МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «**ДОСЛІДЖЕННЯ АВТОМАТИЗОВАНОГО ПІДБОРУ ПАРАМЕТРІВ ДЛЯ ПРОЦЕДУР БІБЛІОТЕКИ SCALAPACK**»

Виконав: студент 2 курсу, групи 8.1218-з
спеціальності 121 інженерія програмного забезпечення
(шифр і спеціальність)
освітньої програми інженерія програмного забезпечення
(шифр і назва спеціальності)
І.В. Говтвян
(ініціали та прізвище)

Керівник — доцент кафедри програмної інженерії доцент, к.ф-м.н, Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент — доцент кафедри фундаментальної математики, доцент к.ф-м.н. Панасенко Є.В
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Математичний

Кафедра програмної інженерії

Рівень вищої освіти Магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 29 ” травня 2019 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТА

Говтвян Ірині Валентинівні
(прізвище, ім'я та по-батькові)

1. Тема роботи Дослідження автоматизованого підбору параметрів для процедур бібліотеки ScaLAPACK

керівник роботи Горбенко В. І., к.ф.-м.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

Затверджені наказом ЗНУ від « 29 » травня 2019 року № 812-с

2. Строк подання студентом роботи 20.05.2019

3. Вихідні дані до роботи 1. Постановка задачі
2. Перелік літератури

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Дослідити автоматизований підбір параметрів бібліотеки ScaLapack.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація до захисту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 29.05.2019

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	29.08.2019	Виконано
2.	Збір вихідних даних.	24.09.2019	Виконано
3.	Обробка методичних та теоретичних	13.10.2019	Виконано
	Джерел		
4.	Розробка першого і другого розділу.	25.11.2019	Виконано
5.	Розробка третього розділу.	13.12.2019	Виконано
6.	Оформлення і нормконтроль	28.12.2019	Виконано
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	10.01.2020	Виконано

Студентка _____ І.В. Говтвян
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Дослідження автоматизованого підбору параметрів для процедур бібліотеки ScaLAPACK»: 72 с., 17 рис., 7 табл., 41 джерел, 2 додатки.

БІБЛІОТЕКА, МАТРИЦЯ, СЛАР, INTEL, MICROSOFT VISUAL STUDIO MPI, SCALAPACK.

Об’єкт дослідження – бібліотека ScaLAPACK та розрахунки, за допомогою неї матриць.

Мета роботи: дослідження автоматизованого підбору параметрів для процедур бібліотеки ScaLAPACK на прикладі проведення операцій з квадратними матрицями.

Метод дослідження – теоретичний аналіз та вивчення літератури, а також математичне моделювання, прикладні методи програмування. Основні результати магістерської роботи (наукові, практичні): розроблено спеціальну програму (комп'ютерна модель) в об'єктному середовищі програмування для обчислення операцій з матрицями.

Завдання, поставлені для досягнення мети роботи:

1) дослідити автоматизований підбір параметрів бібліотеки;

2) проаналізувати джерела інформації з даної теми;

3) написати програму, с застосуванням бібліотеки ScaLAPACK, для проведення математичних операцій з матрицями.

SUMMARY

Master’s Qualification Thesis «Investigation of Automated Selection of Parameters for ScaLAPACK Library Procedures»: 76 pages, 17 figures, 7 tables, 30 references, 2 supplement.

LIBRARY, MATRIX, SLAE, INTEL, MICROSOFT VISUAL STUDIO, MPI, SCALAPACK.

The object of the study is the ScaLAPACK library and its use for calculating matrices.

The aim of the study is research of automated selection of parameters for ScaLAPACK library procedures.

The methods of research ave– theoretical analysis and study of literature, as well as mathematical modeling, applied programming methods. The main results of the master's work (scientific, practical): a special program was developed (computer model) in the object programming environment for calculating operations with matrices.

Tasks set to achieve the goal of the work:

1) research the automated selection of parameters for library procedures;

2) analyze the sources of information on this topic;

3) Write programs, using ScaLAPACK libraries, for mathematical operations with matrices.

ЗМІСТ

Завдання на кваліфікаційну роботу студента 4
Реферат 5
Summary 6
Вступ 8
1 Аналіз предметної області та вимог 9
1.1 Загальний огляд бібліотеки ScaLAPACK 9
1.1.1 Структура бібліотеки ScaLAPACK 9
1.1.2 Функціонал бібліотеки ScaLAPACK 12
1.2 Постановка задачі 14
1.3 Технічне завдання 145
1.4 Вибір засобів розробки 16
2. Створення програмного середовища 21
2.1. Обчислювальна система 21
2.2. Встановлення бібліотеки ScaLAPACK 21
2.3. Постановка задачі для дослідження матричних операцій 25
2.4. Постановка задачі та загальний алгоритм для дослідження рішення СЛАР 27
2.5. Реалізація обчислень із використанням бібліотеки ScaLAPACK 32
3. Дослідження ефективності використання бібліотеки ScaLAPACK 36
3.1. Множення двох квадратних матриць 36
3.2. Рішення системи лінійних алгебраїчних рівнянь 42
Висновки 51
Перелік посилань 52
Додаток А. Програма множення матриць 55
Додаток Б. Рішення системи лінійних рівнянь із матрицею загального виду 62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення;

ПП – програмний продукт;

ЛА – лінійна алгебра

СЛАР – система лінійних алгебраїчних рівнянь;

IDE (англ. Integrated Development Environment) – Інтегроване середовище розробки

MPI (англ. Message Passing Interface) – інтерфейс передачі повідомлень;

BLAS (англ. Basic Linear Algebra Subprograms) – стандарт інтерфейсу бібліотек підпрограм, призначених для виконання основних операцій ЛА;

BLACS (англ. Basic Linear Algebra Communication Subprograms) – бібліотека передачі повідомлень, призначена для ЛА;

PBLAS (англ. Parallel Basic Linear Algebra Subprograms) – реалізація BLAS для архітектур з розподіленою пам'яттю;

LAPACK (англ. Linear Algebra Package) – бібліотека для числової ЛА;

ScaLAPACK (англ. Scalable Linear Algebra Package) – бібліотека для числової ЛА для архітектур з розподіленою пам'яттю;

ВСТУП

На сьогоднішній день розвиток обчислювальних машин та математичного апарату дозволяє вирішувати великий спектр наукових та інженерних завдань. Зазвичай, розв'язання таких задач вимагає виконання великої кількості математичних операцій, найчастіше операцій ЛА. Тому для їх вирішення використовуються суперкомп'ютери – потужні комп'ютерні системи.

Як правило, сучасні суперкомп'ютери складаються з великої кількості високопродуктивних серверних комп'ютерів, з'єднаних між собою локальною високошвидкісною магістраллю для досягнення продуктивності в рамках підходу розпаралелювання обчислювальної задачі, який має назву розподілених обчислень. Розподілені обчислення є окремим випадком паралельних обчислень, тобто одночасного розв'язання різних частин одного обчислювального завдання декількома процесорами одного або кількох комп'ютерів.

Виходячи з усього вищезгаданого, було створено бібліотеку ScaLAPACK з відкритим сирцевим кодом, яка надає можливість вирішувати різноманітні операції ЛА з використанням розподілених систем.

Проте, не дивлячись на бурхливе удосконалення обчислювальних машин та програмних засобів і росту кількості наукових та інженерних задач, світові тенденції розвитку освіти та бізнесу виключили можливість вільного використання суперкомп'ютерів людей без знань у сфері програмування.

Розроблювальний ПП призначений для спрощення вирішення задач ЛА на розподілених комп'ютерних системах людьми без знань у сфері програмування, завдяки автоматизації підбору параметрів для процедур бібліотеки ScaLAPACK.

У першому розділі проводиться дослідження та огляд ScaLAPACK, постановка задачі, огляд аналогів, розробка технічного завдання та засобів

У другому розділі створюється програмне середовище .

У третьому розділі досліджується ефективність використання бібліотеки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ

1.1 Загальний огляд бібліотеки ScaLAPACK

1.1.1 Структура бібліотеки ScaLAPACK

ScaLAPACK – бібліотека з відкритим вихідним кодом, що включає в себе підмножина процедур LAPACK, перероблених для використання на розподілених комп'ютерних системах [23], включаючи: рішення СЛАР і найменших квадратів, знаходження власних значень та векторів, розкладу матриці різними методами та багато інших. ScaLAPACK розроблена з використанням PBLAS і BLACS, і призначена для обчислень на будь-якому комп'ютері або кластері, який підтримує MPI.

Як і LAPACK, підпрограми ScaLAPACK засновані на алгоритмах з розділеними блоками, щоб мінімізувати частоту руху даних між різними рівнями ієрархії пам'яті. Фундаментальними будівельними блоками бібліотеки ScaLAPACK є версії з розподіленою пам'яттю BLAS рівня 1, 2 і 3 рівня, що називаються паралельними BLAS або PBLAS, і набір підпрограм базової лінійної алгебри зв’язку (BLACS) для завдань зв'язку, які часто виникають при паралельних обчисленнях завдань ЛА. У підпрограмах ScaLAPACK більшість міжпроцесорних комунікацій відбувається в межах PBLAS, тому вихідний код верхнього програмного рівня ScaLAPACK виглядає аналогічно тому, як у LAPACK [24].

Ієрархія пакетів бібліотеки ScaLAPACK (див. рис. 1.1) [25]. Пакети, які знаходяться під пунктирною лінією та які позначені як «Локальні», використовуються на тільки одному процесорі з аргументами, які зберігаються лише на одному процесорі. Пакети, які знаходяться над пунктирною лінією та які позначені як «Глобальні», є синхронними паралельними процедурами, аргументи яких включають матриці та вектори, розподілені по декількох процесорах.

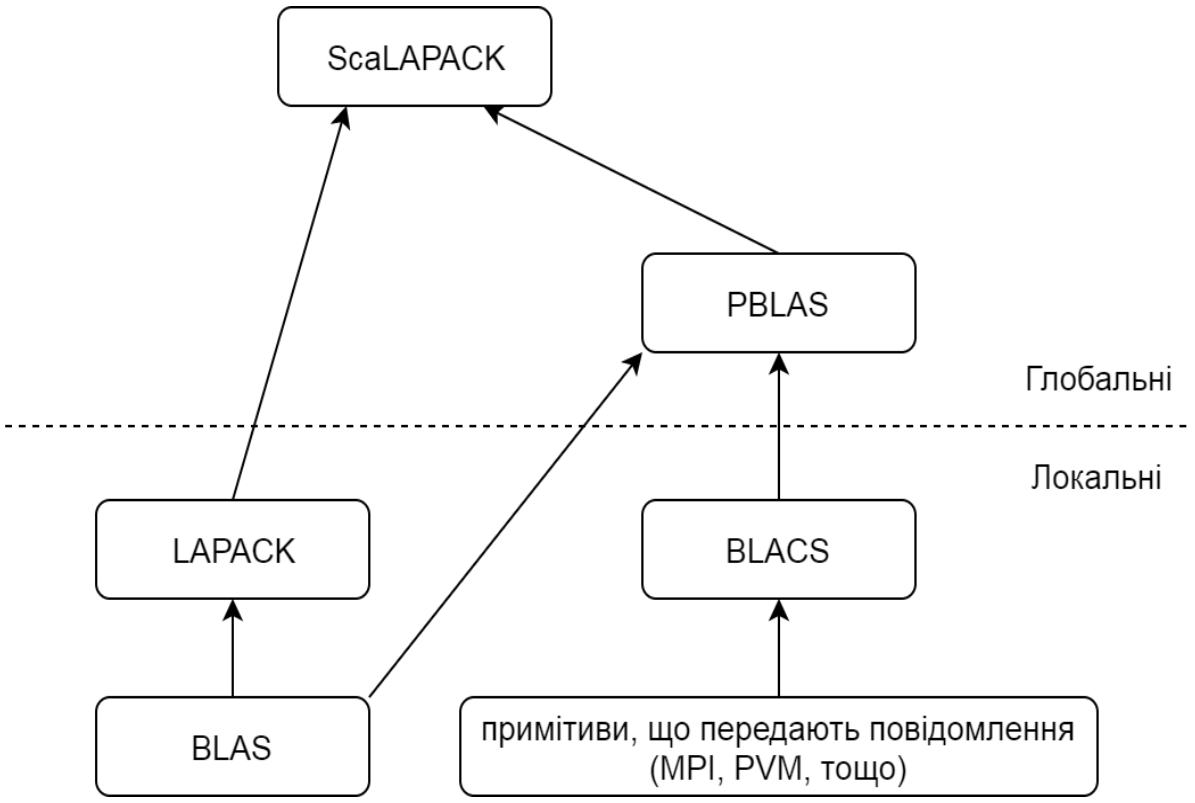


Рисунок 1.1 – Ієрархія бібліотеки ScaLAPACK

Поняття, що входять до складу ScaLAPACK, мають певне призначення.

MPI – це специфікація, що була розроблена в 1993–1994 роках групою MPI Forum, і забезпечує реалізацію моделі обміну повідомленнями між процесами. У моделі програмування MPI програма породжує кілька процесів, що взаємодіють між собою за допомогою виклику підпрограм прийому й передачі повідомлень. Зазвичай, при ініціалізації MPI-програми створюється фіксований набір процесів, причому (що, утім, необов'язково) кожний з них виконується на своєму процесорі. Специфікація MPI забезпечує переносимість програм на рівні вихідних кодів. Підтримується робота на гетерогенних кластерах і симетричних мультипроцесорних системах [26].

BLACS – це бібліотека, що передає повідомлення, розроблена для ЛА. Обчислювальна модель складається з одно- або двовимірної технологічної сітки, де кожен процес зберігає фрагменти матриць і векторів. BLACS включає синхронні підпрограми передачі/прийому для передачі матриці або її частини з одного процесу в інший, для трансляції частин матриць для багатьох процесів або для обчислення глобальних скорочень (сум, максимумів і мінімумів).

Існують також підпрограми для побудови, зміни або запиту в сітці процесів. Оскільки декілька алгоритмів ScaLAPACK вимагають трансляції або скорочення між різними підмножинами процесів, BLACS дозволяє процесу бути членом декількох перекриваючих або неперервних сіток процесів, кожна з яких позначена контекстом. BLACS надає засоби для безпечної взаємодії системних контекстів та контекстів BLACS. Важливою метою BLACS є створення портативного, специфічного для задач ЛА шару зв'язку [27].

BLAS – основні лінійні підпрограми алгебри, які включають підпрограми для загальних обчислень ЛА, такі як скалярні добутки, множення матриці на вектор та множення двох матриць. Як відомо, використання матричних операцій (зокрема, множення матриць), налаштованих на певну архітектуру, може маскувати ефекти ієрархії пам'яті (пропуски кешу, помилки TLB тощо) та дозволяти виконувати операції з плаваючою комою поблизу пікової швидкості машини. Важливою метою BLAS є забезпечення рівня переносимості для обчислення [28].

LAPACK – це сукупність процедур ЛА для вирішення СЛАР, задач найменших квадратів, знаходження власних значень і веткорів, вирішення задач розкладу матриці тощо. Висока продуктивність досягається за допомогою алгоритмів, які проводять більшу частину своєї роботи при викликах до BLAS, з акцентом на множення матриці-матриці. Кожна програма має один або кілька параметрів настройки продуктивності, таких як розміри блоків, якими керується BLAS [29].

PBLAS – це бібліотека, яка була створена для спрощення дизайну ScaLAPACK на основі BLAS. Оскільки BLAS виявився корисним інструментом поза LAPACK, розробники вирішили побудувати паралельний набір BLAS, який би виконував передачу повідомлень між процесорами і інтерфейс якого був би подібний до BLAS, наскільки це можливо. Це рішення дозволило коду ScaLAPACK бути досить схожим, а іноді майже ідентичним аналогічному коду LAPACK [30].

1.1.2 Функціонал бібліотеки ScaLAPACK

Бібліотека ScaLAPACK підтримує 4 типи даних та налічує приблизно 540 процедур [31]. Типи даних, які підтримує ScaLAPACK:

а) дійсний;

б) дійсний з подвійною точністю;

в) комплексний;

г) комплексний з подвійною точністю.

Всі процедури поділяються на три категорії для кожного з 4-х типів даних:

а) auxiliary routines (допоміжні/службові підпрограми) – підпрограми виконують деякі внутрішні допоміжні дії і, як правило, самостійно не використовуються. Здебільшого ці підпрограми використовуються чисельними аналітиками або розробниками програмного забезпечення;

б) computational routines (обчислювальні підпрограми) – ці підпрограми виконують окремі підзадачі, наприклад, LU-розкладання матриці або приведення дійсної симетричної матриці до трьох-діагонального виду;

в) driver routines (драйверні підпрограми) – це підпрограми, які вирішують деякі закінчені завдання, наприклад, рішення СЛАР або знаходження власних значень дійсної симетричної матриці. Таких підпрограм 14 для кожного типу даних. Ці підпрограми використовують набори обчислювальних підпрограм. Виходячи з цього можна зауважити, що обчислювальні підпрограми значно перекривають функціональні потреби драйверних підпрограм і можуть використовуватися самостійно.

Також усі процедури ScaLAPACK неявно припускають використання різноманітної кількості видів матриць для вирішення задач ЛА. Для підтвердження цього факту розглянемо схему іменування підпрограм бібліотеки.

Імена всіх драйверних і обчислювальних підпрограм збігаються з іменами відповідних підпрограм з пакета LAPACK, з тією лише різницею, що на початку імені додається символ P, який вказує на те, що це паралельна версія підпрограми. Відповідно, принцип формування імен підпрограм має ту ж саму

схему, що і в LAPACK [32]. Відповідно до цієї схеми імена підпрограм бібліотеки ScaLAPACK мають вигляд PTXXYYY, де:

а) T – вказує тип вихідних даних (див. табл. 1.1).

Таблиця 1.1 – Перелік можливих значень параметру T

Значення T	Пояснення
S	дійсний з одинарною точністю
D	дійсний з подвійною точністю
C	комплексний з одинарною точністю
Z	комплексний з подвійною точністю

б) XX – вказує тип матриці (див. табл. 1.2).

Таблиця 1.2 – Перелік можливих значень параметру XX

Значення XX	Пояснення
DB	стрічкові загального вигляду з переважаючими діагональними елементами
DT	трьох-діагональні загального вигляду з переважаючими діагональними елементами
GB	стрічкові загального вигляду
GE	загального вигляду
GT	трьох-діагональні загального вигляду
HE	Ермітова матриця
PB	стрічкові симетричні або ермітові додатноозначені
PO	симетричні або ермітові додатноозначені
PT	трьохдіагональні симетричні чи ермітові додатноозначені
ST	симетричні трьох-діагональні
SY	симетричні
TR	трикутні
TZ	трапецієподібні
UN	унітарні

в) YYY – вказує дії, які виконує підпрограма (див. табл. 1.3).

Таблиця 1.3 – Перелік можливих значень параметру YYY

Значення T	Пояснення
TRF	факторизація матриць
TRS	рішення СЛАР після факторизації
CON	оцінка числа обумовленості матриці (після факторизації)
SV	Рішення СЛАР
SVX	Рішення СЛАР з додатковими дослідженнями
EV та EVX	Обчислення власних значень та власних векторів
GVX	рішення узагальненої задачі на власні значення
SVD	обчислення сингулярних значень
RFS	уточнення рішення
LS	знаходження найменших квадратів

1.2 Постановка задачі

Метою роботи є дослідження автоматизованого підбору параметрів для процедур бібліотеки ScaLAPACK за допомогою створення графічного додатку , використовуючи мову програмування C++ у середовищі Microsoft Visual Studio, бібліотеку графічних компонентів Qt та бібліотеку Intel MKL.

