Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Вариант 1 по заданию 3-й поток по курсу «Суперкомпьютерное моделирование и технологии»

Работу выполнил:
Юань Дунлян
Группа: 623

Москва
2024

Содержание

1. Постановка задачи

В данной работе требуется численно решить трехмерное гиперболическое уравнение:

$$\frac{\partial^2 u}{\partial t^2} = \Delta u$$

в области $\Omega = [0 \leqslant x \leqslant L_x] \times [0 \leqslant y \leqslant L_y] \times [0 \leqslant z \leqslant L_z]$ для времени $(0 < t \leqslant T]$ с начальными условиями:

$$u|_{t=0} = \varphi(x, y, z),$$
$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = 0$$

и граничными условиями первого рода.

Задача должна быть реализована с использованием:

- Блочного разбиения расчетной области между процессами
- Явной разностной схемы для аппроксимации уравнения
- Параллельных вычислений на суперкомпьютере IBM Polus

Требуется:

1. Реализовать параллельный алгоритм решения задачи
2. Провести расчеты с различными параметрами сетки
3. Исследовать точность полученного численного решения
4. Оценить эффективность параллельной реализации

В данном варианте задачи аналитическое решение имеет вид:

$$u_{analytical} = \sin\left(\frac{\pi x}{L_x}\right) \cdot \sin\left(\frac{\pi y}{L_y}\right) \cdot \sin\left(\frac{\pi z}{L_z}\right) \cdot \cos(a_t t)$$

где

$$a_t = \pi \sqrt{\frac{1}{L_x^2} + \frac{1}{L_y^2} + \frac{1}{L_z^2}}$$

Легко проверить, что данное решение удовлетворяет граничным условиям первого рода:

$$u(0, y, z, t) = 0, \quad u(L_x, y, z, t) = 0,$$
$$u(x, 0, z, t) = 0, \quad u(x, L_y, z, t) = 0,$$
$$u(x, y, 0, t) = 0, \quad u(x, y, L_z, t) = 0$$

и начальным условиям:

$$u|_{t=0} = \sin\left(\frac{\pi x}{L_x}\right) \cdot \sin\left(\frac{\pi y}{L_y}\right) \cdot \sin\left(\frac{\pi z}{L_z}\right)$$
$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = 0$$

Данное аналитическое решение будет использоваться для оценки точности численного метода.

2. Численный метод решения задачи

Для численного решения поставленной задачи используем метод конечных разностей на равномерной сетке. Введем пространственно-временную сетку:

$$\bar{\omega}_h = \{(x_i = ih_x, y_j = jh_y, z_k = kh_z), i = 0, \ldots, N_x, j = 0, \ldots, N_y, k = 0, \ldots, N_z\},$$
$$h_x = \frac{L_x}{N_x}, \quad h_y = \frac{L_y}{N_y}, \quad h_z = \frac{L_z}{N_z},$$
$$\omega_\tau = \{t_n = n\tau, n = 0, 1, \ldots, K, \tau K = T\},$$

где ω_h - множество внутренних узлов сетки, γ_h - множество граничных узлов.

2.1. Разностная схема

Для аппроксимации уравнения используем явную схему:

$$\frac{u^{n+1}_{ijk} - 2u^n_{ijk} + u^{n-1}_{ijk}}{\tau^2} = \Delta_h u^n_{ijk},$$

где Δ_h - семиточечный разностный оператор Лапласа:

$$\begin{aligned}\Delta_h u^n_{ijk} = &\frac{u^n_{i+1,j,k} - 2u^n_{i,j,k} + u^n_{i-1,j,k}}{h_x^2} + \\ &\frac{u^n_{i,j+1,k} - 2u^n_{i,j,k} + u^n_{i,j-1,k}}{h_y^2} + \\ &\frac{u^n_{i,j,k+1} - 2u^n_{i,j,k} + u^n_{i,j,k-1}}{h_z^2}\end{aligned}$$

2.2. Инициализация и алгоритм решения

Для начала расчета необходимо задать значения решения на первых двух временных слоях.

