::randomDataInitialization(double\*\* pMatrix, double\* pVector, int Size) {

int i, j; // Цикл айнымалылары

srand(unsigned(clock()));

for (i = 0; i < Size; i++) {

pVector[i] = (double) rand() / double(Size);

for (j = 0; j < Size; j++) {

pMatrix[i][j] = pMatrix[j][i] = (double)rand() / double(Size);

}

}

}

void dataGen::DiagonalDataInitialization(double\*\* pMatrix, double\* pVector, int Size) {

srand(unsigned(clock()));

for (int i = 0; i < Size; i++) {

pVector[i] = (double)rand() / double(Size);

for (int j = 0; j < Size; j++) {

if (i == j) {

pMatrix[i][j] = (double) rand() / double(Size);

}

else {

pMatrix[i][j] = 0;

}

}

}

}

void dataGen::ThreeDiagonalDataInitialization(double\*\* pMatrix, double\* pVector, int Size) {

srand(unsigned(clock()));

for (int i = 0; i < Size; i++) {

pVector[i] = (double)rand() / double(Size);

for (int j = 0; j < Size; j++) {

if (fabs(j - i) < 2) {

pMatrix[i][j] = pMatrix[j][i] = (double)rand() / double(Size);

}

else {

pMatrix[i][j] = 0;

}

}

}

}

matrixHelpers.cpp

#include "matrixHelpers.h"

#include "stdio.h"

#include "math.h"

#include <vcruntime\_string.h>

#include <omp.h>

void matrixHelpers::testSolvingResult(double\*\* pMatrix, double\* pVector, double\* pResult, int Size) {

double\* pRightPartVector;

int equal = 0, i, j;

double Accuracy = 0.01f;

pRightPartVector = new double[Size];

for (i = 0; i < Size; i++) {

double tmp = 0;

#pragma omp parallel for reduction(+:tmp)

for (j = 0; j < Size; j++) {

tmp += pMatrix[i][j] \* pResult[j];

}

pRightPartVector[i] = tmp;

}

#pragma omp parallel for reduction(+:equal)

for (i = 0; i < Size; i++) {

if (fabs(pRightPartVector[i] - pVector[i]) > Accuracy) {

equal++;

}

}

if (equal > 0) {

printf(" Wrong.");

}

else {

//printf(" Сorrect.");

}

delete[] pRightPartVector;

}

void matrixHelpers::printVector(double\* matrix, int size) {

for (int i = 0; i < size; i++) {

printf("%.4f ", matrix[i]);

}

printf("\n");

}

void matrixHelpers::printMatrix(double\*\* matrix, int size) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

printf("%.4f ", matrix[i][j]);

}

printf("\n");

}

printf("\n");

}

void matrixHelpers::setDefault(double\*\* originalA, double\* originalB, int size, double\*\* A,double\* B, double\* X) {

#pragma omp parallel for

for (int i = 0; i < size; i++) {

B[i] = originalB[i];

X[i] = 0;

for (int j = 0; j < size; j++)

A[i][j] = originalA[i][j];

}

}

gaussSerial.cpp

#include "gaussSerial.h"

#include "math.h"

#include "stdio.h"

#include "matrixHelpers.h"

#include <iostream>

gaussSerial::gaussSerial(int size) {

mSize = size;

pSerialPivotIter = new int[size];

pSerialPivotPos = new int[size];

for (int i = 0; i < size; i++) {

pSerialPivotIter[i] = -1;

}

}

int gaussSerial::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult) {

serialGaussianElimination(pMatrix, pVector);// Тура жүріс, айнымалыларды Гаусс бойынша жою

serialBackSubstitution(pMatrix, pVector, pResult); // Кері жүріс, айнымалыларды есептеу

return 0;

}

// Тура жүріс, айнымалыларды Гаусс бойынша жою

void gaussSerial::serialGaussianElimination(double\*\* pMatrix, double\* pVector) {

