Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ / В.В. Шайдуров

«___» _____2020 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01 «Математика и компьютерные науки»

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Научный руководитель
кандидат физико-математических наук, доцент 22.06.2020/ Е.В. Кучунова

Выпускник 21.06.2020/ А.А. Дядяшкин

Красноярск 2020

РЕФЕРАТ

Выпускная квалификационная работа по теме «Параллельные алгоритмы решения систем линейных алгебраических уравнений» содержит 31 страницу текстового документа, 10 формул, 6 использованных источников, 4 рисунка и 2 приложения.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ, ПРОГРАММИРОВАНИЕ, КЛАСТЕР, МНОГОПОТОЧНОСТЬ, МЕТОД ГАУСА, МЕТОД ЯКОБИ, МЕТОД ГАУСА, ЗЕЙДЕЛЯ, ВЫЧИСЛИТЕЛЬНЫЙ УЗЕЛ, MPI, OPENMP.

Цель работы:

- Рассмотреть и изучить технологии для организации параллельных вычислений;
- Рассмотреть и изучить методы решения поставленной задачи;
- Создать параллельную программу для решения поставленной задачи на кластерных системах с общей и распределённой памятью, а так же на десктопных устройствах, на ОС Windows 10;
- Разработать программы, наиболее эффективно реализующие решение задач;
- Провести анализ и сравнить полученные результаты.

В ходе работы были изучены технологии распараллеливания MPI и OpenMP, изучены метод Гаусса, итерационные методы Якоби и Гаусса-Зейделя для СЛАУ. Разработаны последовательные и параллельные программы для решения поставленной задачи, проведен анализ и сравнение технологий распараллеливания на кластере и на ноутбуке с ОС Windows 10.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Постановка задачи 5
2 Последовательные алгоритмы решения систем линейных алгебраических уравнений 8
2.1 Метод Гаусса 8
2.2 Метод Якоби 9
2.3 Метод Гаусса-Зейделя 10
2.4 Сравнение последовательных алгоритмов 10
3 Распараллеливание метода Гаусса решения систем линейных алгебраических уравнений 14
3.1 Выделение информационных зависимостей 14
3.2 Масштабирование и распределение подзадач по процессам 15
3.3 Используемые MPI-функции 16
3.4 Общая память 19
3.5 Результаты распараллеливания метода Гаусса 20
4 Распараллеливание итерационных методов решения систем линейных алгебраических уравнений 23
4.1 Параллельный алгоритм решения СЛАУ методом Якоби 23
4.2 Исследование эффективности распараллеливания метода Якоби для систем с распределенной памятью 24
4.3 Особенности использования общей памяти при распараллеливании вычислений 25
4.4 Исследование эффективности распараллеливания метода Якоби для систем с общей памятью 25
ЗАКЛЮЧЕНИЕ 28
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 29
ПРИЛОЖЕНИЯ 31

ВВЕДЕНИЕ

Применение параллельных вычислительных систем является стратегическим направлением развития вычислительной техники [2]. Это вызвано не только принципиальным ограничением максимально возможного быстродействия последовательных ЭВМ, но и практически постоянным наличием вычислительных задач, для решения которых возможностей существующих средств вычислительной техники недостаточно.

Наличие таких задач обусловлено возможностью изучать явления, которые являются либо слишком сложными для исследования аналитическими методами, либо слишком дорогостоящими или опасными для экспериментального изучения, а также быстрым ростом сложности объектов моделирования, необходимостью управления сложными промышленными и технологическими процессами в режиме реального времени и ростом числа задач, для решения которых необходимо обрабатывать гигантские объемы информации. Для подобных задач применение параллельных вычислений является зачастую просто необходимым.

Организация параллельности вычислений осуществляется, в основном, за счет многопроцессорности, что позволяет в один и тот же момент выполнять одновременно несколько операций обработки данных. Если осуществить разделение применяемого алгоритма на информационно независимые части и организовать выполнение каждой части вычислений на разных процессорах, можно достичь ускорения процесса решения вычислительной задачи и выполнить необходимые вычисления с меньшими затратами времени.

На сегодняшний день наиболее используемыми и наиболее эффективными технологиями распараллеливания являются OpenMP (Open Multi-Processing) и MPI (Message Passing Interface) [3]. При использовании интерфейса обмена сообщениями MPI обмен данными между различными процессами в программе осуществляется с помощью механизма передачи и

приемки сообщений [1]. При создании параллельных программ, предназначенных для многопроцессорных вычислительных систем с общей памятью используются методы многопроточного программирования с помощью директив OpenMP, являющимися специальными директивами для компилятора [4]. Они создают и организуют выполнение параллельных процессов (нитей), а также обмен данными между процессами.

Данные технологии часто применяются на системах с различными организациями взаимодействия между вычислительными узлами вычислительных систем и имеют свои недостатки и достоинства, однако на вычислительных системах, организация которых позволяет это сделать, данные технологии объединяются для достижения большей продуктивности параллельных вычислений, используя достоинства каждой из технологий.

В данной работе рассматриваются применение технологий распараллеливания на примере конкретной задачи, различные их реализации, и приводится сравнение и анализ полученных результатов, исследований и проведенных опытов.

1 Постановка задачи

Необходимость решать системы линейных уравнений может возникнуть при решении различных прикладных задач, описываемых дифференциальными, интегральными или системами нелинейных (трансцендентных) уравнений. С такой необходимостью также можно столкнуться при решении задач математического программирования, статистической обработки данных, аппроксимации функций, при дискретизации краевых дифференциальных задач методом сеток. Матрицы коэффициентов систем линейных уравнений могут иметь различные структуры и свойства. Матрицы решаемых систем могут быть плотными, порядок систем может достигать несколько десятков тысяч строк и столбцов.