Основними властивостями проекту є:

а) вирішення задач ЛА, таких як: СЛАР, факторизація матриць, обчислення сингулярних значень тощо;

б) налаштування параметрів вихідних матриць;

в) емуляція розподіленої системи;

г) ведення логу роботи програми;

д) формування звіту ходу виконаної роботи;

е) збереження налаштувань програми;

1.3 Технічне завдання

На основі дослідження автоматизованого підбору параметрів для процедур бібліотеки ScaLAPACK є створювання ПЗ, використовуючи мову програмування C++ у середовищі Microsoft Visual Studio, бібліотеку графічних компонентів Qt та бібліотеку Intel MKL є розробка графічного застосунку для автоматизованого підбору параметрів для процедур бібліотеки ScaLAPACK.

Головними функціями створюваного ПП є:

а) дослідити автоматизований підбір параметрів для бібліотеки на основі роботи з матрицями різного виду та типу вихідних даних;

б) вирішення СЛАР;

в) розклад матриць: LU-розклад, розклад Холецького, сингулярний розклад;

г) знаходження власних значень і власних векторів матриць;

д) оцінка числа обумовленості;

е) уточнення рішення СЛАР;

ж) емуляція розподіленої системи;

з) ведення логу роботи програми;

и) формування звіту ходу виконаної роботи;

к) збереження налаштувань програми;

Для наочної демонстрації майбутнього ПЗ був розроблений макет головного вікна програми (див. рис. 1.2)

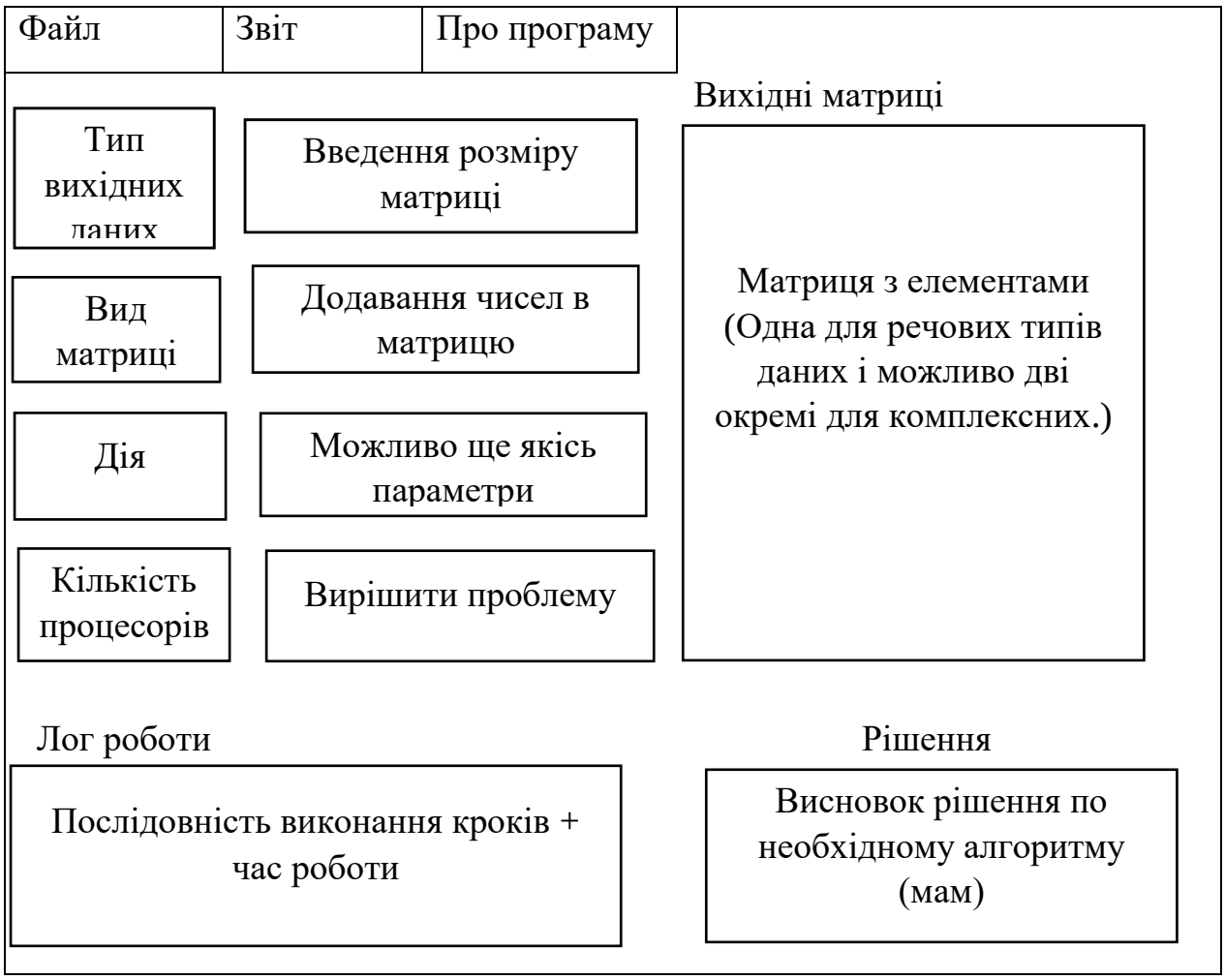


Рисунок 1.2 – Макет головного вікна

1.4 Вибір засобів розробки

MS Visual Studio 2013 – це ефективне і високопродуктивний засіб розробки мовою C ++, який пропонує різноманітні можливості для створення додатків нового покоління для будь-яких платформ та типів пристроїв у найкоротші терміни. MS Visual Studio 2013 виводить C ++ на новий рівень продуктивності без втрати гнучкості, швидкодії та контролю [33].

За допомогою системи MS Visual Studio 2013 Ultimate можна створювати компоненти багаторазового використання, елементи управління ActiveX та додатки для Інтернету в рамках тісно інтегрованої візуального середовища розробки.

MS Visual Studio 2013 містить розширену IDE з інноваційними функціями для підвищення продуктивності розробника, додаткові інструменти з підтримкою розробки додатків для Windows 8.1, веб-розробки, а також поліпшення засобів налагодження та оптимізації виконуваного і керованого коду.

MS Visual Studio 2013 також надає інноваційні засоби взаємодії розробників, розширені можливості ALM і різні нововведення для гнучкого управління портфелем проектів, забезпечення якості та DevOps.

Переваги даної IDE [34]:

– скорочення часу розробки;

– новий рівень продуктивності досягається завдяки новим можливостям, значно скорочують час розробки;

– інтеграція з більшістю популярних бібліотек, таких як: Intel Parallel Studio, Qt Creator, PVS-Studio тощо;

– рекордна швидкодія.

Для візуального програмування на C++ було QtCreator – IDE призначене для створення крос-платформових застосунків з використанням бібліотеки Qt.

Qt – крос-платформовий інструментарій розробки програмного забезпечення мовою програмування C++ [35]. Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем, просто компілюючи текст програми для кожної операційної системи без зміни сирцевого коду. Містить всі основні класи, які можуть бути потрібні для розробки прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею, базами даних тощо. Бібліотека дозволяє керувати потоками, працювати з мережею та забезпечує крос-платформовий доступ до файлів.

Відмінна особливість Qt від інших бібліотек – використання Meta Object Compiller (MOC) – попередньої системи обробки початкового коду (загалом, Qt — це бібліотека не для чистого C++, а для його особливого діалекту, з якого й «перекладає» MOC для подальшої компіляції будь-яким стандартним C++ компілятором)[36]. MOC дозволяє в багато разів збільшити потужність

бібліотек, вводячи такі поняття, як слоти (slots) і сигнали (signals). Qt комплектується графічним середовищем розробки графічного інтерфейсу QTDesigner, що дозволяює створювати діалоги і форми «мишею». Ідеологія створення форм у Qt базується на використанні менеджерів розташування, котрі надають «гумовий» дизайн, при якому розмір і розташування елементів форм визначаються автоматично, що значно прискорює розробку графічного інтерфейсу. В поставці Qt є «Qt Linguist» – могутня графічна утиліта, що дозволяє спростити локалізацію й переклад ПП багатьма мовами, та «Qt Assistant» – довідкова система Qt, що спрощує роботу з документацією для бібліотек і дозволяє створювати крос-платформову довідку для ПЗ, розробленого на основі Qt.

Intel Parallel Studio – пакет розробки розпаралеленого програмного забезпечення, розроблений фірмою Intel, що поєднує в собі провідні в індустрії C/C++ компілятор і Fortran компілятор, різні бібліотеки, інструменти профілювання і багато іншого [37]. У комплект поставки входять програмні продукти:

а) Intel Parallel Advisor;

б) Intel Parallel Composer;

в) Intel Parallel Inspector;

г) Intel VTune Amplifier;

д) Intel MKL.

Intel Parallel Advisor – це допоміжний інструмент для розпаралелювання програм, призначений для розробників на мовах C, C ++, C # і Fortran. Цей засіб виявляє області коду з найбільшим потенціалом для реалізації паралельних обчислень і виявляє основні проблеми синхронізації [38]. Intel Parallel Advisor дозволяє:

- оцінити різні варіанти перед реалізацією в проекті;
- виконати попередню оцінку прискорення розпаралеленої програми;
- визначити проблеми коректності виконання коду;
- виконати вибір варіантів з найбільшою окупністю.

Intel Parallel Composer – великий набір компіляторів і бібліотек C++ і Fortran для Microsoft Visual Studio, що дозволяють спростити і прискорити розробку програм, що використовують багатоядерні процесори. Composer інтегрується в Visual Studio разом з бібліотекою продуктивності IPP і паралельної бібліотекою TBB, що значно полегшує процес розробки паралельного коду для новачків[39]. Наявність відразу декількох компонент в пакеті дозволяє відразу ж почати оптимізувати ПП з використанням паралельних технологій, які містить Composer:

– обчислювальні примітиви, реалізовані у вигляді функцій в бібліотеці IPP, гарантують високу продуктивність алгоритмів на платформах Intel;

– підтримка OpenMP 3.0, яка дозволяє використовувати multitasking;

– новий тип даних Valarray спрощує код, який реалізує векторні операції, компілятор при цьому генерує ефективний бінарний код, використовуючи SIMD-інструкції для збільшення продуктивності;

– підтримка компілятором елементів стандарту C++.

Intel VTune Amplifier – це потужний засіб для оптимізації продуктивності і профілювання паралельних додатків. Цей модуль надає наступні можливості:

– профілювання додатків написаних на C, C ++, C #, Fortran, Assembly;

– отримання докладних даних продуктивності гарячих ділянок коду, потоків, блокувань синхронізації і затримок, пропускної здатності;

– сортування, фільтрація і наочне представлення результатів на часовій шкалі та в вихідному коді;

– використання командного рядка для автоматизації регресійного тестування і віддаленого збору даних.

Intel Parallel Inspector – аналізатор коректності з можливістю перевірки роботи з пам'яттю і потоками [40]. Має окремий призначений для користувача інтерфейс, а також вбудовується в Microsoft Visual Studio. Parallel Inspector надає можливість покращити, захищеність і точність додатків, написаних на мовах C/ C ++ і Fortran:

а) надійність: Пошук взаємних блокувань (deadlocks) і помилок роботи з пам'яттю, що призводять до збоїв в роботі програми;

б) захищеність: Пошук вразливостей в використанні пам'яті і потоків, якими можуть скористатися хакери;

в) точність: Виявлення пошкодження пам'яті і стану гонки для усунення помилкових результатів.

Перевірка пам'яті включає в себе перевірку витоку пам'яті, повисли покажчики, змінні без ініціалізації, використання некоректних посилань на ділянки пам'яті, неспівпадаючі розміри пам'яті з обчисленими раніше, виділення і звільнення пам'яті, перевірки стеків в пам'яті, а також дослідження стеків з керованою глибиною.

Перевірки потоків включають в себе перевірки станів гонки, взаємних блокувань, аналіз стека викликів з настроюваної глибиною, керівництво з діагностики, вбудована підтримка Intel Threading Building Blocks, OpenMP і потоків Windows.

Intel MKL – це бібліотека оптимізованих математичних процедур для наукових, технічних та фінансових застосувань. Основні математичні функції включають BLAS, LAPACK, ScaLAPACK, швидкі перетворення Фур'є та векторну математику [41]. Підпрограми в MKL оптимізовані вручну спеціально для процесорів Intel. Бібліотека підтримує процесори Intel та доступна для операційних систем Windows, Linux та macOS.

2 СТВОРЕННЯ ПРОГРАМНОГО СЕРЕДОВИЩА

2.1 Обчислювальна система

Для проведення обчислювальних експериментів по дослідженню ефективності роботи програм, що застосовують підпрограми ScaLAPCK, використовувався обчислювальний кластер на базі лабораторії паралельних обчислень математичного факультету ЗНУ. При проведені експериментів до кластеру включалось до 4 однорідних вузлів з процесорами Intel Core i3, об'ємом оперативної пам'яті 2 Гб та дисковим накопичувачем об'ємом 160 Гбайт. В якості комунікаційного середовища кластер використовує мережу Gigabit Ethernet. Кожний вузол кластеру виступає в ролі робочого місця і може застосовуватись, як для запуску задач в складі кластеру, так і для виконання окремих задач по створенню програмного забезпечення на робочому місці – набір і збереження коду програм, їх компіляція та відлагодження, перевірка роботи на окремій станції та на декількох станціях кластеру [13-15].

Системною платформою для побудови кластеру обрано ОС Linux Mint тому, що вона вже має в своєму складі пакети комунікаційної бібліотеки MPI – OpenMPI та MPICH. Інсталяція цих пакетів може бути виконана із deb-файлів, взятих або з інсталяційного диску, або через мережу Інтернет з репозиторію відповідної версії. Для виконання як запуску паралельних програм, так і їх створення на вузлах кластеру встановлено компілятори C та Fortran, відповідно, gcc та gfortran.

2.2 Встановлення бібліотеки ScaLAPACK

Складові пакети бібліотеки ScaLAPACK, як правило, на інсталяційних дисках Linux відсутні, тому їх інсталяцію rpm-файлами можна виконати тільки через відповідний репозіторій [8-9]. Крім прямих пакетів з іменами типу blas,

lapack, blacs та scalapack також слід встановити пакети blas-dev, lapack-dev, blacs-dev та scalapack-dev, що дозволить створювати нові програми з використанням цих складових пакетів. Крім того, файли бібліотек є чутливими до встановленого пакету підтримки MPI і, як правило, мають дві версії, кожна з яких встановлюється до відповідної директорії.

Найбільш загальна інсталяція ScaLAPACK виконується за допомогою пакету scalapack_installer, який розміщено на ftp.netlib.org. При такій інсталяції створюються бібліотеки всіх складових пакету ScaLAPACK на основі тих компонент операційної системи, які вже встановлено на вузлах кластеру. Тому створювані файли є найбільш сумісними з існуючою системою. Якщо в системі буде відбуватись заміна версії MPI, наприклад, з OpenMPI на MPICH, то це буде потребувати перекомпіляції і бібліотек ScaLAPACK.

Для роботи інсталяційного пакету scalapack_installer необхідно підключення до мережі Інтернет. В більшості випадків вузли обчислювального кластеру знаходяться в локальній мережі і можуть мати доступ до мережі Інтернет або через застосування технології NAT, або через proxy-сервер. В першому випадку підключення подібно до прямого підключення, в другому інсталятору необхідно вказати параметри proxy-серверу. Параметри proxy-сервера передаються через змінну оточення http_proxy наступним чином:

export http_proxy=http://192.168.1.254:3128.

Бібліотеки ScaLAPACK можуть створюватись як для загального користування усіма користувачами обчислювальної системи, так і для індивідуального окремим користувачем. В першому випадку інсталяція виконується під правами системного адміністратора root, в другому – самим користувачем. Файл scalapack_installer_1.0.3.tgz розпаковується в каталог відповідного користувача і в створеному каталозі запускається файл setup.py. Як правило, запуск потребує уточнення деяким набором ключів. Якщо пакети blas, blacs та lapack не було отримано та скомпільовано заздалегідь, то вказуються ключі: - - downblas, - - downblacs, - - downlapack. Якщо якийсь з цих пакетів вже було скомпільовано і файл відповідної бібліотеки вже отримано. Також важливим є правильна вказівка на файли компіляторів MPI коли вони

викликаються тільки з повним шляхом або якщо в системі використовуються декілька версій MPI і для кожної необхідно створити бібліотечні файли ScaLAPACK. Для цього застосовуються ключі: - - mpicc та - - mpif77. Якщо пакети MPI для файлів заголовків використовує свій каталог, наприклад, /usr/include/openmpi-i386, то про це необхідно вказати наступним чином: - - mpiincdir=/usr/include/openmpi-i386.

Зазвичай setup.py пакету scalapack_installer організовує копіювання файлів бібліотек blas, blacs та lapack з web-ресурсу http://netlib.org. Після копіювання кожного із пакетів виконується їх компіляція на основі makefile, що включено до них. Якщо усі пакети було скомпільовано без помилок scalapack_installer запустить їх тестування. Якщо було виявлено помилку компіляції якоїсь бібліотеки, то інсталяція перерветься одразу ж після її з'явлення. В цьому випадку відповідну бібліотеку можна спробувати скомпілювати самостійно. Відповідний пакет вже отримано та розпаковано в каталозі scalapack_installer в підкаталозі /build. Необхідно виконати всі конфігурації цього пакету та задати команду make. Якщо бібліотечний файл буде створено, то слід повторити запуск setup.py, але вже з відкоригованими ключами на отриманий файл бібліотеки.

Після вдалої компіляції усіх складових ScaLAPACK та успішного їх тестування інсталятор повідомляє по імена створених файлів бібліотек і місці їх розташування. Якщо інсталяція відбувалась з правами адміністратора і подальше використання файлів бібліотек припускається для усіх користувачів кластеру, то найкращим є копіювання отриманих файлів в загальний бібліотечний каталог /usr/lib, до якого вже відкрито шляхи усім користувачам. В іншому випадку при компіляції буде необхідним задавання шляху до бібліотек за допомогою ключа –L, наприклад, –L /home/user1/lib.

Файли бібліотек, що створено, поділяються на ті, ім'я яких починається з префіксу "lib" та ті, що його не мають. Якщо файли перенесено до каталогу /usr/lib, то вони мають починатись з префіксу "lib" при використані ключа –l. Наприклад, якщо в командному рядку компілятору задати –lscalapack, то компілятор буде сприймати це, як необхідність застосувати бібліотеку з ім'ям

libscalapack.a. Тому на файли без префіксу “lib” можна зробити символьне посилання з таким префіксом.

Командний рядок для компіляції паралельної програми, в якій застосовано бібліотеки ScaLAPACK має вигляд:

mpiF77 –o test test.f –lblacsF77 –lblacs –lblacsC –lrefblas –lreflapacs –lscalapack

або

mpicc –o test test.c –lscalapack –lblacsC –lblacsF77 –lblacs–lrefblas –lreflapacs

При прямому посиланні на файли бібліотеки командний рядок компіляції має дещо інший виглід:

mpiF77 –o test test.f libblacsF77.a libblacs.a libblacsC.a librefblas.a libreflapacs.a libscalapack.a

або

mpicc –o test test.c libblacsC.a libblacsF77.a libblacs.a librefblas.a libreflapacs.a libscalapack.a

Запуск завдання на кластері також виконувалось з командного рядку із зазначенням MCA-параметрів для комп’ютерної мережі, кількістю застосованих процесорів та іменами вузлів, на яких воно буде виконуватись:

mpirun – -mca btl_tcp_if_include eth0 –np 4 –H n0,n1 test

Перед запуском завдання файл задачі копіювався в одні і ті ж самі каталоги усіх вузлів, що буде застосовано при її виконанні. Запуск здійснювався в каталозі її розміщення на головному (n0) вузлі, тому повний шлях до цього файлу можна було не вказувати.Також можна створити імітацію кластера на на станціонарному компьютері, що буде відтворювати роботу наближену до роботи на справжнему кластері.