// Ағымдағы айналмалы жолдың саны

for (int Iter = 0; Iter < mSize; Iter++) {

int PivotRow = findPivotRow(pMatrix, Iter); //жетекші жолды максимумы бойынша анықтау жолды табу

pSerialPivotPos[Iter] = PivotRow; // хранить порядок строк по итерации, нужен для обратного хода

pSerialPivotIter[PivotRow] = Iter; // хранить в каком цикле стал главным определенная строка, нужен для прямого хода

serialColumnElimination(pMatrix, pVector, PivotRow, Iter);

}

}

// Iter бағаны үшін макс элементті жолды таңдау жолды табу

//Максимумды іздеу әдісі.

int gaussSerial::findPivotRow(double\*\* pMatrix, int Iter) {

double MaxValue = 0; // бағандағы макс мәні

int PivotRow = -1; // бағандағы макс мән жолдың индексі

// жолдар бойынша макс іздеу және ол жол алдын қолданылмаған болу керек

for (int i = 0; i < mSize; i++) {

if ((pSerialPivotIter[i] == -1) && (fabs(pMatrix[i][Iter]) >= MaxValue)) {

PivotRow = i;

MaxValue = fabs(pMatrix[i][Iter]);

}

}

return PivotRow;

}

// Бағанның басқа элементтерін жою

void gaussSerial::serialColumnElimination(double\*\* pMatrix, double\* pVector, int PivotRow, int Iter) {

double PivotValue, PivotFactor;

PivotValue = pMatrix[PivotRow][Iter];

for (int i = 0; i < mSize; i++) {

if (pSerialPivotIter[i] == -1) {

PivotFactor = pMatrix[i][Iter] / PivotValue;

for (int j = Iter; j < mSize; j++) {

pMatrix[i][j] -= PivotFactor \* pMatrix[PivotRow][j];

}

pVector[i] -= PivotFactor \* pVector[PivotRow];

}

}

}

// Кері жүріс

void gaussSerial::serialBackSubstitution(double\*\* pMatrix, double\* pVector, double\* pResult) {

for (int i = mSize - 1; i >= 0; i--) {

int RowIndex = pSerialPivotPos[i];

pResult[i] = pVector[RowIndex] / pMatrix[RowIndex][i];

for (int j = i + 1; j < mSize; j++) {

pResult[i] -= pMatrix[RowIndex][j] \* pResult[j] / pMatrix[RowIndex][i];

}

}

}

gaussParallel.cpp

#include "gaussParallel.h"

#include "math.h"

#include "stdio.h"

#include <omp.h>

#include "types.h"

#include <iostream>

//конструктор - условие для доступа к классу

gaussParallel::gaussParallel(int size, int threads\_count) {

omp\_set\_num\_threads(threads\_count); // устанавливаем количество потоков в "параллельных" блоках

mSize = size;

pSerialPivotIter = new int[size]; // хранить в каком цикле стал главным определенная строка, нужен для прямого хода

pSerialPivotPos = new int[size]; // хранить порядок строк по итерации, нужен для обратного хода

#pragma omp parallel for

for (int i = 0; i < size; i++) {

pSerialPivotIter[i] = -1; // Бұл жолдарға әлі кірмегенімізді көрсету үшін -1 толтырамыз

}

}

int gaussParallel::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult) {

gaussianElimination(pMatrix, pVector); // Тура жүріс, айнымалыларды Гаусс бойынша жою

backSubstitution(pMatrix, pVector, pResult); // Кері жүріс, айнымалыларды есептеу

return 0;

}

// Тура жүріс, айнымалыларды Гаусс бойынша жою

int gaussParallel::gaussianElimination(double\*\* pMatrix, double\* pVector) {

for (int Iter = 0; Iter < mSize; Iter++) {

int PivotRow = findPivotRow(pMatrix, Iter); // бағандағы макс табу

pSerialPivotPos[Iter] = PivotRow; // хранить порядок строк по итерации, нужен для обратного хода

pSerialPivotIter[PivotRow] = Iter; // хранить в каком цикле стал главным определенная строка, нужен для прямого хода

columnElimination(pMatrix, pVector, PivotRow, Iter); // Бағанның басқа элементтерін жою