Линейное уравнение с n неизвестными $x_0, x_1, \dots, x_{n-1}$ может быть определено при помощи выражения

$$a_0 x_0 + a_1 x_1 + \cdots + a_{n-1} x_{n-1} = b, \tag{1}$$

где величины $a_0, a_1, \dots, a_{n-1}$ и b представляют собой постоянные значения [8].

Множество n линейных уравнений

$$\begin{aligned}
&a_{0,0}x_0 && +a_{0,1}x_1 && +\cdots+a_{0,n-1}x_{n-1} && =b_0\\
&a_{1,0}x_0 && +a_{1,1}x_1 && +\cdots+a_{1,n-1}x_{n-1} && =b_1\\
&\qquad\dots\\
&a_{n-1,0}x_0 && +a_{n-1,1}x_1 && +\cdots+a_{n-1,n-1}x_{n-1} && =b_{n-1}
\end{aligned} \tag{2}$$

называется системой линейных алгебраически уравнений (СЛАУ). В более кратком (матричном) виде система может представлена как

$$\mathbf{A}x = b, \tag{3}$$

где $\mathbf{A} = (a_{i,j})$ есть вещественная матрица размера $n \times n$, а вектора b и x состоят из n элементов.

Под задачей решения системы линейных уравнений для заданных матрицы **A** и вектора b обычно понимается нахождение значения вектора неизвестных x, при котором выполняются все уравнения системы (2).

Цель работы – исследовать эффективность применения распараллеливания вычислений при нахождении решения СЛАУ. Для достижения поставленной цели необходимо решить следующие задачи.

1. Реализовать последовательные алгоритмы численного решения СЛАУ. Для анализа выбраны метод Гаусса с выбором главного элемента и итерационные методы Якоби и Гаусса-Зейделя.
2. Организовать распараллеливание вычислений для систем с распределенной памятью с помощью технологии MPI.
3. Организовать распараллеливание вычислений для систем с общей памятью с помощью технологии OpenMP.
4. Провести анализ полученных параллельных программ на достижение ускорение вычислений.

Для оценки параллельных алгоритмов производятся замеры времени выполнения параллельной части программы, по которым определяются ускорение и эффективность.

Ускорением параллельного алгоритма называют величину

$$S_p = \frac{T}{T_p}, \tag{4}$$

где T – время работы последовательной программы; T_p – время работы параллельной программы; p – число используемых вычислительных узлов.

Параллельный алгоритм может давать большое ускорение, но использовать для этого множество процессов нецелесообразно. В таком случае для оценки параллельного алгоритма используется понятие эффективности:

$$E = \frac{S_p}{p} \times 100\%. \quad (5)$$

Величина эффективности определяет среднюю долю времени выполнения алгоритма, в течении которой процессоры реально задействованы для решения задач. В наилучшем случае эффективность составляет 100%. В практических случаях ускорение может превосходить 100%, что говорит о существовании сверхлинейного ускорения.

Для осуществления вычислений в работе используются следующие вычислительные системы.

- Вычислительная система №1 (ВС №1): малый кластер СФУ с характеристиками: 28 четырехъядерных процессоров Intel Xeon Quad Core E5345 2.33GHz, Управляющий узел IBM x3650 (2xDual-Core Intel Xeon Processor 5160 3.00 GHz), Хранилище IBM System Storage DS3400 (12x300GB Hot-Swap 3.5in 10K RPM Ultra320 SAS HDD).
- Вычислительная система №2 (ВС №2): персональный ноутбук с четырехядерным процессором Intel Pentium N3540, каждое ядро по 2.16 GHz, 4 Гб оперативной памяти, операционная система Windows 10x64, SSD.

2 Последовательные алгоритмы решения систем линейных алгебраических уравнений

2.1 Метод Гаусса

Метод Гаусса является широко известным прямым алгоритмом решения систем линейных уравнений, для которых матрицы коэффициентов являются плотными. Если система линейных уравнений является невырожденной, то метод Гаусса гарантирует нахождение решения с погрешностью, определяемой точностью машинных вычислений. Основная идея метода состоит в приведении матрицы **A** посредством эквивалентных преобразований (не меняющих решение системы (2)) к треугольному виду, после чего значения искомых неизвестных может быть получено непосредственно в явном виде.

Метод Гаусса основывается на возможности выполнения эквивалентных преобразований линейных уравнений, которые не меняют при этом решение рассматриваемой системы. К числу таких преобразований относятся:

• умножение любого из уравнений на ненулевую константу,

• перестановка уравнений,

• прибавление к уравнению любого другого уравнения системы.

Метод Гаусса включает последовательное выполнение двух этапов. На первом этапе – прямой ход метода Гаусса – исходная система линейных уравнений при помощи последовательного исключения неизвестных приводится к диагональному виду

$$\mathbf{U}x = c, \tag{6}$$

где матрица коэффициентов получаемой системы имеет вид

$$U = \begin{pmatrix} u_{0,0} & 0 & \cdots & 0 \\ 0 & u_{1,1} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & u_{n-1,n-1} \end{pmatrix}. \tag{7}$$

На обратном ходе метода Гаусса [8] осуществляется определение значений неизвестных. Из каждого уравнения определяется значение соответствующей переменной: $x_i = \frac{c_i}{u_{i,i}}$ для $i = 0,...,n-1$.

