МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Отчёт по практическим заданиям по дисциплине «Высокоскоростные сетевые технологии суперкомпьютеров»

Студент,
группы 5130201/20101 _____ Тищенко А. А.

Руководитель,
профессор, д.т.н. _____ Курочкин М. А.

«____»_____ 2026г.

Санкт-Петербург, 2026

Содержание

Введение

Современные вычислительные задачи во многих областях науки и техники требуют обработки больших объёмов данных и выполнения ресурсоёмких численных расчётов. В связи с этим особую роль играют высокопроизводительные вычислительные системы (High Performance Computing, HPC), позволяющие значительно ускорить решение сложных задач за счёт параллельной обработки данных и использования специализированных архитектур.

Одной из ключевых задач при работе с вычислительными системами является исследование их производительности. Для этого применяются специальные тесты и бенчмарки, позволяющие оценить эффективность вычислений при различных алгоритмах и конфигурациях оборудования. Одним из наиболее известных тестов является LINPACK, который используется для измерения производительности вычислительных систем при решении систем линейных алгебраических уравнений [1]. Данный тест лежит в основе рейтинга суперкомпьютеров TOP500 и широко применяется для оценки производительности высокопроизводительных вычислительных систем.

Первая лабораторная работа посвящена исследованию производительности вычислительных систем на основе решения систем линейных алгебраических уравнений. В рамках работы изучаются численные методы решения СЛАУ, включая прямые и итерационные методы, а также реализуется собственная версия теста LINPACK с использованием параллельных вычислений. Особое внимание уделяется сравнению производительности стандартной реализации Intel High Performance Linpack и собственной реализации алгоритма.

Вторая лабораторная работа направлена на изучение технологий параллельного программирования и межпроцессного взаимодействия. В рамках данной работы рассматривается технология MPI (Message Passing Interface), широко используемая для разработки масштабируемых параллельных приложений [2]. Разработанный алгоритм реализуется с использованием MPI и исследуется его производительность при запуске на различном количестве вычислительных узлов. Полученные результаты сравниваются с реализацией вычислений с использованием технологии CUDA, предназначенной для выполнения параллельных вычислений на графических процессорах [3].

Все вычислительные эксперименты проводились на вычислительном кластере Суперкомпьютерного центра Политехнического университета (СКЦ Политехнический). Доступ к вычислительным ресурсам осуществлялся через удалённое подключение по протоколу SSH. Для выполнения экспериментов использовалась учётная запись `tm3u21`, вход на кластер производился через узел доступа `login1.hpc.spbstu.ru`.

1 Задача 1

1.1 Постановка задачи

В рамках первого задания требовалось исследовать производительность вычислительной системы при решении плотной системы линейных алгебраических уравнений и сопоставить результаты собственной реализации с эталонным тестом LINPACK. Для достижения этой цели были поставлены следующие задачи:

1. изучить подходы к оценке производительности вычислительных систем;
2. разработать собственную CUDA-реализацию LINPACK-подобного теста;
3. выполнить экспериментальные запуски на вычислительном узле СКЦ Политехнический;
4. сравнить результаты собственной программы и стандартного Intel LINPACK.

Листинг 1. Фрагмент файла `~/.ssh/config`

```
1 Host polytech
2     HostName login1.hpc.spbstu.ru
3     User tm3u21
4     IdentityFile ~/.ssh/09
5     ForwardAgent yes
```

Для подключения к СКЦ Политехнический в файл `~/.ssh/config` была добавлена запись, приведённая в листинге 1. После этого вход на кластер выполнялся командой `ssh polytech`. Подтверждение входа под учётной записью `tm3u21` приведено на рис. 1.

1.2 Математическое описание

Тест LINPACK применяется для оценки производительности вычислительных систем при решении плотной системы линейных уравнений [1]. Рассматривается задача

$$Ax = b, \tag{1}$$

где $A \in \mathbb{R}^{n \times n}$ — плотная квадратная матрица, $x \in \mathbb{R}^n$ — искомый вектор решения, $b \in \mathbb{R}^n$ — вектор правой части.

В классическом варианте LINPACK обычно применяются прямые методы, основанные на LU-разложении [4]. В собственной реализации для первого задания использован итерационный метод Якоби, так как он хорошо распараллеливается на GPU: каждая строка матрицы может обрабатываться независимо [5].

Для метода Якоби очередное приближение вычисляется по формуле

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n. \tag{2}$$

Критерий остановки выбран в виде

$$\max_i \left| x_i^{(k+1)} - x_i^{(k)} \right| \le \varepsilon. \tag{3}$$

Чтобы обеспечить сходимость метода, в программе формируется строго диагонально доминирующая матрица. Для контроля качества решения дополнительно вычисляются:

$$\|Ax - b\|_\infty, \tag{4}$$

$$\|x - x_{true}\|_\infty, \tag{5}$$

где x_{true} — заранее известное опорное решение, использованное при построении тестовой системы.

Для приближённой оценки производительности используется LINPACK-подобная метрика

$$R = \frac{\frac{2}{3}n^3}{t}, \tag{6}$$

где t — время решения, измеренное в секундах. В отчёте далее используется величина R в GFLOPS.

1.3 Особенности реализации

Собственная программа реализована на CUDA C++ и находится в файле `task1/src/main.cu`. В реализации приняты следующие решения:

- для каждой строки матрицы запускается отдельный поток CUDA, который вычисляет новое значение одной компоненты вектора решения;
- матрица A и вектор b один раз копируются на устройство, а далее итерационный процесс выполняется на GPU;
- завершение итераций контролируется через флаг `converged`, который может быть сброшен любым потоком, если изменение соответствующей компоненты превысило ε;
- после завершения расчёта на стороне CPU вычисляются невязка и ошибка относительно заранее известного решения;
- для удобства дальнейшего анализа программа может сохранять результаты в CSV-файл.

Такой вариант не является буквальной реализацией классического LU-LINPACK, однако решает ту же прикладную задачу — измерение производительности при решении плотной СЛАУ — и позволяет исследовать эффективность распараллеливания на GPU. Полный текст исходного кода CUDA-реализации приведён в приложении А.

1.4 Сборка и запуск

Подготовленные файлы для выполнения задания были размещены в каталоге `/home/ipmmstudy1/tm3u21/supercomputers/task1`. Запуск собственной CUDA-реализации выполнялся пакетным файлом `task1/scripts/run_cuda.slurm`, который загружал модули `compiler/gcc/11` и `nvidia/cuda/11.6u2`, собирал программу через `nvcc` и запускал серию вычислительных экспериментов. Полный текст данного файла приведён в приложении Б.

Эталонный CPU-вариант Intel LINPACK запускался из архива Intel oneMKL Benchmarks Suite for Linux, предварительно распакованного в каталог `/home/ipmmstudy1/tm3u21/LINPACK`. В результате файл `xlinpack_xeon64` был получен по пути `LINPACK/benchmarks_2025.3/linux/share/mkl/benchmarks/linpack`. Для запуска использовался пакетный файл `task1/scripts/run_intel_linpack.slurm`, полный текст которого приведён в приложении В.

Для Intel LINPACK был подготовлен отдельный входной файл `lininput_report_xeon64`, в котором зафиксированы те же размеры задач, что и для CUDA-реализации: 1000, 1500, 2000, 2500, 3000, 3500. Полный текст этого файла приведён в приложении Г.

1.5 Результаты эксперимента

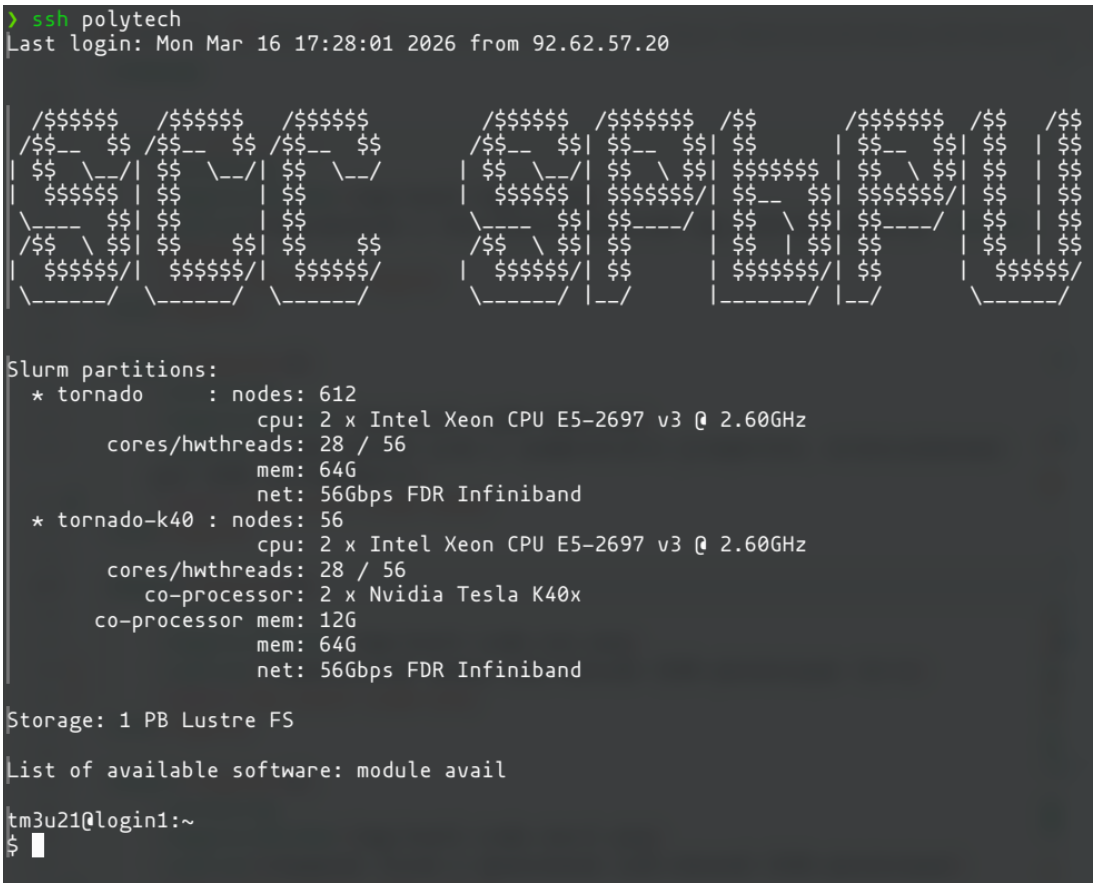

Рис. 1. Подключение к СКЦ Политехнический под учётной записью `tm3u21`

```
===== node config =====
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                56
On-line CPU(s) list:   0-55
Thread(s) per core:    2
Core(s) per socket:    14
Socket(s):             2
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz
Stepping:              2
CPU MHz:               2601.000
BogoMIPS:              5187.61
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
NodeName=n02p009 Arch=x86_64 CoresPerSocket=7
   CPUAlloc=56 CPUTot=56 CPULoad=0.01
   AvailableFeatures=startx
   ActiveFeatures=startx
   Gres=(null)
   NodeAddr=n02p009 NodeHostName=n02p009 Version=19.05.3
   OS=Linux 3.10.0-957.5.1.el7.x86_64 #1 SMP Fri Feb 1 14:54:57 UTC 2019
   RealMemory=64120 AllocMem=0 FreeMem=61950 Sockets=4 Boards=1
   State=ALLOCATED ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=tornado-k40
   BootTime=2026-03-06T14:56:37 SlurmdStartTime=2026-03-06T14:57:33
   CfgTRES=cpu=56,mem=64120M,billing=56
   AllocTRES=cpu=56,mem=64120M,billing=56
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s


GPU 0: Tesla K40m (UUID: GPU-3dae1ca4-d932-de94-06a0-ceebb2307bf8)
GPU 1: Tesla K40m (UUID: GPU-c4a2d484-6fca-6016-440d-42fa4e2457e5)
```

