Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра вычислительной механики

ОТЧЕТ

по лабораторным работам

Дисциплина: «Высокопроизводительные вычисления»

Вариант: 9

Выполнил:
студент группы
ОБ-09.03.01.01-41 Решетникова Е.А.

Принял:
доцент, канд. физ.-мат. наук Новиков А.К.

Ижевск, 2024

Лабораторная работа №1

«Оптимизация программ средствами компилятора.»

Цель работы

Приобретение навыков: оптимизации программ средствами компилятора и преобразования циклов, и оценки производительности программ. Объектом исследования является исходный и исполняемый код программ, реализующих операции вычислительной линейной алгебры, отличающиеся арифметической сложностью и интенсивностью.

Задание

1. Проанализировать заданные алгоритмы и оценить их арифметическую интенсивность.
2. Реализовать заданные алгоритмы на языке программирования C++ (компилятор GNU GCC).
3. Используя созданные программы определить время их выполнения, время записи данных в память, объем этих данных.
4. По полученным данным о выполнении программы построить модель Roofline для вычислительной системы.
5. Оценить производительность программ, построенных с ключами оптимизации `-O0`, `-O1`, `-O2`, `-O3`.

Теоретическая часть

Высокопроизводительные вычисления предполагают оптимизацию программного кода, как на уровне исходного, так и исполняемого кода.

Обмен или «перестановка циклов» — перестановка внутренних и внешних циклов, в случае вложенных циклов. Такое преобразование может улучшить локализацию доступа к данным.

Оптимизация исполняемого кода регулируется опциями (ключами) оптимизации компилятора. Опции могут группироваться по целям оптимизации (см. таблицу 1) и включать в себя несколько (десятков) более низкоуровневых опций [2].

Таблица 1 — Опции оптимизации компилятора GNU GCC g++.

Опция	Цель оптимизации	Время выполнения	Размер кода	Использование памяти	Время компиляции
-O0	Время компиляции	++	++	-	-
-O1 или -O	Размер кода и время выполнения	0	0	0	0
-O2	Время выполнения и размер кода	0	+	+	+
-O3	Время вып. размера кода	-	+	++	+++

Здесь + — увеличение, ++ — большее увеличение, +++ — наибольшее увеличение, - — уменьшение, -- — большее уменьшение, --- — наибольшее уменьшение.

В модели Roofline производительность P вычислительной системы определяется следующим образом:

$$P = \begin{cases} B \cdot I, & 0 < I \leqslant I_*, \\ P_{max}, & I_* \leqslant I < \infty, \end{cases} \tag{1}$$

где: P_{max} — максимальная производительность вычислительной системы, GFLOP/s; B — пропускная способность подсистемы памяти, Gbyte/s; I — арифметическая интенсивность, FLOP/byte. $I_* = P_{max}/B$ — пороговая арифметическая интенсивность, начиная с которой производительность данной вычислительной системы ограничивается производительностью процессора.

Практическая часть №1.

1. Определить арифметическую интенсивность (4)

$$\gamma = (x, y) = \sum_{i=1}^{N} x_i y_i \quad x, y \in \mathbb{R}^N, \gamma \in \mathbb{R} \tag{2}$$

   Полагаем, что длины векторов принадлежат множеству $\{32 \cdot (10^3 + v), 32 \cdot (10^5 + v), 32 \cdot (10^6 + v)\}$, где v — номер варианта.

2. Реализовать (4) на языке C++ в виде теста производительности, показать в отчете фрагмент кода.

3. Выполнить вычислительные эксперименты на одном узле кластера из миникомпьютеров Raspberry Pi 4 Model B.

4. Построить Roofline модель по максимальным значениям производительности и пропускной способности.

5. Построить графики зависимости производительности от арифметической интенсивности для случаев: оптимизации средствами компилятора (опции `O0`, `O1`, `O2`, `O3`).

Для построения Roofline модели одного миникомпьютера Raspeberry Pi 4 Model B Rev. 1.4 полагаем, что вычисления осуществляются одним ядром процессора, подсистема памяти работает в одноканальном режиме.

Таким образом для вычислений с одинарной точностью теоретические значения P_{max} составят 3 GFLOP/s, пропускной способности памяти B — 12,8 Gbyte/s; для вычислений с двойной точностью теоретические значения P_{max} составят 1,5 GFLOP/s, пропускной способности памяти B — 6,4 Gbyte/s.

Фрагменты вариантов исходного кода для (4).

```
1 begin=omp_get_wtime();
2
3       gamma=0;
4       for(int  i=0;i<N;i++){
5       gamma += (x[i]*y[i]);
6       }
7 end = omp_get_wtime();
```

Листинг 1 — Базовый вариант

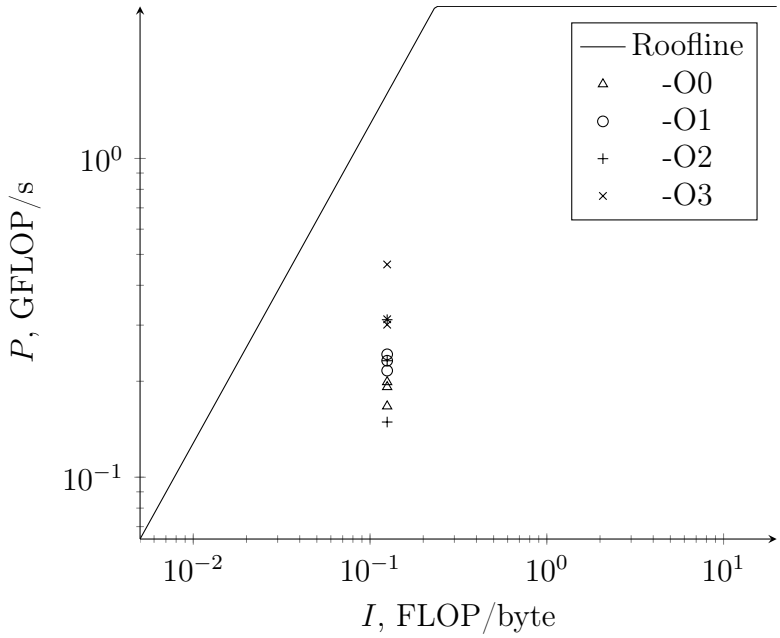

Рисунок 1 — Зависимости производительности от арифметической интенсивности для (4).

Практическая часть №2.

1. Определить арифметическую интенсивность (5) произведения матриц

$$C = A \cdot B, \quad C \in \mathbb{R}^{N \times N}, A \in \mathbb{R}^{N \times N}, B \in \mathbb{R}^{N \times N}. \tag{3}$$

Полагаем, что размеры матриц принадлежат множеству $\{32 \cdot (10+v) \times 32 \cdot (10+v), 32 \cdot (20+v) \times 32 \cdot (20+v), 32 \cdot (40+v) \times 32 \cdot (40+v)\}$, где v — номер варианта.

2. Реализовать (5) на языке C++ в виде теста производительности, показать в отчете фрагмент кода.

3. Выполнить вычислительные эксперименты на одном узле кластера из миникомпьютеров Raspberry Pi 4 Model B.

4. Построить Roofline модель по максимальным значениям производительности и пропускной способности.

5. Построить графики зависимости производительности от арифметической интенсивности для случаев: оптимизации средствами компилятора (опции `O0`, `O1`, `O2`, `O3`).