2.3 Постановка задачі для дослідження матричних операцій

Матричні обчислення є складовою частиною багатьох наукових та інженерних розрахунків. Вони часто пов’язані з рішенням систем рівнянь частинних похідних (задачі розрахунку на міцність, перенос тепла, квантово-хімічні розрахунки тощо). З врахуванням значності ефективного виконання обчислень з матрицями більшість стандартних бібліотек програм вміщують процедури для різноманітних матричних операцій. Обчислення, пов’язані з матрицями є достатньо трудомісткими, тому представляють собою основну задачу для паралельних систем.

При послідовному програмуванні матричного множення застосовуються оператори циклу „do” на мові Fortran [22] та „for” – на мові Ci[20,21]. Такий підхід є типовим для однопроцесорних систем, але він не дозволяє достигнути максимальної продуктивності при обчисленнях на багатоядерних процесорах та системах з багатьма процесорами.

Нехай A,B,C - квадратні матриці розміром $n \times n$. Матриця C є додатком матриць A та B. Компоненти матриці C розраховуються за формулою:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \tag{2.1}$$

де i, j – індекси, що змінюються від $1, \ldots, n$.

Для обчислення одного елемента матриці C необхідно виконати n множень та $n-1$ додавань. Загальна кількість операцій з плаваючою точкою для усіх елементів матриці буде визначатись формулою

$$op = n^3 + (n-1)n^2 = (2n-1)n^2. \tag{2.2}$$

Оцінку продуктивності комп’ютера можна визначити в кількості, що приходиться на один мільйон операцій з плаваючою точкою за одну секунду - 1 Mflops. Для цього кількість операцій необхідно поділити на час їх виконання, який представлено в мікросекундах. Таким чином, для задачі перемноження квадратних матриць отримуємо продуктивність:

$$prod = \frac{(2n-1)n^2}{1000000 \cdot t}, \tag{2.3}$$

де t – час виконання n операцій в секундах.

Бібліотека ScaLAPACK потребує, щоб усі об’єкти (вектори та матриці), які є параметрами підпрограм, було попередньо розподілено по процесорам. Для виконання цього необхідно, щоб глобальні матриці A,B,C було представлено у вигляді окремих блоків. Ці блоки необхідно за блочно-циклічним засобом розподілити по процесорам або по ядрам процесорів.

Так множення матриць має вигляд:

$$C = A \times B. \tag{2.4}$$

З точки зору блочно-циклічного розподілу це множення можна представити матричними блоками, а саме

$$\begin{bmatrix} C_{11} & \cdots & C_{1N} \\ \cdots & \cdots & \cdots \\ C_{N1} & \cdots & C_{NN} \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \cdots & \cdots & \cdots \\ A_{N1} & \cdots & A_{NN} \end{bmatrix} \times \begin{bmatrix} B_{11} & \cdots & B_{1N} \\ \cdots & \cdots & \cdots \\ B_{N1} & \cdots & B_{NN} \end{bmatrix}. \tag{2.5}$$

Таким чином формула множення матриць буде мати вигляд:

$$C_{ij}=\sum_{k=1}^{N} A_{ik}B_{kj}, \tag{2.6}$$

де A_{ik}, B_{kj}, C_{ij} – блочні матриці,

N – кількість блоків по рядкам або стовпцям.

Розмір блочних матриць визначається наступною формулою

$$NB=\left(\frac{n}{N}\right)\times\left(\frac{n}{N}\right). \tag{2.7}$$

З формули видно, що найбільш сприятливим є такий розмір блоків, в яких відношення (n/N) є цілим числом.

Таким чином, необхідно виконати дослідження, що встановлюють залежність ефективності використання ресурсів обчислювального кластеру від розміру матриць, кількості застосованих процесорів та розміру блоків, на які розподіляються глобальні матриці по процесорах.

2.4 Постановка задачі та загальний алгоритм для дослідження рішення СЛАР

Велика кількість обчислювальних моделей будується на базі рішення системи лінійних алгебраїчних рівнянь (СЛАР), при цьому число рівнянь може досягати багатьох тисяч. Розглядається рішення великих систем лінійних алгебраїчних рівнянь, що має вигляд:

$$A\times X=B \tag{2.8}$$

де A – невироджена матриця порядку $n\times n$,

B - вектор правої частини,

X - вектор довжини n, що вміщує рішення.

Рішення системи здійснюється з використанням підпрограми PDGESV з бібліотеки ScaLAPACK. Загальний алгоритм показано у послідовності змістовних етапів паралельної програми.

Етап 1. Підключення необхідних бібліотек. Включення в їхній склад спеціалізованої бібліотеки mpi.h.

Етап 2. Опис прототипів використовуваних функцій.

Етап 3. Оголошення змінних.

Етап 4. Ініціалізація MPI процесу.

Етап 5. Розбір параметрів, прийнятих з командного рядка. У програму передаються імена вхідних файлів, що містять значення матриці A и вектора правої частини B, розмірність блоку NB, на які буде розбиватися глобальна матриця та ім'я вихідного файлу, до якого буде записано рішення - вектор X.

Етап 6. Обчислення формату сітки процесорів, найбільш наближеного до квадратного виду.

Етап 7. Формування робочої сітки процесорів.

Етап 8. Зчитування з файлу, що містить значення матриці та розсилання її по всіх процесах.

Етап 9. Корегування розмірності блоку. Якщо розмірність блоку виявилася більше розмірності матриці, то взяти розмірність блоку рівної розмірності матриці.

Етап 10. Визначення точного числа рядків і стовпців розподіленої матриці в кожному процесі.

Етап 11. Виділення пам'яті під матриці. У випадку неможливості виділення виконується аварійний вихід із програми.

Етап 12. Формування дескрипторів матриць.

Етап 13. Зчитування матриць із файлів і розподіл їх по процесах.

Етап 14. Створення копії вхідної матриці A, для подальшої перевірки правильності рішення.

Етап 15. Виклик обчислювальної підпрограми PDGESV з бібліотеки ScaLAPACK.

Етап 16. Перевірка правильності рішення системи рівнянь. Добуток вихідної матриці A та вектору рішення X повинен співпасти з вектором правої частини B.

Етап 17. Запис результату у файл.

Етап 18. Звільнення пам'яті, виділеної під матриці.

Етап 19. Звільнення сітки процесорів.

Етап 20. Закриття процесу MPI.

Блок-схему програми наведено на рисунку 2.1.

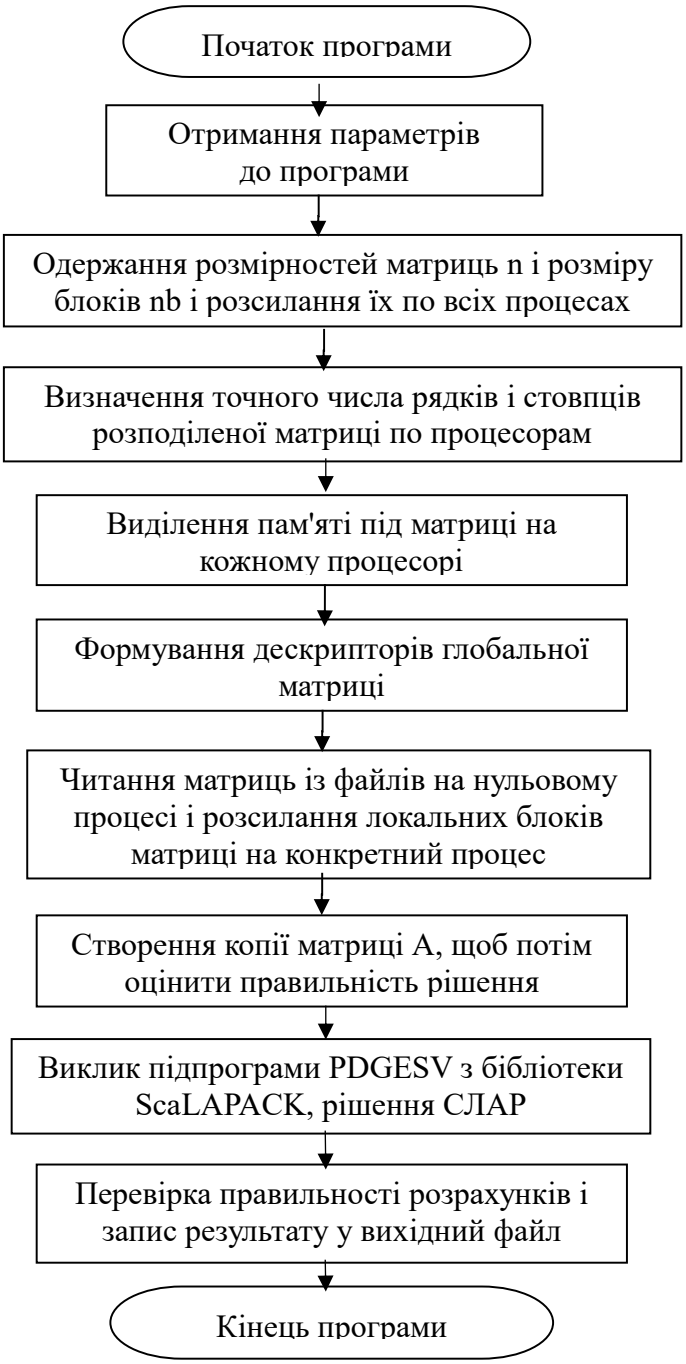


Рисунок 2.1 Блок-схема програми рішення СЛАР

Множина паралельних процесів, що використовується для рішення завдання, утворює сітку процесів (process grid), яка може бути як двовимірною, так і одномірною. Кожний окремий процес ідентифікується парою цілих чисел, що визначають його положення в цій сітці процесів (номер рядка, номер стовпця). Нумерація починається з номера 0.

Ініціалізація сітки процесорів виконується за допомогою підпрограм з бібліотеки BLACS.

Cblacs_pinfo(&iam, &nprocs) - ініціалізує бібліотеку BLACS, встановлює деякий стандартний контекст для ансамблю процесорів, повідомляє процесору його номер в ансамблі (iam) і кількість доступних завданню процесорів (nprocs).

Cblacs_get(-1, 0, &ictxt) - виконує опитування встановленого контексту (ictxt) для використання в інших підпрограмах.

Cblacs_gridinit(&ictxt, "Row", nprow, npcol) – виконує ініціалізацію сітки процесорів розміру nprow×npcol з нумерацією уздовж рядка.

Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol) – повідомляє процесору його позицію (myrow, mycol) у сітці процесорів.

Тому що кожний процес робить операції над своєю частиною вихідної (глобальної) матриці, то до звертання до обчислювальних підпрограм бібліотеки ScaLAPACK необхідно розподілити матрицю системи рівнянь *A* і вектор правих частин *B* за сіткою процесів. Точне значення кількості рядків і стовпців для кожного із процесорів обчислюється за допомогою підпрограми-функції *numroc_* бібліотеки PTOOLS:

np = numroc_(&n , &nb, &myrow, &izero, &nprow),

nq = numroc_(&n , &nb, &mycol, &izero, &npcol),

де np і nq - число рядків і стовпців у локальних подматрицах у рядку (стовпці) процесорів myrow (mycol); n - число рядків і стовпців вихідної матриці; nb - розмірність блоків; (myrow, mycol) - координати процесора в сітці процесорів; (izero, izero) - координати процесора, починаючи з якого виконано розподіл глобальної матриці; nprow і npcol - число рядків і стовпців в сітці процесорів.

Кожній глобальній матриці, що розподіляється по сітці процесорів, приписується дескриптор, який містить інформацію про те, як саме було зроблено її розподіл. Дескриптор може бути заповнено або безпосередньо за допомогою операторів присвоювання, або за допомогою спеціальної підпрограми бібліотеки PTOOLS:

descinit_(desc, &m, &n, &nb, &nb, &izero, &izero, &ictxt, &itemp, &info),

Якщо параметр info=0, то підпрограма виконалася успішно, а якщо info=-I, то I-й параметр некоректний. Призначення полів дескриптора й опис їх значень наведено в таблиці 1.2 з тією різницею, що в Сі-версії підпрограми масив дескриптору починається з нульового індексу.

Звертання до підпрограми pdgesv бібліотеки ScaLAPACK має вигляд:

pdgesv_(&n, &nrhs, A, &ia, &ja, desc, ippiv, B, &ib, &jb, desc, &info),

де n – розмірність вихідної матриці A; nrhs – кількість правих частин у системі, тобто кількість стовпців у матриці B; A – на вході представляє локальну частину розподіленої матриці A, а на виході - локальну частину LU розкладання; ia та ja – індекси лівого верхнього елемента підматриці матриці A, для якої знаходиться рішення; desc – дескриптор матриці A; ippiv – робочий масив цілого типу, що на виході містить інформацію про перестановки в процесі факторизации; B – на вході представляє локальну частину розподіленої матриці B, а на виході - локальну частину отриманого рішення; ib та jb – подібні до ia та ja, тільки для матриці B; desc – дескриптор матриці B; info – ціла змінна, яка вказує на успішність завершення підпрограми або про причину її аварії.

Підпрограма pdgesv_ викликає підпрограми нижнього рівня, а саме: PDGETRF та PDGETRS. Перша зних є LU - розкладанням матриці загального вигляду методом Гаусса з вибором провідного елемента по стовпцям. Друга – це рішення системи $AX = B$, $A^T X = B$ або $A^H X = B$ на основі LU - розкладання, яке отримується з підпрограми PDGETRF.

Звільнення сітки процесорів відбувається наприкінці програми за допомогою підпрограми ScaLAPACK Cblacs_gridexit(0). Виклик цієї підпрограми приводить до закриття контексту і всіх BLACS-процесів.

Крім підпрограм ScaLAPACK для таких операцій як пересилання даних, синхронізації процесів, виконання глобальних операцій по всьому ансамблі процесорів або для його частини застосовувались процедури бібліотеки MPI [6-18].

2.5 Реалізація обчислень із використанням бібліотеки ScaLAPACK

Варіанти розрахункових програм множення матриць представлено на мовах C та Fortran в Додатку А. Обидві програми застосовують підпрограму pdgemm бібліотеки ScaLAPACK. Оскільки поставленою задачею є дослідження ефективності використання підпрограми, а не розрахунок реальних завдань, то перед застосуванням pdgemm формуються матриці A та B з довільними значеннями за допомогою генератора випадкових чисел.

Обчислювальну підпрограму pdgemm створено за типом SMPD, що означає - одна програма для множини даних. Один і той самий файл програми застосовується для створення усіх паралельних процесів, які замовляються користувачем через командний рядок при запуску на розрахунок завдання. У той же час, кожний процес займається обробкою тільки своєї частини даних, що є складовою від деякої загальної (глобальної) частини даного завдання. Ці дані окремого процесу є локальними даними і безпосередньо недоступні іншим процесам. Вони зберігаються в окремій локальній пам'яті, що виділяється для кожного процесу на вузлах кластеру, які застосовуються у розрахунках. Необхідний обмін даними між різними процесами виконується тільки за допомогою передачі повідомлень за технологією MPI.

Для визначення часу, що витрачається на роботу підпрограми pdgemm, застосовано функцію MPI_Wtime() комунікаційної бібліотеки MPI. Спочатку час визначається безпосередньо перед викликом підпрограми, а потім – після повернення з неї. Різниця визначених величин є часом роботи підпрограми, до якого включається як передавання вихідних матриць через мережу кожному вузлу, що виникає при виконанні завдання, так і безпосереднє виконання частини

завдання на кожному із процесорів, а також повернення завдання головному процесу.

Варіанти розрахункових програм, що призначено для рішення системи лінійних алгебраїчних рівнянь (СЛАР) представлено на мовах С та Fortran в додатку Б. В основі програми на мові Fortran лежить алгоритм, в якому спочатку за допомогою підпрограми pdgetrf виконується факторизація матриці А, а після цього вона застосовується в підпрограмі pdgetrs разом з матрицею правої частини В для отримання рішення системи рівнянь. Матриця В на вході підпрограми pdgetrs вміщує праві частини рівнянь, а на виході з підпрограми – значення корнів. В програмі на мові С було застосовано іншу підпрограму бібліотеки ScaLAPACK – pdgesv_. Її призначено для рішення СЛАР методом Гауса для матриць загального типу і попередніх перетворювань матриць не потребує. Як і у випадку з підпрограмою pdgemm усі застосовані підпрограми створено за типом SMPD.

Як і у вмпадку з програмою множення матриць після компіляції отримана копія програми завантажується на всі вузли кластера, що будуть застосовуватись для обчислень, і використовується для породження необхідної для користувача кількості паралельних процесів, які замовляються черех командний рядок.

У ході підготовки робочої копії програми було встановлено суттєву залежність між розміром матриць, що оголошуються в програмі та часом компіляції програми. Дійсно, розпаралелювання програм та застосування кластерів має сенс якщо задача, що вирішується, занадто велика для одного процесору або одного комп’ютера. На відміну від розрахункової паралельної програми компіляція може відбуватись тільки за допомогою одного процесору і застосуванням пам’яті одного комп’ютера. На рисунку 2.2 наведено діаграми зміни об’єму оперативної пам’яті при компіляції програми компілятором gcc, які отримано за допомогою стандартного системного монітору операційної системи Linux.

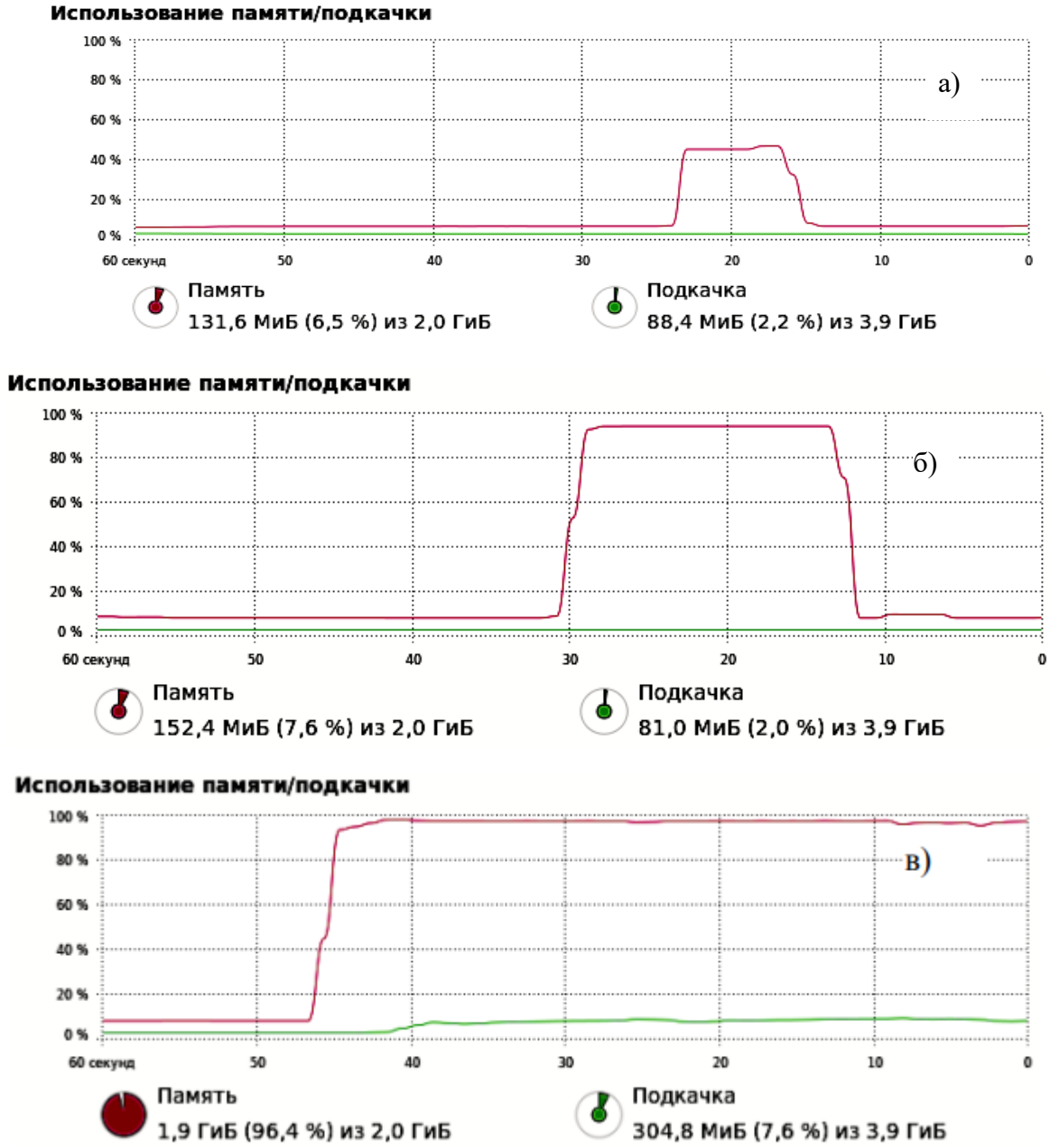


Рисунок 2.2 Зміна об’єму пам’яті, що застосовується компілятором, при збільшенні розміру матриць в програмі рішення СЛАР: а) 10000 рівнянь; б) 15000 рівнянь; в) 16000 рівнянь

З наведених діаграм також видно збільшення часу компіляції програми при збільшені розміру матриць (числа рівнянь) в системі. Так час компіляції подвоюється якщо максимальне замовлене число рівнянь збільшується з 10000 до 15000. Для використовуваної обчислювальної системи (об’єм оперативної пам’яті 2 Гбайти) критичним був ромір матриці ~16000×16000 (див.рис. 2.2 в). При такому розмірі матриць необхідний компілятору об’єм пам’яті перевищує існуючий в системі, тому починає застосовуватись swap-розділ. Останне

унеможливлює ефективне використання процесорних ядер, в наслідок чого значно збільшується час компіляції програми. Так, час компіляції програми рішення СЛАР з матрицями 15000×15000 був в межах 20 секунд, а при збільшенні розміру матриць до 16000×16000 став перевищувати 20 хвилин.