Рис. 2. Конфигурация узла и графического ускорителя, использованных для CUDA-эксперимента

```
tm3u21@login1:~/supercomputers/task1
$ tail -n 12 ~/supercomputers/task1/results/task1-cuda-6616336.out

===== benchmark =====
CUDA Jacobi LINPACK-like benchmark
device = Tesla K40m, compute capability = 3.5, threads = 256, repeat = 3, warmup = 1, eps = 1.000000e-06

N       Time(ms)      Iter        ResidualInf       XerrInf          GFLOPS        Status
1000    5.0083        6           2.242e-06         4.390e-09        133.114       converged
1500    7.4931        6           1.107e-06         1.466e-09        300.277       converged
2000    8.3563        5           2.443e-05         2.489e-08        638.244       converged
2500    10.4837       5           1.593e-05         1.297e-08        993.608       converged
3000    12.6709       5           1.288e-05         8.451e-09        1420.573      converged
3500    14.8861       5           9.516e-06         5.432e-09        1920.138      converged
tm3u21@login1:~/supercomputers/task1
```

Рис. 3. Терминальный вывод собственной CUDA-реализации теста

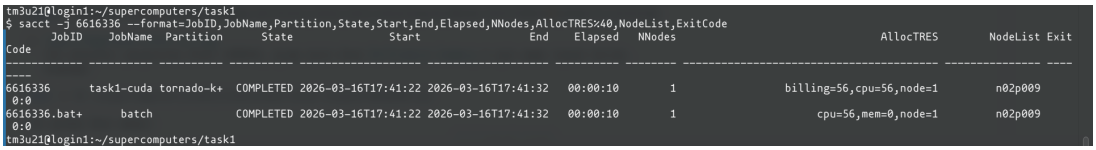

Рис. 4. Сведения Slurm о выполнении собственной CUDA-реализации

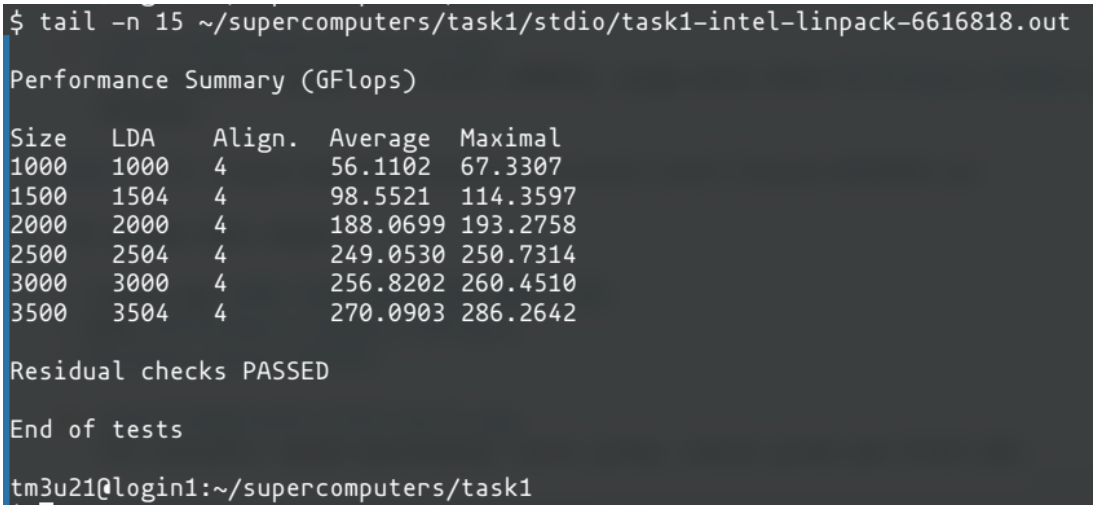

Рис. 5. Терминальный вывод стандартного Intel LINPACK

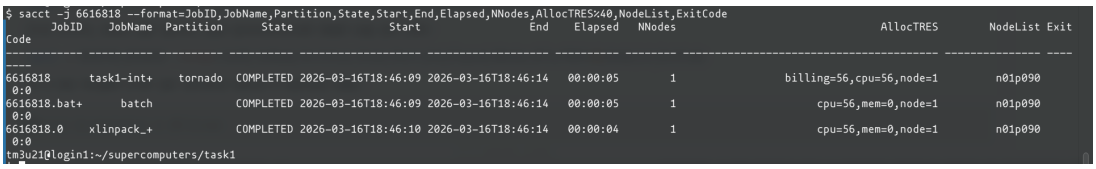

Рис. 6. Сведения Slurm о выполнении Intel LINPACK

Для сравнения были использованы два фактических запуска на СКЦ Политехнический: собственная CUDA-реализация (задание `6616336`, узел `n02p009`, раздел `tornado-k40`) и стандартный Intel LINPACK (задание `6616818`, узел `n01p090`, раздел `tornado`). Для Intel LINPACK использовался отдельный входной файл, содержащий те же размеры задач, что и в CUDA-реализации.

Подключение к кластеру под учётной записью `tm3u21` показано на рис. 1. Конфигурация вычислительного узла и графических ускорителей, задействованных при CUDA-эксперименте, приведена на рис. 2. Терминальный вывод собственной CUDA-реализации и сведения Slurm о её выполнении приведены на рис. 3 и рис. 4. Аналогичные материалы для эталонного CPU-запуска Intel LINPACK приведены на рис. 5 и рис. 6.

Численные результаты сведены в таблицу 1. Для столбцов CUDA использованы значения из файла `results/task1-cuda-6616336.csv`. Для Intel LINPACK в таблицу внесены минимальное время из серии прогонов и максимальная производительность из секции `Performance Summary`.

Таблица 1. Сравнение собственной CUDA-реализации и Intel LINPACK

N	t_{CUDA}, мс	Iter	$\|Ax-b\|_\infty$	R_{CUDA}, GFLOPS	t_{Intel}, с	R_{Intel}, GFLOPS	S_t
1000	5.0083	6	$2.242 \cdot 10^{-6}$	133.114	0.010	67.331	2.00
1500	7.4931	6	$1.107 \cdot 10^{-6}$	300.277	0.020	114.360	2.67
2000	8.3563	5	$2.443 \cdot 10^{-5}$	638.244	0.028	193.276	3.35
2500	10.4837	5	$1.593 \cdot 10^{-5}$	993.608	0.042	250.731	4.01
3000	12.6709	5	$1.288 \cdot 10^{-5}$	1420.573	0.069	260.451	5.45
3500	14.8861	5	$9.516 \cdot 10^{-6}$	1920.138	0.100	286.264	6.72

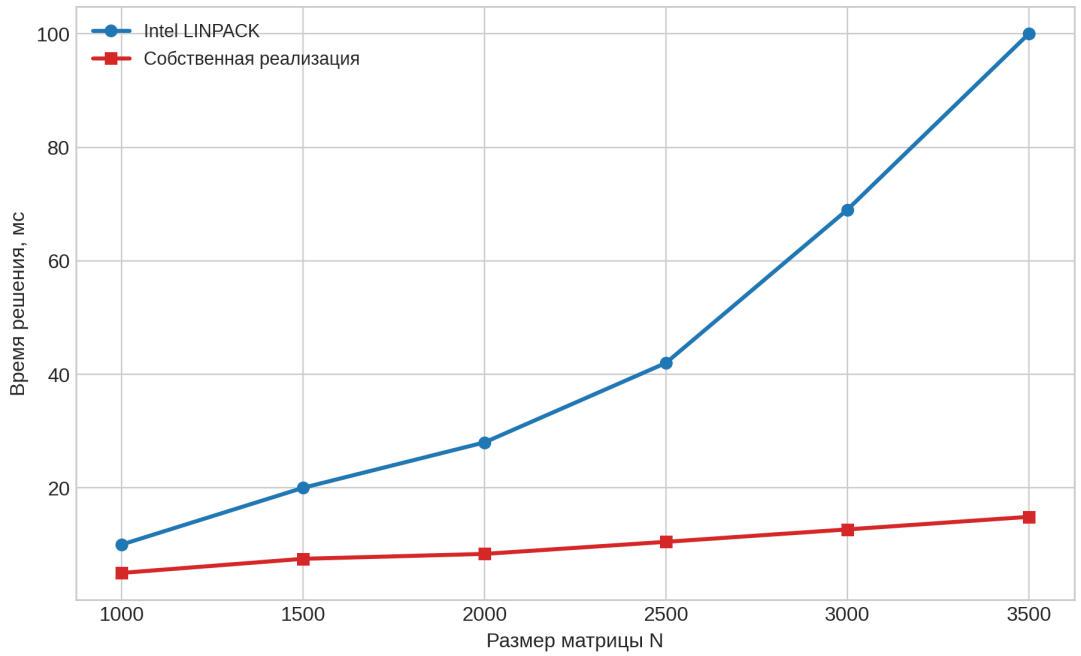

Рис. 7. Сравнение времени решения эталонной и собственной реализаций

Графическое сравнение времени решения приведено на рис. 7. Скрипт, использованный для построения данного графика, приведён в приложении Д.

По полученным результатам можно сделать следующие наблюдения:

- все тестовые случаи для размеров от 1000 до 3500 успешно сошлись за 5–6 итераций;
- время решения возрастает плавно: от 5.0083 мс при $N = 1000$ до 14.8861 мс при $N = 3500$;
- ошибка по известному решению остаётся на уровне порядка 10^{-9}–10^{-8}, а невязка — на уровне 10^{-6}–10^{-5}, что подтверждает корректность полученного решения;

- на всех рассмотренных размерах задач собственная реализация работает быстрее Intel LINPACK, а выигрыш по времени возрастает от 2.00 до 6.72 раза.

1.6 Выводы

В рамках первого задания была подготовлена собственная CUDA-реализация LINPACK-подобного теста для решения плотной СЛАУ методом Якоби. Программа поддерживает запуск серии экспериментов, измерение времени выполнения, а также вычисление невязки и ошибки относительно известного точного решения.

Дополнительно были подготовлены пакетные файлы запуска для собственной CUDA-реализации и для стандартного Intel LINPACK. Полные тексты этих файлов приведены в приложениях Б и В.

На текущем этапе можно зафиксировать, что собственная CUDA-реализация корректно работает на узле `tornado-k40` и обеспечивает сходимость на всём исследованном диапазоне размеров. В сопоставлении со стандартным Intel LINPACK на CPU она показывает меньшее время решения на всех исследованных размерах: собственная реализация оказывается более производительной, а выигрыш по времени возрастает от 2.00 раза при $N = 1000$ до 6.72 раза при $N = 3500$.