Фрагменты вариантов исходного кода для (5).

```
1  begin = omp_get_wtime();
2
3   for(int i=0;i<M;i++){
4     for(int j=0;j<M;j++){
5       C[i*M+j]=0;
6     }
7   }
8  // ijk
9   for(int i=0;i<M;i++){
10    for(int j=0;j<M;j++){
11      for(int k=0;k<N;k++){
12          C[i*M+j]+=A[i*N+k]*B[k*M+j];
13      }
14    }
15  }
```

Листинг 2 — Вариант (5) в виде «ijk»

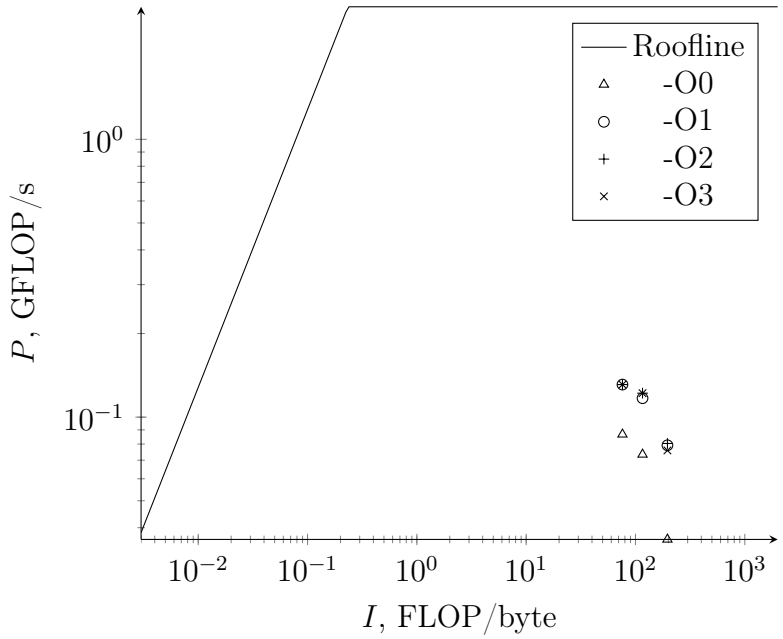

Рисунок 2 — Зависимости производительности от арифметической интенсивности (5) в варианте «ijk».

Выводы

1. Арифметическая интенсивность вычисления (4) составляет $I_1 = 0.125$ FLOP/byte, а для (5) — $I_2 = \dfrac{N}{2N_b} \in \{76, 116, 196\}$ FLOP/byte [1,5], $N_b = 4$ байта при одинарной точности вычислений.

2. При выполнении работы на языке C++ реализованы (4) и (5) в виде теста производительности, фрагменты кода представлены в отчете. Выполнены вычислительные эксперименты на одном узле кластера из миникомпьютеров Raspberry Pi 4 Model B при $N_1 \in \{32288, 3200288, 32000288\}$, $N_2 \times N_2 \in \{608 \times 608, 928 \times 928, 1658 \times 1568\}$ и вычислениях с одинарной точностью.

3. Получены максимальные значения производительности и пропускной способности для (4) и (5), которые составили 0.465 GFLOP/s и 1.16 Gbyte/s (для оптимизации -O3) для (4), 0.131 GFLOP/s и 1.02 Gbyte/s (для оптимизации -O3) для (5).

4. С применением максимальных значений производительности и пропускной способности, построены Roofline модели для (4) и (5). Построены графики (рисунки 1,2) зависимости производительности от арифметической интенсивности для случаев: оптимизации средствами компилятора (опции `O0`, `O1`, `O2`, `O3`).

5. По полученным результатам можно сделать вывод, что при вычислении (4) производительность ограничивается пропускной способностью памяти, а для (5) — производительностью процессора.

Список литературы

1. Копысов С.П., Кузьмин И.М., Недожогин Н.С. Масштабируемые вычисления для гетерогенных платформ: учебное пособие. — Ижевск: Издательский центр «Удмуртский университет», 2020. — 272 с.

2. Optimize Options (Using the GNU Compiler Collection (GCC)) [Электронный ресурс]. URL: `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Optimize-Options` Дата обращения 11.02.2024.

3. compile c++ gcc online [Электронный ресурс]. URL: `https://rextester.com/l/cpp_online_compiler_gcc` Дата обращения 11.02.2024.

4. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. М.: «Мир». 1991. — 367 с.

5. Nichols D. Understanding the Roofline Model [Электронный ресурс]. URL: `https://dando18.github.io/posts/2020/04/02/roofline-model` Дата обращения 25.03.2024.

Лабораторная работа №2

«Оптимизация циклов в исходном коде.»

Цель работы

Приобретение навыков: оптимизации при помощи и преобразования циклов, и оценки производительности программ. Объектом исследования является исходный и исполняемый код программ, реализующих операции вычислительной линейной алгебры, отличающиеся арифметической сложностью и интенсивностью.

Задание

1. Проанализировать заданные алгоритмы и оценить их арифметическую интенсивность.
2. Реализовать заданные алгоритмы на языке программирования C++ (компилятор GNU GCC).
3. Используя созданные программы определить время их выполнения, время записи данных в память, объем этих данных.
4. По полученным данным о выполнении программы построить модель Roofline для вычислительной системы.
5. Преобразовать циклы в исходном коде по заданию и выполнить для созданного кода п.5.
6. Оценить производительность программ, построенных с ключом оптимизации `O0`.

Теоретическая часть

Высокопроизводительные вычисления предполагают оптимизацию программного кода, как на уровне исходного, так и исполняемого кода.

Наиболее распространенный способ оптимизации исходного кода состоит в применении последовательности различных преобразований циклов [1], которые включают: разделение (loop fission) или распределение (loop distribution) цикла; слияние или объединение (loop fusion, loop combining) циклов; обмен или перестановка (loop interchange, loop permutation); инверсию цикла (loop inversion); инвариантное к циклу движение кода (loop invariant code motion); реверсию; перекос цикла (loop skewing); блочную обработку (loop tiling, loop blocking, loop nest optimization (LNO)); развертывание цикла (loop unrolling); вынос ветвления из тела цикла (loop unswitching).

В этой работе используются развертывание и перестановка циклов. «Развертывание цикла» дублирует тело цикла несколько раз, чтобы уменьшить количество проверок условия цикла и количество переходов, которые могут снизить производительность из-за нарушения конвейера инструкций. Полное развертывание цикла устраняет все накладные расходы (за исключением

выборки нескольких инструкций и увеличения времени загрузки программы), но требует, чтобы количество итераций было известно во время компиляции (за исключением случая компиляции Just-in-time).

Обмен или «перестановка циклов» — перестановка внутренних и внешних циклов, в случае вложенных циклов. Такое преобразование может улучшить локализацию доступа к данным.

Практическая часть №1.

1. Определить арифметическую интенсивность (4)

$$\gamma = (x, y) = \sum_{i=1}^{N} \quad x, y \in \mathbb{R}^N, \gamma \in \mathbb{R} \tag{4}$$

2. Реализовать (4) на языке C++ в виде теста производительности, показать в отчете фрагмент кода.

3. Выполнить вычислительные эксперименты на одном узле кластера из миникомпьютеров Raspberry Pi 4 Model B.

4. Построить Roofline модель по максимальным значениям производительности и пропускной способности.

5. Построить графики зависимости производительности от арифметической интенсивности для случаев: оптимизации средствами компилятора (опции `O0`, `O1`, `O2`, `O3`) и применения «развертывания циклов» с шагами 2,4 в исходном коде программы.

Для построения Roofline модели одного миникомпьютера Raspeberry Pi 4 Model B Rev. 1.4 полагаем, что вычисления осуществляются одним ядром процессора, подсистема памяти работает в одноканальном режиме.

Таким образом для вычислений с одинарной точностью теоретические значения P_{max} составят 3 GFLOP/s, пропускной способности памяти B — 12,8 Gbyte/s; для вычислений с двойной точностью теоретические значения P_{max} составят 1,5 GFLOP/s, пропускной способности памяти B — 6,4 Gbyte/s.