1. На нулевом слое используем начальное условие:

$$u^0_{ijk} = \sin\left(\frac{\pi x_i}{L_x}\right)\sin\left(\frac{\pi y_j}{L_y}\right)\sin\left(\frac{\pi z_k}{L_z}\right)$$

2. Для первого временного слоя используем аппроксимацию второго порядка:

$$u^1_{ijk} = u^0_{ijk} + \frac{\tau^2}{2}\Delta_h u^0_{ijk}$$

3. Для последующих временных слоев используем явную схему:

$$u^{n+1}_{ijk} = 2u^n_{ijk} - u^{n-1}_{ijk} + \tau^2 \Delta_h u^n_{ijk}$$

3. Параллельная реализация MPI + OpenMP и тестирование на Polus

3.1. Тестирование на Polus

Тестирование проводилось на вычислительном комплексе Polus МГУ со следующими характеристиками:

- Процессоры: IBM POWER8
- Число ядер на узел: 20
- Объём оперативной памяти: 256GB на узел

3.1.1. Параметры тестовой задачи

- Временной интервал: T
- Шаг по времени: τ
- Число шагов по времени: K

3.1.2. Вычисление погрешности δ

$$\delta = \|e\|_{L^2} = \left(\sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \sum_{k=1}^{N_z} |u_{i,j,k}^{numerical} - u_{i,j,k}^{exact}|^2 \Delta x \Delta y \Delta z \right)^{\frac{1}{2}}$$

3.1.3. Результаты расчетов на IBM Polus для программы OpenMP версии

Область: $[0,1] \times [0,1] \times [0,1]$, $\tau = 0.001$, $K = 20$

Число OpenMP потоков N_p	Число точек сетки N^3	Время решения T /(s)	Ускорение S	Погрешность δ
1	128^3	0.46	1.00	1.00e-07
2	128^3	0.28	1.64	1.00e-07
4	128^3	0.14	3.29	1.00e-07
8	128^3	0.07	6.57	1.00e-07
10	128^3	0.06	7.67	1.00e-07
16	128^3	0.04	11.50	1.00e-07
20	128^3	0.03	15.33	1.00e-07
32	128^3	0.043	10.70	1.00e-07
40	128^3	0.036	12.78	1.00e-07
64	128^3	0.034	13.53	1.00e-07
120	128^3	0.032	14.38	1.00e-07
160	128^3	0.029	15.86	1.00e-07
1	256^3	3.75	1.00	2.12e-08
2	256^3	2.33	1.61	2.12e-08
4	256^3	1.175	3.19	2.12e-08
8	256^3	0.593	6.32	2.12e-08
10	256^3	0.491	7.64	2.12e-08
16	256^3	0.302	12.42	2.12e-08
20	256^3	0.243	15.43	2.12e-08
32	256^3	0.484	7.75	2.12e-08
40	256^3	0.279	13.44	2.12e-08
64	256^3	0.267	14.04	2.12e-08
120	256^3	0.227	16.52	2.12e-08
160	256^3	0.219	17.12	2.12e-08
1	512^3	30.08	1.00	1.40e-09
2	512^3	18.79	1.60	1.40e-09
4	512^3	9.52	3.16	1.40e-09
8	512^3	4.83	6.23	1.40e-09
10	512^3	3.92	7.67	1.40e-09
16	512^3	2.48	12.13	1.40e-09
20	512^3	2.01	14.96	1.40e-09
32	512^3	2.81	10.70	1.40e-09
40	512^3	2.27	13.25	1.40e-09
64	512^3	2.20	13.67	1.40e-09
120	512^3	1.79	16.80	1.40e-09
160	512^3	1.73	17.39	1.40e-09