2 Задача 2

2.1 Постановка задачи

В рамках второго задания требовалось решить следующие задачи:

1. изучить технологию межпроцессного взаимодействия MPI;
2. разработать параллельный масштабируемый алгоритм для решения вычислительной задачи;
3. реализовать разработанный алгоритм с использованием технологии MPI;
4. исследовать производительность реализации на 1, 2 и 4-х узлах СКЦ «Политехнический»;
5. сравнить время решения вычислительной задачи с использованием MPI и CUDA.

В качестве вычислительной задачи использовалась задача построения пути движения робота по полигону, решавшаяся в предыдущем семестре с помощью CUDA. Для текущего задания тот же алгоритм был реализован на MPI и исследована масштабируемость при увеличении числа узлов.

2.2 Математическое описание

Дано:

- массив P — полигон размера $n \times n$ целых чисел;
- точки $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$ на полигоне;
- контуры V, запрещённые для движения (ячейки со значением -1).

Требуется построить L — кратчайшую траекторию, соединяющую P_1 и P_2, либо сообщить, что такой траектории не существует. Траектория не может проходить через ячейки со значением -1.

Для решения задачи используется волновой алгоритм [6]. Алгоритм состоит из двух фаз.

Фаза 1 — распространение волны. От начальной точки к конечной распространяется волна. На каждом шаге волна пополняется свободными ячейками, которые ещё не принадлежат волне и являются соседями ячеек, попавших в волну на предыдущем шаге. Для определения соседей используется окрестность фон Неймана (рис. 8): четыре ячейки сверху, снизу, слева и справа. Каждая новая ячейка заполняется значением

$$d_{i,j} = \min\big(d_{i,j},\ d_{i-1,j} + 1,\ d_{i+1,j} + 1,\ d_{i,j-1} + 1,\ d_{i,j+1} + 1\big), \tag{7}$$

где $d_{i,j}$ — текущее расстояние от стартовой точки до ячейки (i, j). Ячейки с препятствиями ($P_{i,j} = -1$) игнорируются.

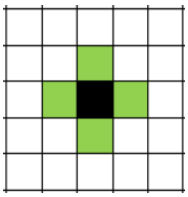

Рис. 8. Окрестность фон Неймана

Начальная ячейка P_1 заполняется нулём. Остальные свободные ячейки инициализируются значением ∞. Итерации продолжаются до тех пор, пока хотя бы одна ячейка обновляется.

Иллюстрации первого, третьего и последнего шагов распространения волны приведены на рис. 9–11.

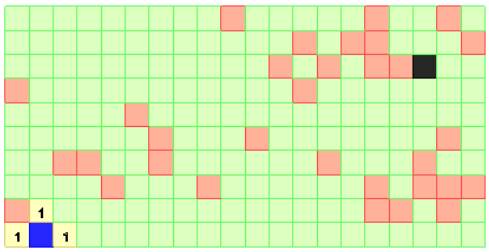

Рис. 9. Первый шаг распространения волны

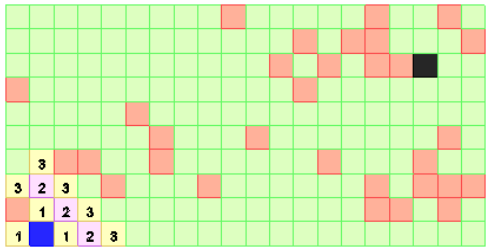

Рис. 10. Третий шаг распространения волны

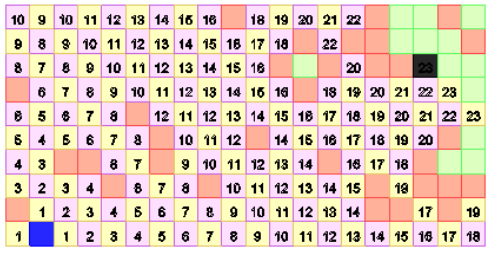

Рис. 11. Последний шаг распространения волны

Фаза 2 — восстановление пути. Из конечной ячейки P_2 к начальной P_1 выполняется обратный ход: на каждом шаге выбирается соседняя ячейка со значением на единицу меньше текущего (рис. 12).

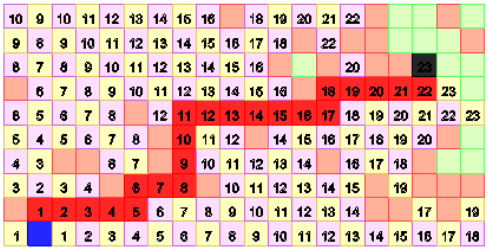

Рис. 12. Восстановление пути

2.3 Особенности реализации MPI

Для распараллеливания волнового алгоритма с использованием MPI [2] применена декомпозиция полигона по строкам. Полигон размера $n \times n$ разбивается на горизонтальные полосы, каждая из которых обрабатывается отдельным MPI-процессом. При использовании P процессов каждый из них получает примерно n/P строк.

Для корректной обработки граничных ячеек каждый процесс дополнительно хранит по одной теневой строке (ghost row) сверху и снизу, содержащей актуальные значения расстояний из соседних процессов.

Алгоритм работы программы:

1. Процесс с рангом 0 генерирует полигон P.
2. Полигон рассылается всем процессам через `MPI_Bcast`.
3. Массив расстояний *dist* распределяется по процессам через `MPI_Scatterv`.
4. Итеративно выполняется:
   (a) обмен теневыми строками с соседними процессами через `MPI_Sendrecv`;
   (b) волновой шаг на локальных строках;
   (c) проверка глобальной сходимости через `MPI_Allreduce` с операцией `MPI_LOR`.
5. Результат собирается на процессе 0 через `MPI_Gatherv`.
6. Процесс 0 выводит длину пути и время выполнения (замер через `MPI_Wtime`).

Полный текст MPI-реализации приведён в приложении Е.

2.4 CUDA-реализация для сравнения

Для сопоставления с MPI-реализацией подготовлена CUDA-версия того же волнового алгоритма, основанная на программе прошлого семестра. В отличие от исходной версии, размер полигона задаётся через аргумент командной строки (вместо `#define`). Используется ядро с глобальной памятью: каждый поток обрабатывает несколько ячеек с шагом, равным общему числу потоков. Полный текст CUDA-реализации приведён в приложении Ж.

2.5 Сборка и запуск

Файлы для второго задания были размещены в каталоге `/home/ipmmstudy1/tm3u21/supercomp`

MPI-версия запускалась пакетным файлом `task2/scripts/run_mpi.slurm` в разделе `tornado` на 1, 2 и 4 узлах. Для каждого запуска число узлов задавалось при отправке задачи:

```
sbatch --nodes=1 scripts/run_mpi.slurm
sbatch --nodes=2 scripts/run_mpi.slurm
sbatch --nodes=4 scripts/run_mpi.slurm
```

Полный текст данного файла приведён в приложении З.

CUDA-версия запускалась пакетным файлом `task2/scripts/run_cuda.slurm` в разделе `tornado-k40` на одном узле. Полный текст приведён в приложении И.

Обе программы последовательно запускались на полигонах размером 500, 1000, 2000, 3000, 5000.

2.6 Результаты эксперимента

```
===== benchmark (4 nodes / 4 ranks) =====
--- n=500 ---
n=500  path_len=990  time=17.85 ms  iters=24  procs=4
--- n=1000 ---
n=1000  path_len=1990  time=55.69 ms  iters=42  procs=4
--- n=2000 ---
n=2000  path_len=3990  time=343.95 ms  iters=70  procs=4
--- n=3000 ---
n=3000  path_len=5990  time=1200.42 ms  iters=107  procs=4
--- n=5000 ---
n=5000  path_len=9990  time=5349.19 ms  iters=166  procs=4

===== done =====
tm3u21@login1:~/supercomputers/task2
$
```

Рис. 13. Терминальный вывод MPI-реализации волнового алгоритма

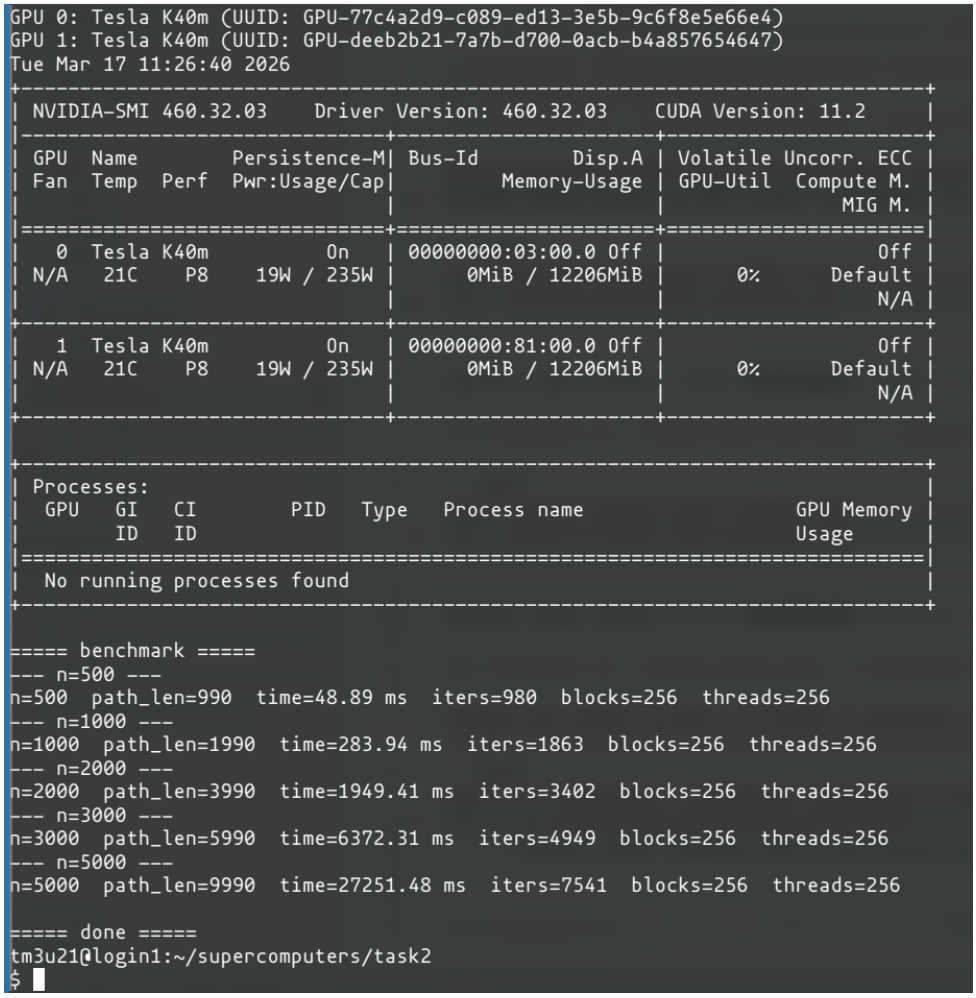

Рис. 14. Терминальный вывод CUDA-реализации волнового алгоритма

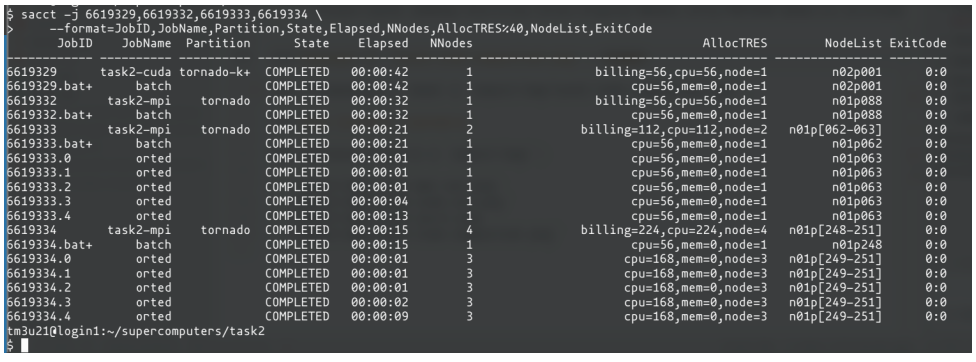

Рис. 15. Сведения Slurm о выполнении задач второго задания

Для сравнения были использованы четыре фактических запуска на СКЦ «Политехнический»: CUDA-реализация (задание `6619329`, узел `n02p001`, раздел `tornado-k40`) и три запуска MPI-реализации на 1, 2 и 4 узлах (задания `6619332`, `6619333`, `6619334`, раздел `tornado`).