Таким чином було виявлено, що обмеження на розмір вирішуванної задачі (розмір матриць, кількість рівнянь та інш.) обумовлено як характеристиками застосованного кластеру, так і властивостями компілятора та комп’ютера, на якому виконується компіляція.

3 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ БІБЛІОТЕКИ SCALAPACK

3.1 Множення двох квадратних матриць

Складність застосування підпрограм бібліотеки ScaLAPACK полягає в тому, що є необхідність автоматичного визначення вхідних параметрів цих підпрограм. Для нерозпаралелених програм або для тих програм, що виконуються тільки на одному процесорі (одному ядрі процесора) вхідними параметрами будуть самі матриці, наприклад, A та B, а як результат роботи програми буде отримано матрицю C. При цьому, ніяких питань, щодо розподілу A та B, а також щодо збірки результату множення в матрицю C вирішувати не доводиться, тому що всі операції буде виконано на одному процесорі і він буде мати повний доступ до елементів усіх матриць в будь-який час.

Бібліотека ScaLAPACK надає підпрограми з паралельним алгоритмом, але питання щодо розподілу матриць A та B між процесорами зостається відкритим. Розробник програмного забезпечення, що намагається застосувати підпрограму pdgemm повинен не тільки задати значення елементів матриць A та B, як це відповідає загальному алгоритму множення матриць, а також задати розмір блоків та схему розбивки цих матриць по процесорах. Можна припустити, що якщо блоки будуть замалі, наприклад, їх розмір буде складатись з одного елементу, то це значно буде навантажувати комп’ютерну мережу і досягти значного підвищення продуктивності обчислювальної системи за рахунок збільшення кількості застосованих процесорів (ядер) вряд чи вдасться. З іншого боку, занадто великі блоки можуть непропорційно поділити матриці, що незбалансовано навантажить вузли кластеру і також понизить ефективність його застосування.

Першим було досліджено залежність часу обчислення множення матриць від кількості застосованих процесорів та від розміру самих матриць (див.рис. 3.1-3.3).

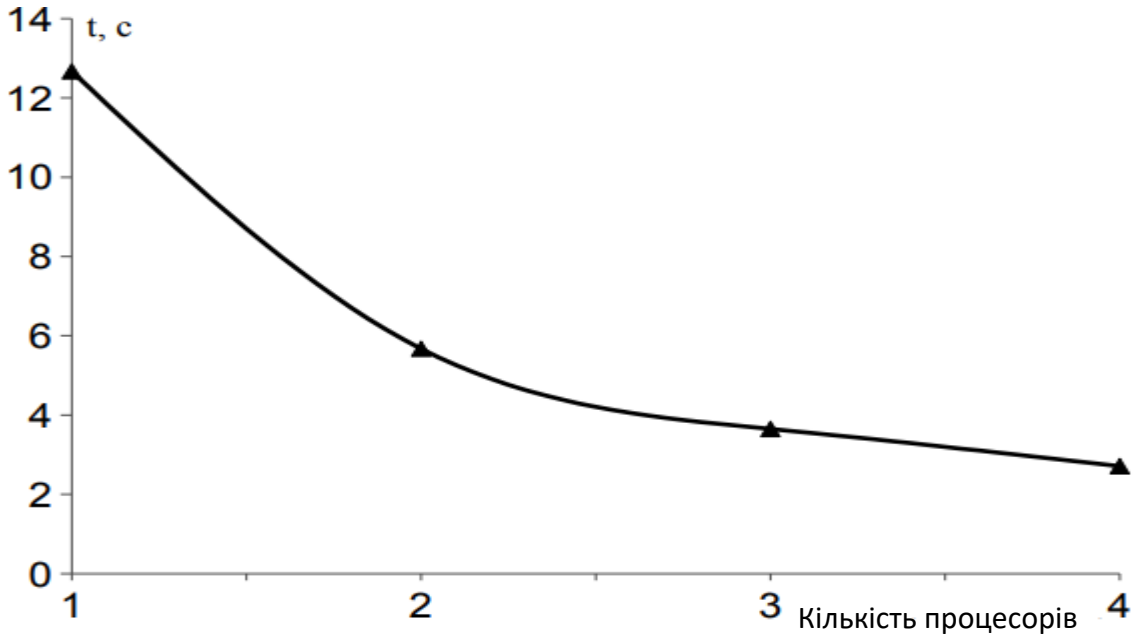


Рисунок 3.1 Залежність часу обчислення множення матриць 2000×2000 від кількості застосованих процесорів.

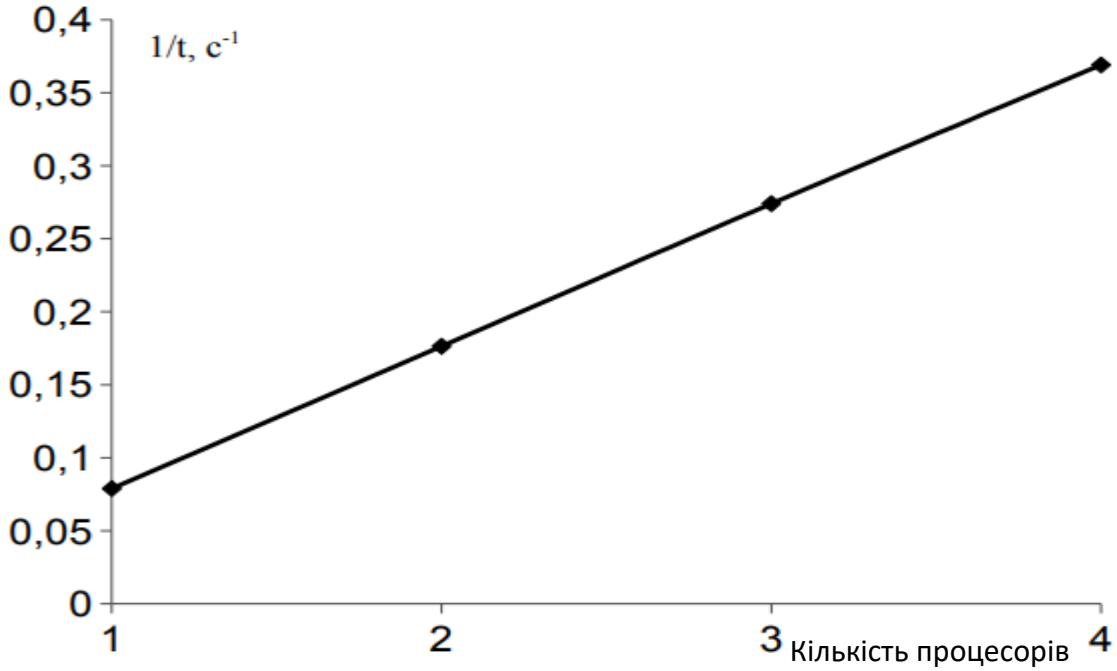


Рисунок 3.2 Обернена величина часу обчислення множення матриць 2000×2000 від кількості застосованих процесорів

Як показали проведені експерименти збільшення кількості застосованих для розрахунків процесорів зменшує час обчислень з оберненою пропорційністю до неї. Вимірювання часу було проведено для матриць 2000×2000 та блоком розбивки 50×50 при кількості процесорів, що не перевищувала 4.

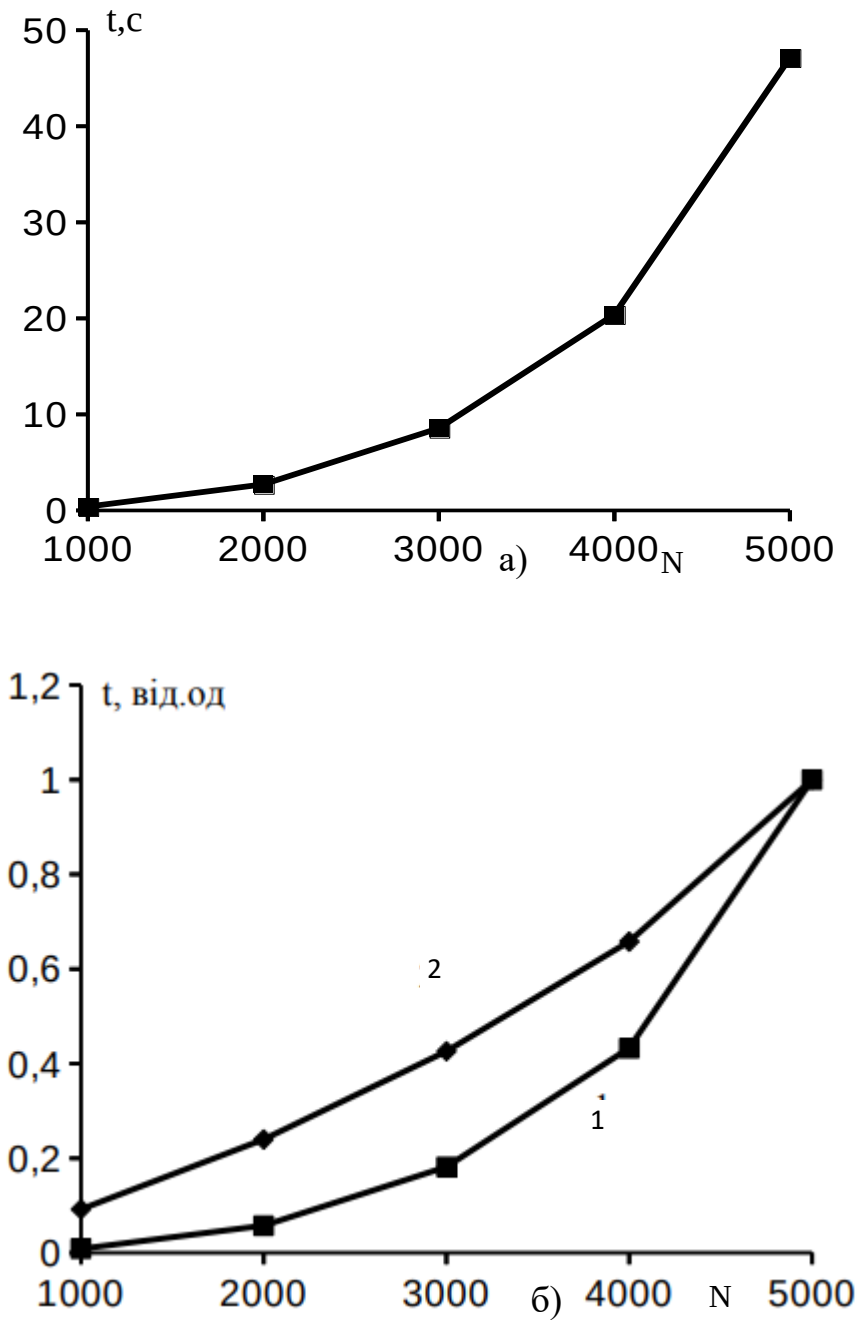


Рисунок 3.3 Часова характеристика множення матриць від їх розміру N×N: а) беспосередні значення отриманого часу; б) відносні залежності часу 1 та квадратного кореню від нього 2

Залежності часу обчислень від розміру матриць показують достатнє наближення до квадратичної форми (див.рис. 3.3а). Для з’ясування наскільки близька отримана залежність до квадратичної було також побудовано графік квадратних коренів отриманих величин (див.рис. 3.3б, крива 2) та порівняно його з первинно отриманими (див.рис. 3.3б, крива 1). З наведених графіків видно, що часова залежність множення матриць дещо відхиляється від квадратичної,

інакше крива 2 була б представлена прямою лінією. Це може бути пояснено особливістю обчислювальної системи та виконуваної програми.

По-перше, систему побудовано на звичайній розповсюдженій мережі, а саме за технологією Ethernet, а всі обміни даними відбуваються за допомогою протоколів TCP/IP. Це безумовно повинно додавати часу на обмін даними в мережі, який буде зростати пропорційно зростанню кількості переданих даних. В досліджуваному випадку об’єм переданих даних росте пропорційно N×N. З іншого боку, слід встановити як програми виконують обмін даними. На рис.3.4 наведено діаграми змін у навантаженні процесорів та завантаженості комп’ютерної мережі при виконанні програми множення матриць 5000×5000 за блоками 50×50, що виконувалась на двох вузлах і загальною кількістю ядер процесорів 4.

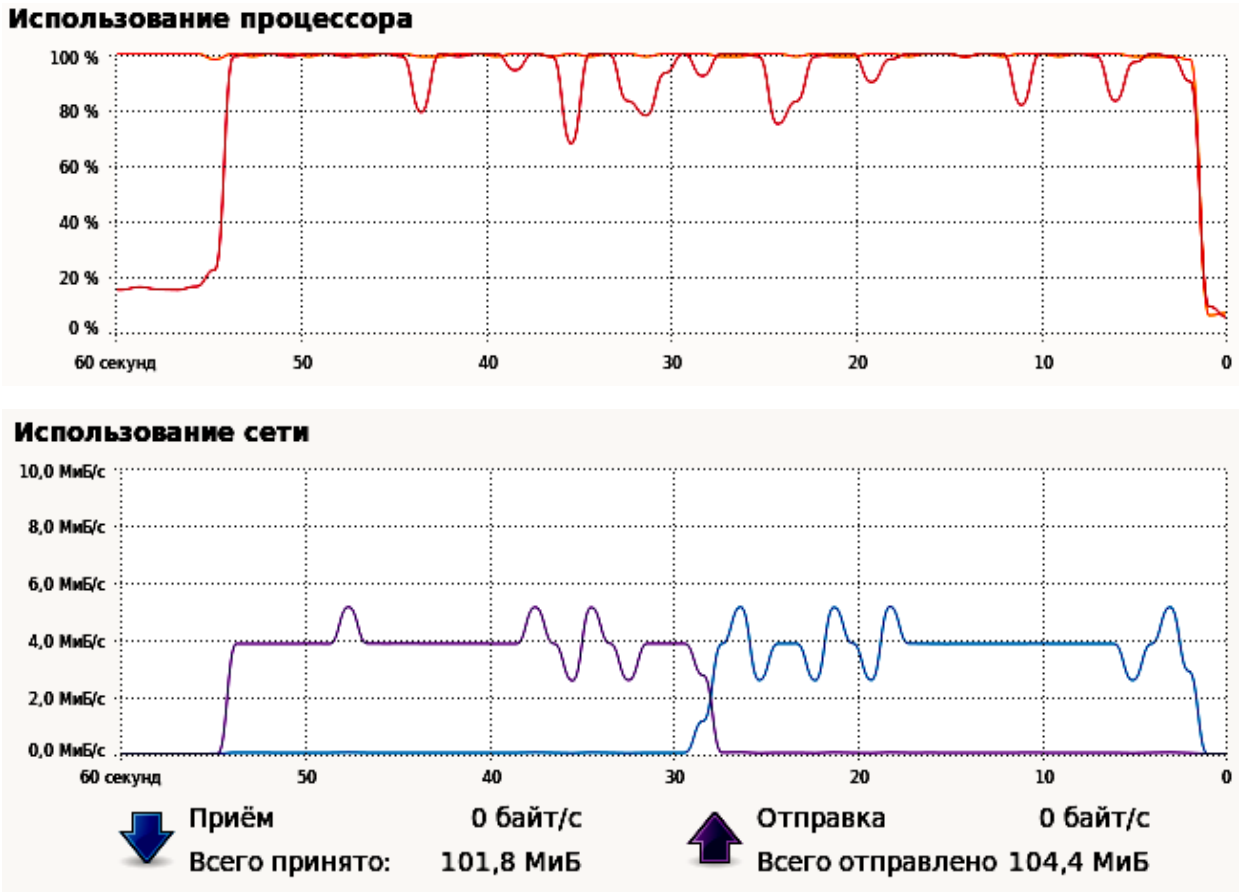


Рисунок 3.4 Діаграми навантаженості процесорів та мережі при виконанні обчислень множення матриць 5000×5000 за блоками 50×50 на двох вузлах і кількістю ядер процесорів 4. Отримано за допомогою стандартного системного монітору операційної системи Linux

З наведених графіків видно, що процес обміну даним відбувається протягом усього часу проведення обчислень, а завантаження мережі приблизно на рівні 4.0 Мбайт/с. Очевидно, що така поведінка програми також дещо знижує продуктивність обчислень і, як слід, збільшує час обчислень, хоча відносно загального часу обчислень це не суттєва величина.

На рисунках 3.5-3.7 наведено залежності часу обчислень множення матриць від розміру блоку, за яким вони розподіляються по процесорах.

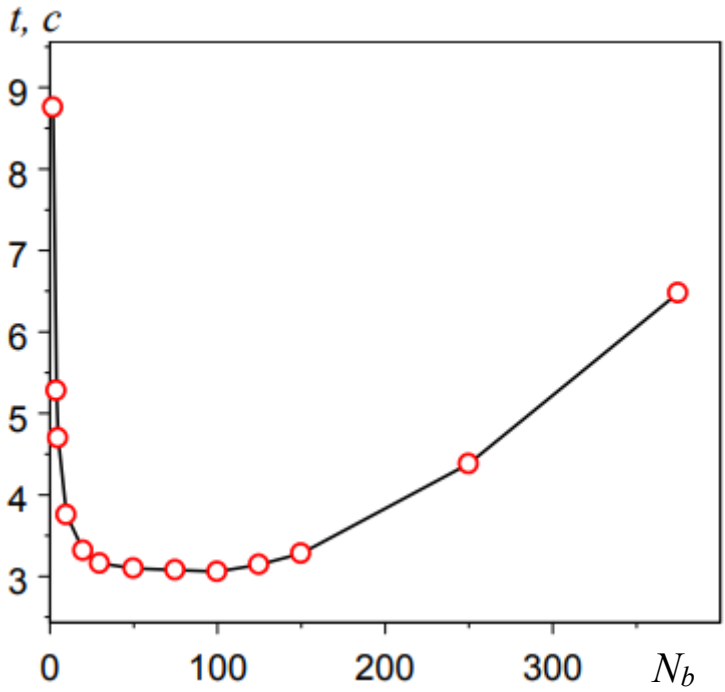


Рисунок 3.5 Залежність часу обчислення множення матриць 3000x3000 на 4 процесорах від розміру блоків

Отримані результати демонструють найбільшу продуктивність системи при застосуванні блоків в межах від 50×50 до 100×100. Як і припускалось занадто малі блоки будуть робить обчислювальну систему малоефективною.