}

return 0;

}

// Iter бағаны үшін макс элементті жолды таңдау жолды табу

int gaussParallel::findPivotRow(double\*\* pMatrix, int Iter) {

double MaxValue = 0; // бағандағы макс элементтің мәні

int PivotRow = -1; // бағандағы макс элементті жолдың индексі

int i;

#pragma omp parallel

{

TThreadPivotRow ThreadPivotRow;

ThreadPivotRow.MaxValue = 0; //задаем макс переменные каждому потоку

ThreadPivotRow.PivotRow = -1; //задаем ид строки со макс значением каждому потоку

#pragma omp for

for (i = 0; i < mSize; i++) {

if ((pSerialPivotIter[i] == -1) && (fabs(pMatrix[i][Iter]) > ThreadPivotRow.MaxValue)) {

ThreadPivotRow.PivotRow = i;

ThreadPivotRow.MaxValue = fabs(pMatrix[i][Iter]);

}

}

#pragma omp critical

{

if (ThreadPivotRow.MaxValue > MaxValue) {

MaxValue = ThreadPivotRow.MaxValue;

PivotRow = ThreadPivotRow.PivotRow;

}

} // pragma omp critical

}// pragma omp parallel

return PivotRow;

}

// Бағанның басқа элементтерін жою

int gaussParallel::columnElimination(double\*\* pMatrix, double\* pVector, int Pivot, int Iter) {

double PivotValue, PivotFactor;

PivotValue = pMatrix[Pivot][Iter];

#pragma omp parallel for private (PivotFactor) schedule(dynamic, 1)

for (int i = 0; i < mSize; i++) { // ti1:i=0; ti2:i=1; --> ti2:i=2;-->ti2:i=3;-->ti1

if (pSerialPivotIter[i] == -1) {

PivotFactor = pMatrix[i][Iter] / PivotValue;

for (int j = Iter; j < mSize; j++) {

pMatrix[i][j] -= PivotFactor \* pMatrix[Pivot][j];

}

pVector[i] -= PivotFactor \* pVector[Pivot];

}

}

return 0;

}

// Кері жүріс

int gaussParallel::backSubstitution(double\*\* pMatrix, double\* pVector, double\* pResult) {

int RowIndex, Row;

for (int i = mSize - 1; i >= 0; i--) {

RowIndex = pSerialPivotPos[i];

double tmp = pVector[RowIndex] / pMatrix[RowIndex][i];

#pragma omp parallel for reduction (-:tmp)

for (int j = i + 1; j < mSize; j++) {

tmp -= pMatrix[RowIndex][j] \* pResult[j] / pMatrix[RowIndex][i];

}

pResult[i] = tmp;

}

return 0;

}

int gaussSerial::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult) {

serialGaussianElimination(pMatrix, pVector);// Тура жүріс, айнымалыларды Гаусс бойынша жою

serialBackSubstitution(pMatrix, pVector, pResult); // Кері жүріс, айнымалыларды есептеу

return 0;

}

// Тура жүріс, айнымалыларды Гаусс бойынша жою

void gaussSerial::serialGaussianElimination(double\*\* pMatrix, double\* pVector) {

// Ағымдағы айналмалы жолдың саны

for (int Iter = 0; Iter < mSize; Iter++) {

int PivotRow = findPivotRow(pMatrix, Iter); //жетекші жолды максимумы бойынша анықтау жолды табу

pSerialPivotPos[Iter] = PivotRow; // хранить порядок строк по итерации, нужен для обратного хода

pSerialPivotIter[PivotRow] = Iter; // хранить в каком цикле стал главным определенная строка, нужен для прямого хода

serialColumnElimination(pMatrix, pVector, PivotRow, Iter);