Область: $[0,\pi]\times[0,\pi]\times[0,\pi]$, $\tau = 0.001$, $K = 20$

Число OpenMP потоков N_p	Число точек сетки N^3	Время решения T /(s)	Ускорение S	Погрешность δ
1	128^3	0.45	1.00	1.07e-08
2	128^3	0.28	1.57	1.07e-08
4	128^3	0.14	3.14	1.07e-08
8	128^3	0.07	6.29	1.07e-08
10	128^3	0.05	8.80	1.07e-08
16	128^3	0.037	11.89	1.07e-08
20	128^3	0.031	14.19	1.07e-08
32	128^3	0.044	10.00	1.07e-08
40	128^3	0.036	12.22	1.07e-08
64	128^3	0.034	12.94	1.07e-08
120	128^3	0.032	13.75	1.07e-08
160	128^3	0.029	15.17	1.07e-08
1	256^3	3.79	1.00	2.62e-09
2	256^3	2.33	1.55	2.62e-09
4	256^3	1.175	3.06	2.62e-09
8	256^3	0.593	6.07	2.62e-09
10	256^3	0.491	7.33	2.62e-09
16	256^3	0.302	11.92	2.62e-09
20	256^3	0.243	14.81	2.62e-09
32	256^3	0.484	7.44	2.62e-09
40	256^3	0.279	12.90	2.62e-09
64	256^3	0.267	13.48	2.62e-09
120	256^3	0.227	15.86	2.62e-09
160	256^3	0.219	16.44	2.62e-09
1	512^3	29.57	1.00	6.14e-10
2	512^3	18.75	1.54	6.14e-10
4	512^3	9.52	3.03	6.14e-10
8	512^3	4.81	6.00	6.14e-10
10	512^3	3.91	7.39	6.14e-10
16	512^3	2.49	11.60	6.14e-10
20	512^3	2.01	14.37	6.14e-10
32	512^3	2.82	10.24	6.14e-10
40	512^3	2.27	12.72	6.14e-10
64	512^3	2.23	12.95	6.14e-10
120	512^3	1.81	15.96	6.14e-10
160	512^3	1.74	16.60	6.14e-10

3.1.4. Результаты расчетов на IBM Polus для программы MPI версии

Область: $[0,1] \times [0,1] \times [0,1]$, $\tau = 0.001$, $K = 20$

Число MPI процессов N_p	Число точек сетки N^3	Время решения T /(s)	Ускорение S	Погрешность δ
1	128^3	0.46	1.00	1.00e-07
2	128^3	0.29	1.59	1.00e-07
4	128^3	0.14	3.29	1.00e-07
8	128^3	0.12	3.83	1.00e-07
10	128^3	0.09	5.11	1.00e-07
16	128^3	0.11	4.18	1.00e-07
20	128^3	0.14	3.29	1.00e-07
32	128^3	0.08	5.75	1.00e-07
1	256^3	3.75	1.00	2.12e-08
2	256^3	2.35	1.60	2.12e-08
4	256^3	1.17	3.21	2.12e-08
8	256^3	0.61	6.15	2.12e-08
10	256^3	0.53	7.08	2.12e-08
16	256^3	0.70	5.36	2.12e-08
20	256^3	0.51	7.35	2.12e-08
32	256^3	0.28	13.39	2.12e-08
1	512^3	30.08	1.00	1.40e-09
2	512^3	19.34	1.56	1.40e-09
4	512^3	9.67	3.11	1.40e-09
8	512^3	7.34	4.10	1.40e-09
10	512^3	4.71	6.39	1.40e-09
16	512^3	4.06	7.41	1.40e-09
20	512^3	3.14	9.58	1.40e-09
32	512^3	1.29	23.31	1.40e-09