Фрагменты вариантов исходного кода для (4).

```
1 begin = omp_get_wtime();
2       gamma=0;
3       #pragma GCC unroll 2
4       for(int i=0;i<N;i++){
5       gamma+=x[i];
6       }
7 end = omp_get_wtime();
```

Листинг 3 — Вариант развертывания на 2 итерации (unroll 2)

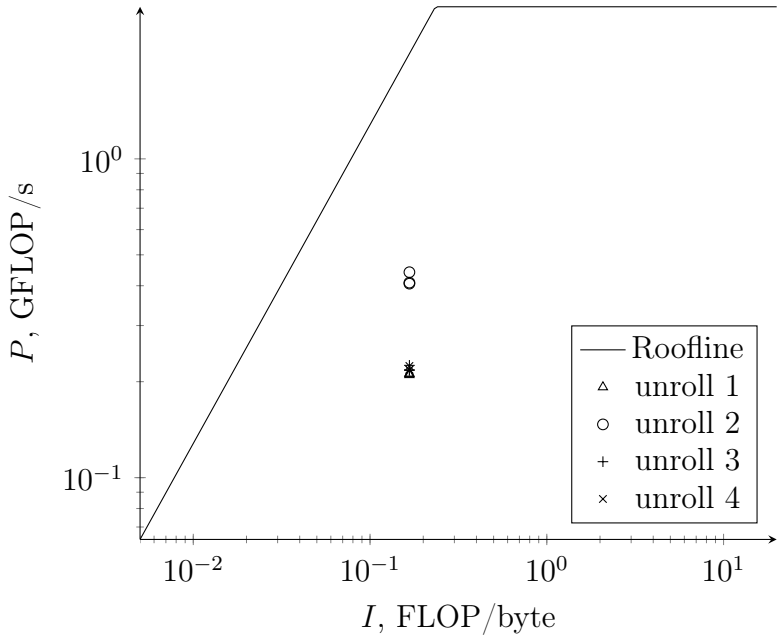

Рисунок 3 — Зависимости производительности от арифметической интенсивности для (4) в базовом варианте и при развертывании на 2–4 итерации.

Практическая часть №2.

1. Определить арифметическую интенсивность (5) произведения матриц

$$C = A \cdot B, \quad C \in \mathbb{R}^{M \times M}, A \in \mathbb{R}^{M \times N}, B \in \mathbb{R}^{N \times M}. \tag{5}$$

2. Реализовать (4) на языке C++ в виде теста производительности, показать в отчете фрагмент кода.

3. Выполнить вычислительные эксперименты на одном узле кластера из миникомпьютеров Raspberry Pi 4 Model B.

4. Построить Roofline модель по максимальным значениям производительности и пропускной способности.

5. Построить графики зависимости производительности от арифметической интенсивности для случаев: оптимизации средствами компилятора (опции `O0`, `O1`, `O2`, `O3`) и применения «перестановки циклов» в исходном коде программы.

Фрагменты вариантов исходного кода для (5).

```
1 begin = omp_get_wtime();
2
3  for(int i=0;i<M;i++){
4    for(int j=0;j<M;j++){
5        C[i*M+j]=0;
```

```
6       }
7     }
8   //  ijk
9     for(int  i=0;i<M; i++){
10      for(int  j=0;j<M; j++){
11        for(int  k=0;k<N; k++){
12              C[i*M+j]+=A[i*N+k]*B[k*M+j];
13          }
14        }
15    }
16
17  end = omp_get_wtime();
```

Листинг 4 — Вариант (5) в виде «ijk»

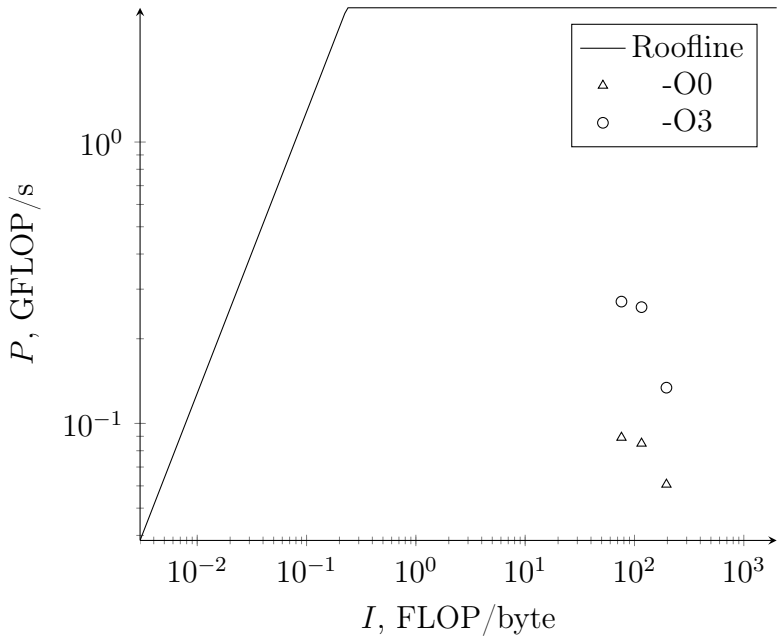

Рисунок 4 — Зависимости производительности от арифметической интенсивности (5) в варианте «ijk».

Выводы

1. Арифметическая интенсивность вычисления (4) составляет $I_1 = 0.167$ FLOP/byte, а для (5), полагая, что все матрицы загружаются и хранятся в памяти [5] — $I_2 = \dfrac{N_2 \cdot M}{4(M + N_2)} \in \{76, 116, 196\}$ FLOP/byte.

2. При выполнении работы на языке C++ реализованы (4) и (5) в виде теста производительности, фрагменты кода представлены в отчете. Выполнены вычислительные эксперименты на одном узле кластера из миникомпьютеров Raspberry Pi 4 Model B при $N_1 \in \{32288, 3200288, 32000288\}$, $M \times N_2 \in \{608 \times 608, 928 \times 928, 1568 \times 1568\}$ и вычислениях с одинарной точностью.

3. Получены максимальные значения производительности и пропускной способности для (4) и (5), которые составили 0.441 GFLOP/s и 0.352 Gbyte/s (для оптимизации -O2 при unroll 1), и 0.271 GFLOP/s и 1.15 Gbyte/s (для оптимизации -O3 в варианте «ijk»).

4. С применением максимальных значений производительности и пропускной способности, построены Roofline модели для (4) и (5). Построены графики (рисунки 3, 4) зависимости производительности от арифметической интенсивности для случаев: оптимизации средствами компилятора (опции `O0`, `O1`, `O2`, `O3`), и применения «развертывания циклов» и «перестановки циклов».

5. По полученным результатам можно сделать вывод, что при вычислении (4) производительность ограничивается пропускной способностью памяти, а для (5) — производительностью процессора.

Список литературы

1. LaForest E. ECE 1754 Survey of Loop Transformation Techniques [Электронный ресурс]. URL: `http://fpgacpu.ca/writings/SurveyLoopTransformations.pdf` Дата обращения 11.02.2024.
2. Optimize Options (Using the GNU Compiler Collection (GCC)) [Электронный ресурс]. URL: `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Optimize-Options` Дата обращения 11.02.2024.
3. compile c++ gcc online [Электронный ресурс]. URL: `https://rextester.com/l/cpp_online_compiler_gcc` Дата обращения 11.02.2024.
4. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. М.: «Мир». 1991. — 367 с.
5. Nichols D. Understanding the Roofline Model [Электронный ресурс]. URL: `https://dando18.github.io/posts/2020/04/02/roofline-model` Дата обращения 25.03.2024.