2.2 Метод Якоби

Метод Якоби — один из наиболее простых методов приведения системы матрицы к виду, удобному для итерации: из 1-го уравнения матрицы выражаем неизвестное x_1, из 2-го выражаем неизвестное x_2 и т.д.

Формула для нахождения результата:

$$x_i^{(s+1)} = \frac{1}{a_{ii}}\left(y_i - \sum_j a_{ij} x_j^{(s)}\right), \tag{8}$$

где s – номер итерации.

Итерационный процесс продолжается, пока значение вычисленной невязки, определяемой следующим выражением

$$\max_{0 \le i < n} \left| \sum_{j=0}^{n-1} a_{i,j} \cdot x_j - b_i \right| \tag{9}$$

превышает заранее заданное значение ε. В качестве этого значения берется значение невязки, полученное при решения этой же СЛАУ методом Гаусса. В данном случае предварительное вычисление невязки необходимо для того, чтобы результаты, вычисляемые точным и итерационным методом, были примерно одинаковые [8].

2.3 Метод Гаусса-Зейделя

Метод Гаусса-Зейделя — метод является модификацией метода Якоби.

При вычислении очередного s-го приближения к неизвестному x_i при $i > 1$ используют уже найденные s-е приближения к неизвестным $x_1, x_2, \dots, x_{i-1}$, а не (s-1)−ое приближение, как в методе Якоби.

Матричная запись:

$$x_i^{(s)} = \frac{1}{a_{ii}}\left(y_i - \sum_j a_{ij} x_j^{(s)}\right), \tag{10}$$

где s – номер итерации.

Критерий окончания итерационного процесса аналогичен изложенному в п. 2.2 [8].

2.4 Сравнение последовательных алгоритмов

Для тестовых систем были проведены замеры времени выполнения программ на вычислительной системе 2. В таблице 1 показаны результаты замеров времени метода Гаусса и метода Якоби, а также отношение между ними.

Таблица 1 – Сравнение последовательного метода Гаусса и Якоби

Размерность системы	Время работы программ (секунды)		Отношение метода Гаусса к методу Якоби (T_1 / T_2)
	Метод Гаусса (T_1)	Метод Якоби (T_2)	
1000	4,56	0,38	12,0
2000	36,38	1,48	24,7
3000	117,90	3,24	36,4
4000	316,33	6,12	51,7
5000	523,68	8,68	60,3
6000	875,70	12,47	70,2
7000	1432,80	16,85	85,0
8000	2019,70	21,63	93,4
9000	2890,32	27,59	104,7
10000	4150,24	34,95	118,7

На рисунке 1 представлен график отношения времени вычислений по методу Гаусса к времени вычислений по методу Якоби (T_1 / T_2). Из приведенных данных видим, что время работы итерационного метода Якоби значительно меньше времени работы метода Гаусса.

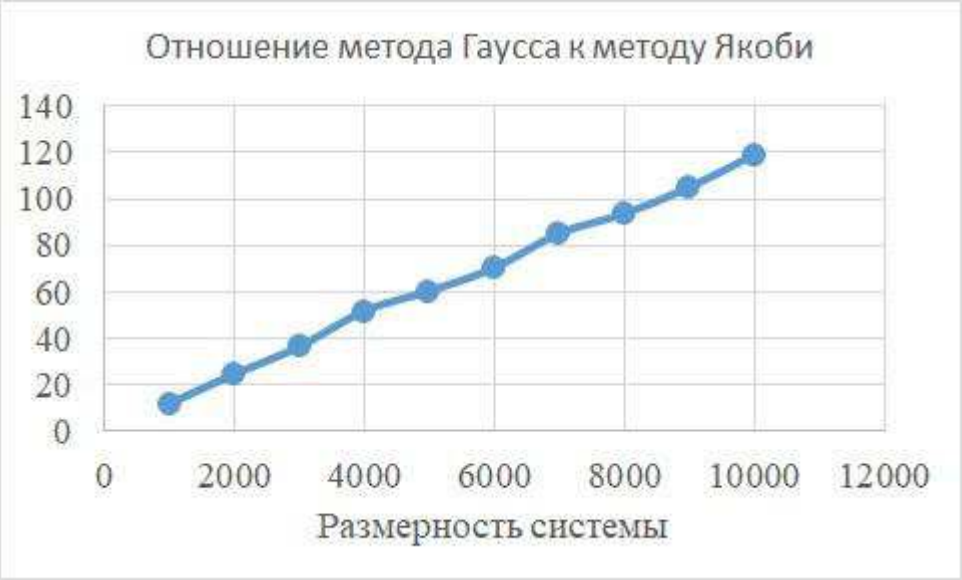

Рисунок 1 – График отношения времени вычислений (метод Гаусса / метод Якоби)

В таблице 2 приведены результаты замеров времени проведения вычислений по методам Гаусса и Гаусса-Зейделя, а также отношение между ними.

Таблица 2 – Сравнение последовательного метода Гаусса и Гаусса-Зейделя

Размерность системы	Время работы программ (секунды)		Отношение метода Гаусса к методу Гаусса-Зейделя (T_1 / T_3)
	Метод Гаусса (T_1)	Метод Гаусса-Зейделя (T_3)	
1000	4,56	0,132	34,5
2000	36,38	0,492	73,9
3000	117,90	1,114	105,8
4000	316,33	2,056	153,9
5000	523,68	3,015	173,7
6000	875,70	4,296	203,8
7000	1432,80	5,887	243,4
8000	2019,70	7,513	268,8
9000	2890,32	9,556	302,5
10000	4150,24	12,173	340,9

На рисунке 2 представлен график отношения времени вычислений по методу Гаусса к времени вычислений по методу Гаусса-Зейдея (T_1 / T_3). Аналогично отмечаем, что время работы метода итерационного метода Гаусса-Зейделя значительно меньше времени работы метода Гаусса.