Терминальный вывод MPI- и CUDA-реализаций приведён на рис. 13 и рис. 14. Сведения Slurm о выполнении задач показаны на рис. 15.

Численные результаты сведены в таблицу 2.

Таблица 2. Сравнение времени выполнения MPI (1, 2, 4 узла) и CUDA

n	$t_{\text{MPI-1}}$, мс	$t_{\text{MPI-2}}$, мс	$t_{\text{MPI-4}}$, мс	t_{CUDA}, мс	S_2	S_4
500	30.60	14.94	17.85	48.89	2.05	1.71
1000	191.56	97.56	55.69	283.94	1.96	3.44
2000	1304.08	654.28	343.95	1949.41	1.99	3.79
3000	4616.74	2363.96	1200.42	6372.31	1.95	3.85
5000	19897.95	9960.07	5349.19	27251.48	2.00	3.72

В таблице $S_2 = t_{\text{MPI-1}}/t_{\text{MPI-2}}$ и $S_4 = t_{\text{MPI-1}}/t_{\text{MPI-4}}$ — ускорение при увеличении числа узлов.

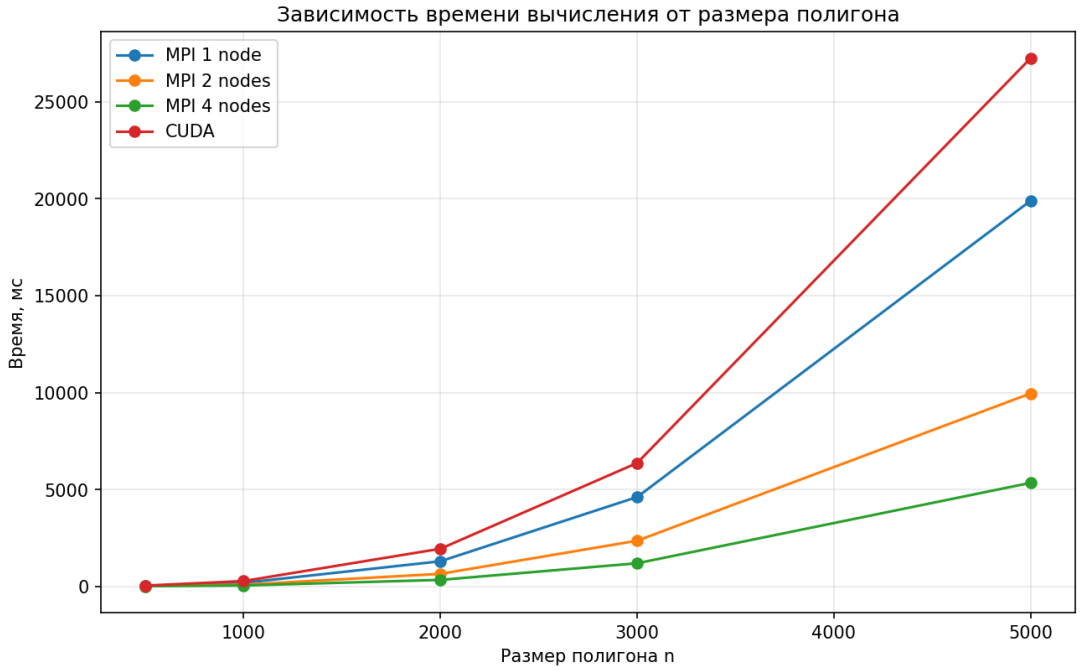

Рис. 16. Зависимость времени вычисления от размера полигона

Графическое сравнение времени решения приведено на рис. 16. Скрипт, использованный для построения данного графика, приведён в приложении К.

По полученным результатам можно сделать следующие наблюдения:

- при увеличении числа узлов с 1 до 2 наблюдается практически линейное ускорение: S_2 составляет от 1,95 до 2,05 на всех размерах задач;
- при увеличении числа узлов до 4 ускорение S_4 зависит от размера задачи: для $n = 500$ оно составляет лишь 1,71 из-за доминирования накладных расходов на обмен, а для $n \geq 1000$ достигает 3,44–3,85;

- MPI-реализация на одном узле оказывается быстрее CUDA-реализации на всех размерах задач: например, при $n = 5000$ MPI выполняется за 19,9 с, а CUDA — за 27,3 с;
- различие в производительности объясняется тем, что при последовательном обходе строк в MPI-версии волна может распространяться на несколько ячеек за одну итерацию, тогда как в CUDA все ячейки обрабатываются параллельно и за одну итерацию волна продвигается не более чем на одну ячейку (для $n = 5000$: 166 итераций MPI против 7541 итерации CUDA);
- на 4 узлах MPI-реализация работает в 3,7–5,1 раза быстрее CUDA.

2.7 Выводы

В рамках второго задания была реализована MPI-версия волнового алгоритма для поиска кратчайшего пути на полигоне. Программа использует декомпозицию по строкам с обменом теневых строк между соседними процессами. Для сравнения подготовлена CUDA-реализация того же алгоритма, основанная на программе прошлого семестра.

Экспериментальные запуски показали, что MPI-реализация хорошо масштабируется: при переходе с 1 на 2 узла достигается практически линейное ускорение ($S_2 \approx 2{,}0$), а на 4 узлах — ускорение до 3,85 раза при достаточном размере задачи ($n \geq 1000$). MPI-реализация даже на одном узле оказалась быстрее CUDA благодаря меньшему числу итераций алгоритма.

Заключение

В рамках данной работы были выполнены два практических задания, направленных на изучение технологий высокопроизводительных вычислений.

В первом задании была разработана собственная CUDA-реализация LINPACK-подобного теста и проведено сравнение с эталонным Intel LINPACK. Собственная реализация показала меньшее время решения на всех исследованных размерах задач с ускорением от 2,00 до 6,72 раза.

Во втором задании волновой алгоритм поиска пути был реализован с использованием MPI и исследована его масштабируемость при запуске на 1, 2 и 4 узлах СКЦ «Политехнический». MPI-реализация продемонстрировала хорошую масштабируемость: ускорение при переходе с 1 на 2 узла составило около 2,0, а на 4 узлах — до 3,85 раза. При сравнении с CUDA MPI-версия оказалась быстрее на всех размерах задач благодаря меньшему числу итераций при последовательном обходе строк.

Список литературы

1. *Dongarra J.*, *Luszczek P.*, *Petitet A.* The LINPACK Benchmark: Past, Present and Future // Concurrency and Computation: Practice and Experience. — 2003. — Т. 15, № 9. — С. 803—820.
2. *Gropp W.*, *Lusk E.*, *Skjellum A.* Using MPI: Portable Parallel Programming with the Message-Passing Interface. — 3-е изд. — MIT Press, 2014.
3. *Kirk D. B.*, *Hwu W.-m. W.* Programming Massively Parallel Processors: A Hands-on Approach. — 3-е изд. — Morgan Kaufmann, 2016.
4. *Golub G. H.*, *Van Loan C. F.* Matrix Computations. — 4-е изд. — Johns Hopkins University Press, 2013.
5. *Saad Y.* Iterative Methods for Sparse Linear Systems. — SIAM, 2003.
6. *Lee C. Y.* An Algorithm for Path Connections and Its Applications // IRE Transactions on Electronic Computers. — 1961. — Т. EC—10, № 3. — С. 346—365.