Лабораторная работа №3

«Параллельные вычисления в модели общей памяти.»

Цель работы

Приобретение навыков: распараллеливания программ и оценки ускорения, эффективности и производительности параллельных алгоритмов в модели общей памяти. Объектом исследования является исходный и исполняемый код программ, реализующих операции вычислительной линейной алгебры, отличающиеся арифметической сложностью и интенсивностью.

Задание

1. Реализовать заданные алгоритмы на языке программирования C++ с использованием параллельных технологий программирования OpenMP.

2. Используя созданные программы определить время выполнения и программ при вычислениях на заданном числе процессорных ядер и потоков/процессов.

3. Оценить ускорение и эффективность параллельных вычислений, производительность программ, используя Roofline модель.

Теоретическая часть

Параллельные вычисления являются одним из основных инструментов получения высокой производительности вычислений, которая достигается распределением вычислений между вычислительными ресурсами: кластерами, модулями, процессорами, процессорными ядрами [1]. Взаимодействия параллельных потоков реализуются в рамках многопоточного программирования, использующего модель общей памяти (общего адресного пространства).

Многопоточное программирование не обеспечивает взаимодействие параллельных процессов, когда данные находятся разных адресных пространствах. В этом случае, для взаимодействия параллельных процессов в основном применяется модель передачи сообщений. Многопоточное программирование реализуется и средствами языка, и в виде внешних библиотек, как, например, технология OpenMP [2], обеспечивающая межплатформенную переносимость исходного кода и его высокоуровневое (циклов) и низкоуровневое (циклов, блоков кода, функций) распараллеливание.

Параллельные вычисления характеризуются понятиями ускорение и эффективность.

Ускорение S_p параллельных вычислений определяется следующим образом:

$$S_p = \frac{T_1}{T_p}, \tag{6}$$

где T_1 — время выполнения одним процессом/потоком, T_p — время выполнения p процессами/-потоками.

Эффективность E_p определяется как отношение:

$$E_p = \frac{S_p}{p} \cdot 100\%, \tag{7}$$

Ускорение показывает во сколько раз параллельные вычисления выполнены быстрее последовательных, а эффективность — насколько при этом были задействованы вычислительные ресурсы, т. к. полагается равным числу использованных процессоров или процессорных ядер.

Закон Амдала (1967 г.) описывает изменение ускорения параллельных вычислений при решении задачи фиксированного размера при увеличении числа используемых (ядер) процессоров:

$$S_p = \frac{1}{f + \dfrac{1-f}{p}} = \frac{p}{f \cdot (p-1) + 1}, \tag{8}$$

где $0 \leqslant f \leqslant 1$ — доля последовательных вычислений; p — число используемых (ядер) процессоров. Масштабирование времени в этом случае называется сильным.

Если размер задачи увеличивается пропорционально числу (ядер) процессоров, то ускорение изменяется согласно закону Густафсона–Барсиса (1988 г.):

$$S_p = p - f \cdot (p-1). \tag{9}$$

Как следствие, задача большего размера может быть решена за то же время с использованием большего числа (ядер) процессоров. Масштабирование времени в этом случае называется слабым.

Наиболее затратной частью программ, подвергающихся распараллеливанию, являются циклы [3,4]. Циклы применяются при поиске экстремальных значений в структурах данных, суммировании данных, в других операциях вычислительной линейной алгебры. В технологии OpenMP реализовано несколько способов распараллеливания циклов:

1. «Низкоуровневое», в котором итерации цикла разделяются в зависимости от номера потока;

2. С применением директивы `#pragma omp for` внутри параллельной области или ее краткой формы `#pragma omp parallel for`;

3. С применением `#pragma omp sections` внутри параллельной области и ее краткой формы `#pragma omp parallel sections`, но данный способ чаще применяется к распараллеливанию блоков программного кода.

В данной работе применяется второй способ распределения итераций циклов. В директиве `#pragma omp for` может задаваться опция `schedule(type,chunk)`, где `type` — порядок распределения итераций, а необязательный параметр `chunk` — число итераций, приходящихся на поток. Параметр `type` может принимать значения `static`, `dynamic`, `guided`, `auto`, `runtime` [2,4,6].

Практическая часть.

Реализовать задания из Лабораторной работы №1, используя технологии параллельного программирования OpenMP для модели общей памяти:

1. Выполнить вычислительные эксперименты на кластере из миникомпьютеров.
2. Построить графики ускорения и эффективности в зависимости от числа вычислительных потоков/процессов.
3. Построить графики зависимости производительности от арифметической интенсивности, ограниченные Roofline ломанной.

Рассмотреть случаи оптимизации исполняемого кода средствами компилятора (опции -O0,-O3).

Аппаратное обеспечение работы: кластер из миникомпьютеров. Кластер состоит из 8 миникомпьютеров Raspeberry Pi 4 Model B Rev. 1.4, соединенных сетью Gigabit Ethernet. Каждый миникомпьютер оснащен 4-ядерным ARM процессором (ядра Cortex-A72 (ARM v8) с частотой 1,5 GHz) и 8 GB оперативной памяти LPDDR4-3200.

Программное обеспечение: Операционная система 64-разрядная Ubuntu 20.04 LTS, компилятор GNU g++ 10.3.0, библиотеки Open MPI 4.0.3.

Практическая часть №1.

На листинге 5 представлен фрагмент исходного кода на языке C++ с применением распараллеливания циклов в технологии OpenMP для параллельного вычисления $\gamma = (x, y),\ x, y \in \mathbb{R}^N, \gamma \in \mathbb{R}$, обозначено (1).

```
1  begin = omp_get_wtime();
2
3  gamma = 0.0;
4
5  omp_set_num_threads(threads);
6  #pragma omp parallel
7  {
8  #pragma omp for reduction(+:gamma)
9      for (int i = 0; i < N; i++) {
10         gamma += (x[i] * y[i]);
11     }
12 }
13
14 end = omp_get_wtime();
```

Листинг 5 — Фрагмент кода на C++, реализующий (1) с применением OpenMP

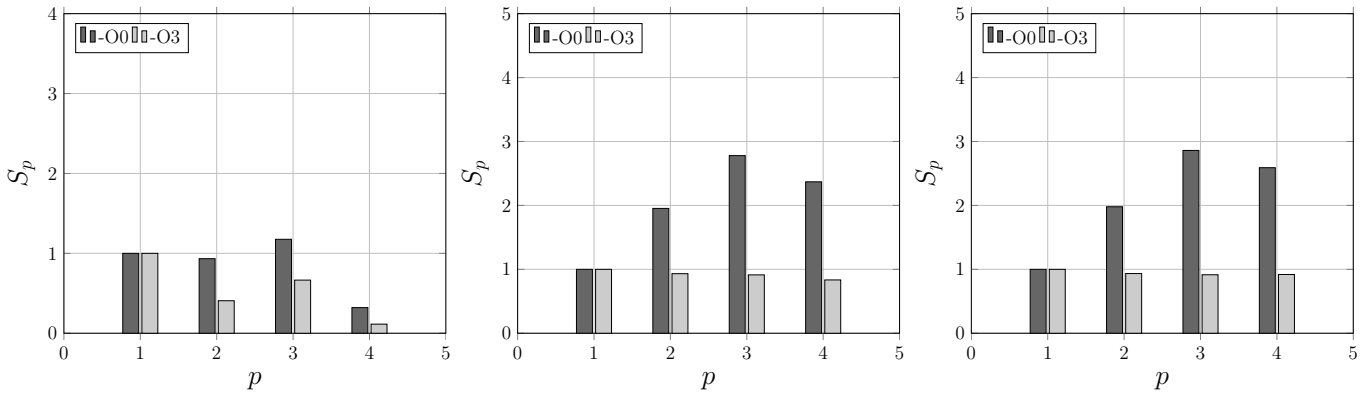

Рисунок 5 — Ускорение $S_p(p)$ параллельных вычислений, технология OpenMP, при $N \in \{32288, 3200288, 32000288\}$