Із результатів проведених обчислювальних експериментів видно, що занадто малі блоки ведуть до збільшення часу обчислень в 3-5 разів у залежності від розміру матриць. Також уповільнення розрахунків відбувається при збільшені розміру блоків відносно оптимального, але це уповільнення більш пологе.

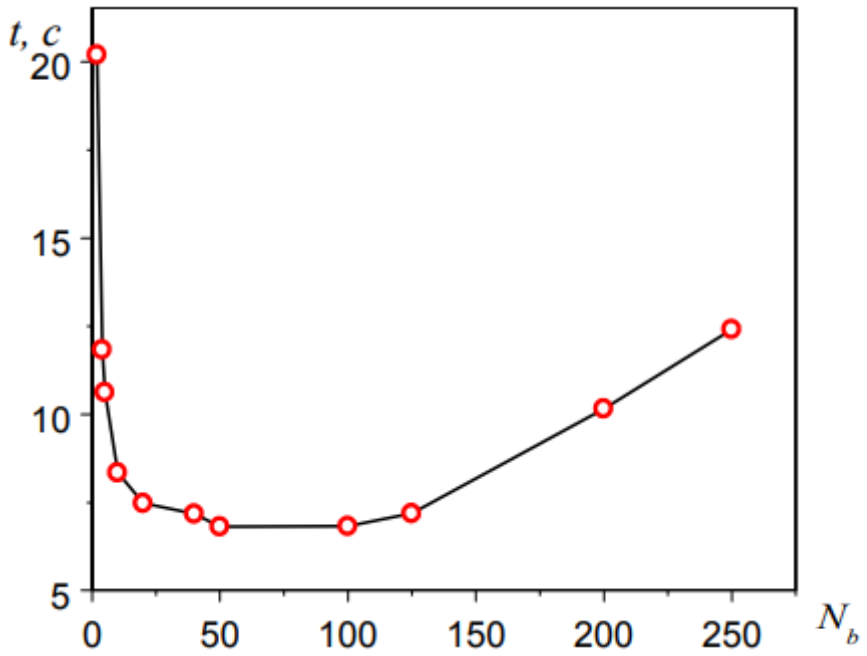
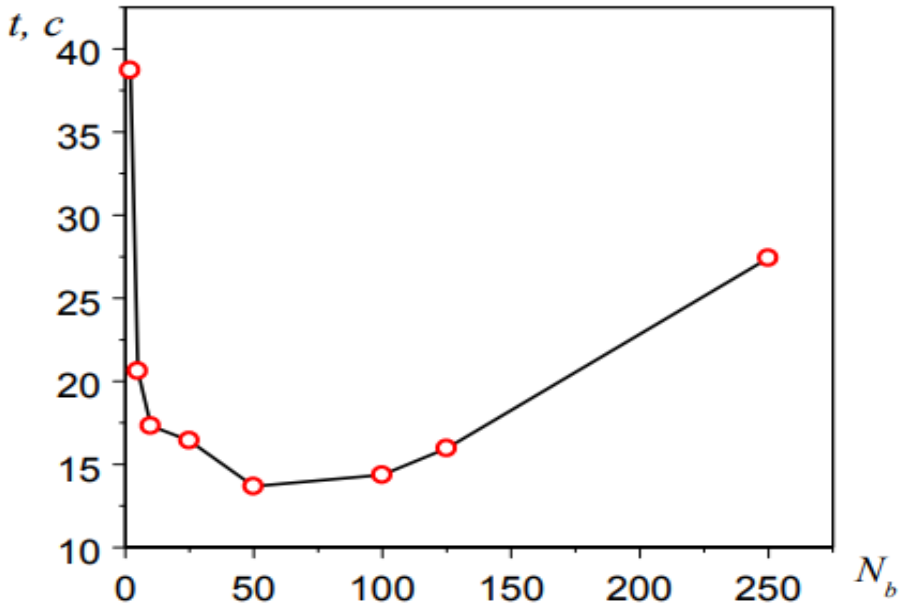


Рисунок 3.6 Залежність часу обчислення множення матриць 4000x4000 на 4 процесорах від розміру блоків



Рисинук 3.7 Залежність часу обчислення множення матриць 5000x5000 на 4 процесорах від розміру блоків N_b

Тому можна зробити висновок, що для ефективного застосування кластеру краще застосовувати блоки з розміром в межах від 50×50 до 100×100, при цьому обраний розмір блоку повинен відповідати цілому числу вкладань у вхідні матриці.

3.2 Рішення системи лінійних алгебраїчних рівнянь

Дослідження рішення СЛАР також починались з визначення залежності часу обчислень від збільшення числа процесорів. Час, який вимірявся, є сумарною величиною як арифметичних операцій, так і пересилок в меріжі, що відбуваються під час обчислень. Тому одним із факторів, що впливає на загальну швидкість обчислень є розсилка матриць між процесами, отже і латентність мережі. Але при оптимальному розподілі матриці між процесами можна досягти зниження числа запитів на передачу даних між вузлами, що дозволить знизити час на пересілку даних і тим самим знизити загальний час виконання завдання.

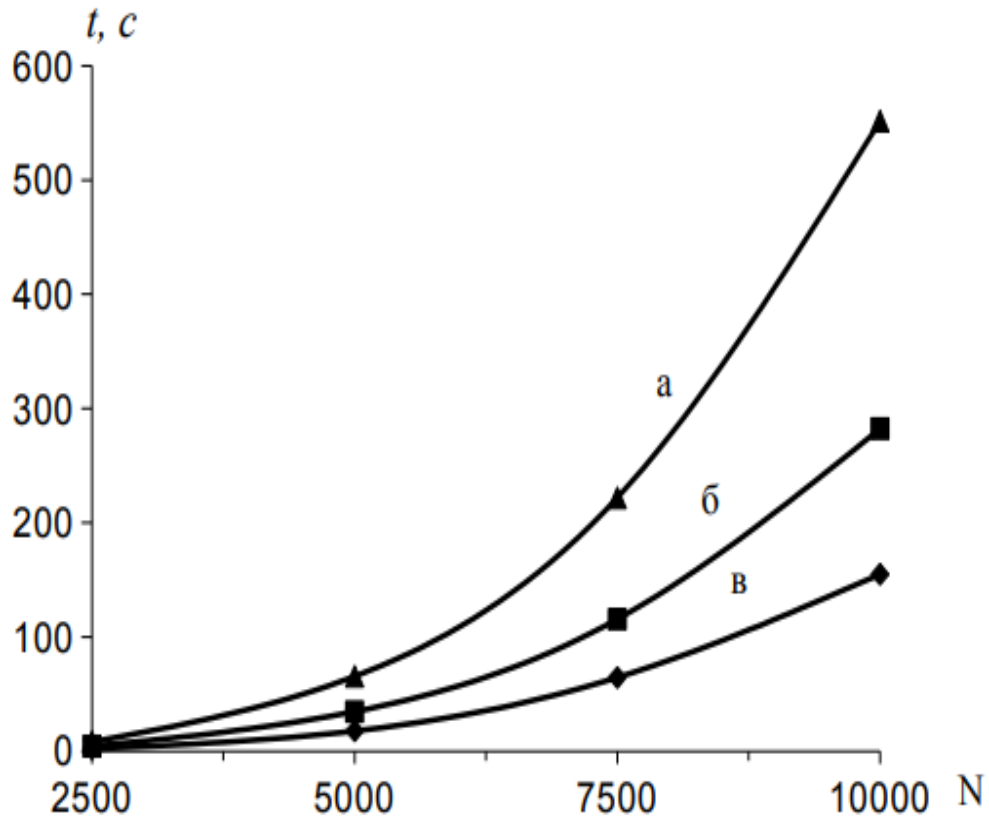
Час рішення завдання визначався при поступовому збільшені розмірності системи рівнянь. Було розглянуто наступну послідовність размірності СЛАР: 2500×2500, 5000×5000, 7500×7500, 10000×10000. Запуск завдань виконувався послідовно з різним числом процесорів, а розмір блоку розбивки становив 50×50. Отримані результати часу, що вимірюється в секундах, представлено в таблиці 3.1, в якій кожний стовпець відповідає розміру СЛАР, а кожний рядок – кількості задіяних в обчисленнях процесорів.

Таблиця 3.1 Залежність часу обчислень від кількості рівнянь в СЛАР

	2500	5000	7500	10000
1	8.243	65.51	221.9	551.6
2	4.519	34.70	115.6	282.3
4	2.161	17.71	64.48	154.9

Обчислення проводились наступним чином. Якщо треба було задіяти 4 процесора, то застосовувались 2 вузли. Якщо треба було задіяти 2 процесори, то також застосовувались 2 вузли і по-одному одному ядру на комному вузлі. Звичайно одне ядро застосовувалось тільки на одному вузлі.

Графічне представлення отриманого результату наведено на рис.3.8, з якого видно, що обчислення на декількох процесорах дають кращий результат у порівнянні з одним процесором. Графіки показують степеневу залежність часу обчислень *t* від кількості рівнянь *N*, що погоджується з викладками, наведеними в розділі 2. Ступінь, якій вони повинні відповідати дорівнює 3.



Рисинук 3.8 Залежність часу обчислень від кількості рівнянь СЛАР та кількості задіяних ядер: а) одне ядро; б) два ядра на двух вузлах; в) чотирі ядра на двух вузлах

На рисунку 3.9 показано залежність t від кількості рівнянь в системі *N* для різної кількості застосованих ядер. Графіки достатньо близькі до лінійних, що вказує на наближення залежності часу обчислень системи рівнянь до кубічної. Однак з таблиці 3.1 маємо, що час рішення СЛАР на 4 ядрах при кількості рівнянь 2500 становить ~2.16 с, а при 5000 – 17.71 с. Тобто кількість рівнянь в системі збільшилось у двічі, а час розрахунків збільшився у 8.19 разів. При збільшені числа рівнянь в СЛАР з 5000 до 10000 час рішення системи

збільшується у 8.75 разів. Таким чином спостерігається поступове відхілення залежності часу рішення СЛАР від кубічної при збільшені числа рівнянь в ній.

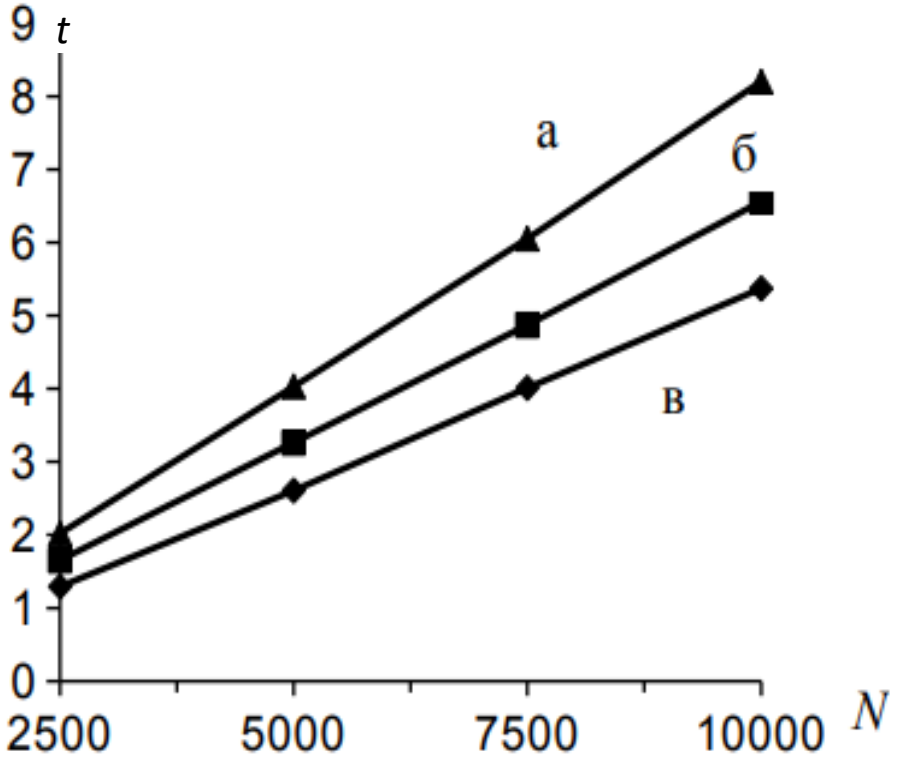


Рисунок 3.9 Залежність (t - час обчислень) від кількості рівнянь СЛАР та кількості задіяних ядер: а) одне ядро; б) два ядра на двух вузлах; в) чотирі ядра на двух вузлах

Одним із факторів, що може впливати на зміни в залежності є комунікаційне середовище. Зі збільшенням розміру СЛАР і ростом числа задіяних в розрахунках процесорів відзначається прискорення обчислень. Кратність прискорень приведено в таблиці 3.2.

При збільшенні розміру розв'язуваної системи рівнянь обсяг обчислювальної роботи росте пропорційно n^3, а обсяг обмінів між процесорами пропорційно n^2. Таким чином відносна частка комунікаційних витрат з розміром системи рівнянь змінюється. Якщо обчисленя виконуються на двох ядрах одного вузла, t о кратність прискорення достатньо близька до 2, що пояснюється набагато більшою швидкістю обміну між ядрами усередині однієї комп’ютерної системи, чим при використанні локальної мережі.

Таблиця 3.2 Залежність кратності прискорення від обсягу вихідних даних

	2500	5000	7500	10000
1:4	3.814	3.699	3.441	3.561
1:2	1.824	1.888	1.920	1.954
2:4	2.091	1.959	1.793	1.822

На рисунку 3.10 показано як збільшуєтья об’єм передавання даних у мережі кластеру при збільшені числа рівнянь в СЛАР.

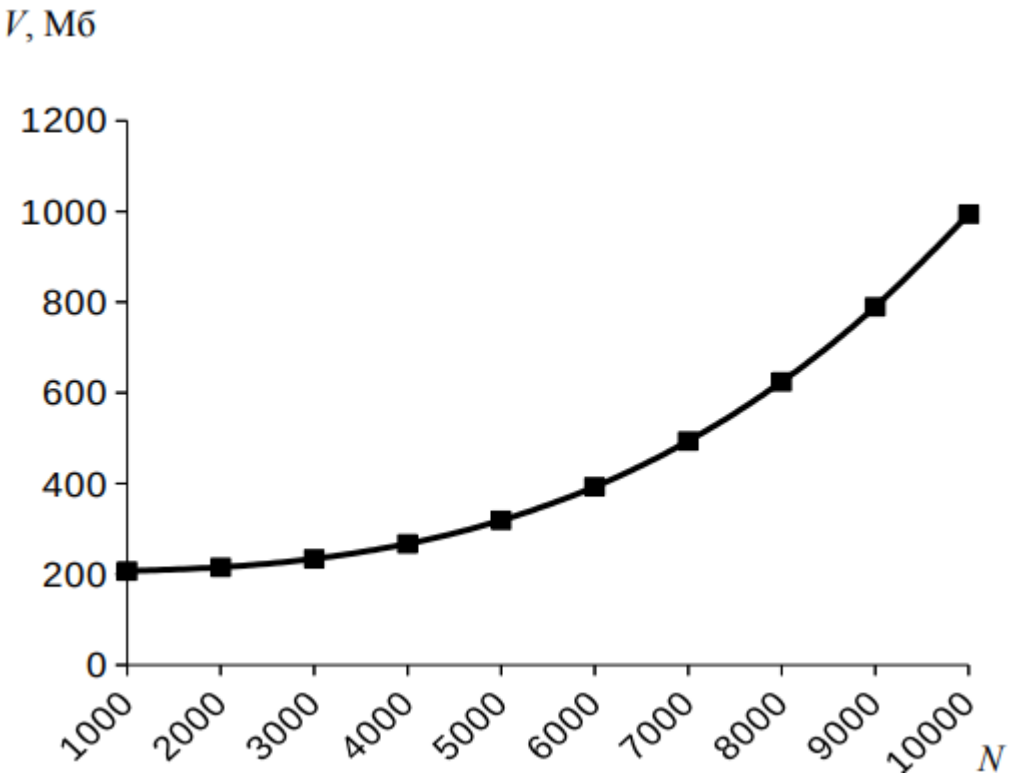
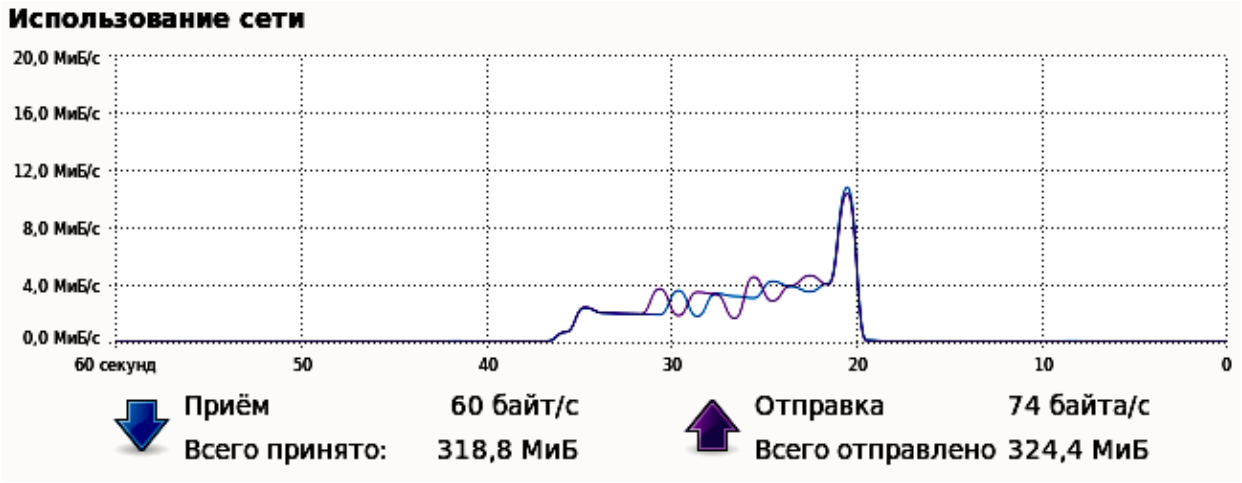
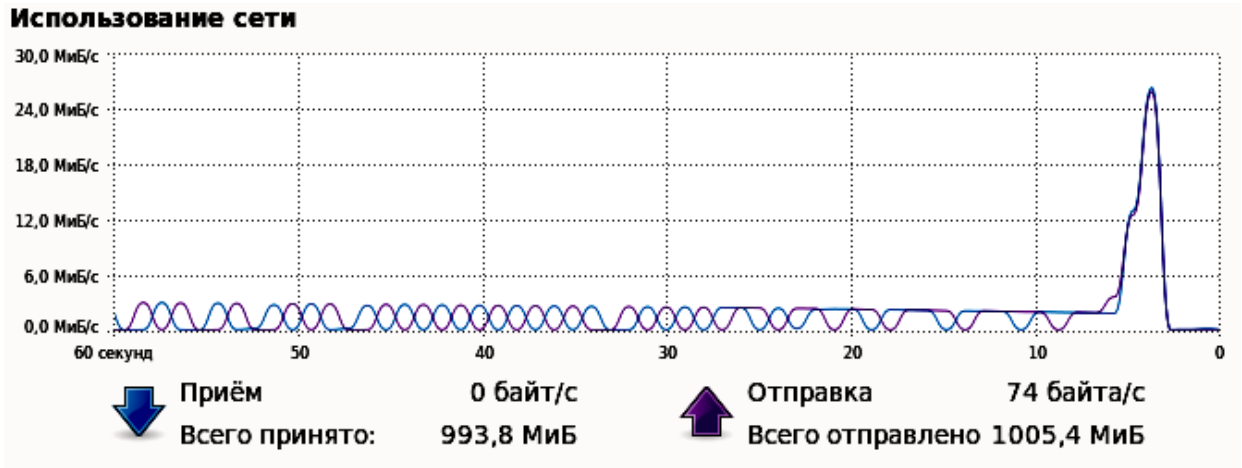


Рисунок 3.10 Залежність об’єму переданної в мережі інформації *V* від кількості рівнянь *N* при виконанні рішення СЛАР

Інтенсивність обміну в мережі обчислювального кластеру при виконані рішення СЛАР показано на рисунку 3.11.