Рисунок 2 – График отношения времени вычислений (метод Гаусса / метод Гаусса-Зейделя)

В таблице 3 приведены результаты замеров времени у методов Якоби и Гаусса-Зейделя, а также отношение между ними.

Таблица 3 – Сравнение последовательного метода Якоби и Гаусса-Зейделя

Размерность системы	Время работы программ (секунды)		Отношение метода Якоби к методу Гаусса-Зейделя (T_2 / T_3)
	Метод Якоби (T_2)	Метод Гаусса-Зейделя (T_3)	
1000	0,381	0,132	2,9
2000	1,475	0,492	3,0
3000	3,239	1,114	2,9
4000	6,12	2,056	3,0
5000	8,683	3,015	2,9
6000	12,474	4,296	2,9
7000	16,854	5,887	2,9
8000	21,626	7,513	2,9
9000	27,593	9,556	2,9
10000	34,951	12,173	2,9

На рисунке 3 представлен график отношения времени вычислений по методу Якоби к времени вычислений по методу Гаусса-Зейдея (T_2 / T_3), т.е. итерационный метод Гаусса-Зейделя работает примерно в три раза быстрее итерационного метода Якоби.

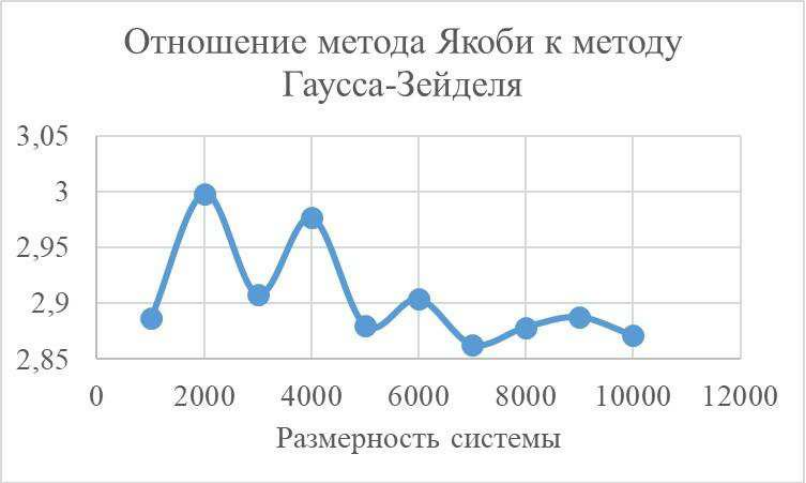

Рисунок 3 – График отношения времени вычислений (метод Якоби / метод Гаусса-Зейделя)

3 Распараллеливание метода Гаусса решения систем линейных алгебраических уравнений

При внимательном рассмотрении метода Гаусса можно заметить, что все вычисления сводятся к однотипным вычислительным операциям над строками матрицы коэффициентов системы линейных уравнений. Как результат, в основу параллельной реализации алгоритма Гаусса может быть положен принцип распараллеливания по данным. В качестве *базовой подзадачи* можно принять тогда все вычисления, связанные с обработкой одной строки матрицы A и соответствующего элемента вектора b.

3.1 Выделение информационных зависимостей

Рассмотрим общую схему параллельных вычислений и возникающие при этом информационные зависимости между базовыми подзадачами.

Для выполнения *прямого хода* метода Гаусса необходимо осуществить $(n-1)$ итерацию по исключению неизвестных для преобразования матрицы коэффициентов A к диагональному виду.

Выполнение итерации i, $0 \leq i < n-1$, прямого хода метода Гаусса включает ряд последовательных действий. Прежде всего, в самом начале итерации необходимо выбрать ведущую строку, которая при использовании метода главных элементов определяется поиском строки с наибольшим по абсолютной величине значением среди элементов столбца i, соответствующего исключаемой переменной x_i. Поскольку строки матрицы **A** распределены по подзадачам, для поиска максимального значения подзадачи с номерами k, $k>i$, должны обменяться своими элементами при исключаемой переменной x_i. После сбора всех необходимых данных в каждой подзадаче может быть определено, какая из подзадач содержит ведущую строку и какое значение является ведущим элементом.

Далее для продолжения вычислений ведущая подзадача должна разослать свою строку матрицы **A** и соответствующий элемент вектора b всем остальным подзадачам с номерами k, $k>i$. Получив ведущую строку, подзадачи выполняют

вычитание строк, обеспечивая тем самым исключение соответствующей неизвестной x_i.

При выполнении *обратного хода* метода Гаусса подзадачи выполняют необходимые вычисления для нахождения значения неизвестных. Как только какая-либо подзадача i, $0 \le i < n-1$, определяет значение своей переменной x_i, это значение должно быть разослано всем подзадачам с номерами k, $k<i$. Далее подзадачи подставляют полученное значение новой неизвестной и выполняют корректировку значений для элементов вектора b.

3.2 Масштабирование и распределение подзадач по процессам

Выделенные базовые подзадачи характеризуются одинаковой вычислительной трудоемкостью и сбалансированным объемом передаваемых данных. В случае, когда размер матрицы, описывающей систему линейных уравнений, оказывается большим, чем число доступных процессов (p<n), базовые подзадачи можно укрупнить, объединив в рамках одной подзадачи несколько строк матрицы. Воспользуемся ленточной схемой разделения данных: каждому процессу выделяется непрерывная последовательность строк матрицы линейных уравнений.