Приложение А

Листинг 2. Файл `task1/src/main.cu`

```
#include <cuda_runtime.h>

#include <algorithm>
#include <cctype>
#include <cmath>
#include <cstdint>
#include <cstdlib>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <limits>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>

#define CUDA_CHECK(call)                                                    \
    do {                                                                    \
        cudaError_t err__ = (call);                                         \
        if (err__ != cudaSuccess) {                                         \
            std::cerr << "CUDA error at " << __FILE__ << ":" << __LINE__    \
                      << " -> " << cudaGetErrorString(err__) << std::endl;  \
            std::exit(EXIT_FAILURE);                                        \
        }                                                                   \
    } while (0)

struct Options {
    int start = 1000;
    int step = 500;
    int count = 6;
    std::vector<int> sizes;
    int threads = 256;
    int max_iters = 10000;
    int repeat = 3;
    int warmup = 1;
    unsigned int seed = 42U;
    double eps = 1e-6;
    std::string csv_path;
};

struct Metrics {
    double elapsed_ms = std::numeric_limits<double>::max();
    int iterations = 0;
    double residual_inf = std::numeric_limits<double>::infinity();
    double x_error_inf = std::numeric_limits<double>::infinity();
    double gflops = 0.0;
    bool converged = false;
};

__global__ void jacobi_iteration(const double *a,
                                 const double *b,
```

```
52                              const double *x_in,
53                              double *x_out,
54                              int n,
55                              double eps,
56                              int *converged) {
57      const int row = blockIdx.x * blockDim.x + threadIdx.x;
58      if (row >= n) {
59          return;
60      }
61
62      const int row_offset = row * n;
63      double sum = 0.0;
64      for (int col = 0; col < n; ++col) {
65          if (col != row) {
66              sum += a[row_offset + col] * x_in[col];
67          }
68      }
69
70      const double next = (b[row] - sum) / a[row_offset + row];
71      x_out[row] = next;
72      if (fabs(next - x_in[row]) > eps) {
73          atomicAnd(converged, 0);
74      }
75  }
76
77  static void print_usage(const char *program) {
78      std::cout
79          << "Usage: " << program << " [options]\n"
80          << "Options:\n"
81          << "  --start N        First matrix size (default: 1000)\n"
82          << "  --step N         Size increment (default: 500)\n"
83          << "  --count N        Number of tests (default: 6)\n"
84          << "  --sizes a,b,c    Comma-separated matrix sizes\n"
85          << "  --threads N      Threads per block (default: 256)\n"
86          << "  --max-iters N    Max Jacobi iterations (default: 10000)\n"
87          << "  --eps X          Convergence epsilon (default: 1e-6)\n"
88          << "  --repeat N       Timed repetitions, best is kept (default: 3)\n"
89          << "  --warmup N       Warmup repetitions (default: 1)\n"
90          << "  --seed N         RNG seed (default: 42)\n"
91          << "  --csv PATH       Write CSV summary to PATH\n"
92          << "  --help           Print this help\n";
93  }
94
95  static bool parse_int_arg(const std::string &text, int &out) {
96      try {
97          size_t pos = 0;
98          const int parsed = std::stoi(text, &pos);
99          if (pos != text.size()) {
100             return false;
101         }
102         out = parsed;
103         return true;
104     } catch (...) {
105         return false;
106     }
107 }
```

```
108
109 static bool parse_uint_arg(const std::string &text, unsigned int &out) {
110     try {
111         size_t pos = 0;
112         const unsigned long parsed = std::stoul(text, &pos);
113         if (pos != text.size()) {
114             return false;
115         }
116         out = static_cast<unsigned int>(parsed);
117         return true;
118     } catch (...) {
119         return false;
120     }
121 }
122
123 static bool parse_double_arg(const std::string &text, double &out) {
124     try {
125         size_t pos = 0;
126         const double parsed = std::stod(text, &pos);
127         if (pos != text.size()) {
128             return false;
129         }
130         out = parsed;
131         return true;
132     } catch (...) {
133         return false;
134     }
135 }
136
137 static bool parse_sizes_arg(const std::string &text, std::vector<int> &sizes) {
138     std::stringstream ss(text);
139     std::string token;
140     std::vector<int> parsed;
141
142     while (std::getline(ss, token, ',')) {
143         token.erase(
144             std::remove_if(token.begin(),
145                            token.end(),
146                            [](unsigned char c) { return std::isspace(c) != 0; }),
147             token.end());
148         if (token.empty()) {
149             continue;
150         }
151
152         int value = 0;
153         if (!parse_int_arg(token, value) || value <= 0) {
154             return false;
155         }
156         parsed.push_back(value);
157     }
158
159     if (parsed.empty()) {
160         return false;
161     }
162
163     sizes = parsed;
```

```
164     return true;
165 }
166
167 static bool parse_options(int argc, char **argv, Options &options) {
168     for (int i = 1; i < argc; ++i) {
169         const std::string arg = argv[i];
170         auto require_value = [&](const char *name) -> const char * {
171             if (i + 1 >= argc) {
172                 std::cerr << "Missing value for " << name << '\n';
173                 return nullptr;
174             }
175             return argv[++i];
176         };
177
178         if (arg == "--help") {
179             print_usage(argv[0]);
180             return false;
181         }
182         if (arg == "--start") {
183             const char *v = require_value("--start");
184             if (!v || !parse_int_arg(v, options.start) || options.start <= 0) {
185                 std::cerr << "Invalid --start value\n";
186                 return false;
187             }
188             continue;
189         }
190         if (arg == "--step") {
191             const char *v = require_value("--step");
192             if (!v || !parse_int_arg(v, options.step) || options.step <= 0) {
193                 std::cerr << "Invalid --step value\n";
194                 return false;
195             }
196             continue;
197         }
198         if (arg == "--count") {
199             const char *v = require_value("--count");
200             if (!v || !parse_int_arg(v, options.count) || options.count <= 0) {
201                 std::cerr << "Invalid --count value\n";
202                 return false;
203             }
204             continue;
205         }
206         if (arg == "--threads") {
207             const char *v = require_value("--threads");
208             if (!v || !parse_int_arg(v, options.threads) || options.threads <= 0 ||
209                 options.threads > 1024) {
210                 std::cerr << "Invalid --threads value\n";
211                 return false;
212             }
213             continue;
214         }
215         if (arg == "--max-iters") {
216             const char *v = require_value("--max-iters");
217             if (!v || !parse_int_arg(v, options.max_iters) ||
218                 options.max_iters <= 0) {
219                 std::cerr << "Invalid --max-iters value\n";
```

```
220             return false;
221         }
222         continue;
223     }
224     if (arg == "--eps") {
225         const char *v = require_value("--eps");
226         if (!v || !parse_double_arg(v, options.eps) || options.eps <= 0.0) {
227             std::cerr << "Invalid --eps value\n";
228             return false;
229         }
230         continue;
231     }
232     if (arg == "--repeat") {
233         const char *v = require_value("--repeat");
234         if (!v || !parse_int_arg(v, options.repeat) || options.repeat <= 0) {
235             std::cerr << "Invalid --repeat value\n";
236             return false;
237         }
238         continue;
239     }
240     if (arg == "--warmup") {
241         const char *v = require_value("--warmup");
242         if (!v || !parse_int_arg(v, options.warmup) || options.warmup < 0) {
243             std::cerr << "Invalid --warmup value\n";
244             return false;
245         }
246         continue;
247     }
248     if (arg == "--seed") {
249         const char *v = require_value("--seed");
250         if (!v || !parse_uint_arg(v, options.seed)) {
251             std::cerr << "Invalid --seed value\n";
252             return false;
253         }
254         continue;
255     }
256     if (arg == "--sizes") {
257         const char *v = require_value("--sizes");
258         if (!v || !parse_sizes_arg(v, options.sizes)) {
259             std::cerr << "Invalid --sizes value\n";
260             return false;
261         }
262         continue;
263     }
264     if (arg == "--csv") {
265         const char *v = require_value("--csv");
266         if (!v) {
267             return false;
268         }
269         options.csv_path = v;
270         continue;
271     }
272
273     std::cerr << "Unknown option: " << arg << '\n';
274     return false;
275 }
```

```
276
277     if (options.sizes.empty()) {
278         options.sizes.reserve(static_cast<size_t>(options.count));
279         for (int i = 0; i < options.count; ++i) {
280             options.sizes.push_back(options.start + i * options.step);
281         }
282     }
283
284     return true;
285 }
286
287 static double next_random_value(uint32_t &state) {
288     state = state * 1664525U + 1013904223U;
289     const double normalized =
290         static_cast<double>(state & 0x00FFFFFFU) /
291         static_cast<double>(0x00FFFFFFU);
292     return normalized * 2.0 - 1.0;
293 }
294
295 static void build_system(int n,
296                          unsigned int seed,
297                          std::vector<double> &a,
298                          std::vector<double> &x_true,
299                          std::vector<double> &b) {
300     const size_t nn = static_cast<size_t>(n) * static_cast<size_t>(n);
301     a.assign(nn, 0.0);
302     x_true.assign(static_cast<size_t>(n), 0.0);
303     b.assign(static_cast<size_t>(n), 0.0);
304
305     uint32_t state = seed;
306     for (int i = 0; i < n; ++i) {
307         x_true[static_cast<size_t>(i)] = next_random_value(state);
308     }
309
310     for (int row = 0; row < n; ++row) {
311         double off_diag_sum = 0.0;
312         const size_t row_offset = static_cast<size_t>(row) * static_cast<size_t>(n);
313         for (int col = 0; col < n; ++col) {
314             if (col == row) {
315                 continue;
316             }
317             const double value = next_random_value(state);
318             a[row_offset + static_cast<size_t>(col)] = value;
319             off_diag_sum += std::fabs(value);
320         }
321         a[row_offset + static_cast<size_t>(row)] =
322             off_diag_sum + 2.0 + std::fabs(next_random_value(state));
323     }
324
325     for (int row = 0; row < n; ++row) {
326         const size_t row_offset = static_cast<size_t>(row) * static_cast<size_t>(n);
327         double sum = 0.0;
328         for (int col = 0; col < n; ++col) {
329             sum += a[row_offset + static_cast<size_t>(col)] *
330                    x_true[static_cast<size_t>(col)];
331         }
```

```
        b[static_cast<size_t>(row)] = sum;
    }
}

static double compute_residual_inf(const std::vector<double> &a,
                                   const std::vector<double> &x,
                                   const std::vector<double> &b,
                                   int n) {
    double max_residual = 0.0;
    for (int row = 0; row < n; ++row) {
        const size_t row_offset = static_cast<size_t>(row) * static_cast<size_t>(n);
        double sum = 0.0;
        for (int col = 0; col < n; ++col) {
            sum += a[row_offset + static_cast<size_t>(col)] *
                   x[static_cast<size_t>(col)];
        }
        max_residual =
            std::max(max_residual, std::fabs(sum - b[static_cast<size_t>(row)]));
    }
    return max_residual;
}

static double compute_x_error_inf(const std::vector<double> &x,
                                  const std::vector<double> &x_true) {
    double max_error = 0.0;
    for (size_t i = 0; i < x.size(); ++i) {
        max_error = std::max(max_error, std::fabs(x[i] - x_true[i]));
    }
    return max_error;
}

static Metrics solve_once(const Options &options,
                          int n,
                          const std::vector<double> &a,
                          const std::vector<double> &b,
                          const std::vector<double> &x_true,
                          double *d_a,
                          double *d_b,
                          double *d_x_old,
                          double *d_x_new,
                          int *d_converged) {
    Metrics metrics;
    std::vector<double> x(static_cast<size_t>(n), 0.0);

    CUDA_CHECK(cudaMemset(d_x_old, 0, static_cast<size_t>(n) * sizeof(double)));
    CUDA_CHECK(cudaMemset(d_x_new, 0, static_cast<size_t>(n) * sizeof(double)));

    cudaEvent_t start = nullptr;
    cudaEvent_t stop = nullptr;
    CUDA_CHECK(cudaEventCreate(&start));
    CUDA_CHECK(cudaEventCreate(&stop));
    CUDA_CHECK(cudaEventRecord(start, 0));

    const int block = options.threads;
    const int grid = (n + block - 1) / block;