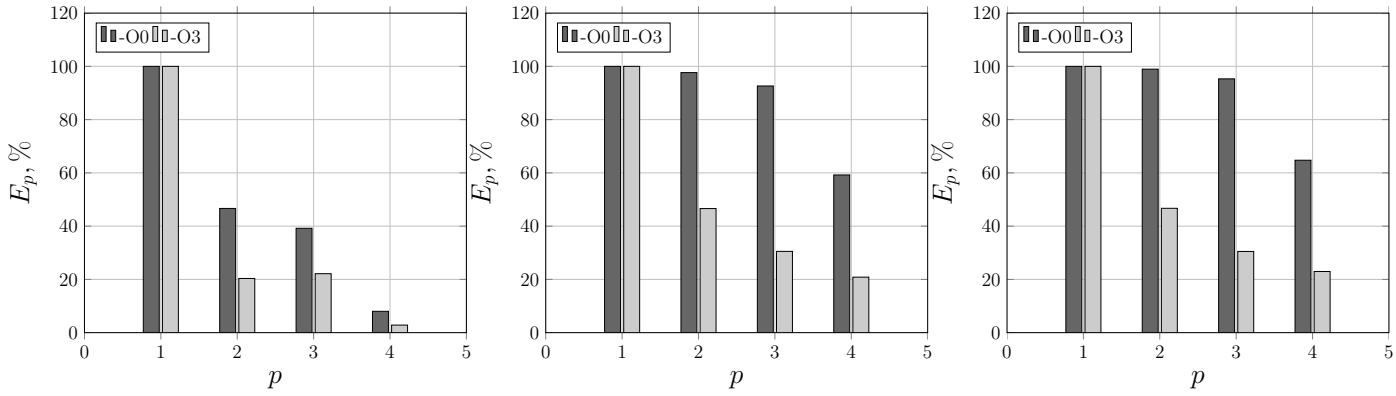

Рисунок 6 — Эффективность $E_p(p)$ параллельных вычислений, %, технология OpenMP при $N \in \{32288\,3200288, 32000288\}$

Практическая часть №2.

На листинге 6 представлен фрагмент исходного кода на языке C++ с применением распараллеливания циклов в технологии OpenMP для параллельного вычисления $C = A \cdot B$ в виде «ikj», обозначено (2), где $A \in \mathbb{R}^{M \times N}$, $B \in \mathbb{R}^{N \times M}$, $C \in \mathbb{R}^{M \times M}$.

```
1  begin = omp_get_wtime();
2  #pragma omp parallel
3  {
4  #pragma omp for
5    for(int i=0;i<M;i++){
6      for(int j=0;j<M;j++){
7        C[i*M+j]=0;
8      }
9    }
10 // ijk
11 #pragma omp for
```

```
12   for(int  i=0;i<M; i++){
13     for(int  j=0;j<M; j++){
14       for(int  k=0;k<N;k++){
15           C[i*M+j]+=A[i*N+k]*B[k*M+j];
16       }
17     }
18   }
19 }
20 end = omp_get_wtime();
```

Листинг 6 — Фрагмент кода на C++, реализующий (2) с применением OpenMP

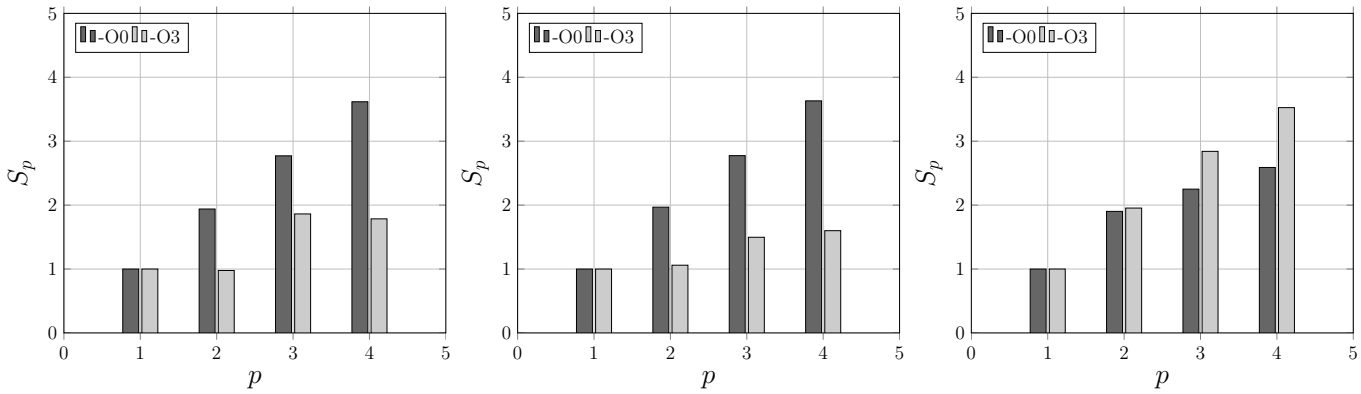

Рисунок 7 — Ускорение $S_p(p)$ параллельных вычислений, технология OpenMP, при $N \times N \in \{608 \times 608, 928 \times 928, 1568 \times 1568\}$

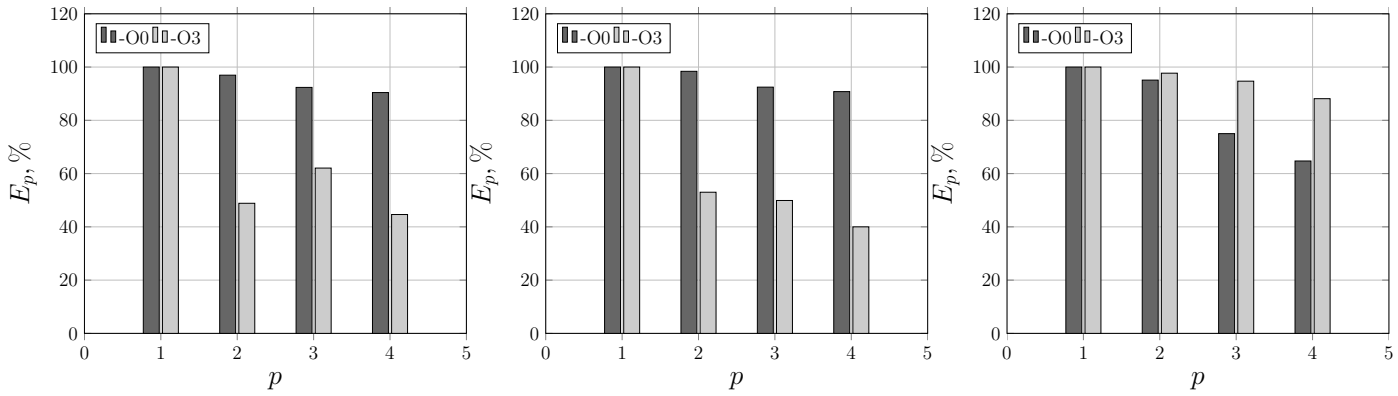

Рисунок 8 — Эффективность $E_p(p)$ параллельных вычислений, %, технология OpenMP при $N \times N \in \{608 \times 608, 928 \times 928, 1568 \times 1568\}$

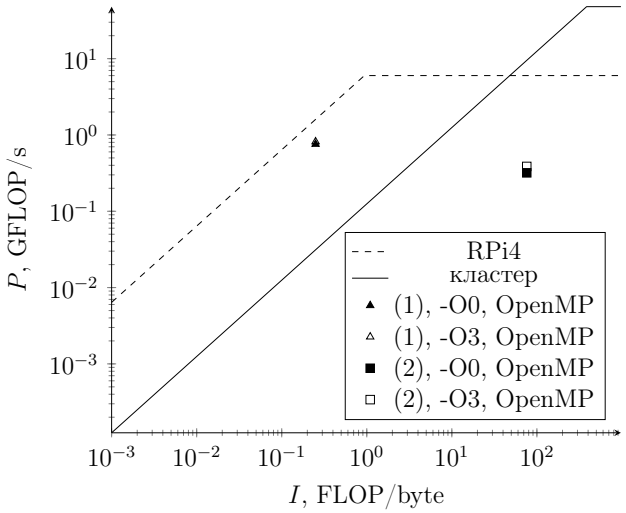

Рисунок 9 — Производительность параллельных вычислений для заданий (1) и (2) с применением технологии OpenMP, и Roofline модели кластера и миникомпьютера (RPi4).