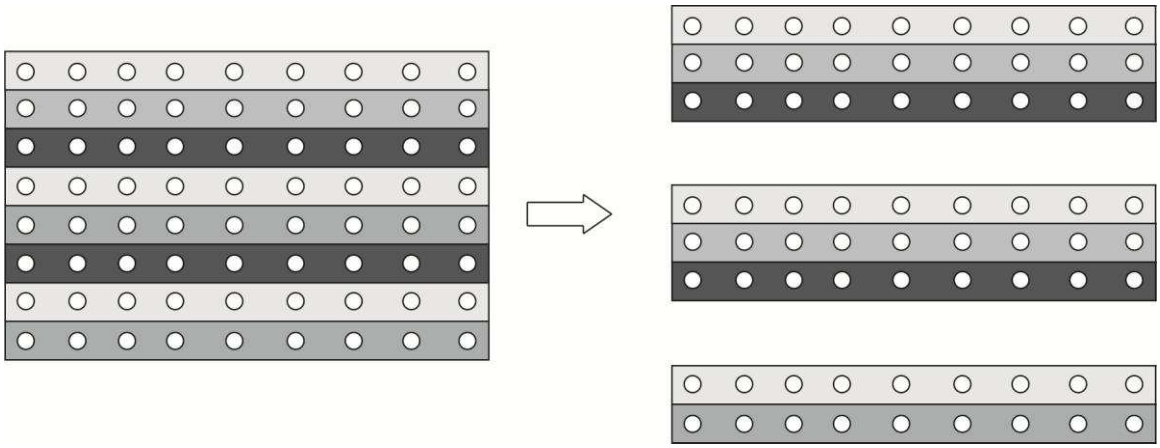

Рисунок 4 - Пример использования ленточной циклической схемы распределения строк матрицы между тремя процессорами

Распределение подзадач между процессами должно учитывать характер выполняемых в методе Гаусса коммуникационных операций. Основным видом

информационного взаимодействия подзадач является операция передачи данных от одного процесса всем процессам вычислительной системы. Как результат, для эффективной реализации требуемых информационных взаимодействий между базовыми подзадачами топология сети передачи данных должны иметь структуру гиперкуба или полного графа [1].

3.3 Используемые MPI-функции

Рассмотрим MPI-операции, используемые в работе при распараллеливании вычислений.

1. Инициализация и завершение MPI – программ.

Первой вызываемой функцией MPI должна быть функция:

```
int MPI_Init(int *argc, char **argv),
```

где

- `argc` – указатель на количество параметров командной строки,
- `argv` – параметр командой строки.

Последней вызываемой функцией MPI обязательно должна являться функция:

```
int MPI_Finalize(void).
```

Определение количества и ранга процессов.

2. Определение количество процессов

Определение количества процессов в выполняемой параллельной программе осуществляется при помощи функции:

```
int MPI_Comm_size(MPI_Comm comm, int *size),
```

где

- `comm` – коммуникатор, размер которого определяется,
- `size` – определяемое количество процессов в коммуникаторе.

3. Определение номер процесса

Для определения ранга процесса используется функция:

`int MPI_Comm_rank(MPI_Comm comm, int *rank),`

где

- `comm` – коммуникатор, в котором определяется ранг процесса,
- `rank` – ранг процесса в коммуникаторе.

4. Замер вренени

Определение времени выполнения MPI – программы.

Получение текущего времени обеспечивается при помощи функции:

`double MPI_Wtime(Void).`

5. Передача данных от одного процесса всем процессам программы

Достижение эффективного выполнения передачи данных от одного процесса всем процессам программы (широковещательная рассылка данных) может быть обеспечено при помощи функции MPI:

`int MPI_Bcast(void *buf, int count, MPI_Datatype, int root, MPI_Comm comm),`

где

- `buf`, `count`, `type` – буфер памяти с отправляемым сообщением (для процесса с рангом 0) и для приёма сообщений (для всех остальных процессов),
- `root` – ранг процесса, выполняющего рассылку данных,
- `comm` – коммуникатор, в рамках которого выполняется передача данных.

Функция MPI_Bcast осуществляет рассылку данных из буфера buf, содержащего count элементов типа type, с процесса, имеющего номер root, всем процессам, входящим в коммуникатор comm.

6. Передача данных от всех процессов одному процессу (операция редукции)

Для наилучшего выполнения действий, связанных с редукцией данных, в MPI предусмотрена функция:

```
int MPI_Reduce(void *sendbuf, void recvbuf, int count, MPI_Datatype type, MPI_Op op, int root, MPI_Comm comm),
```

где

- sendbuf – буфер памяти с отправляемым сообщением,
- recvbuf – буфер памяти для результирующего сообщения (только для процесса с рангом root),
- cout – количество элементов в сообщениях,
- type – тип элементов сообщений,
- op – операция, которая должна быть выполнена над данными,
- root – ранг процесса, на котором должен быть получен результат,
- comm – коммуникатор, в рамках которого выполняется операция.

Дополнительная операция редукции данных.

Функция MPI_Reduce обеспечивает получение результатов редукции только на одном процессе. Для получения результатов редукции данных на каждом из процессов коммуникатора необходимо использовать функцию редукции и рассылки:

```
int MPI_Allreduce(void *sendbuf, void recvbuf, int count, MPI_Datatype type, MPI_Op op, MPI_Comm comm),
```

где

- `sendbuf` – буфер памяти с отправляемым сообщением,
- `recvbuf` – буфер памяти для результирующего сообщения (только для процесса с рангом root),
- `cout` – количество элементов в сообщениях,
- `type` – тип элементов сообщений,
- `op` – операция, которая должна быть выполнена над данными,
- `comm` – коммуникатор, в рамках которого выполняется операция.