```

```
388     int h_converged = 0;
389     int iterations = 0;
390     while (iterations < options.max_iters) {
391         h_converged = 1;
392         CUDA_CHECK(cudaMemcpy(
393             d_converged, &h_converged, sizeof(int), cudaMemcpyHostToDevice));
394 
395         jacobi_iteration<<<grid, block>>>(
396             d_a, d_b, d_x_old, d_x_new, n, options.eps, d_converged);
397         CUDA_CHECK(cudaGetLastError());
398 
399         CUDA_CHECK(cudaMemcpy(
400             &h_converged, d_converged, sizeof(int), cudaMemcpyDeviceToHost));
401 
402         std::swap(d_x_old, d_x_new);
403         ++iterations;
404 
405         if (h_converged == 1) {
406             metrics.converged = true;
407             break;
408         }
409     }
410 
411     CUDA_CHECK(cudaEventRecord(stop, 0));
412     CUDA_CHECK(cudaEventSynchronize(stop));
413 
414     float elapsed_ms = 0.0f;
415     CUDA_CHECK(cudaEventElapsedTime(&elapsed_ms, start, stop));
416     CUDA_CHECK(cudaEventDestroy(start));
417     CUDA_CHECK(cudaEventDestroy(stop));
418 
419     CUDA_CHECK(cudaMemcpy(x.data(),
420                           d_x_old,
421                           static_cast<size_t>(n) * sizeof(double),
422                           cudaMemcpyDeviceToHost));
423 
424     metrics.elapsed_ms = static_cast<double>(elapsed_ms);
425     metrics.iterations = iterations;
426     metrics.residual_inf = compute_residual_inf(a, x, b, n);
427     metrics.x_error_inf = compute_x_error_inf(x, x_true);
428 
429     const double seconds = metrics.elapsed_ms / 1000.0;
430     const double flops =
431         (2.0 / 3.0) * static_cast<double>(n) * static_cast<double>(n) *
432         static_cast<double>(n);
433     metrics.gflops = (seconds > 0.0) ? (flops / seconds) / 1e9 : 0.0;
434     return metrics;
435 }
436 
437 static Metrics benchmark_size(const Options &options,
438                               int n,
439                               const std::vector<double> &a,
440                               const std::vector<double> &b,
441                               const std::vector<double> &x_true) {
442     const size_t matrix_bytes =
443         static_cast<size_t>(n) * static_cast<size_t>(n) * sizeof(double);
```

```
444     const size_t vector_bytes = static_cast<size_t>(n) * sizeof(double);
445
446     double *d_a = nullptr;
447     double *d_b = nullptr;
448     double *d_x_old = nullptr;
449     double *d_x_new = nullptr;
450     int *d_converged = nullptr;
451
452     CUDA_CHECK(cudaMalloc(reinterpret_cast<void **>(&d_a), matrix_bytes));
453     CUDA_CHECK(cudaMalloc(reinterpret_cast<void **>(&d_b), vector_bytes));
454     CUDA_CHECK(cudaMalloc(reinterpret_cast<void **>(&d_x_old), vector_bytes));
455     CUDA_CHECK(cudaMalloc(reinterpret_cast<void **>(&d_x_new), vector_bytes));
456     CUDA_CHECK(cudaMalloc(reinterpret_cast<void **>(&d_converged), sizeof(int)));
457
458     CUDA_CHECK(cudaMemcpy(
459         d_a, a.data(), matrix_bytes, cudaMemcpyHostToDevice));
460     CUDA_CHECK(cudaMemcpy(
461         d_b, b.data(), vector_bytes, cudaMemcpyHostToDevice));
462
463     for (int w = 0; w < options.warmup; ++w) {
464         (void)solve_once(options, n, a, b, x_true, d_a, d_b, d_x_old, d_x_new,
465                          d_converged);
466     }
467
468     Metrics best;
469     for (int run = 0; run < options.repeat; ++run) {
470         Metrics current = solve_once(
471             options, n, a, b, x_true, d_a, d_b, d_x_old, d_x_new, d_converged);
472         if (current.elapsed_ms < best.elapsed_ms) {
473             best = current;
474         }
475     }
476
477     CUDA_CHECK(cudaFree(d_a));
478     CUDA_CHECK(cudaFree(d_b));
479     CUDA_CHECK(cudaFree(d_x_old));
480     CUDA_CHECK(cudaFree(d_x_new));
481     CUDA_CHECK(cudaFree(d_converged));
482
483     return best;
484 }
485
486 int main(int argc, char **argv) {
487     Options options;
488     if (!parse_options(argc, argv, options)) {
489         return 0;
490     }
491
492     int device = 0;
493     CUDA_CHECK(cudaGetDevice(&device));
494     cudaDeviceProp prop{};
495     CUDA_CHECK(cudaGetDeviceProperties(&prop, device));
496
497     std::ofstream csv_file;
498     if (!options.csv_path.empty()) {
499         csv_file.open(options.csv_path.c_str(), std::ios::out | std::ios::trunc);
```

```
500         if (!csv_file) {
501             std::cerr << "Failed to open CSV file: " << options.csv_path << '\n';
502             return 1;
503         }
504         csv_file
505             << "n,elapsed_ms,iterations,residual_inf,x_error_inf,gflops,converged\n";
506     }
507
508     std::cout << "CUDA Jacobi LINPACK-like benchmark\n";
509     std::cout << "device = " << prop.name << ", compute capability = "
510               << prop.major << '.' << prop.minor << ", threads = "
511               << options.threads << ", repeat = " << options.repeat
512               << ", warmup = " << options.warmup
513               << ", eps = " << std::scientific << options.eps << std::defaultfloat
514               << "\n\n";
515
516     std::cout << std::left << std::setw(8) << "N" << std::setw(14) << "Time(ms)"
517               << std::setw(12) << "Iter" << std::setw(18) << "ResidualInf"
518               << std::setw(18) << "XerrInf" << std::setw(14) << "GFLOPS"
519               << "Status\n";
520
521     for (size_t i = 0; i < options.sizes.size(); ++i) {
522         const int n = options.sizes[i];
523         if (n <= 0) {
524             continue;
525         }
526
527         std::vector<double> a;
528         std::vector<double> x_true;
529         std::vector<double> b;
530         build_system(
531             n, options.seed + static_cast<unsigned int>(n), a, x_true, b);
532
533         Metrics metrics = benchmark_size(options, n, a, b, x_true);
534
535         std::cout << std::left << std::setw(8) << n << std::setw(14)
536                   << std::fixed << std::setprecision(4) << metrics.elapsed_ms
537                   << std::setw(12) << metrics.iterations << std::setw(18)
538                   << std::scientific << std::setprecision(3)
539                   << metrics.residual_inf << std::setw(18) << metrics.x_error_inf
540                   << std::setw(14) << std::fixed << std::setprecision(3)
541                   << metrics.gflops
542                   << (metrics.converged ? "converged" : "max_iters") << '\n';
543
544         if (csv_file) {
545             csv_file << n << ',' << std::fixed << std::setprecision(6)
546                      << metrics.elapsed_ms << ',' << metrics.iterations << ','
547                      << std::scientific << std::setprecision(8)
548                      << metrics.residual_inf << ',' << metrics.x_error_inf << ','
549                      << std::fixed << std::setprecision(6) << metrics.gflops
550                      << ',' << (metrics.converged ? 1 : 0) << '\n';
551         }
552     }
553
554     if (csv_file) {
555         csv_file.close();
```

```
556         }
557
558         return 0;
559     }
```

Приложение Б

Листинг 3. Файл `task1/scripts/run_cuda.slurm`

```
#!/usr/bin/env bash
#SBATCH --job-name=task1-cuda
#SBATCH --partition=tornado-k40
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=00:20:00
#SBATCH --output=results/%x-%j.out
#SBATCH --error=results/%x-%j.err

set -euo pipefail

cd "${SLURM_SUBMIT_DIR}"
ROOT_DIR="${SLURM_SUBMIT_DIR}"

module purge
module load compiler/gcc/11
module load nvidia/cuda/11.6u2

mkdir -p results bin

./scripts/build.sh

echo "===== account info ====="
whoami
hostname
date

echo
echo "===== slurm info ====="
echo "SLURM_JOB_ID=${SLURM_JOB_ID:-unknown}"
echo "SLURM_JOB_NAME=${SLURM_JOB_NAME:-unknown}"
echo "SLURM_JOB_PARTITION=${SLURM_JOB_PARTITION:-unknown}"
echo "SLURM_JOB_NUM_NODES=${SLURM_JOB_NUM_NODES:-unknown}"
echo "SLURM_NODELIST=${SLURM_NODELIST:-unknown}"
echo "CUDA_VISIBLE_DEVICES=${CUDA_VISIBLE_DEVICES:-unset}"
scontrol show job "${SLURM_JOB_ID}" || true

echo
echo "===== node config ====="
lscpu | sed -n '1,20p'
if [ -n "${SLURMD_NODENAME:-}" ]; then
    scontrol show node "${SLURMD_NODENAME}" || true
fi
nvidia-smi -L || true
nvidia-smi || true

echo
echo "===== benchmark ====="
./bin/linpack_cuda \
    --start 1000 \
    --step 500 \
```

```
52      --count 6 \
53      --eps 1e-6 \
54      --max-iters 15000 \
55      --threads 256 \
56      --repeat 3 \
57      --warmup 1 \
58      --csv "results/task1-cuda-${SLURM_JOB_ID}.csv"
```

Приложение В

Листинг 4. Файл `task1/scripts/run_intel_linpack.slurm`

```
#!/usr/bin/env bash
#SBATCH --job-name=task1-intel-linpack
#SBATCH --partition=tornado
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=56
#SBATCH --time=00:20:00
#SBATCH --output=stdio/%x-%j.out
#SBATCH --error=stdio/%x-%j.err

set -euo pipefail

TASK1_DIR="${SLURM_SUBMIT_DIR:-$PWD}"
module purge

LINPACK_DIR="${LINPACK_DIR:-$HOME/LINPACK}"
LINPACK_INPUT="${LINPACK_INPUT:-$TASK1_DIR/intel/lininput_report_xeon64}"

if [ ! -d "${LINPACK_DIR}" ]; then
    echo "LINPACK directory not found: ${LINPACK_DIR}"
    echo "Prepare Intel LINPACK in your home directory first."
    echo "Example:"
    echo "  mkdir -p \$HOME/LINPACK"
    echo "  tar -xzf \$HOME/linpack.tgz -C \$HOME/LINPACK"
    exit 1
fi

resolve_linpack_dir() {
    if [ -x "${LINPACK_DIR}/xlinpack_xeon64" ]; then
        printf '%s\n' "${LINPACK_DIR}"
        return 0
    fi

    local found
    found="$(find "${LINPACK_DIR}" -name xlinpack_xeon64 2>/dev/null | head -n 1 || true)
      "
    if [ -n "${found}" ]; then
        dirname "${found}"
        return 0
    fi

    return 1
}

if ! LINPACK_DIR="$(resolve_linpack_dir)"; then
    echo "Intel LINPACK binary not found under: ${LINPACK_DIR}"
    echo "Check archive contents with:"
    echo "  find ${LINPACK_DIR} -name xlinpack_xeon64 2>/dev/null"
    exit 1
fi

```

```
51 cd "${LINPACK_DIR}"
52
53 chmod +x ./* || true
54 chmod -x ./*.* || true
55 mkdir -p stdio
56
57 echo "===== account info ====="
58 whoami
59 hostname
60 date
61
62 echo
63 echo "===== slurm info ====="
64 echo "SLURM_JOB_ID=${SLURM_JOB_ID:-unknown}"
65 echo "SLURM_JOB_NAME=${SLURM_JOB_NAME:-unknown}"
66 echo "SLURM_JOB_PARTITION=${SLURM_JOB_PARTITION:-unknown}"
67 echo "SLURM_JOB_NUM_NODES=${SLURM_JOB_NUM_NODES:-unknown}"
68 echo "SLURM_NODELIST=${SLURM_NODELIST:-unknown}"
69 echo "OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-56}"
70 scontrol show job "${SLURM_JOB_ID}" || true
71
72 echo
73 echo "===== node config ====="
74 lscpu | sed -n '1,20p'
75 if [ -n "${SLURMD_NODENAME:-}" ]; then
76     scontrol show node "${SLURMD_NODENAME}" || true
77 fi
78
79 echo
80 echo "===== intel linpack ====="
81 echo "LINPACK_DIR=${LINPACK_DIR}"
82 echo "LINPACK_INPUT=${LINPACK_INPUT}"
83 export OMP_NUM_THREADS="${SLURM_CPUS_PER_TASK:-56}"
84 export MKL_NUM_THREADS="${SLURM_CPUS_PER_TASK:-56}"
85
86 srun ./xlinpack_xeon64 "${LINPACK_INPUT}"
```