а)



б)

Рисунок 3.11 Обмін в мережі кластеру при рішенні СЛАР: а) з 5000 рівнянь; б) з 10000 рівнянь

З наведених діаграм, які отримано за допомогою системного монітору ОС Linux, видно, що мережний обмін відбувається в ході всього виконання задачі і знаходиться на рівні 3-4 Мб/с. Перед завершенням роботи програми інтенсивність мережного обміну різько зростає, що пояснюється процесом збірки рішення задачі на головному вузлі. При цьому, кількість рівнянь у СЛАР практично не впливає на інтенивність мережного обміну в ході її рішення, а зростає тільки загальний об’єм переданної інформації. На останьому етапі роботи програми – збірці рішення, найбільш інтенсивним є обмін для більшої

системи рівнянь, що пояснюється одночасного виникнення вимоги передачі усіх даних для усіх задіяних процесорів.

Ефективність обчислювального кластеру проявляється у збалансованому застосувані як обладнання вузлів, так і комунікаційної мережі. Було проведено розрахунки з одними і тими ж самими вихідними даними в першому випадку із застосуванням двох процесорів (ядер) на одному вузлі, а в другому – двох процесорів (ядер) на різних вузлах кластеру. Розрахунки було проведено для СЛАР з кількість рівнянь від 2000 до 10000. Результат виміру часу наведено в таблиці 3.3, а залежність t від кількості рівнянь в системі показано на рисунку 3.12. На графіках видно, як дві криві, що відносяться до різних схем розміщення процесорів пертинаються на значенні близько 5500 рівнянь. Нижче цього значення більша швидкість обчислень для схеми 2 процесори на одному вузлі, вижче цього значення – для схеми 2 процесори на різних вузлах.

Таблиця 3.3 Залежність часу рішення СЛАР від схеми розміщення процесорів

	2000	3000	4000	5000	6000	7000	8000	9000	10000
2 CPU на 1 вузлі	1.690	6.248	16.04	33.74	63.34	106.1	172.8	262.4	383.2
2 CPU на 2 вузлах	2.284	7.663	17.69	35.40	59.76	95.88	143.3	201.7	277.9

Скоріш за все це можна пояснити наступним чином. Коли рівнянь не так багато, то обидва ядра на одному вузлі достатньо завантажені, а час на передачу даних між ними настільки малий, що суттево не впливає на загальний час обчислень. При значному підвищенні кількості рівнянь в СЛАР системі, на якій застосовано обидва ядра процесора потрібень деякий час на передачу даних. Це приводить до необхідності звільнення на деякий час хоча б одного із ядер, а отже і поступове зниження продуктивності системи у порівнянні з тою схемою, у якій в розрахунках задіяно 2 ядра на різних вузлах. В такій схемі є ще два вільних процесори, які можуть виконувати усі необхідні системні операції, що також обумовлено розрахунками.

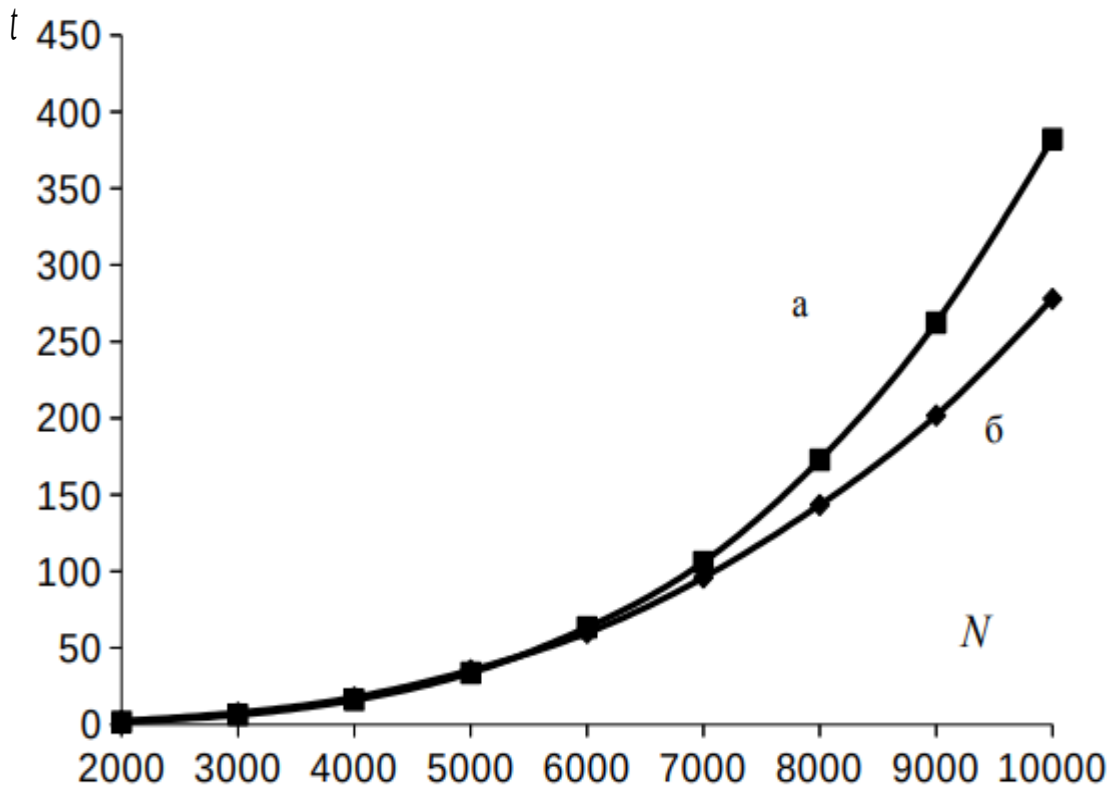


Рисунок 3.12 Вплив схеми розміщення застосованих процесорів (ядер) на час рішення СЛАР: а) 2 процесори на одному вузлі; б) 2 процесори на двох різних вузлах

При використанні ScaLAPACK також необхідно визначити яка розбивка даних на блоки буде найбільш ефективною. Розмір блоку визначається обсягом даних, переданих на кожний вузол. Якщо розмір блоку вказано некоректно, то це негативно позначається на продуктивності обчислень. Блокові алгоритми, а це практично усі алгоритми бібліотеки ScaLAPACK, дозволяють варіювати розмірами блоків, на які діляться вхідні матриці. Таким чином, можна підібрати розміри локальних підматриць так, щоб блок матриці, який багаторазово використовується в матрично-векторних операціях, поміщався у кеш-пам'ять. Це дасть можливість значно скоротити кількість звертань до основної оперативної пам'яті, а значить може вплинути на підвищення продуктивності обчислень. Для зєясування цього, рішення СЛАР, яку складало 7500 рівнянь, було ви но з послідовною зміною розміру блоку, за якими розбивається вхідна матриця. Отримані результати представлено в таблиці 3.4.

Таблиця 3.4 Залежність часу рішення СЛАР від розміру блоку розбивки

блок	32	64	128	140	150	160	256	512
t, с	68.38	62.70	54.39	50.90	48.87	49.41	51.73	60.94

Графічне зображення залежності представлено на рисунку 3.13. Отримані результати показують, що розмір блоку суттєво впливає на час обчислень СЛАР. Занадто малий розмір і занадто великий розмір блоків розбивки вудуть до заниження швидкості обчислень.

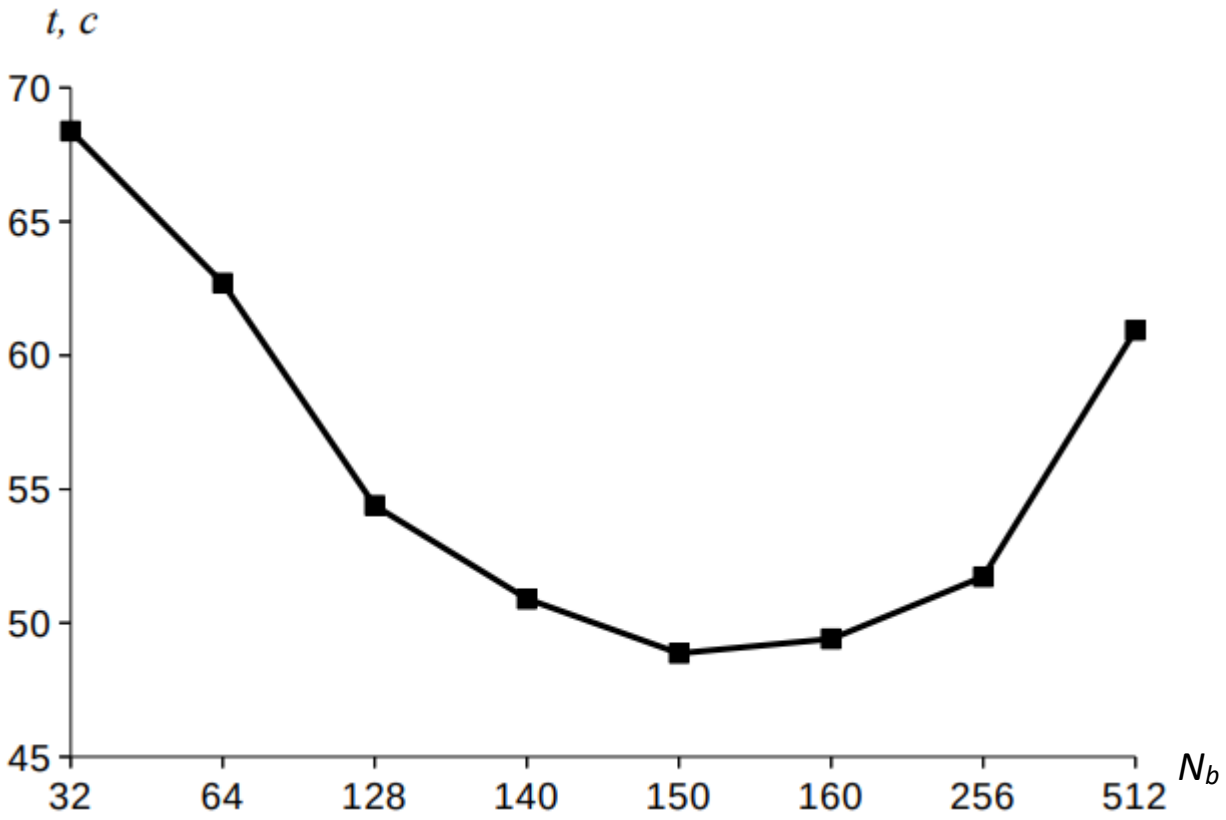


Рисунок 3.13 Залежність часу рішення СЛАР від розміру блоків розбивки N_b

З графіку видно, що оптимальним розміром блоку для даного завдання є блок розмірністю 150×150. Різниця в часі на краях визначеного діапазону та при оптимальному розмірі блоку становить більш ніж 1.5 рази. Вибір блоків занадто малих або занадто великих розмірів веде до неповного завантаження процесорів, а тому і до зниження продуктивності кластеру.

Існують загальні рекомендації розроблювачів стандарту MPI, що стосуються пересіланню данних, а саме, рекомендується застосовувати розмір даних, об’ємом близько 128-256 кбайт. В досліджуваному випадку оптимальний блок має розмір 150×150, тобто 22500 елементів. Типу елементів double потребує 8 байтів для кожного, а тоді їхній сумарний обсяг буде становити 22500×8=180000 байт. Таким чином емпірічно визначений розмір блоку є також оптимальним з точкі зору рекомендацій стандарту MPI.

ВИСНОВКИ

Для ефективного застосування паралельних підпрограм сучасних бібліотек необхідно проводити комп’ютерні експерименти з тестовими програмами для визначення діапазонів використовуваних параметрів та отримання їх оптимальних значень.

Розмір та тип оголошення матриць, що застосовуються у розрахунковій програмі для передачі даних підпрограмам бібліотеки ScaLAPACK, суттєво впливає на завантаженість оперативної пам’яті і, як наслідок, на завантаженість процесору під час компіляції.

Долідження таких підпрограм бібліотеки ScaLAPACK, як pdgemm, pdgetrs та pdgesv показало, що збільшення кількості процесорів, які задіються в обчисленнях, веде до близького до лінійного зменшення часу. Збільшення об’ємів розрахункових систем збільшує час розрахунків на величину, яка достатньо близька до теоретичної. Для множення матриць час збільшується за квадратом від кількості рядків та стовпців, а для рішення СЛАР – за кубом від кількості рівнянь в системі.

Було також встановлено, що на швидкість обчислень впливає кількість задіяних процесорів (ядер), характеристики комп’ютерної мережі, що застосовано для з’єднання вузлів кластеру та такі програмні параметри, як розмір обчислювальної задачі (розмер матриць, кількість рівнянь в СЛАР) та розмір блоків розбивки даних, що передаються між процесорами. Розмір блоку є саме тим параметром, що необхідно перебачувати та встановлювати в алгоритмах розрахункових програм, тому саме цей параметр і критерії вибору його значення є важливими для розробника програмного забезпечення, в якому застосовуються підпрограми ScaLAPACK.

ПЕРЕЛІК ПОСИЛАНЬ

1. Воеводин В.В. Вычислительная математика и структура алгоритмов. М.: МГУ, 2006. 112 с.

2. Многопроцессорные системы / ITспец. Журнал для IT-профессионалов, №3, 2008. 132с.

3. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. М.: Вильямс, 2003. 506с.

4. Воеводин В.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608с.

5. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных систем. СПб.: БХВ-Петербург, 2002. 400с.

6. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI. Минск: БГУ, 2002. 324с.

7. Шпаковский Г.И. Организация параллельных ЭВМ и суперскалярных процессоров: Учеб. пособие. Мн.: Белгосуниверситет, 1996. 284 с.

8. The ScaLAPACK Project. URL: http://www.netlib.org/scalapack.

9. LAPACK – Linear Algebra PACKage. URL: http://www.netlib.org/lapack.

10. Высокопроизводительные параллельные вычисления на кластерных системах / Материалы 7-ой международной конференции-семинара. Нижний Новгород: Изд-во Нижегородского госуниверситета,2007. 443с.

11. Таненбаум Э. Распределенные системы. Принципы и парадигмы. СПб.: Питер, 2003. 877с.

12. Богачев К.Ю. Основы параллельного программирования. М.: БИНОМ. Лаборатория знаний, 2003. 342с.

13. Букатов А.А., Дацюк В.Н., Жегуло А.И. Программирование многопроцессорных вычислительных систем. Ростов-на-Дону: Изд-во ООО “ЦВВР”, 2003. 208с.

14. Корнеев В.В. Параллельные вычислительные системы. М.: “Нолидж”, 1999. 320 с.

15. Кластерные установки в России. URL: http://parallel.ru/parallel/russia /russian_clusters.html.

16. Кофлер М. Весь Линукс. Установка, конфигурирование, использование. М.: Бином-Пресс, 2006. 880с.

17. Кузюрин Н.Н., Фрумкин М.А. Параллельные вычисления: теория и алгоритмы / Программирование. 1991. N 2. С. 3–19.

18. MPI: Message Passing Interface - русскоязычная страница. URL: http://parallel.ru/tech/tech_dev/mpi.html.

19. Специализированные параллельные библиотеки. URL: http://parallel.ru/tech/tech_dev/par_libs.html

20. Шилдт Г. Полный справочник по C++. М.: Издательский дом “Вильямс”, 2003. 800 с.

21. Подбельский В.В., Фомин С.С. Программирование на языке Си. [учеб. пособие.] М.: Финансы и статистика, 2004. 600с.

22. Немнюгин М.А., Стесик О.Л. Современный Фортран. Самоучитель. СПб.: БХВ-Петербург, 2004. 496с.

23. ScaLAPACK – Википедия. URL: https://ru.wikipedia.org/wiki /ScaLAPACK (Дата звернення: 01.11.2019).

24. ScaLAPACK User’s Guide. URL: http://scask.ru/i_book_vlin.php?id=39 (Дата звернення: 01.11.2019).

25. ScaLAPACK Users' Guide – The Netlib. URL: http://www.netlib.org/ scalapack/slug/ (Дата звернення: 01.11.2019).

26. Standarts of Message-Passing in a Distributed Maemory. URL: https://technicalreports.ornl.gov/1992/3445603661204.pdf (Дата звернення: 05.11.2019).

27. BLACS – Process Grid and scoped operations. URL: https://www.netlib.org/blacs/BLACS/Pgrid.html (Дата звернення: 05.11.2019).

28. Basic Linear Algebra Subprograms – Wikipedia. URL: https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms (Дата звернення: 05.11.2019).

29. LAPACK – Linear Algebra PACKage. URL: http://www.netlib.org /lapack/ (Дата звернення: 01.11.2019).

30. High Performance Portable Libraries for Dense Linear Algebra. URL: https://www8.cs.umu.se/kurser/5DV050/VT08/utdelat/F7.pdf (Дата звернення: 08.11.2019).

31. БИБЛИОТЕКА ПРОГРАММ ScaLAPACK. URL: https://wwwinfo. jinr.ru/programs/publ/scalapack/scalapack.html (Дата звернення: 08.11.2019).

32. Society for Industrial And Applied Mathematics. URL: https://epubs.siam.org/doi/book/10.1137/1.9781611971163 (Дата звернення: 08.11.2019).

33. Visual Studio Code Editing. URL: https://code.visualstudio.com/ (Дата звернення: 10.11.2019)

34. Visual Studio – Microsoft IDE for C/C++. URL: http://cppstudio.com /cat/337/ (Дата звернення: 10.11.2019)

35. Qt Documentation. URL: https://doc.qt.io/ (Дата звернення: 15.11.2019)

36. Программирование с Qt – IBM Opensource Developer. URL: https://www.ibm.com/developerworks/ru/library/l-qt_1/index.html (Дата звернення: 15.11.2019).

37. Intel Parallel Studio XE – Maximize Code Performance. URL: https://software.intel.com/en-us/parallel-studio-xe (Дата звернення: 20.11.2019)

38. Intel Parallel Studio XE 2016 Cluster Edition for C++ and Fortran. URL: https://itpro.ua/product/intel-parallel-studio-xe-2016-cluster-edition-c-and-fortran/?tab=description (Дата звернення: 20.11.2019).

39. Intel Parallel Composer – Википедия. URL: https://ru.wikipedia.org/ wiki/Intel_Parallel_Composer (Дата звернення: 20.11.2019).

40. Intel® Inspector XE 2013: автоматическая верификация и отладка в реальном времени. URL: https://habr.com/ru/company/intel/blog/156289/ (Дата звернення: 20.11.2019).

41. Developer Reference for Intel® Math Kernel Library – C. URL: https://software.intel.com/en-us/mkl-developer-reference-c (Дата звернення: 24.11.2019).