}

}

// Iter бағаны үшін макс элементті жолды таңдау жолды табу

int gaussSerial::findPivotRow(double\*\* pMatrix, int Iter) {

double MaxValue = 0; // бағандағы макс мәні

int PivotRow = -1; // бағандағы макс мән жолдың индексі

// жолдар бойынша макс іздеу және ол жол алдын қолданылмаған болу керек

for (int i = 0; i < mSize; i++) {

if ((pSerialPivotIter[i] == -1) && (fabs(pMatrix[i][Iter]) >= MaxValue)) {

PivotRow = i;

MaxValue = fabs(pMatrix[i][Iter]);

}

}

return PivotRow;

}

// Бағанның басқа элементтерін жою

void gaussSerial::serialColumnElimination(double\*\* pMatrix, double\* pVector, int PivotRow, int Iter) {

double PivotValue, PivotFactor;

PivotValue = pMatrix[PivotRow][Iter];

for (int i = 0; i < mSize; i++) {

if (pSerialPivotIter[i] == -1) {

PivotFactor = pMatrix[i][Iter] / PivotValue;

for (int j = Iter; j < mSize; j++) {

pMatrix[i][j] -= PivotFactor \* pMatrix[PivotRow][j];

}

pVector[i] -= PivotFactor \* pVector[PivotRow];

}

}

}

// Кері жүріс

void gaussSerial::serialBackSubstitution(double\*\* pMatrix, double\* pVector, double\* pResult) {

for (int i = mSize - 1; i >= 0; i--) {

int RowIndex = pSerialPivotPos[i];

pResult[i] = pVector[RowIndex] / pMatrix[RowIndex][i];

for (int j = i + 1; j < mSize; j++) {

pResult[i] -= pMatrix[RowIndex][j] \* pResult[j] / pMatrix[RowIndex][i];

}

}

}

CGSerial.cpp

#include "CGSerial.h"

#include "math.h"

#include "stdio.h"

void CGSerial::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult, int Size) {

double\* g, \* d, \* A\_prev\_d;

double step = 0, beta = 0, sum\_new\_g = 0, sum\_prev\_g = 0, sum\_PR;

int Iter = 0, i, j;

float Accuracy = 0.01f; //шешім қателігі

g = new double[Size];

d = new double[Size];

A\_prev\_d = new double[Size];

// Бастапқы мәндерді енгізіп қою

for (i = 0; i < Size; i++) {

pResult[i] = pVector[i];

g[i] = pVector[i];

for (j = 0; j < Size; j++) {

g[i] -= pMatrix[i][j] \* pVector[j];

}

d[i] = g[i];

sum\_new\_g += g[i] \* g[i];

}

sum\_PR = sum\_new\_g;

do {

Iter++;

sum\_prev\_g = sum\_new\_g;

double ip = 0;

// 1 этап вычисление step - қадам ұзындығы

for (i = 0; i < Size; i++) {

A\_prev\_d[i] = 0;

for (j = 0; j < Size; j++) {

A\_prev\_d[i] += pMatrix[i][j] \* d[j];

}

ip += A\_prev\_d[i] \* d[i]; // бөлімі

}

step = sum\_prev\_g / ip;

// 2 - 3 этап вычисление new\_x - жаңа шешім, new\_g - жаңа градиент

sum\_new\_g = 0;

sum\_PR = 0;

for (i = 0; i < Size; i++) {

pResult[i] += step \* d[i];

g[i] -= step \* A\_prev\_d[i];

sum\_new\_g += g[i] \* g[i];

}

if (sqrt(sum\_new\_g) <= Accuracy) break;

// 4 этап вычисление beta - параметр сопряженности

beta = fmax(sum\_new\_g / sum\_prev\_g, 0);

// 5 этап вычисление new\_d - жаңа бағыт

for (i = 0; i < Size; i++) {

d[i] = g[i] + beta \* d[i];

}

iterationsCount = Iter;

} while (sqrt(sum\_new\_g) > Accuracy);