Область: $[0,\pi]\times[0,\pi]\times[0,\pi]$, $\tau = 0.001$, $K = 20$

Число MPI процессов N_p	Число точек сетки N^3	Время решения T /(s)	Ускорение S	Погрешность δ
1	128^3	0.45	1.00	1.07e-08
2	128^3	0.28	1.61	1.07e-08
4	128^3	0.15	3.00	1.07e-08
8	128^3	0.12	3.75	1.07e-08
10	128^3	0.05	9.00	1.07e-08
16	128^3	0.11	4.09	1.07e-08
20	128^3	0.08	5.63	1.07e-08
32	128^3	0.073	6.16	1.07e-08
1	256^3	3.79	1.00	2.62e-09
2	256^3	2.32	1.63	2.62e-09
4	256^3	1.20	3.16	2.62e-09
8	256^3	0.62	6.11	2.62e-09
10	256^3	0.47	8.06	2.62e-09
16	256^3	0.81	4.68	2.62e-09
20	256^3	0.30	12.63	2.62e-09
32	256^3	0.379	10.00	2.62e-09
1	512^3	29.57	1.00	6.14e-10
2	512^3	18.57	1.59	6.14e-10
4	512^3	9.38	3.15	6.14e-10
8	512^3	4.77	6.20	6.14e-10
10	512^3	4.78	6.19	6.14e-10
16	512^3	2.54	11.64	6.14e-10
20	512^3	2.11	14.01	6.14e-10
32	512^3	1.27	23.28	6.14e-10

3.1.5. Результаты расчетов на IBM Polus для гибритной программы MPI+OpenMP версии

Область: $[0,1]\times[0,1]\times[0,1]$, $\tau = 0.001$, $K = 20$

Число MPI процессов N_p	Число OpenMP нитей в процессе	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	4	128^3	0.16	2.88	1.00e-07
2	4	128^3	0.08	5.75	1.00e-07
4	4	128^3	0.04	11.50	1.00e-07
8	4	128^3	0.076	6.05	1.00e-07
1	4	256^3	1.30	2.88	2.12e-08
2	4	256^3	0.64	5.86	2.12e-08
4	4	256^3	0.33	11.36	2.12e-08
8	4	256^3	0.238	15.76	2.12e-08
1	4	512^3	10.36	2.90	1.40e-09
2	4	512^3	5.62	5.35	1.40e-09
4	4	512^3	2.65	11.35	1.40e-09
8	4	512^3	1.38	21.80	1.40e-09

Область: $[0, \pi] \times [0, \pi] \times [0, \pi]$, $\tau = 0.001$, $K = 20$

Число MPI процессов N_p	Число OpenMP нитей в процессе	Число точек сетки N^3	Время решения T	Ускорение S	Погрешность δ
1	4	128^3	0.16	2.81	1.07e-08
2	4	128^3	0.08	5.63	1.07e-08
4	4	128^3	0.04	11.25	1.07e-08
8	4	128^3	0.063	7.14	1.07e-08
1	4	256^3	1.30	2.92	2.62e-09
2	4	256^3	0.64	5.92	2.62e-09
4	4	256^3	0.48	7.90	2.62e-09
8	4	256^3	0.22	17.23	2.62e-09
1	4	512^3	10.42	2.84	6.14e-10
2	4	512^3	5.19	5.70	6.14e-10
4	4	512^3	2.63	11.24	6.14e-10
8	4	512^3	1.36	21.74	6.14e-10

4. Параллельная реализация MPI + CUDA и тестирование на Polus

4.1. Тестирование на Polus

Тестирование проводилось на вычислительном комплексе Polus МГУ со следующими характеристиками:

- Графические ускорители: NVIDIA Tesla P100-SXM2-16GB
- Процессоры: IBM POWER8
- Число ядер на узел: 20

4.1.1. Параметры тестовой задачи

- Сетка: $512 \times 512 \times 512$
- Временной шаг: $\tau = 0.001$
- Число временных шагов: $K = 20$

4.2. Команды компиляции и скрипты запуска

4.2.1. Последовательная версия

Команда компиляции:

```
g++ -O3 wave.cpp -o wave -std=c++11
```

Скрипт запуска:

```
#!/bin/bash
#BSUB -J "Sequential"
#BSUB -o "Sequential_%J.out"
#BSUB -e "Sequential_%J.err"
#BSUB -n 1
#BSUB -W 0:30
#BSUB -R "span[hosts=1]"
#BSUB -m "polus-c3-ib"
export OMP_NUM_THREADS=1
GRID_SIZES=(512)
for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3"
    ./wave $grid 0.001 20
done
```

4.2.2. MPI версия

Команда компиляции:

```
g++ -O3 mpi.cpp -o mpi -std=c++11
```

module load SpectrumMPI/10.1.0 mpicxx -O3 mpi.cpp -o mpi -std=c++11 Скрипты запуска для различных конфигураций:

Один процесс на узле:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 0:30
#BSUB -o n1.%J.out
#BSUB -e n1.%J.err
#BSUB -R "span[ptile=1]"
#BSUB -m "polus-c3-ib"

GRID_SIZES=(512)
for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 1 processes"
    mpirun -np 1 ./mpi $grid 0.001 20
done
```

Десять процессов на узле:

```
#!/bin/bash
#BSUB -n 10
#BSUB -W 0:30
#BSUB -o n10.%J.out
#BSUB -e n10.%J.err
#BSUB -R "span[ptile=10]"
#BSUB -m "polus-c3-ib"

GRID_SIZES=(512)
for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 10 processes"
    mpirun -np 10 ./mpi $grid 0.001 20
done
```

Двадцать процессов на узле:

```
#!/bin/bash
#BSUB -n 20
#BSUB -W 0:30
#BSUB -o n20.%J.out
#BSUB -e n20.%J.err
#BSUB -R "span[ptile=20]"
#BSUB -m "polus-c3-ib"

GRID_SIZES=(512)
for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 20 processes"
    mpirun -np 20 ./mpi $grid 0.001 20
done
```

4.2.3. MPI + OpenMP версия

Команда компиляции:

```
mpicxx -fopenmp -O3 mpiomp.cpp -o mpiomp -std=c++11
```

Скрипт запуска для одного MPI процесса:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 0:30
#BSUB -o mpiomp.%J.out
#BSUB -e mpiomp.%J.err
#BSUB -R "affinity[core(20)]"
#BSUB -R "span[ptile=1]"
#BSUB -m "polus-c3-ib"

export OMP_NUM_THREADS=20
for grid in 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 20
done

export OMP_NUM_THREADS=40
for grid in 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 40
done

export OMP_NUM_THREADS=80
for grid in 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 80
done

export OMP_NUM_THREADS=160
```

```
for grid in 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 160
done
```

Скрипт запуска для двух MPI процессов:

```
#!/bin/bash
#BSUB -n 2
#BSUB -W 0:30
#BSUB -o mpiomp.%J.out
#BSUB -e mpiomp.%J.err
#BSUB -R "affinity[core(10)]"
#BSUB -R "span[ptile=2]"
#BSUB -m "polus-c3-ib"

export OMP_NUM_THREADS=20
for grid in 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 20
done

export OMP_NUM_THREADS=40
for grid in 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 40
done

export OMP_NUM_THREADS=80
for grid in 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 80
done
```

4.2.4. MPI + GPU версия

Файл Makefile:

```
# Обязательные переменные
ARCH = sm_60
HOST_COMP = mpicc

# Компилятор и флаги
NVCC = nvcc
NVCC_FLAGS = -O3 -std=c++11 -arch=$(ARCH)
MPI_FLAGS = -I/opt/ibm/spectrum_mpi/include
MPI_LIBS = -L/opt/ibm/spectrum_mpi/lib -lmpi_ibm -lmpiprofilesupport
EXTRA_FLAGS = -Xcompiler -fopenmp

# Имя исполняемого файла
TARGET = mpigpu-1
```

```
# Исходный файл
SRC = mpigpu-1.cu

# Цель по умолчанию
all: $(TARGET)

# Правило компиляции
$(TARGET): $(SRC)
    $(NVCC) $(NVCC_FLAGS) $(MPI_FLAGS) $(EXTRA_FLAGS) -o $@ $< $(MPI_LIBS)

# Правило очистки
clean:
    rm -f $(TARGET)
```

Скрипт запуска для одного MPI процесса и одного GPU:

```
#!/bin/bash
#BSUB -n 1
#BSUB -q normal
#BSUB -W 0:30
#BSUB -gpu "num=1:mode=exclusive_process"
#BSUB -o job_%J.out
#BSUB -e job_%J.err

mpirun -np 1 ./mpigpu-1 512 0.001 20 1
```

Скрипт запуска для двух MPI процессов и двух GPU:

```
#!/bin/bash
#BSUB -n 2
#BSUB -q normal
#BSUB -W 0:30
#BSUB -gpu "num=2:mode=exclusive_process"
#BSUB -o job_%J.out
#BSUB -e job_%J.err

mpirun -np 2 ./mpigpu-1 512 0.001 20 1
```

4.3. Проверка корректности MPI+GPU реализации

Для проверки корректности параллельной реализации MPI+GPU было проведено сравнение результатов с последовательной версией программы. Тестирование проводилось для двух конфигураций: с одним MPI процессом и одним GPU, а также с двумя MPI процессами и двумя GPU.