Функция `MPI_Allreduce` выполняет рассылку между процессами всех результатов операции редукции.

3.4 Общая память

При реализации параллельной программы для систем с общей памятью используется технология OpenMP. В этом случае для создания параллельной программы необходимо в последовательной версии программы расставить специальные директивы препроцессора (в циклах), которые заставят определенный участок вычисления выполняться параллельно. OpenMP предоставляет набор директив, процедур и переменных окружения, которые используются для преобразования последовательного кода в параллельный.

Для поставленной задачи при использовании OpenMP не возникает необходимости в распределении данных. В этом случае одной из основных задач является организация взаимной блокировки потоков при доступе к общим данным [4].

В программе прежде всего необходимо установить количество потоков, которые можно создать во время использования параллельной секции. Данную операцию выполняет функция

```
omp_set_num_threads(n);
```

где n – необходимое количество потоков. Далее необходимо указать, какие циклы необходимо выполнять параллельно. Для этого используется директива

```
#pragma omp parallel for ...
```

Для корректности распараллеливания необходимо все переменные, в которые насчитываются результаты промежуточных расчетов, объявлять локальными для нити (`private`). Для получения полной информации о технологии OpenMP можно обратиться к источнику [4], [5].

3.5 Результаты распараллеливания метода Гаусса

Вычисления по методу Гаусса распараллеливаются для систем с распределенной памятью по технологии MPI и для систем с общей памятью по технологии OpenMP. При проведении тестовых вычислений замерялись следующие величины:

- время работы последовательной программы (в секундах);
- время работы параллельной программы (в секундах) на различном количестве ядер (от одного до восьми).

По этим величинам вычислялись ускорение параллельной программы S_p и эффективность распараллеливания E_p по методике, описанной в п.1.

Результаты замеров времени вычислений для последовательных и параллельных программ для метода Гаусса, а также ускорение и эффективность параллельных программ приведены в таблицах 4-7.

В таблице 4 показаны результаты, полученные на вычислительной системе №1 с использованием MPI. Для этой вычислительной системы не удалось получить ускорения для параллельной программы, что может быть связано с особенностями вычислительной системы.

Таблица 4 – Результаты замеров времени метода Гаусса при помощи MPI (ВС №1)

Порядок системы			2000	5000	10000
Последовательный алгоритм			32,59	506,56	4029,39
Параллельный алгоритм	1 ядро	время	31,62	496,49	3960,69
		S_1	1,0	1,0	1,0
	2 ядра	время	28,51	447,38	3581,60
		S_2	1,1	1,1	1,1
		E_2	55%	55%	55%
	4 ядра	время	26,71	447,12	3579,63
		S_4	1,2	1,1	1,1
		E_4	30%	27,5%	27,5%
	6 ядер	время	27,34	445,70	3562,55
		S_6	1,2	1,1	1,1
		E_8	15%	18,3%	18,3%
	8 ядер	время	24,82	443,07	3537,02
		S_8	1,3	1,1	1,1
		E_8	16,25%	13,75%	13,75%

В таблице 5 показаны результаты, полученные на вычислительной системе №1 с использованием OpenMP. Полученные данные также свидетельствуют о том, на используемой вычислительной системе не удалось получить сокращения времени вычислений по методу Гаусса при применении технологий распараллеливания OpenMP.

Таблица 5 – Результаты замеров времени метода Гаусса при помощи OpenMP (ВС №1)

Порядок системы			2000	5000	10000
Последовательный алгоритм			32,59	506,56	4029,39
Параллельный алгоритм	1 ядро	время	34,15	513,41	4016,28
		S_1	1,0	1,0	1,0
	2 ядра	время	33,65	489,40	3721,24
		S_2	1,1	1,1	1,1
		E_2	55%	55%	55%
	4 ядра	время	33,12	496,35	3802,63
		S_4	1,2	1,1	1,1
		E_4	30%	27,5%	27,5%
	8 ядер	время	39,28	526,12	3885,07
		S_8	1,3	1,1	1,1
		E_8	16,25%	13,75%	13,75

В таблице 6 приведены результаты, полученные на вычислительной системе №2 с использованием MPI. Для данной вычислительной системы максимальное ускорение пдостигается при использовании четырех ядер.

Таблица 6 – Результаты замеров времени метода Гаусса при помощи MPI (ВС №2)

Порядок системы	Время последовательной программы	Параллельный алгоритм						
		1 ядро	2 ядра			4 ядра		
		время	время	S_2	E_2	время	S_4	E4
1000	13,39	15,0	7,57	1,8	90%	5,4	2,5	62,5%
2000	105,22	128,2	58,78	1,8	90%	39,3	2,7	67,5%
4000	822,32	954,8	469,59	1,8	90%	259,4	3,2	80%
5000	1600,69	1796,2	903,36	1,8	90%	528,0	3,0	75%
10000	12736,00	14304,5	7259,60	1,8	90%	4007,8	3,2	80%

В таблице 7 показаны результаты, полученные на вычислительной системе №2 с использованием OpenMP. Наибольшее ускорение достигается при использовании четырех ядер для систем меньшей размерности.