Выводы

1. По результатам вычислительных экспериментов на кластере из миникомпьютеров Raspeberry Pi 4, определены ускорение и эффективность параллельного вычисления (1) для векторов размерности 32288, 3200288, 32000288 (рисунки 5 и 6), и параллельного вычисления (2) — произведения двух вещественных матриц трех размеров: 608×608, 928×928 и 1568×1568 (рисунки 7 и 8), определена максимальная производительность для (1) и (2) (рисунок 9).

2. Максимальные ускорения и эффективность составили: 3.631 и 98.4% для 4 и 2 вычислительных потоков при распараллеливании циклов в технологии OpenMP при размере матрицы 928×928.

3. Получены максимальные значения производительности, в случае применения OpenMP для вычисления (1) — 0.816 GFLOP/s и (2) — 0.387 GFLOP/s, при этом, пиковая производительность кластера (теоретическая) для вычислений с одинарной точностью составляет 96 GFLOP/s (8 миникомпьютеров, по 4х-ядерному процессору в каждом, тактовая частота 1.5 ГГц и 2 арифметических операции с плавающей точкой за цикл).

Список литературы

1. Копысов С.П., Новиков А.К. Промежуточное программное обеспечение параллельных вычислений. — Ижевск: Изд-во «Удмуртский университет». 2012. — 140 с.

2. OpenMP. [Электронный ресурс]. URL: `https://www.openmp.org`. Дата обращения 21.03.2024.

3. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. — М.: «Мир». 1991. — 367 с.

4. Малявко А.А. Параллельное программирование на основе технологий OpenMP, CUDA, OpenCL, MPI: учеб. пособие для вузов. — М.: Изд-во Юрайт, 2022. — 135 с.

5. Копысов С. П., Кузьмин И. М., Недожогин Н. С. Масштабируемые вычисления для гетерогенных платформ: учебное пособие. — Ижевск: Издательский центр «Удмуртский университет», 2020. — 272 с.

6. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: учебное пособие. — М.: Изд-во МГУ, 2009. — 77 с.

Лабораторная работа №4

«Параллельные вычисления в модели передачи сообщений.»

Цель работы

Приобретение навыков: распараллеливания программ и оценки ускорения, эффективности и производительности параллельных алгоритмов в модели передачи сообщений. Объектом исследования является исходный и исполняемый код программ, реализующих операции вычислительной линейной алгебры, отличающиеся арифметической сложностью и интенсивностью.

Задание

1. Реализовать заданные алгоритмы на языке программирования C++ с использованием параллельных технологии программирования MPI.

2. Используя созданные программы определить время выполнения и программ при вычислениях на заданном числе процессорных ядер и процессов.

3. Оценить ускорение и эффективность параллельных вычислений, производительность программ, используя Roofline модель.

Теоретическая часть

Параллельные вычисления являются одним из основных инструментов получения высокой производительности вычислений, которая достигается распределением вычислений между вычислительными ресурсами: кластерами, модулями, процессорами, процессорными ядрами [1].

Модель передачи сообщений осуществляет взаимодействие параллельных процессов через передачу данных в виде сообщений, которая реализуется к в разделенной, так и в общей памяти. Модель передачи сообщений реализуется в виде внешних библиотек, как например, MPI [3], на уровне стандартов обеспечивает переносимость исходного кода и низкоуровневое распараллеливание при помощи коммуникационных функций.

Передача сообщений в MPI осуществляется вызовом коммуникационных функций, подразделяющихся на функции «точка-точка» и коллективные, в которых участвуют соответственно два и более процессов. В коллективных коммуникациях участвуют все процессы данного коммуникатора (описатель группы процессов) и они не взаимодействуют с функциями «точка-точка». Коллективные функции, кроме непосредственно передачи данных, могут выполнять синхронизацию и вычисления, распределенные между процессами коммуникатора. Функции «точка-точка» осуществляют только передачу данных.

Практическая часть.

Реализовать задания (1) и (2) из Лабораторной работы №2, используя технологии параллельного программирования OpenMP для модели общей памяти и MPI для модели передачи сообщений:

Выполнить вычислительные эксперименты на кластере из миникомпьютеров.

Построить графики ускорения и эффективности в зависимости от числа вычислительных потоков/процессов. Построить графики зависимости производительности от арифметической интенсивности, ограниченные Roofline ломанной. Рассмотреть случаи оптимизации исполняемого кода средствами компилятора (опции -O0,-O3).

Аппаратное обеспечение работы: кластер из миникомпьютеров. Кластер состоит из 8 миникомпьютеров Raspeberry Pi 4 Model B Rev. 1.4, соединенных сетью Gigabit Ethernet. Каждый миникомпьютер оснащен 4-ядерным ARM процессором (ядра Cortex-A72 (ARM v8) с частотой 1,5 GHz) и 8 GB оперативной памяти LPDDR4-3200.

Программное обеспечение: Операционная система 64-разрядная Ubuntu 20.04 LTS, компилятор GNU g++ 10.3.0, библиотеки Open MPI 4.0.3.

Практическая часть №1.

На листинге 7 представлен фрагмент исходного кода на языке C++ с применением распараллеливания циклов в технологии MPI для параллельного вычисления $\gamma = (x, y),\ x, y \in \mathbb{R}^N, \gamma \in \mathbb{R}$, обозначено (1).

```
1  MPI_Barrier(world);
2  begin=MPI_Wtime();
3        gamma=0.0;
4  int  Ni=N/np;
5        for(int  i=Ni*id;i<(Ni*(id+1));i++){
6        gamma += (x[i]*y[i]);
7        }
8        MPI_Allreduce(&gamma,&gamma_sum,1,mpireal,MPI_SUM,world);
9  MPI_Barrier(world);
10 end = MPI_Wtime();
```

Листинг 7 — Фрагмент кода на C++, реализующий (1) с применением MPI

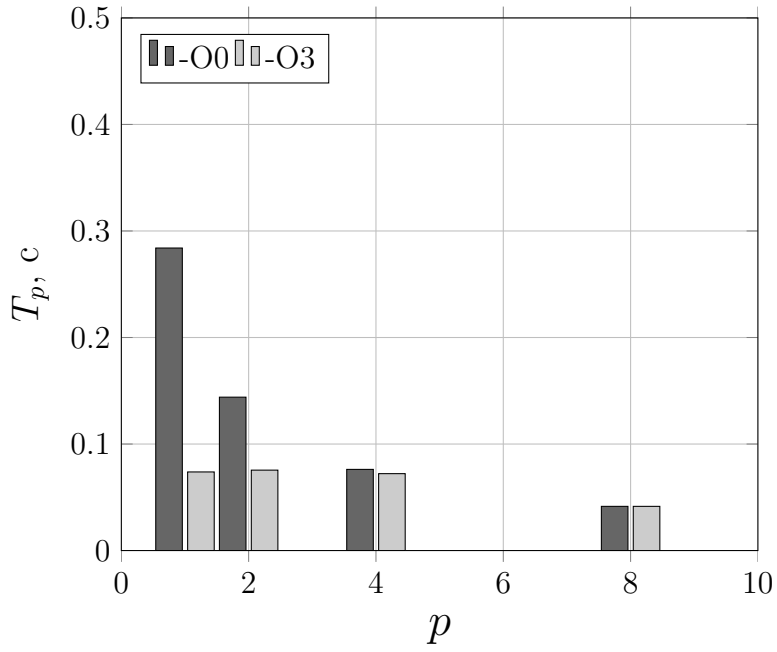

Рисунок 10 — Время $T_p(p)$ параллельных вычислений, %, технология MPI при $N = 32000288$