ДОДАТОК А

ПРОГРАМА МНОЖЕННЯ МАТРИЦЬ

Варіант на мові FORTRAN:

```
      program abcsl
      include 'mpif.h'
c Параметр nt визначає максимальну розмірність блоку матриці
c на одному процесорі, масиви описані як одномірні
      parameter (nm = 1000, nxn = nm*nm)
      double precision a(nxn), b(nxn), c(nxn), mem(nm)
      double precision time(6), ops, total, t1
c Параметр NOUT - номер вихідного пристрою (термінал)
      PARAMETER ( NOUT = 6 )
      DOUBLE PRECISION ONE, ZERO
      PARAMETER ( ONE = 1.0D+0, ZERO = 0.0D+0)
      INTEGER DESCA(9), DESCB(9), DESCC(9)
c Ініціалізація BLACS
     CALL BLACS_PINFO( IAM, NPROCS )
c Обчислення формату сітки процесорів,
c найбільш наближеного до квадратного
      NPROW = INT(SQRT(REAL(NPROCS)))
      NPCOL = NPROCS/NPROW
c Зчитування параметрів розв'язуваного завдання ( N - розмір матриць і
c NB - розмір блоків ) 0-м процесором і печатка цієї інформації
      IF( IAM.EQ.0) THEN
      WRITE(*,* ) ' Input N and NB: '
   READ( *, * ) N, NB
   WRITE( NOUT, FMT = * )
   WRITE( NOUT, FMT = 9999 )
  $     'The following parameter values will be used:'
   WRITE( NOUT, FMT = 9998 ) 'N ', N
   WRITE( NOUT, FMT = 9998 ) 'NB ', NB
   WRITE( NOUT, FMT = 9998 ) 'P ', NPROW
   WRITE( NOUT, FMT = 9998 ) 'Q ', NPCOL
   WRITE( NOUT, FMT = * )
   END IF
c Розсилання ліченої інформації всім процесорам
   call MPI_BCAST(N, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
   call MPI_BCAST(NB,1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
c Теоретична кількість операцій при перемножуванні
c двох квадратних матриць
   ops = (2.0d0*dfloat(n)-1)*dfloat(n)*dfloat(n)
c Ініціалізація сітки процесорів
   CALL BLACS_GET( -1, 0, ICTXT )
   CALL BLACS_GRIDINIT( ICTXT, 'Row-major', NPROW, NPCOL )
   CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
c Якщо процесор не ввійшов у сітку, то він нічого не робить;
```

```
c таке може трапитися, якщо замовлено, наприклад, 5 процесорів
      IF( MYROW.GE.NPROW .OR. MYCOL.GE.NPCOL )
     $ GO TO 500
c Обчислення реальних розмірів матриць на процесорі
      NP = NUMROC( N, NB, MYROW, 0, NPROW )
      NQ = NUMROC( N, NB, MYCOL, 0, NPCOL )
c Ініціалізація дескрипторів для 3-х матриць
 CALL DESCINIT( DESCA, N, N, NB, NB, 0, 0, ICTXT, MAX(1,NP),    INFO )
 CALL DESCINIT( DESCB, N, N, NB, NB, 0, 0, ICTXT, MAX(1,NP), INFO )
 CALL DESCINIT( DESCC, N, N, NB, NB, 0, 0, ICTXT, MAX(1,NP), INFO )
      lda = DESCA(9)
c Виклик процедури генерації матриць A и B

   call pmatgen(a, DESCA, np, nq, b, DESCB, nprow, npcol, myrow, mycol)
      t1 = MPI_Wtime()

c Виклик процедури перемножування матриць
      CALL PDGEMM('N','N', N, N, N, ONE, A, 1, 1, DESCA,
     $  B, 1, 1, DESCB, 0.0, C, 1, 1, DESCC)
      time(2) = MPI_Wtime() - t1
c Печатка кутових елементів матриці C
c за допомогою службової підпрограми
      if (IAM.EQ.0) write(*,*) 'Matrix C...'
      CALL PDLAPRNT( 1, 1, C, 1, 1, DESCC, 0, 0, 'C', 6, MEM )
      CALL PDLAPRNT( 1, 1, C, 1, N, DESCC, 0, 0, 'C', 6, MEM )
      CALL PDLAPRNT( 1, 1, C, N, 1, DESCC, 0, 0, 'C', 6, MEM )
      CALL PDLAPRNT( 1, 1, C, N, N, DESCC, 0, 0, 'C', 6, MEM )
c Обчислення часу, витраченого на перемножування,
c і оцінка продуктивності в Mflops.
   total = time(2)
   time(4) = ops/(1.0d6*total)
   if (IAM.EQ.0) then
   write(6,80) lda
 80 format(' times for array with leading dimension of',i4)
   write(6,110) time(2), time(4)
 110 format(2x,'Time calculation: ',f12.4, ' sec.',
  $       ' Mflops = ',f12.4)
   end if
c Закриття BLACS-процесів
   CALL BLACS_GRIDEXIT( ICTXT )
   CALL BLACS_EXIT(0)
9998 FORMAT( 2X, A5, '   :   ', I6 )
9999 FORMAT( 2X, 60A )
 500 continue
   stop
   end

   subroutine pmatgen(a, DESCA, np, nq, b, DESCB, nprow, npcol, myrow, mycol)
   integer n, i, j, DESCA(*), DESCB(*), nprow, npcol, myrow, mycol
   double precision a(*), b(*)
   nb = DESCA(5)
c Заповнення локальних частин матриць A і B,
c матриця A формується по алгоритму A(I,J) = I,
c матриця B(I,J) = 1./J
```

```
c Тут маються на увазі глобальні індекси.
   k = 0
   do 250 j = 1,nq
   jc = (j-1)/nb
   jb = mod(j-1,nb)
   jg = mycol*nb + jc*npcol*nb + jb + 1
   do 240 i = 1,np
   ic = (i-1)/nb
   ib = mod(i-1,nb)
   ig = myrow*nb + ic*nprow*nb + ib + 1
   k = k + 1
   a(k) = dfloat(ig)
   b(k) = 1.D+0/dfloat(jg)
 240 continue
 250 continue
   return
   end
```

Варіант на мові С:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#ifdef F77_WITH_NO_UNDERSCORE
#define   numroc_     numroc
#define   descinit_   descinit
#define   pdgemm_     pdgemm
#endif

extern void Cblacs_pinfo( int* mypnum, int* nprocs);
extern void   Cblacs_get( int context, int request, int* value);
extern int Cblacs_gridinit( int* context, char * order, int np_row, int np_col);
extern void Cblacs_gridinfo(int context,int*  np_row, int* np_col, int*  my_row,int*  my_col);
extern void Cblacs_gridexit( int context);
extern void Cblacs_exit( int error_code);
extern void Cdgesd2d(int ictxt, int m, int n, double* A, int ld, int rdest, int cdest);
extern void Cdgerv2d(int ictxt, int m, int n, double* A, int ld, int rsrc, int csrc);
extern int numroc_( int *n, int *nb, int *iproc, int *isrcproc, int *nprocs);
extern void descinit_(int *desc,int *m,int *n,int *mb,int *nb,int *irsrc,int *icsrc,int *ictxt,int *lld, int
*info);

extern void pdgemm_(char *TRANSA,char *TRANSB,int *M,int *N,int *K,double *ALPHA,
double * A, int * IA, int * JA, int * DESCA, double * B, int * IB, int * JB, int * DESCB, double *
BETA, double * C, int * IC, int * JC, int * DESCC );

int main(int argc,char**argv){

/* Розмір матриці n x n та розмір блоку nb x nb */
int n=1000,nb=20;
/* контекст та ідентифікатори процесорів в сітці BLACS */
int ictxt,nprow=2,npcol=2,myrow,mycol;
/* ідентифікатори ті кількість процесорів в середовищі MPI */
```

```
int iam,nprocs;
int zero=0,one=1,ia,ja;
double alpha=1.0,beta=0.;
int myArows,myAcols;

clock_t start_pdgemm,end_pdgemm;
double time_pdgemm;
double *A,*B,*C;
double *myA,*myB,*myC,*SA,*SB;
int descA[9],descB[9],descC[9];

int blocks,rowproc,colproc,rows,cols,jstart,istart,cb,rb,jl,il,k1,k2,i,j;
int rowblocks,colblocks;

/*Визначаємо кількість процесів та порівнюємо їх з кількістю процесорів*/
Cblacs_pinfo(&iam,&nprocs) ;
if (nprow*npcol>nprocs){
if (iam==0)
printf("*ERROR: Ви хачете виконати %d процесів на %d CPU \n", (nprow*npcol),nprocs);
return -1;
}

/*Ініціалізація сітки BLACS*/
Cblacs_get( -1, 0, &ictxt);
Cblacs_gridinit(&ictxt,"Row",nprow,npcol);
Cblacs_gridinfo(ictxt,&nprow,&npcol,&myrow,&mycol);

/* Визначаємо кількість рядків та стовпців для розводілу матриць по процесорах*/
myArows=numroc_(&n,&nb,&myrow,&zero,&nprow);
myAcols=numroc_(&n,&nb,&mycol,&zero,&npcol);
myA=(double*)calloc(myArows*myAcols,sizeof(double));
myB=(double*)calloc(myArows*myAcols,sizeof(double));
myC=(double*)calloc(myArows*myAcols,sizeof(double));
if (!(myA&&myB&&myC)){
printf("\nПомилка виділення пам’яті для proc[%d][%d]\n",myrow,&mycol);
Cblacs_gridexit(ictxt);
Cblacs_exit(1);
return -1;
}

/*Розміщення та визначення глобальних матриць на iam=0*/
if (mycol==0 && myrow==0)
{
    A=(double*)calloc(n*n,sizeof(double));
    B=(double*)calloc(n*n,sizeof(double));
    C=(double*)calloc(n*n,sizeof(double));
    SA=(double*)calloc(myArows*myAcols,sizeof(double));
    SB=(double*)calloc(myArows*myAcols,sizeof(double));
       if(!(A&&B&&C&&SA&&SB)){
          printf("\n Error : proc[0][0]\n");
          Cblacs_gridexit(ictxt);
          Cblacs_exit(1);
          return -1;
          }
```

```
for (i=0;i<n;i++)
   for(j=0;j<n;j++){
       A[i*n+j]=(rand()%1000);
       B[i*n+j]=(rand()%1000);
   }
}
/* Розподіл матриць A та B */
if(mycol==0 && myrow==0)
{
 if(n%nb==0)
        blocks=n/nb;
 else
        blocks=n/nb+1;

for(rowproc=0;rowproc<nprow;rowproc++)
for(colproc=0;colproc<npcol;colproc++)
{
rows=numroc_(&n,&nb,&rowproc,&zero,&nprow);
cols=numroc_(&n,&nb,&colproc,&zero,&npcol);
for(jstart=colproc*nb,cb=colproc,jl=0;cb<blocks;cb+=nprow,jstart+=npcol*nb)
for(k2=0;k2<nb && k2+jstart<n;k2++,jl++)
for(istart=rowproc*nb,rb=rowproc,il=0;rb<blocks;rb+=nprow,istart+=nprow*nb)
for(k1=0;k1<nb && k1+istart<n;k1++,il++)
if(rowproc!=0||colproc!=0)
{
SA[jl*rows+il]=A[(jstart+k2)*n+(istart+k1)];
SB[jl*rows+il]=B[(jstart+k2)*n+(istart+k1)];
}
else
{
myA[jl*rows+il]=A[(jstart+k2)*n+(istart+k1)];
myB[jl*rows+il]=B[(jstart+k2)*n+(istart+k1)];
}
if(rowproc!=0||colproc!=0)
{
Cdgesd2d(ictxt,rows,cols,SA,rows,rowproc,colproc);
Cdgesd2d(ictxt,rows,cols,SB,rows,rowproc,colproc);
}
}
}
else
{
rows=numroc_(&n,&nb,&myrow,&zero,&nprow);
cols=numroc_(&n,&nb,&mycol,&zero,&npcol);
Cdgerv2d(ictxt,rows,cols,myA,rows,0,0);
Cdgerv2d(ictxt,rows,cols,myB,rows,0,0);
}
/* Ініціалізуємо дескриптори для матриць */
descinit_(descA,&n,&n,&nb,&nb,&zero,&zero,&ictxt,&myArows,&one);
descinit_(descB,&n,&n,&nb,&nb,&zero,&zero,&ictxt,&myArows,&one);
descinit_(descC,&n,&n,&nb,&nb,&zero,&zero,&ictxt,&myArows,&one);

/* Застосовуємо підпрограму множення матриць pdgemm з бібліотки BLACS */
```

```
ia=myrow+1;
ja=mycol+1;
start_pdgemm=clock();
pdgemm_("T","N",&myAcols,&myArows,&myArows,&alpha,myA,&ia,&ja,descA,myB,&ia,&ja,
descB,&beta,myC,&ia,&ja,descC);
end_pdgemm=clock();

/* Збираємо результуючу матрицю */
if(mycol==0&&myrow==0)
{
for(rowproc=0;rowproc<nprow;rowproc++)
for(colproc=0;colproc<npcol;colproc++)
{
rows = numroc_(&n,&nb,&rowproc,&zero,&nprow);
cols = numroc_(&n,&nb,&colproc,&zero,&npcol);
if(rowproc!=0||colproc!=0)
Cdgerv2d(ictxt,rows,cols,myC,rows,rowproc,colproc);
rowblocks=rows/nb;
colblocks=cols/nb;
   for(jstart=colproc*nb,cb=0;cb<colblocks;cb++,jstart+=npcol*nb)
    for(k2=0;k2<nb;k2++)
    {
     for(istart=rowproc*nb,rb=0;rb<rowblocks;rb++,istart+=nprow*nb)
      for(k1=0;k1<nb;k1++)
       C[(jstart+k2)*n+(istart+k1)]=myC[(cb*nb+k2)*rows+(rb*nb+k1)];
     if(rows%nb!=0)
      for(k1=0;rowblocks*nb+k1<rows;k1++)
       C[(jstart+k2)*n+(istart+k1)]=myC[(cb*nb+k2)*rows+(rb*nb+k1)];
    }
   if(cols%nb!=0)
   for(k2=0;(colblocks*nb)+k2<cols;k2++)
   {
    for(rb=0;rb<rowblocks;rb++,istart+=nprow*nb)
     for(k1=0;k1<nb;k1++)
      C[(jstart+k2)*n+(istart+k1)]=myC[(cb*nb+k2)*rows+(rb*nb+k1)];
    if(rows%nb!=0)
    for(k1=0;rowblocks*nb+k1<rows;k1++)
    C[(jstart+k2)*n+(istart+k1)]=myC[(cb*nb+k2)*rows+(rb*nb+k1)];
   }
  }
}
else
{
rows=numroc_(&n,&nb,&myrow,&zero,&nprow);
cols=numroc_(&n,&nb,&mycol,&zero,&npcol);
Cdgesd2d(ictxt,rows,cols,myC,rows,0,0);
}
/*  Звільнена матриць */
free(myA);
free(myB);
free(myC);
if (myrow==0&&mycol==0)
{
free(A);
```

```
free(B);
free(C);
free(SA);
free(SB);
}
/* Приведення часового значення та друк його значення для цього завдання*/
time_pdgemm=((double)(end_pdgemm -start_pdgemm))/CLOCKS_PER_SEC;
if (iam==0)
{
printf("\n procs = %d   n=%d    nb=%d   MM= %6.3f\n",nprocs,n,nb,time_pdgemm);
}
Cblacs_gridexit(ictxt);
Cblacs_exit(0);
return 0;
}
```

ДОДАТОК Б

РІШЕННЯ СИСТЕМИ ЛІНІЙНИХ РІВНЯНЬ ІЗ МАТРИЦЕЮ ЗАГАЛЬНОГО ВИДУ

Варіант на мові FORTRAN:

```
      program pdlusl
      include 'mpif.h'
      parameter (nsize = 16000, nxn = nsize*nsize)
      double precision a(nxn), b(nsize)
      double precision time(6), cray, ops, total, norma, normx, t1
      double precision resid, residn, eps, epslon, rab, RANN
      integer ipvt(nsize), iwork(5), init(4)
      PARAMETER ( NOUT = 6 )
      DOUBLE PRECISION ONE
      PARAMETER ( ONE = 1.0D+0 )
      INTEGER DESCA(9), DESCB(9)
c Параметр NRHS – кількість правих частин
      NRHS = 1
      CALL BLACS_PINFO( IAM, NPROCS )
*
      NPROW = INT(SQRT(REAL(NPROCS)))
      NPCOL = NPROCS/NPROW
c Формування одвімерної сітки процесорів
      CALL BLACS_GET( -1, 0, ICTXT )
      CALL BLACS_GRIDINIT( ICTXT, 'Row-major', 1, NPROCS )
c На 0-му процесі зчитуємо параметри, пакуємо їх до масиву
c та розсилаємо усім за допомогою процедури IGEBS2D.

c 1000  continue
      IF( IAM.EQ.0 ) THEN
      WRITE( *,* ) ' Введіть N та NB: '
      READ ( *,* ) N, NB
      IWORK( 1 ) = N
      IWORK( 2 ) = NB
      CALL IGEBS2D( ICTXT, 'All', ' ', 2, 1, IWORK, 2 )
      WRITE( NOUT, FMT = 9999 )
     +'Будуть використовуватись наступні значення параметрів:'
      WRITE( NOUT, FMT = 9998 ) 'N ', N
      WRITE( NOUT, FMT = 9998 ) 'NB ', NB
      WRITE( NOUT, FMT = 9998 ) 'P ', NPROW
      WRITE( NOUT, FMT = 9998 ) 'Q ', NPCOL
      WRITE( NOUT, FMT = * )
      ELSE
c На не 0-м процесі отримуємо массив з процесору (0,0)
      CALL IGEBR2D(ICTXT, 'All', ' ', 2, 1, IWORK, 2, 0, 0)
      N  = IWORK( 1 )
      NB = IWORK( 2 )
```

```
      END IF
c Закриваємо тимчасову сітку процесів
      CALL BLACS_GRIDEXIT( ICTXT )
c Формуємо робочу сітку процесів
      CALL BLACS_GET( -1, 0, ICTXT )
      CALL BLACS_GRIDINIT( ICTXT, 'Row-major', NPROW, NPCOL )
      CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )

c Теоретична кількість операцій, які необхідно виконати для рішення системи
      ops = (2.0d0*dfloat(n)**3)/3.0d0 + 2.0d0*dfloat(n)**2
c
c Перевірка процессорів, що неувійшли до сітки
      IF( MYROW.GE.NPROW .OR. MYCOL.GE.NPCOL )
     +GO TO 500
c Визначаємо точну кількість рядків та стовбців розподіленної матриці  в процесорі
      NP = NUMROC( N, NB, MYROW, 0, NPROW )
      NQ = NUMROC( N, NB, MYCOL, 0, NPCOL )
c
c Формуємо дескриптори
      CALL DESCINIT(DESCA,N,N,NB,NB,0,0,ICTXT,MAX(1,NP),INFO)
      CALL DESCINIT(DESCB,N,NRHS,NB,NB,0,0,ICTXT,MAX(1,NP),INFO)
      lda = DESCA(9)
c  Викликаємо процедуру генерації елементів матриці A та вектору B
      call pmatgenl(a,DESCA,NP,NQ,b,DESCB,nprow,npcol,myrow,mycol)
      t1 = MPI_Wtime()
c Викликаємо процедуру факторизації матриці A
      CALL PDGETRF(N, N, A, 1, 1, DESCA, ipvt, INFO )

      time(1) = MPI_Wtime() - t1
      t1 = MPI_Wtime()
c
c Виклик процедуру рішення СЛАР з факторизованною матрицею A
      CALL PDGETRS('No',N,NRHS,A,1,1,DESCA,ipvt,B,1,1,DESCB,INFO)
*
      time(2) = MPI_Wtime() - t1
      total = time(1) + time(2)
c Виводимо частину результату
      if (iam.eq.0) then
      write(6,40)
 40   format( '      x(1)         x(np)')
      write(6,50) b(1),b(np)
 50   format(1p5e16.8)
c
      write(6,60) n
 60   format(' Витраченний час для матриць порядку  ',i5)
      write(6,70)
 70   format(6x,'факт-ція',5x,'рішення',6x,'загалом',5x,'mflops',7x,'unit',
     + 6x,'ratio')
c
      time(3) = total
      time(4) = ops/(1.0d6*total)
      time(5) = 2.0d0/time(4)
      time(6) = total/cray
      write(6,80) lda
```