Таблица 7 – Результаты замеров времени метода Гаусса при помощи OpenMP (ВС №2)

Порядок системы	Время последовательной программы	Параллельный алгоритм						
		1 ядро	2 ядро			4 ядра		
		время	время	S_2	E_2	время	S_4	E_4
1000	3,62	7,19	4,96	1,4	70%	6,48	1,8	45%
2000	25,22	44,86	34,53	1,4	70%	48,01	1,9	47,5%
4000	193,78	262,22	230,74	1,2	60%	294,15	1,5	37,5%
5000	380,83	515,97	445,26	1,2	60%	535,57	1,4	35%
10000	2991,24	4066,47	3409,78	1,1	55%	3778,29	1,3	32,5%

Полученные результаты позволяют сделать следующий вывод. Можно отметить, что удалось сократить время расчетов в три раза при использовании четырех ядер вычислительной системы №2. Однако, несмотря на то, что время расчетов не всегда сокращается при использовании параллельных программ, с другой стороны становится возможным проведение вычислений для систем большой размерности, для которых применения последовательных программ могло быть невозможным из-за ограниченности оперативной памяти.

4 Распараллеливание итерационных методов решения систем линейных алгебраических уравнений

4.1 Параллельный алгоритм решения СЛАУ методом Якоби

При большом числе неизвестных метод Гаусса становится весьма сложным в плане вычислительных и временных затрат. Поэтому иногда удобнее использовать приближенные (итерационные) численные методы.

При распараллеливании алгоритма предполагается, что размерность системы больше числа процессов. Каждый процесс считает «свои» элементы вектора $\mathrm{X} = (x_1, x_2, x_3, \ldots, x_n)$.

Перед началом расчетов по методу Якоби в программе выполняются следующие подготовительные действия.

1. Каждый процесс знает размерность системы и максимальную погрешность ε.
2. Каждый процесс получает точное решение и начальное приближение.
3. Матрицу разбивается на «полосы», где у каждого процесса своё количество строк в полосе N_p.
4. Зная количество строк на процессе заполняем матрицу **A** так, чтобы выполнялось диагональное преобладание.
5. С помощью точного решения заполним вектор свободных членов b, так же построчно.

После получения необходимой информации каждый процесс будет вычислять соответствующие компоненты вектора X. Далее необходимо вычислять максимальную погрешность (m_p), то есть вычисляем по модулю разницу между результатами с предыдущей итерации и текущей. Когда все процессы найдут свою максимальную погрешность, при помощи функции `MPI_Allreduce` определяется глобальная максимальная погрешности (M). Кроме того, на каждой итерации производится сборка вектора X со всех

процессов для использования в следующей итерации. Перед переходом к следующей итерации выполняется проверка условия останова: $M < \varepsilon$[1].

4.2 Исследование эффективности распараллеливания метода Якоби для систем с распределенной памятью

Результаты замеров времени для параллельной MPI программы замерялись на двух вычислительных системах. В таблицах 8-9 ускорение вычисляется не относительно времени работы последовательной программы, а относительно времени работы параллельной программы на одном вычислительном узле.

В таблице 8 представлены результаты, полученные на вычислительной системе №1 для с использованием MPI. Полученные данные говорят о том, что при увеличении размерности системы растет ускорение на данной вычислительной системе. Наибольший прирост достигается при использовании восьми ядер и при размерности системы в двадцать тысяч, однако эффективность использования 8 ядер ниже, чем при использовании 4 ядер.

Таблица 8 – Время работы и ускорение параллельной программы для метода Якоби на разном количестве вычислительных узлов (ВС №1)

Размерность системы	1 ядро	2 ядра	S_2	E_2	4 ядра	S_4	E_4	8 ядер	S_8	E_8
10000	5,82	3,3	1,8	90%	2,9	2,0	50%	2,8	2,7	27%
20000	67,4	13,3	5,0	250 %	12,6	5,3	133%	8,7	7,8	108%

В таблице 9 представлены результаты, полученные на вычислительной системе № 2 с использованием MPI. Полученные данные свидетельствуют о том, что на используемой вычислительной системе получено значительное сокращение времени работы программы, в особенности при использовании 4 процессов.

Таблица 9 – Время работы и ускорение параллельной программы для метода Якоби на разном количестве вычислительных узлов (ВС №2)

Размерность системы	1 ядро	2 ядра	S_2	E_2	4 ядра	S_4	E_4
10000	14,37	7,26	2,0	100%	4,18	3,4	85%
20000	310,84	48,80	6,4	320%	53,04	5,9	147,5%
30000	927,21	659,75	1,4	70%	544,15	1,7	42,5%
40000	1539,22	1099,40	1,4	70%	834,25	1,8	45%

4.3 Особенности использования общей памяти при распараллеливании вычислений

При реализации параллельной программы на системе с общей памятью для данной задачи нет необходимости в особенном распределении данных для каждого из рассматриваемых методов.

Общее в распараллеливание итерационных методов в данной работе было заполнении массива начального приближения.

После начала работы основного цикла программы необходимо распараллелить цикл, где присваивается текущему приближению результаты, вычисленные на предыдущей итерации. После этого нужно распараллелить цикл по i, j где вычисляются значения вектора $x[i]$.

В методе Гаусса-Зейделя необходимо распараллелить только один цикл, где вычисляется вектор решений $x[i]$.

4.4 Исследование эффективности распараллеливания метода Якоби для систем с общей памятью

Вычисления по методу Якоби распараллеливаются для систем с общей памятью по технологии OpenMP. Результаты замеров времени вычислений для параллельных программ, а так же их ускорение приведены в таблицах 10-11.

В таблице 10 представлены результаты замеров времени (в секундах), на ВС№1 для вычислений по методу Якоби с использованием OpenMP, и ускорение. По полученным данным можно сказать что при применении

технологии OpenMP для метода Якоби использование 8 узлов не имеет смысла из-за разбиения памяти. Было получено максимальное ускорение равное двум, при использовании четырёх и восьми узлов. Как уже отмечалось ранее, эффективность использования восьми узлов ниже, чем эффективность использования четырех узлов. Как показывают расчеты, наибольшая эффективность распараллеливания при использовании двух узлов.