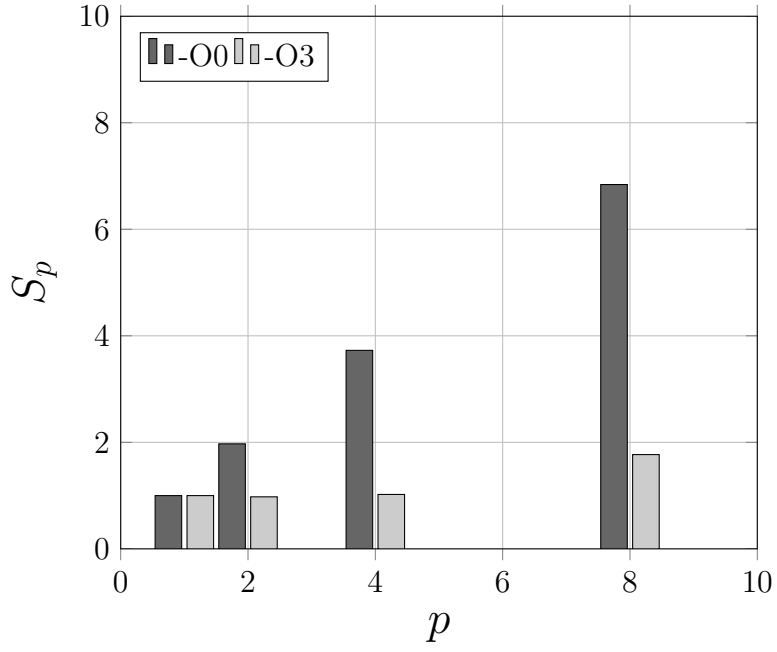

Рисунок 11 — Ускорение $S_p(p)$ параллельных вычислений, технология MPI, при $N = 32000288$

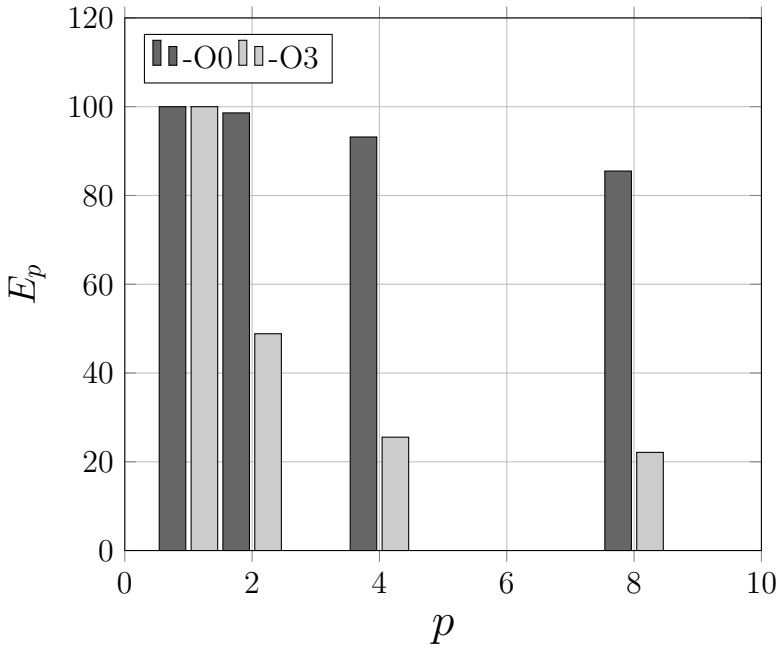

Рисунок 12 — Эффективность $E_p(p)$ параллельных вычислений, %, технология MPI при $N = 32000288$

Практическая часть №2.

На листинге 8 представлен фрагмент исходного кода на языке C++ с применением распараллеливания циклов в технологии MPI для параллельного вычисления $C = A \cdot B$ в виде «ijk», обозначено (2), где $A \in \mathbb{R}^{M \times N}$, $B \in \mathbb{R}^{N \times M}$, $C \in \mathbb{R}^{M \times M}$.

```
1  begin=MPI_Wtime();
2  if(!id)
3  {
4      for(int i=1;i<np;i++){
5          MPI_Send(&M,1,MPI_INT,i,0,world);
6          MPI_Send(&N,1,MPI_INT,i,0,world);
7
8          MPI_Send(A+(M/np)*N*i,(M/np)*N,MPI_FLOAT,i,0,world);
9          MPI_Send(B,N*M,MPI_FLOAT,i,0,world);
10     }
11
12     Ci=C; Ai=A; Bi=B;
13 }
14 else
15 {
16     MPI_Recv(&M,1,MPI_INT,0,0,world,&status);
17     MPI_Recv(&N,1,MPI_INT,0,0,world,&status);
```

```
18
19            Ai = new real[(M/np)*N];
20        MPI_Recv(Ai,(M/np)*N,MPI_FLOAT,0,0,world,&status);
21
22        Bi = new real[N*M];
23        MPI_Recv(Bi,N*M,MPI_FLOAT,0,0,world,&status);
24
25        Ci = new real[(M/np)*M];
26  }
27
28   mi=(M/np);
29
30  real tmp;
31
32        for(int i=0;i<mi;i++)
33            for(int j=0;j<M;j++){
34            tmp=0;
35            for(int k=0;k<N;k++) tmp+=Ai[i*N+k]*Bi[k*M+j];
36            Ci[i*M+j]=tmp;
37        }
38
39  if(!id){
40        for(int i=1;i<np;i++){
41        MPI_Recv(C+mi*M*i,mi*M,MPI_FLOAT,i,0,world,&status);
42        }
43  }
44  else{
45        MPI_Send(Ci,mi*M,MPI_FLOAT,0,0,world);
46  }
47  end=MPI_Wtime();
```

Листинг 8 — Фрагмент кода на C++, реализующий (2) с применением MPI

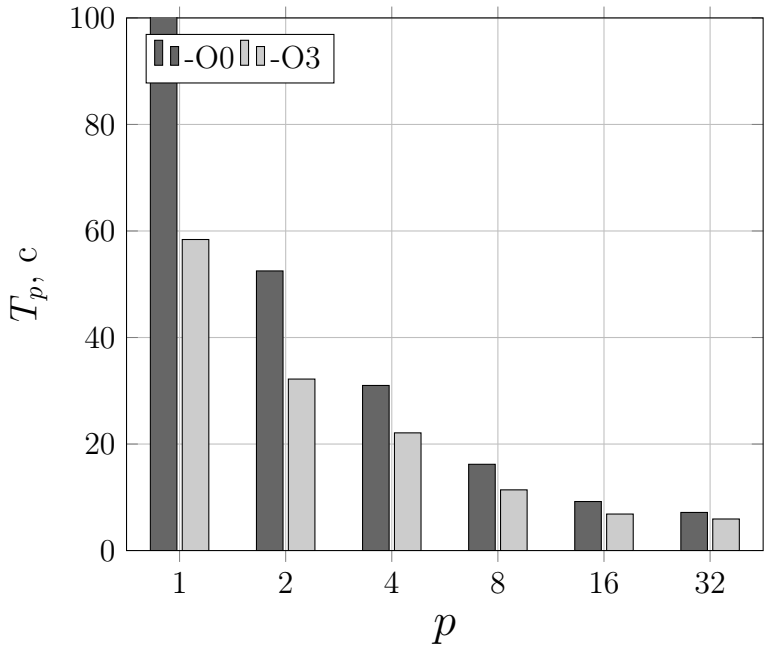

Рисунок 13 — Время $T_p(p)$ параллельных вычислений, %, технология MPI при $N \times N = 1568 \times 1568$