}

CGParallel.cpp

#include "CGParallel.h"

#include "math.h"

#include <omp.h>

#include <iostream>

void CGParallel::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult, int Size, int threads\_count) {

omp\_set\_num\_threads(threads\_count); // устанавливаем количество потоков в "параллельных" блоках

double\* g, \* d, \* A\_prev\_d;

double step = 0, beta = 0, sum\_new\_g = 0, sum\_prev\_g = 0;

int Iter = 0, i, j;

float Accuracy = 0.01f; //шешім қателігі

g = new double[Size];

d = new double[Size];

A\_prev\_d = new double[Size];

// Бастапқы мәндерді енгізіп қою

#pragma omp parallel for reduction(+:sum\_new\_g) private (i)

for (i = 0; i < Size; i++) {

pResult[i] = 0;

g[i] = pVector[i];

d[i] = -pVector[i];

sum\_new\_g += g[i] \* g[i];

}

do {

Iter++;

sum\_prev\_g = sum\_new\_g;

double ip = 0;

// 1 этап вычисление step - қадам ұзындығы

#pragma omp parallel private (i,j)

#pragma omp for reduction(+:ip)

for (i = 0; i < Size; i++) {

A\_prev\_d[i] = 0;

for (j = 0; j < Size; j++){

A\_prev\_d[i] += pMatrix[i][j] \* d[j];

}

ip += A\_prev\_d[i] \* d[i];

}

step = sum\_prev\_g / ip;

// 2 - 3 этап вычисление new\_x - жаңа шешім, new\_g - жаңа градиент

sum\_new\_g = 0;

#pragma omp parallel for reduction(+:sum\_new\_g) private (i)

for (i = 0; i < Size; i++) {

pResult[i] = pResult[i] - step \* d[i];

g[i] = g[i] + step \* A\_prev\_d[i];

sum\_new\_g += g[i] \* g[i];

}

if (sqrt(sum\_new\_g) <= Accuracy) break;

// 4 этап вычисление beta - параметр сопряженности

beta = fmax(sum\_new\_g / sum\_prev\_g, 0);

// 5 этап вычисление new\_d - жаңа бағыт

#pragma omp parallel for private (i)

for (i = 0; i < Size; i++) {

d[i] = -g[i] + beta \* d[i];

}

iterationsCount = Iter;

} while (sqrt(sum\_new\_g) > Accuracy);

}

csvExport.cpp

#include "csvExport.h"

#include <iostream>

#include <fstream>

#include <string>

#include "string.h"

using namespace std;

int csvExport::write(int \*threads\_array, int experiment\_number, int size) { // метод записи результатов в файл

ofstream myfile;

string filename = "result";

filename += to\_string(experiment\_number);

filename +=".csv";

cout << '\n' << filename;

myfile.open(filename);

myfile << "Размер;Гаусс линейно;";

for (int i = 0; i < size; i++) {

myfile << "Гаусс параллель(" << threads\_array[i] << ");";

}

myfile << "CG линейно;";

for (int i = 0; i < size; i++) {

myfile << "CG параллель(" << threads\_array[i] << ");";

}

myfile.close();

return 0;

}

int csvExport::addTimes(int Size, int experiment\_number, const char\* times)

{

ofstream myfile;

string filename = "result";

filename += to\_string(experiment\_number);

filename += ".csv";

myfile.open(filename,ios::app); // файлды ашып, соңына алып барады(так, чтобы предыдущие не стерлись)

myfile << "\n" <<Size << "x" << Size << ";" << times;

myfile.close();

return 0;

}

void csvExport::replaceAll(std::string& str, const std::string& from, const std::string& to){

if (from.empty())

return;

size\_t start\_pos = 0;

while ((start\_pos = str.find(from, start\_pos)) != std::string::npos) {

str.replace(start\_pos, from.length(), to);

start\_pos += to.length(); // In case 'to' contains 'from', like replacing 'x' with 'yx'

}

}