Приложение Г

Листинг 5. Файл `task1/intel/lininput_report_xeon64`

```
1 Shared−memory version of Intel(R) Distribution for LINPACK* Benchmark. *Other names and brands
      may be claimed as the property of others.
2 Custom data file for task1 report.
3 6                 # number of tests
4 1000 1500 2000 2500 3000 3500 # problem sizes
5 1000 1504 2000 2504 3000 3504 # leading dimensions
6 8 6 6 5 5 5 # times to run a test
7 4 4 4 4 4 4 # alignment values (in KBytes)
```

Приложение Д

Листинг 6. Файл `task1/scripts/plot_task1_results.py`

```python
from __future__ import annotations

from pathlib import Path

import matplotlib.pyplot as plt


N_VALUES = [1000, 1500, 2000, 2500, 3000, 3500]

CUDA_TIME_MS = [5.0083, 7.4931, 8.3563, 10.4837, 12.6709, 14.8861]
INTEL_TIME_S = [0.010, 0.020, 0.028, 0.042, 0.069, 0.100]
INTEL_TIME_MS = [value * 1000.0 for value in INTEL_TIME_S]

CUDA_GFLOPS = [133.114, 300.277, 638.244, 993.608, 1420.573, 1920.138]
INTEL_GFLOPS = [67.331, 114.360, 193.276, 250.731, 260.451, 286.264]


def configure_plot() -> None:
    plt.style.use("seaborn-v0_8-whitegrid")
    plt.rcParams["figure.figsize"] = (8.4, 5.2)
    plt.rcParams["figure.dpi"] = 140
    plt.rcParams["savefig.dpi"] = 220
    plt.rcParams["font.size"] = 11
    plt.rcParams["axes.labelsize"] = 11
    plt.rcParams["axes.titlesize"] = 12
    plt.rcParams["legend.fontsize"] = 10


def plot_time(output_path: Path) -> None:
    plt.figure()
    plt.plot(
        N_VALUES,
        INTEL_TIME_MS,
        marker="o",
        linewidth=2.2,
        color="#1f77b4",
        label="Intel LINPACK",
    )
    plt.plot(
        N_VALUES,
        CUDA_TIME_MS,
        marker="s",
        linewidth=2.2,
        color="#d62728",
        label="Собственная реализация",
    )
    plt.xlabel("Размер матрицы N")
    plt.ylabel("Время решения, мс")
    plt.legend(loc="upper left")
    plt.tight_layout()
    plt.savefig(output_path, bbox_inches="tight")
```

```
52      plt.close()
53
54
55  def plot_gflops(output_path: Path) -> None:
56      plt.figure()
57      plt.plot(
58          N_VALUES,
59          INTEL_GFLOPS,
60          marker="o",
61          linewidth=2.2,
62          color="#1f77b4",
63          label="Intel LINPACK",
64      )
65      plt.plot(
66          N_VALUES,
67          CUDA_GFLOPS,
68          marker="s",
69          linewidth=2.2,
70          color="#d62728",
71          label="Собственная реализация",
72      )
73      plt.xlabel("Размер матрицы N")
74      plt.ylabel("Производительность, GFLOPS")
75      plt.legend(loc="upper left")
76      plt.tight_layout()
77      plt.savefig(output_path, bbox_inches="tight")
78      plt.close()
79
80
81  def main() -> None:
82      configure_plot()
83
84      output_dir = Path(__file__).resolve().parents[2] / "report" / "img"
85      output_dir.mkdir(parents=True, exist_ok=True)
86
87      plot_time(output_dir / "task1-time-comparison.png")
88      plot_gflops(output_dir / "task1-gflops-comparison.png")
89
90
91  if __name__ == "__main__":
92      main()
```

Приложение Е

Листинг 7. Файл `task2/src/wave_mpi.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>
#include <mpi.h>

#define INF UINT_MAX
#define OBSTACLE_PROB 10

static void generate_polygon(int *P, int n) {
    srand(42);
    for (int i = 0; i < n * n; i++)
        P[i] = (rand() % 100 < OBSTACLE_PROB) ? -1 : 0;
    int sx = 2, sy = 2;
    int fx = n - 3, fy = n - 3;
    P[sx * n + sy] = 0;
    P[fx * n + fy] = 0;
}

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (argc < 2) {
        if (rank == 0)
            fprintf(stderr, "Usage: mpirun -np <P> %s <matrix_size> [csv_file]\n", argv[0]);
        MPI_Finalize();
        return 1;
    }

    int n = atoi(argv[1]);
    const char *csv_path = (argc >= 3) ? argv[2] : NULL;
    int sx = 2, sy = 2;
    int fx = n - 3, fy = n - 3;

    int *P = (int *)malloc(n * n * sizeof(int));
    unsigned int *dist = NULL;

    if (rank == 0) {
        generate_polygon(P, n);
        dist = (unsigned int *)malloc(n * n * sizeof(unsigned int));
        for (int i = 0; i < n * n; i++)
            dist[i] = INF;
        dist[sx * n + sy] = 0;
    }

    MPI_Bcast(P, n * n, MPI_INT, 0, MPI_COMM_WORLD);

```

```
52      int base_rows = n / size;
53      int remainder = n % size;
54      int local_rows = base_rows + (rank < remainder ? 1 : 0);
55      int start_row = rank * base_rows + (rank < remainder ? rank : remainder);
56
57      int ghost_top = (rank > 0) ? 1 : 0;
58      int ghost_bot = (rank < size - 1) ? 1 : 0;
59      int total_local = (ghost_top + local_rows + ghost_bot) * n;
60
61      unsigned int *local_dist = (unsigned int *)malloc(total_local * sizeof(unsigned int));
62      int *local_P = (int *)malloc(total_local * sizeof(int));
63
64      for (int i = 0; i < total_local; i++) {
65          local_dist[i] = INF;
66          local_P[i] = -1;
67      }
68
69      int *sendcounts = NULL, *displs = NULL;
70      if (rank == 0) {
71          sendcounts = (int *)malloc(size * sizeof(int));
72          displs = (int *)malloc(size * sizeof(int));
73          int off = 0;
74          for (int r = 0; r < size; r++) {
75              int rr = base_rows + (r < remainder ? 1 : 0);
76              sendcounts[r] = rr * n;
77              displs[r] = off;
78              off += rr * n;
79          }
80      }
81
82      MPI_Scatterv(
83          (rank == 0) ? dist : NULL, sendcounts, displs, MPI_UNSIGNED,
84          local_dist + ghost_top * n, local_rows * n, MPI_UNSIGNED,
85          0, MPI_COMM_WORLD);
86
87      for (int i = 0; i < local_rows; i++)
88          memcpy(local_P + (ghost_top + i) * n, P + (start_row + i) * n, n * sizeof(int));
89
90      if (ghost_top) {
91          memcpy(local_P, P + (start_row - 1) * n, n * sizeof(int));
92      }
93      if (ghost_bot) {
94          memcpy(local_P + (ghost_top + local_rows) * n,
95                 P + (start_row + local_rows) * n, n * sizeof(int));
96      }
97
98      MPI_Barrier(MPI_COMM_WORLD);
99      double t_start = MPI_Wtime();
100
101     int prev_rank = (rank > 0) ? rank - 1 : MPI_PROC_NULL;
102     int next_rank = (rank < size - 1) ? rank + 1 : MPI_PROC_NULL;
103
104     int iteration = 0;
105     int global_changed;
106     do {
107         /* exchange ghost rows */
```

```
108         MPI_Sendrecv(
109             local_dist + ghost_top * n, n, MPI_UNSIGNED, prev_rank, 0,
110             local_dist + (ghost_top + local_rows) * n, n, MPI_UNSIGNED, next_rank, 0,
111             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
112
113         MPI_Sendrecv(
114             local_dist + (ghost_top + local_rows - 1) * n, n, MPI_UNSIGNED, next_rank, 1,
115             local_dist, n, MPI_UNSIGNED, prev_rank, 1,
116             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
117
118         int local_changed = 0;
119
120         for (int li = ghost_top; li < ghost_top + local_rows; li++) {
121             for (int j = 0; j < n; j++) {
122                 int idx = li * n + j;
123                 if (local_P[idx] == -1) continue;
124
125                 unsigned int cur = local_dist[idx];
126                 unsigned int mn = cur;
127
128                 if (li > 0 && local_dist[(li - 1) * n + j] != INF)
129                     mn = (local_dist[(li - 1) * n + j] + 1 < mn) ? local_dist[(li - 1) * n + j] + 1 : mn;
130                 if (li < ghost_top + local_rows + ghost_bot - 1 && local_dist[(li + 1) * n + j] != INF)
131                     mn = (local_dist[(li + 1) * n + j] + 1 < mn) ? local_dist[(li + 1) * n + j] + 1 : mn;
132                 if (j > 0 && local_dist[li * n + j - 1] != INF)
133                     mn = (local_dist[li * n + j - 1] + 1 < mn) ? local_dist[li * n + j - 1] + 1 : mn;
134                 if (j < n - 1 && local_dist[li * n + j + 1] != INF)
135                     mn = (local_dist[li * n + j + 1] + 1 < mn) ? local_dist[li * n + j + 1] + 1 : mn;
136
137                 if (mn < cur) {
138                     local_dist[idx] = mn;
139                     local_changed = 1;
140                 }
141             }
142         }
143
144         MPI_Allreduce(&local_changed, &global_changed, 1, MPI_INT, MPI_LOR,
      MPI_COMM_WORLD);
145         iteration++;
146     } while (global_changed && iteration < 2 * n);
147
148     double t_end = MPI_Wtime();
149     double elapsed_ms = (t_end - t_start) * 1000.0;
150
151     MPI_Gatherv(
152         local_dist + ghost_top * n, local_rows * n, MPI_UNSIGNED,
153         (rank == 0) ? dist : NULL, sendcounts, displs, MPI_UNSIGNED,
154         0, MPI_COMM_WORLD);
155
156     if (rank == 0) {
157         unsigned int path_len = dist[fx * n + fy];
158         if (path_len == INF)
159             printf("n=%d  Path not found!  time=%.2f ms  iters=%d  procs=%d\n",
160                    n, elapsed_ms, iteration, size);
161         else
162             printf("n=%d  path_len=%u  time=%.2f ms  iters=%d  procs=%d\n",
```