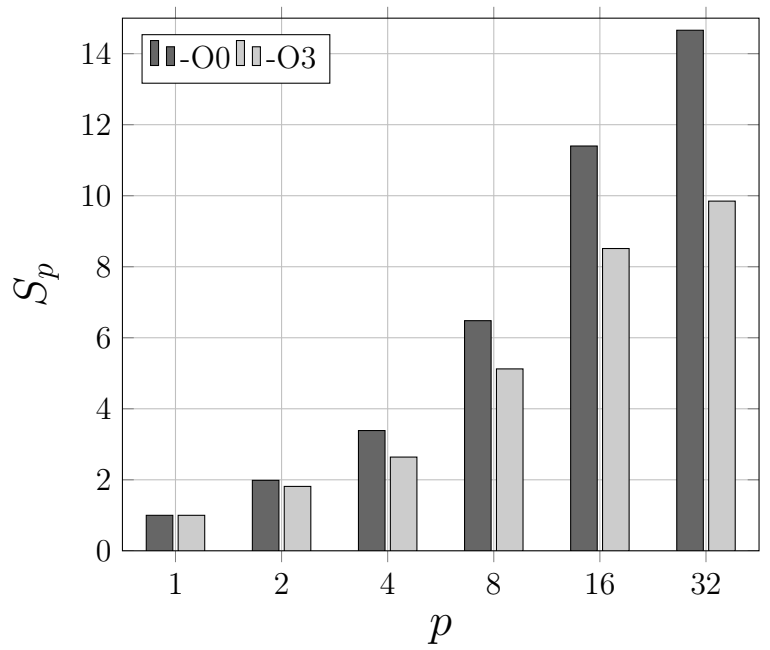

Рисунок 14 — Ускорение $S_p(p)$ параллельных вычислений, технология MPI, при $N \times N = 1568 \times 1568$

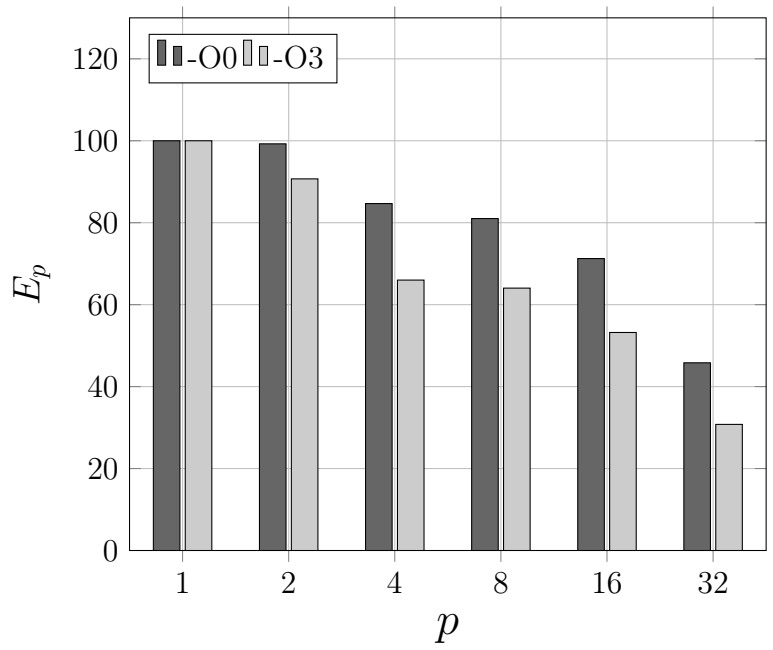

Рисунок 15 — Эффективность $E_p(p)$ параллельных вычислений, %, технология MPI при $N \times N = 1568 \times 1568$

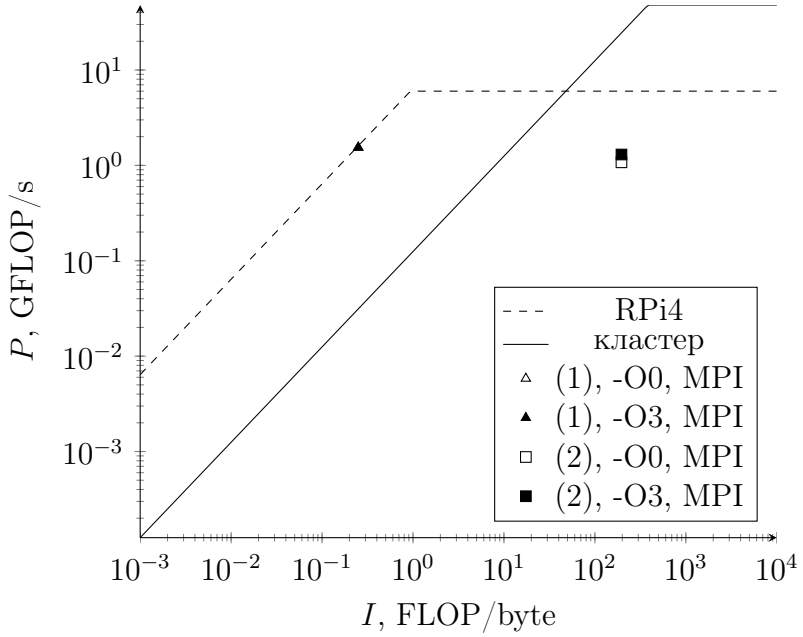

Рисунок 16 — Производительность параллельных вычислений для заданий (1) и (2) с применением технологии MPI, и Roofline модели кластера и миникомпьютера (RPi4).

Выводы

1. По результатам вычислительных экспериментов на кластере из миникомпьютеров Raspeberry Pi 4, определены ускорение и эффективность параллельного вычисления линейной комбинации двух векторов размерности 32000288 и произведения двух вещественных матриц размеров 1536×1536, определена производительность.

2. Максимальные ускорения и эффективность составили: 6.84 и 98.6% для 8 и 2 вычислительных процессов соответственно в технологии MPI при векторе размерности 32000288, 14.66 и 99.25% для 32 и 2 вычислительных процессов при размере матрицы 1536×1536.

3. Получены максимальные значения производительности, в случае применения MPI: 1.54 GFLOP/s, теоретически максимальная производительность составляет 8 * 1 * 4 * 1.5 * 2 = 96 GFLOP/s (8 миникомпьютеров, по 4х-ядерному процессору в каждом, частота 1.5 ГГц и 2 операции одинарной точности, выполняемых за цикл).

Список литературы

1. Копысов С.П., Новиков А.К. Промежуточное программное обеспечение параллельных вычислений. Ижевск: Изд-во «Удмуртский университет». 2012. — 140 с.
2. OpenMP. [Электронный ресурс]. URL: `https://www.openmp.org`. Дата обращения 21.03.2024.
3. MPI Forum [Электронный ресурс]. URL: `https://www.mpi-forum.org`. Дата обращения 21.03.2024.