4.3.1. Результаты проверки для одного MPI процесса и одного GPU:

```
Step 2, t = 0.002000, Max Error = 3.967859e-11, L2 Error = 1.406978e-11
Step 4, t = 0.004000, Max Error = 1.587058e-10, L2 Error = 5.627573e-11
Step 6, t = 0.006000, Max Error = 3.570526e-10, L2 Error = 1.266079e-10
Step 8, t = 0.008000, Max Error = 6.346724e-10, L2 Error = 2.250497e-10
Step 10, t = 0.010000, Max Error = 9.915001e-10, L2 Error = 3.515776e-10
```

```
Step 12, t = 0.012000, Max Error = 1.427449e-09, L2 Error = 5.061618e-10
Step 14, t = 0.014000, Max Error = 1.942419e-09, L2 Error = 6.887656e-10
Step 16, t = 0.016000, Max Error = 2.536286e-09, L2 Error = 8.993458e-10
Step 18, t = 0.018000, Max Error = 3.208909e-09, L2 Error = 1.137852e-09
Step 20, t = 0.020000, Max Error = 3.960129e-09, L2 Error = 1.404229e-09
```

4.3.2. Результаты проверки для двух MPI процессов и двух GPU:

```
Step 2, t = 0.002000, Max Error = 3.967859e-11, L2 Error = 1.406978e-11
Step 4, t = 0.004000, Max Error = 1.587058e-10, L2 Error = 5.627573e-11
Step 6, t = 0.006000, Max Error = 3.570526e-10, L2 Error = 1.266079e-10
Step 8, t = 0.008000, Max Error = 6.346724e-10, L2 Error = 2.250497e-10
Step 10, t = 0.010000, Max Error = 9.915001e-10, L2 Error = 3.515776e-10
Step 12, t = 0.012000, Max Error = 1.427449e-09, L2 Error = 5.061618e-10
Step 14, t = 0.014000, Max Error = 1.942419e-09, L2 Error = 6.887656e-10
Step 16, t = 0.016000, Max Error = 2.536286e-09, L2 Error = 8.993458e-10
Step 18, t = 0.018000, Max Error = 3.208909e-09, L2 Error = 1.137852e-09
Step 20, t = 0.020000, Max Error = 3.960129e-09, L2 Error = 1.404229e-09
```

Как видно из результатов, погрешность вычислений MPI+GPU версии находится в пределах допустимых значений (порядка 10^{-9}) и полностью соответствует погрешности последовательной версии, а также версиям OpenMP и MPI+OpenMP. Это подтверждает корректность работы параллельной реализации как для одного, так и для двух GPU.

4.4. Результаты расчетов на IBM Polus для программы MPI+CUDA версии

Область: $[0, 1] \times [0, 1] \times [0, 1]$

Число MPI процессов N_p	Число GPU	Общее время (s)	Время вычислений (s)	Время копирования H2D/D2H (s)	Время обменов MPI (s)	Ускорение (общее) S_{total}	Ускорение (вычисления) S_{comp}
1	1	0.752	0.655	0.087	0.007	29.23	32.65
2	2	0.505	0.344	0.135	0.054	43.52	63.38