```
163                     n, path_len, elapsed_ms, iteration, size);
164
165         if (csv_path) {
166             FILE *fp = fopen(csv_path, "a");
167             if (fp) {
168                 fprintf(fp, "%d,%d,%.4f,%u,%d\n",
169                         n, size, elapsed_ms, path_len, iteration);
170                 fclose(fp);
171             }
172         }
173
174         free(sendcounts);
175         free(displs);
176         free(dist);
177     }
178
179     free(P);
180     free(local_dist);
181     free(local_P);
182
183     MPI_Finalize();
184     return 0;
185 }
```

Приложение Ж

Листинг 8. Файл `task2/src/wave_cuda.cu`

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>

#define INF UINT_MAX
#define OBSTACLE_PROB 10
#define DEFAULT_BLOCKS  256
#define DEFAULT_THREADS 256

static void generate_polygon(int *P, int n) {
    srand(42);
    for (int i = 0; i < n * n; i++)
        P[i] = (rand() % 100 < OBSTACLE_PROB) ? -1 : 0;
    int sx = 2, sy = 2;
    int fx = n - 3, fy = n - 3;
    P[sx * n + sy] = 0;
    P[fx * n + fy] = 0;
}

__global__ void wave_step(int *P, unsigned int *dist, int n, bool *changed) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;

    while (tid < n * n) {
        int i = tid / n;
        int j = tid % n;

        if (P[tid] != -1) {
            unsigned int cur = dist[tid];
            unsigned int mn = cur;

            if (i > 0     && dist[(i-1)*n + j] != INF) mn = min(mn, dist[(i-1)*n + j] + 1);
            if (i < n - 1 && dist[(i+1)*n + j] != INF) mn = min(mn, dist[(i+1)*n + j] + 1);
            if (j > 0     && dist[i*n + j - 1] != INF) mn = min(mn, dist[i*n + j - 1] + 1);
            if (j < n - 1 && dist[i*n + j + 1] != INF) mn = min(mn, dist[i*n + j + 1] + 1);

            if (mn < cur) {
                dist[tid] = mn;
                *changed = true;
            }
        }
        tid += blockDim.x * gridDim.x;
    }
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <matrix_size> [blocks] [threads] [csv_file]\n", argv[0]);
        return 1;
    }
```

```
52
53      int n       = atoi(argv[1]);
54      int blocks  = (argc >= 3) ? atoi(argv[2]) : DEFAULT_BLOCKS;
55      int threads = (argc >= 4) ? atoi(argv[3]) : DEFAULT_THREADS;
56      const char *csv_path = (argc >= 5) ? argv[4] : NULL;
57
58      int sx = 2, sy = 2;
59      int fx = n - 3, fy = n - 3;
60
61      int *P = (int *)malloc(n * n * sizeof(int));
62      generate_polygon(P, n);
63
64      unsigned int *dist_h = (unsigned int *)malloc(n * n * sizeof(unsigned int));
65      for (int i = 0; i < n * n; i++) dist_h[i] = INF;
66      dist_h[sx * n + sy] = 0;
67
68      int *d_P;
69      unsigned int *d_dist;
70      bool *d_changed;
71      cudaMalloc(&d_P,       n * n * sizeof(int));
72      cudaMalloc(&d_dist,    n * n * sizeof(unsigned int));
73      cudaMalloc(&d_changed, sizeof(bool));
74
75      cudaMemcpy(d_P,    P,      n * n * sizeof(int),          cudaMemcpyHostToDevice);
76      cudaMemcpy(d_dist, dist_h, n * n * sizeof(unsigned int), cudaMemcpyHostToDevice);
77
78      cudaEvent_t t0, t1;
79      cudaEventCreate(&t0);
80      cudaEventCreate(&t1);
81      cudaEventRecord(t0);
82
83      int iterations = 0;
84      bool changed;
85      do {
86          changed = false;
87          cudaMemcpy(d_changed, &changed, sizeof(bool), cudaMemcpyHostToDevice);
88          wave_step<<<blocks, threads>>>(d_P, d_dist, n, d_changed);
89          cudaDeviceSynchronize();
90          cudaMemcpy(&changed, d_changed, sizeof(bool), cudaMemcpyDeviceToHost);
91          iterations++;
92      } while (changed && iterations < 2 * n);
93
94      cudaEventRecord(t1);
95      cudaEventSynchronize(t1);
96      float elapsed_ms = 0;
97      cudaEventElapsedTime(&elapsed_ms, t0, t1);
98
99      cudaMemcpy(dist_h, d_dist, n * n * sizeof(unsigned int), cudaMemcpyDeviceToHost);
100
101     unsigned int path_len = dist_h[fx * n + fy];
102     if (path_len == INF)
103         printf("n=%d  Path not found!  time=%.2f ms  iters=%d  blocks=%d  threads=%d\n",
104                n, elapsed_ms, iterations, blocks, threads);
105     else
106         printf("n=%d  path_len=%u  time=%.2f ms  iters=%d  blocks=%d  threads=%d\n",
107                n, path_len, elapsed_ms, iterations, blocks, threads);
```

```
108
109     if (csv_path) {
110         FILE *fp = fopen(csv_path, "a");
111         if (fp) {
112             fprintf(fp, "%d,cuda,%.4f,%u,%d\n", n, elapsed_ms, path_len, iterations);
113             fclose(fp);
114         }
115     }
116
117     free(P);
118     free(dist_h);
119     cudaFree(d_P);
120     cudaFree(d_dist);
121     cudaFree(d_changed);
122     cudaEventDestroy(t0);
123     cudaEventDestroy(t1);
124
125     return 0;
126 }
```

Приложение З

Листинг 9. Файл `task2/scripts/run_mpi.slurm`

```
#!/usr/bin/env bash
#SBATCH --job-name=task2-mpi
#SBATCH --partition=tornado
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=56
#SBATCH --time=00:20:00
#SBATCH --output=results/%x-%j.out
#SBATCH --error=results/%x-%j.err

set -euo pipefail

cd "${SLURM_SUBMIT_DIR}"

module purge
module load compiler/gcc/11
module load mpi/openmpi

mkdir -p results bin

./scripts/build_mpi.sh

RANKS=${SLURM_JOB_NUM_NODES}

echo "===== account info ====="
whoami; hostname; date

echo
echo "===== slurm info ====="
echo "SLURM_JOB_ID=${SLURM_JOB_ID:-unknown}"
echo "SLURM_JOB_PARTITION=${SLURM_JOB_PARTITION:-unknown}"
echo "SLURM_JOB_NUM_NODES=${SLURM_JOB_NUM_NODES:-unknown}"
echo "SLURM_NODELIST=${SLURM_NODELIST:-unknown}"
echo "RANKS=${RANKS}"
scontrol show job "${SLURM_JOB_ID}" || true

echo
echo "===== node config ====="
lscpu | head -20
if [ -n "${SLURMD_NODENAME:-}" ]; then
    scontrol show node "${SLURMD_NODENAME}" || true
fi

CSV="results/task2-mpi-${RANKS}n-${SLURM_JOB_ID}.csv"
echo "n,procs,time_ms,path_len,iterations" > "$CSV"

echo
echo "===== benchmark (${RANKS} nodes / ${RANKS} ranks) ====="
for N in 500 1000 2000 3000 5000; do
    echo "--- n=$N ---"
    mpirun -np "${RANKS}" --map-by ppr:1:node --bind-to none ./bin/wave_mpi "$N" "$CSV
      "
```

```
51  done
52
53  echo
54  echo "=====␣done␣====="
```

Приложение И

Листинг 10. Файл `task2/scripts/run_cuda.slurm`

```
#!/usr/bin/env bash
#SBATCH --job-name=task2-cuda
#SBATCH --partition=tornado-k40
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=00:20:00
#SBATCH --output=results/%x-%j.out
#SBATCH --error=results/%x-%j.err

set -euo pipefail

cd "${SLURM_SUBMIT_DIR}"

module purge
module load compiler/gcc/11
module load nvidia/cuda/11.6u2

mkdir -p results bin

./scripts/build_cuda.sh

echo "===== account info ====="
whoami; hostname; date

echo
echo "===== slurm info ====="
echo "SLURM_JOB_ID=${SLURM_JOB_ID:-unknown}"
echo "SLURM_JOB_PARTITION=${SLURM_JOB_PARTITION:-unknown}"
echo "SLURM_NODELIST=${SLURM_NODELIST:-unknown}"
scontrol show job "${SLURM_JOB_ID}" || true

echo
echo "===== node config ====="
lscpu | head -20
nvidia-smi -L || true
nvidia-smi || true

CSV="results/task2-cuda-${SLURM_JOB_ID}.csv"
echo "n,impl,time_ms,path_len,iterations" > "$CSV"

echo
echo "===== benchmark ====="
for N in 500 1000 2000 3000 5000; do
    echo "--- n=$N ---"
    ./bin/wave_cuda "$N" 256 256 "$CSV"
done

echo
echo "===== done ====="
```

Приложение К

Листинг 11. Файл `task2/scripts/plot_task2_results.py`

```
#!/usr/bin/env python3
"""Строит
 график зависимости времени вычисления от размера полигонадля
 MPI (1, 2, 4 узла) и CUDA.Использование

:
    python3 plot_task2_results.py \
        --mpi1 results/task2-mpi-1n-XXXXX.csv \
        --mpi2 results/task2-mpi-2n-XXXXX.csv \
        --mpi4 results/task2-mpi-4n-XXXXX.csv \
        --cuda results/task2-cuda-XXXXX.csv \
        -o report/img/task2-time-comparison.png
"""
import argparse
import csv
from pathlib import Path

import matplotlib.pyplot as plt


def read_mpi_csv(path: str) -> tuple[list[int], list[float]]:
    sizes, times = [], []
    with open(path) as f:
        reader = csv.DictReader(f)
        for row in reader:
            sizes.append(int(row["n"]))
            times.append(float(row["time_ms"]))
    return sizes, times


def read_cuda_csv(path: str) -> tuple[list[int], list[float]]:
    sizes, times = [], []
    with open(path) as f:
        reader = csv.DictReader(f)
        for row in reader:
            sizes.append(int(row["n"]))
            times.append(float(row["time_ms"]))
    return sizes, times


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--mpi1", required=True, help="CSV for MPI 1 node")
    parser.add_argument("--mpi2", required=True, help="CSV for MPI 2 nodes")
    parser.add_argument("--mpi4", required=True, help="CSV for MPI 4 nodes")
    parser.add_argument("--cuda", required=True, help="CSV for CUDA")
    parser.add_argument("-o", "--output", default="task2-time-comparison.png")
    args = parser.parse_args()

    fig, ax = plt.subplots(figsize=(10, 6))

```

```
52      for label, path in [
53          ("MPI 1 node", args.mpi1),
54          ("MPI 2 nodes", args.mpi2),
55          ("MPI 4 nodes", args.mpi4),
56          ("CUDA", args.cuda),
57      ]:
58          sizes, times = read_mpi_csv(path) if "mpi" in label.lower() else read_cuda_csv(path)
59          ax.plot(sizes, times, marker="o", label=label)
60
61      ax.set_xlabel("Размер полигона n")
62      ax.set_ylabel("Время, мс")
63      ax.set_title("Зависимость времени вычисления от размера полигона")
64      ax.legend()
65      ax.grid(True, alpha=0.3)
66
67      Path(args.output).parent.mkdir(parents=True, exist_ok=True)
68      fig.savefig(args.output, dpi=150, bbox_inches="tight")
69      print(f"Saved: {args.output}")
70
71
72  if __name__ == "__main__":
73      main()
```