```
 80   format(' lda = ',i4)
      write(6,110) (time(i),i=1,6)
 110  format(6(1pe11.3))
      write(6,*) ' Кінець тестування '
      end if
c      goto 1000
      CALL BLACS_GRIDEXIT( ICTXT )
      CALL BLACS_EXIT(0)
      goto 501
 9998 FORMAT( 2X, A5, '  :   ', I6 )
 9999 FORMAT(2X, 60A )
 500  continue
      call MPI_FINALIZE(ierr)
501   continue
      stop
      end

c Процедура генерації матриць A та B за допомогою генератора RAND
      subroutine pmatgenl(a, DESCA, NP, NQ, b, DESCB,
     +nprow, npcol, myrow, mycol)
      integer n, init(4), i, j, DESCA(*), DESCB(*), nprow,
     +npcol, myrow, mycol
      double precision a(*),b(*)
      nb = DESCA(5)
      ICTXT = DESCA(2)
c Ініціалізація генератора
      init(1) = 1
      init(2) = myrow + 2
      init(3) = mycol + 3
      init(4) = 1325
c Заповнення матриці A
      k = 0
      do 250 j = 1,nq
      do 240 i = 1,np
      k = k + 1
      a(k) = rand(0) - 0.5
240   continue
250   continue
      na = k
c Обчислення вектора B
      do 350 i = 1,np
      k = i
      b(i) = 0
      do 351 j = 1,nq
      b(i) = b(i) + a(k)
      k = k + np
351   continue
      CALL BLACS_BARRIER( ICTXT, 'All' )
      CALL DGSUM2D( ICTXT, 'Row', ' ', 1, 1, b(i), 1, -1, 0)
350   continue
      return
      end
```

Варіант на мові С:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include <sys/time.h>
#include <math.h>

static int max( int a, int b ){
      if (a>b) return(a); else return(b);
}

static int min( int a, int b ){
      if (a<b) return(a); else return(b);
}

#ifdef F77_WITH_NO_UNDERSCORE
#define   numroc_     numroc
#define   descinit_     descinit
#define   pdlamch_     pdlamch
#define   pdlange_     pdlange
#define   pdlacpy_     pdlacpy
#define   pdgesv_     pdgesv
#define   pdgemm_     pdgemm
#define   indxg2p_     indxg2p
#define   pdlawrite_   pdlawrite
#endif

static char nul_fname[] = "/../../";
static int nul_len = 7;

extern void   Cblacs_pinfo( int* mypnum, int* nprocs);
extern void Cblacs_barrier(int ConTxt, char *scope);
extern void   Cblacs_get( int context, int request, int* value);
extern int    Cblacs_gridinit( int* context, char * order, int np_row, int np_col);
extern void Cblacs_gridinfo(int context, int* np_row, int* np_col, int* my_row, int*  my_col);
extern void   Cblacs_gridexit( int context);
extern void   Cblacs_exit( int error_code);
extern void Cigsum2d(int ConTxt, char *scope, char *top, int m, int n, int *A, int lda, int rdest, int
cdest);
extern void Cigebs2d(int ConTxt, char *scope, char *top, int m, int n, int *A, int lda);
void Cigebr2d(int ConTxt, char *scope, char *top, int m, int n, int *A, int lda, int rsrc, int csrc);
extern void Csgerv2d(int ConTxt, int m, int n, float *A, int lda, int rsrc, int csrc);
extern void Csgesd2d(int ConTxt, int m, int n, float *A, int lda, int rdest, int cdest);
extern void Cdgamx2d(int ConTxt, char *scope, char *top, int m, int n, double *A, int lda, int *rA,
int *cA, int ldia, int rdest, int cdest);
extern void Cdgerv2d(int ConTxt, int m, int n, double *A, int lda, int rsrc, int csrc);
extern void Cdgesd2d(int ConTxt, int m, int n, double *A, int lda, int rdest, int cdest);
extern int    numroc_( int *n, int *nb, int *iproc, int *isrcproc, int *nprocs);
extern void   descinit_( int *desc, int *m, int *n, int *mb, int *nb, int *irsrc, int *icsrc, int *ictxt, int
*lld, int *info);
extern double pdlamch_( int *ictxt , char *cmach);
```

```
extern double pdlange_( char *norm, int *m, int *n, double *A, int *ia, int *ja, int *desca, double *work);
extern void pdlaprnt_ (int *m, int *n, double *A, int *ia, int *ja, int *descA, int *prow, int *pcol, char *NAME, int *NOUT, double *work);

extern void pdlacpy_( char *uplo, int *m, int *n, double *a, int *ia, int *ja, int *desca, double *b, int *ib, int *jb, int *descb);
extern void pdgesv_( int *n, int *nrhs, double *A, int *ia, int *ja, int *desca, int* ipiv, double *B, int *ib, int *jb, int *descb, int *info);
extern void pdgemm_( char *TRANSA, char *TRANSB, int * M, int * N, int * K, double * ALPHA, double * A, int * IA, int * JA, int * DESCA, double * B, int * IB, int * JB, int * DESCB, double * BETA, double * C, int * IC, int * JC, int * DESCC );
extern int  indxg2p_( int *indxglob, int *nb, int *iproc, int *isrcproc, int *nprocs);
extern void pdlawrite_(char *FILENAM, int *M, int *N, double *A, int *IA, int *JA, int *DESCA, int *IRWRIT, int *ICWRIT, double *WORK);

static void
pdlaread( char *filnam, double *a, int *desca, int irread, int icread, double *work);

int main(int argc, char **argv) {
  int iam, nprocs;//ідентифікатори та число процесорів
  int myrank_mpi, nprocs_mpi;//ідентифікація процесорів в MPI
  int ictxt, nprow, npcol, myrow, mycol;//контекст та ідентифікатори в сітці BLACS
  int np, nq, n, nb, nqrhs, nrhs;//локальне число рядків та стовпців, загальний розмір блоків, число правих частин
  int i, j, k, info, itemp, seed;
  int descA[9], descB[9];//дескриптори масивів
  double *A, *Acpy, *B, *X, *R, eps, *work;//покажчики на необхідні масиви
     double AnormF, XnormF, RnormF, BnormF, residF;//покажчики на масиви для контролю вірності рішення
  int *ippiv, *iwork, iiwork[2];
  int izero=0,ione=1;//цілочислені змінні зі значеннями 0 та 1
  double mone=(-1.0e0),pone=(1.0e0);

FILE *matA, *matB, *answX;//покажчики на файли

  double MPIt1, MPIt2, MPIelapsed;//змінні заміру часу
  char mA[100]; //ім'я вхідної матриці A
  char mB[100];//ім'я вхідного вектору B
  char mX[100];//ім'я файлу з решенням
/**/
  MPI_Init( &argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank_mpi);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs_mpi);

//Визначаемо число правих частин та розмір блоків
   nrhs = 1;  nb = 150;
//Параметри командного рядку
        for (i=1; i<argc; i++){
                if ( strcmp( argv[i], "-a" ) == 0 ){
                  strcpy(mA, argv[i+1]); i++;
                }

                if ( strcmp( argv[i], "-b" ) == 0 ) {
```

```
                strcpy(mB, argv[i+1]); i++;
            }

            if ( strcmp( argv[i], "-x" ) == 0 ){
                strcpy(mX, argv[i+1]); i++;
            }

            if( strcmp( argv[i], "-nb" ) == 0 ) {
            nb = atoi(argv[i+1]);
            i++;
            }
        }
/* Розрахунок формату сітки процесорів */
        nprow = (int) sqrt(nprocs_mpi);
        npcol = nprocs_mpi/nprow;

//Формування робочої сітки процесорів
  Cblacs_pinfo( &iam, &nprocs ) ;
  Cblacs_get( -1, 0, &ictxt );
  Cblacs_gridinit( &ictxt, "Row", nprow, npcol );
  Cblacs_gridinfo( ictxt, &nprow, &npcol, &myrow, &mycol );

        if (myrow == izero && mycol == izero) {
            if (! mA || ! (matA = fopen( mA, "r" ))) {
            printf("Не можу відкрити для зчитування %s файлу \n", mA);
            Cblacs_abort( ictxt, 0 );
            exit( EXIT_FAILURE );}
            fscanf( matA, "%d", &n );
            fclose(matA);

            iiwork[0] = n;
            iiwork[1] = nb;
            if (nprocs>1)
            Cigebs2d( ictxt, "All", " ", 2, 1, iiwork, 2 );
          }
        else  {
            Cigebr2d( ictxt, "All", " ", 2, 1, iiwork, 2, izero, izero );
            n = iiwork[0];
            nb = iiwork[1];
                }
//      printf("n=%d nb=%d\n", n, nb);

        if (nb>n) nb = n; //блок локальної матриці не може бути більше самої матриці
  if (nprow*npcol>nprocs_mpi){
    if (myrank_mpi==0)
      printf(" **** Error : Недостатньо процесорів для побудови сітки {%d, %d} ***\n", nprow,
npcol);
      printf(" **** Необхідно прервати програму ****\n");
    MPI_Finalize(); exit(1);
  }

//Робота продовжується тільки з процесорами, які попали в сітку процесорів
  if ((myrow>-1)&(mycol>-1)&(myrow<nprow)&(mycol<npcol)) {
```

```
//Визначаємо точне число рядків та стовпців розподіленної матриці
     np    = numroc_( &n   , &nb, &myrow, &izero, &nprow );
     nq    = numroc_( &n   , &nb, &mycol, &izero, &npcol );
     nqrhs = numroc_( &nrhs, &nb, &mycol, &izero, &npcol );

//Виділяємо пам’ять для матриць

     A = (double *)calloc(np*nq,sizeof(double)) ;
     if (A= =NULL){ printf("Error помилка розподілення в пам’яті A на
proc %dx%d\n",myrow,mycol); exit(0); }
     Acpy = (double *)calloc(np*nq,sizeof(double)) ;
     if (Acpy= =NULL){ printf("Error помилка розподілення в пам’яті Acpy на
proc %dx%d\n",myrow,mycol); exit(0); }
     B = (double *)calloc(np*nqrhs,sizeof(double)) ;
     if (B= =NULL){ printf("Error помилка розподілення в пам’яті B на
proc %dx%d\n",myrow,mycol); exit(0); }
     X = (double *)calloc(np*nqrhs,sizeof(double)) ;
     if (X= =NULL){ printf("Error помилка розподілення в пам’яті X на
proc %dx%d\n",myrow,mycol); exit(0); }
     R = (double *)calloc(np*nqrhs,sizeof(double)) ;
     if (R= =NULL){ printf("Error помилка розподілення в пам’яті R на
proc %dx%d\n",myrow,mycol); exit(0); }
     ippiv = (int *)calloc(np+nb,sizeof(int)) ;
     if (ippiv= =NULL){ printf("Error помилка розподілення в пам’яті IPIV на
proc %dx%d\n",myrow,mycol); exit(0); }

//Формуємо дескриптори

     itemp = max( 1, np );
     descinit_( descA, &n, &n   , &nb, &nb, &izero, &izero, &ictxt, &itemp, &info );
     descinit_( descB, &n, &nrhs, &nb, &nb, &izero, &izero, &ictxt, &itemp, &info );

//Зчитуємо матриці із файлів

if( iam==0 ) {
       printf("\tЗчитування \n");
     }
pdlaread( mA, A, descA, izero, izero, work);
pdlaread( mB, B, descB, izero, izero, work);

if( iam==0 ) {
       printf("\tКінець зчитування\n");
     }

//for (i=0; i<9; i++) printf("descB[%d]=%d\n",i,descB[i]); //вміст дескриптора B
/** Створюємо копію матриці A, для подальшої оцінки правільності рішення */
   pdlacpy_( "All", &n, &n , A, &ione, &ione, descA, Acpy, &ione, &ione, descA );
   pdlacpy_( "All", &n, &nrhs, B, &ione, &ione, descB, X   , &ione, &ione, descB );
/*
if (iam= =0){
for(i=0; i<n*n; i++) printf("%f\n",A[i]);
for(i=0; i<n; i++) printf("B:\t%f\n",B[i]);
} */
```

```
/*
*************************************************************************
*     Call ScaLAPACK PDGESV routine
*************************************************************************
*/
    if( iam==0 ) {
      printf("                                        \n");
      printf("*********************************************************\n");
      printf("   Рішення СЛАР  з підпрограмою PDGESV ScaLAPACK \n");
      printf("*********************************************************\n");
      printf("                                        \n");
      printf("\tn = %d\tnrhs = %d\tсітка (%d,%d)\t з блоками: %dx%d\n", n, nrhs, nprow, npcol, nb,
nb);
      printf("                                        \n");
    }

printf("Початок рішення\n");
    MPIt1 = MPI_Wtime();
        pdgesv_( &n, &nrhs, A, &ione, &ione, descA, ippiv, X, &ione, &ione, descB, &info );
    MPIt2 = MPI_Wtime();
    MPIelapsed=MPIt2-MPIt1;
printf("Кінець рішення\n");

    if( iam= =0 ) {
      printf("\tчас виконання = %f s\n",MPIelapsed);
    }
    if( iam= =0 ) {
      printf("                                        \n");
      printf("\t PDGESV повернула код INFO = %d             \n",info);
      printf("\t----------------------------------------\n");

answX=fopen(mX,"wt");
for (i=0; i<n; i++) fprintf(answX, "%f\n",X[i]);
fclose(answX);
          printf("                                        \n");
    }
        pdlacpy_( "All", &n, &nrhs, B, &ione, &ione, descB, R   , &ione, &ione, descB );
        eps = pdlamch_( &ictxt, "Epsilon" );
    pdgemm_( "N", "N", &n, &nrhs, &n, &pone, Acpy, &ione, &ione, descA, X, &ione, &ione,
descB,
        &mone, R, &ione, &ione, descB);
    AnormF = pdlange_( "F", &n, &n   , A, &ione, &ione, descA, work);
    BnormF = pdlange_( "F", &n, &nrhs, B, &ione, &ione, descB, work);
    XnormF = pdlange_( "F", &n, &nrhs, X, &ione, &ione, descB, work);
    RnormF = pdlange_( "F", &n, &nrhs, R, &ione, &ione, descB, work);
    residF = RnormF / ( AnormF * XnormF * eps * ((double) n));
    if ( iam==0 ){
      printf("                                        \n");
      printf("\t||A * X  - B||_F / ( ||X||_F * ||A||_F * eps * N ) = %e \n",residF);
      printf("                                        \n");
      if (residF<10.0e+0)
        printf("\tОтримана коректна відповідь\n");
      else
```

```
        printf("\tОтримано некоректну відповідь\n");
    }

    free(A);
    free(Acpy);
    free(B);
    free(X);
    free(ippiv);
    Cblacs_gridexit( 0 );
  }

  if (iam==0){
    printf("\tКінець роботи програми\n");
    printf("**********************************************\n");
    printf("                                      \n");
  }
  MPI_Finalize();
  exit(0);
}

/*
Підпрограма, що зчитує з файлу з ім'ям FILNAM матрицю та розподіляє її по сітці процессів.
Файл зчитує тільки процес з координатою {'iread', 'icread'}.
 */
static void pdlaread( char *filnam, double *a, int *desca, int iread, int icread, double *work)
{
  int h, i, j, k, m, n, npcol, mycol, ctxt, iwork[2], nprow, myrow, ib, jb, mb,nb;
  int ii, jj, lda, icurcol, icurrow, matrnd;
  double rcp = 1.0 / RAND_MAX;
  FILE *fin = NULL; char fmt[6] = "%d %d";
  int scaleDiag = 0, useBLACS = 1, tag = 1, dest;
        double *mem;
//  char gridOrder = 'R';
  MPI_Status stat;
//  int opcode = 1;
  /* Отримання параметрів в сітці */
  ctxt = desca[1];
  Cblacs_gridinfo( ctxt, &nprow, &npcol, &myrow, &mycol );
/*
printf("\tReading in pdlaread\n\tnprow=%d col=%d myr=%d myc=%d", nprow, npcol, myrow,
mycol);
*/
  if (myrow == iread && mycol == icread) {
    if (strncmp( nul_fname, filnam, nul_len ) == 0) {
      iwork[0] = iwork[1] = -1;    }
    else {
      if (! filnam || ! (fin = fopen( filnam, "rb" ))) {
        printf("PDLAREAD: Не можу відкрити на зчитування файл: %s\n", filnam);
        Cblacs_abort( ctxt, 0 );
        exit( EXIT_FAILURE );}
      //while (fgetc( fin ) != '\n');
      fscanf( fin, fmt, iwork, iwork+1 );
        //printf("m=%d n=%d\n", iwork[0], iwork[1]);
      while (fgetc( fin ) != '\n') ; /*пропустити лінії, що зостались */
```

```
   }
     Cigebs2d( ctxt, "All", " ", 2, 1, iwork, 2 );
    }
 else
{
     Cigebr2d( ctxt, "All", " ", 2, 1, iwork, 2, irread, icread );
// printf("I am {%d, %d} m=%d n=%d\n",myrow,mycol, iwork[0], iwork[1]);
  }
  m = iwork[0];
  n = iwork[1];
//printf("m=%d n=%d myrow=%d mycol=%d \n", m,n, myrow, mycol);
  if (m > desca[2] || n > desca[3]) {
     printf("PDLAREAD: Matrix too big to fit in\nAbort ...\n");
     Cblacs_abort( ctxt, 0 );
     }
  ii = 0;
  jj = 0;
  mb = desca[4];
  nb=desca[5];
  icurrow = desca[6];
  icurcol = desca[7];
  lda = desca[8];
//printf("icurrow=%d icurcol=%d lda=%d\n", icurrow, icurcol, lda);
  /* Прохід за стовбцями блоків */
  for (j = 1; j <= n; j += desca[5]) {
    jb = min( desca[5], n - j + 1 );
    for (h = 0; h < jb; h++) {
      /* Проход за рядками блоків */
      for (i = 1; i <= m; i += desca[4]){
        ib = min( desca[4], m - i + 1 );
        if (icurrow == irread && icurcol == icread) {
            if (myrow == irread && mycol == icread) {
                // fread( a+ii+(size_t)(jj+h)*lda, sizeof(double)*ib, ib, fin );
               for (k = 0; k < ib; k++) fscanf(fin,"%lf",a+ii+(size_t)(jj+h)*lda+k);
            }
        }
            else {
            if (myrow == icurrow && mycol == icurcol) {
            printf("Процес {%d, %d} отримав інформацію \n", myrow, mycol);
            Cdgerv2d( ctxt, 1, ib, a + ii + (size_t)(jj + h) * lda, lda, irread, icread );
                    printf("2- Процес {%d, %d} отримав інформацію\n", myrow, mycol);
          //        Cblacs_barrier(ctxt, "All");
          //  fflush(stdout);
            }
                    else
                    if (myrow == irread && mycol == icread) {
                    printf("Процес {%d, %d} відправив інформацію\n", myrow, mycol);
              //fread( work, sizeof *work, ib, fin );
                    for (k = 0; k < ib; k++) fscanf(fin,"%lf",work+k);
                    printf("2- Процес {%d, %d} відправив інформацію\n", myrow, mycol);
              Cdgesd2d( ctxt, 1, ib, work, desca[4], icurrow, icurcol );
                    //Cblacs_barrier(ctxt, "All");
                    }
      }
```

```
        if (myrow == icurrow) ii += ib;
        icurrow = (icurrow + 1) % nprow;
      }
      ii = 0;
      icurrow = desca[6];
    }
//    if(myrow == 0 && mycol == 0) printf( "%d %d\n", i, j );
    if (mycol == icurcol) jj += jb;
    icurcol = (icurcol + 1) % npcol;
  }
  if (myrow == irread && mycol == icread && ! matrnd) fclose( fin );
}          /* pdlaread */
```