Таблица 10 – Время работы и ускорение параллельной программы для метода Якоби на разном количестве вычислительных узлов (ВС №1)

Размерность системы	Последовательная программа	Время				Ускорение и эффиктивность относительно последовательной программы			
		1 ядро	2 ядра	4 ядра	8 ядер	1 ядро	2 ядра	4 ядра	8 ядер
10000	10,17	9,10	7,18	5,07	5,04	1,1	1,4 / 70%	2 / 50%	2 / 25%
15000	22,22	20,40	12,68	11,38	11,38	1,1	1,8 / 90%	2 / 50%	2 / 25%
20000	39,63	36,67	28,80	20,29	20,37	1,1	1,4 / 70%	2 / 50%	1,9 / 24%

В таблице 11 представлены результаты, полученные на ВС№2 с использованием OpenMP. По полученным данным заметен рост ускорения при увеличении и размерности системы, и количества вычислительных узлов. Также было получено максимальное ускорение в четыре с половиной раза при использовании четырех узлов. Эффективность распараллеливания уменьшается при использовании четырех узлов по сравнению с использованием двух узлов.

Таблица 11 – Время работы и ускорение параллельной программы для метода Якоби на разном количестве вычислительных узлов (ВС №2)

Размерность системы	Последовательная программа	Время			Ускорение и эффективность относительно последовательной программы		
		1 ядро	2 ядра	4 ядра	1 ядро	2 ядра	4 ядра
1000	0,14	0,17	0,06	0,14	0,8	2,3 / 115%	1,1 / 27,5%
5000	2,63	2,47	1,31	0,91	1,1	2 / 100%	2,9 / 72,5%
10000	10,50	9,78	5,16	3,5	1,1	2 / 100%	3 / 75%
15000	40,24	28,33	13,02	9,02	1,4	3,1 / 155%	4,5 / 112,5 %

ЗАКЛЮЧЕНИЕ

По результатам данной работы были рассмотрены и изучены технологии организации параллельных вычислений. В ходе работы был рассмотрен метод Гаусса, итерационные методы Якоби и Гаусса – Зейделя.

Для решения поставленной задачи были составлены последовательные и параллельные программы с применением технологий MPI и OpenMP.

Для реализации параллельных программы с использованием технологии MPI была рассмотрена ленточная схема распределения данных между вычислительными узлами. Были проведены тесты на малом кластере СФУ и на персональном ноутбуке. Во всех тестовых расчетах было получено ускорения вычислений при использовании технологии MPI.

Для технологии OpenMP так же проводились тесты на двух вычислительных системах. Полученные результаты не однозначные, в некоторых случаях, при увеличении узлов не было достигнуто сокращение работы параллельной программы по сравнению с последовательной.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Антонов А.С. Параллельное программирование с использованием технологии MPI: учебное пособие / А. С. Антонов. – Москва: Изд-во МГУ, 2004. – 71с.
2. Гергель В.П. Теория и практика параллельных вычислений: учебное пособие / В. П. Гергель. – Москва: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 423 с.
3. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.
4. Карепова Е. Д. Основы многопоточного и параллельного программирования : учебное пособие для студентов вузов, обучающихся по направлениям "Прикладная математика и информатика" и "Фундаментальная информатика и информационные технологии" / Е. Д. Карепова ; Сиб. федер. ун-т, Ин-т математики и фундамент. информатики. – 2016
5. Левин М.П. Параллельное программирование с использованием OpenMP: учебное пособие /М. П. Левин. – М.: Интернет-Университет Информационных Технологий; БИНОМ, Лаборатория знаний, 2012. – 118 с.
6. Миллер Р. Последовательные и параллельные алгоритмы: Общий подход / Р. Миллер, Л. Боксер; Пер. с англ. – М.: БИНОМ, Лаборатория знаний, 2013. – 406с.
7. Объектно-ориентированное программирование на C++: учебник / И.В. Баранова, С.Н. Баранов, И.В. Баженова [и др.]. – Красноярск: Сиб. федер. ун-т, 2019. – 288с.

8. Распопов В.В. Численные методы: учебное пособие / В. Е. Распопов, М. М. Клунникова, В. А. Сапожников; Красноярский университет. Математический факультет. – Красноярск: Красноярский университет [КрасГУ], 2006. - 182 с.

9. Самарский А.А. Численные методы: учебное пособие / А. А. Самарский. – Москва: Наука. Главная редакция физико-математической литературы, 1989. – 432 с.

ПРИЛОЖЕНИЯ

Приложение А. Блок-схема параллельного алгоритма решения СЛАУ методом Гаусса

Используемые обозначения:

P – текущий процесс

A – матрица системы

X – точное решение

x – вычисленное решение

N – размерность системы

N_p – локальная размерность системы

m_p –локальный максимум

Приложение В. Блок-схема параллельного алгоритма решения СЛАУ методом Якоби

Используемые обозначения

X – вектор точного решения

x – вектор вычисленного решения

T – вектор начального приближения

m_p – локальный максимум

M – глобальный максимум

A – матрица системы

P – текущий процесс

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ / В.В. Шайдуров

«___» _____2020 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01 «Математика и компьютерные науки»

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Научный руководитель
кандидат физико-математических наук, доцент 22.06.2020/ Е.В. Кучунова

Выпускник 21.06.2020/ А.А. Дядяшкин

Красноярск 2020