Версия программы	Конфигурация	Общее время (s)	Время инициализации (s)	Время вычислений (s)	Время обработки границ (s)	Время обменов (s)
Последовательная	1 процесс	21.98	0.32	21.55	0.11	-
MPI	1 процесс	39.21	0.34	31.71	6.95	0.21
MPI	10 процессов	6.40	0.08	5.17	1.09	2.22
MPI	20 процессов	2.73	0.08	1.88	0.57	0.62
MPI+OpenMP	1×20	3.00	0.25	2.60	0.01	0.14
MPI+OpenMP	1×40	2.41	0.25	2.01	0.02	0.14
MPI+OpenMP	1×80	2.39	0.25	1.96	0.01	0.17
MPI+OpenMP	1×160	2.54	0.25	2.04	0.01	0.24
MPI+OpenMP	2×20	1.81	0.13	1.60	0.01	0.09
MPI+OpenMP	2×40	1.56	0.13	1.33	0.01	0.10
MPI+OpenMP	2×80	1.83	0.13	1.56	0.01	0.13

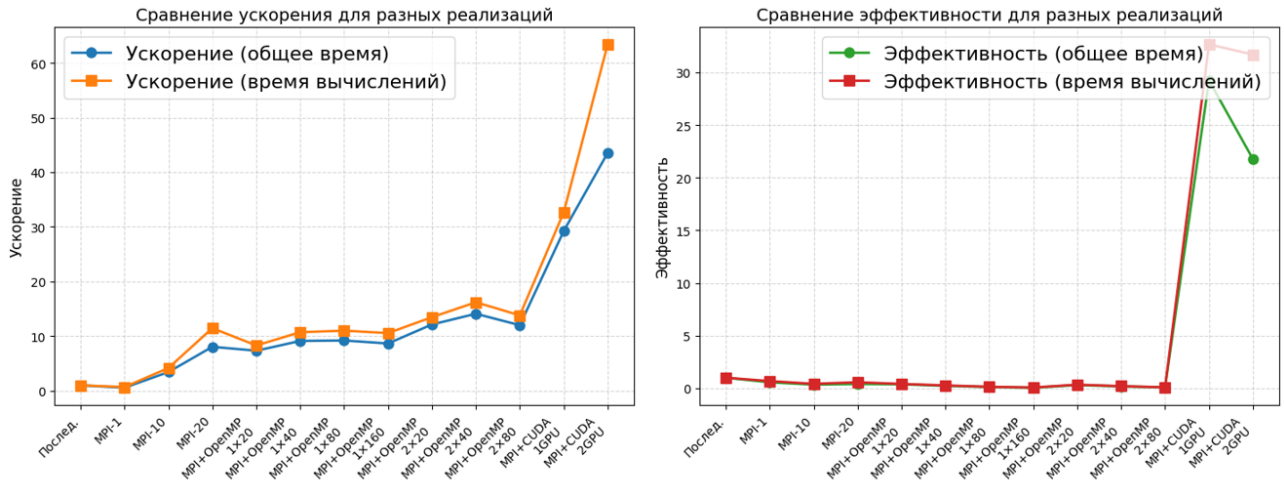
4.5. Сравнение производительности на основе общего времени выполнения

Версия программы	Конфигурация	Время выполнения (s)	Ускорение S	Эффективность E
Последовательная	1 процесс	21.98	1.00	1.00
MPI	1 процесс	39.21	0.56	0.56
MPI	10 процессов	6.40	3.43	0.34
MPI	20 процессов	2.73	8.05	0.40
MPI+OpenMP	1 MPI процесс × 20 OpenMP потоков	3.00	7.33	0.37
MPI+OpenMP	1 MPI процесс × 40 OpenMP потоков	2.41	9.12	0.23
MPI+OpenMP	1 MPI процесс × 80 OpenMP потоков	2.39	9.20	0.12
MPI+OpenMP	1 MPI процесс × 160 OpenMP потоков	2.54	8.65	0.05
MPI+OpenMP	2 MPI процесса × 20 OpenMP потоков	1.81	12.14	0.30
MPI+OpenMP	2 MPI процесса × 40 OpenMP потоков	1.56	14.09	0.18
MPI+OpenMP	2 MPI процесса × 80 OpenMP потоков	1.83	12.01	0.08
MPI+CUDA	1 MPI процесс + 1 GPU	0.75	29.23	29.23
MPI+CUDA	2 MPI процесса + 2 GPU	0.50	43.52	21.76

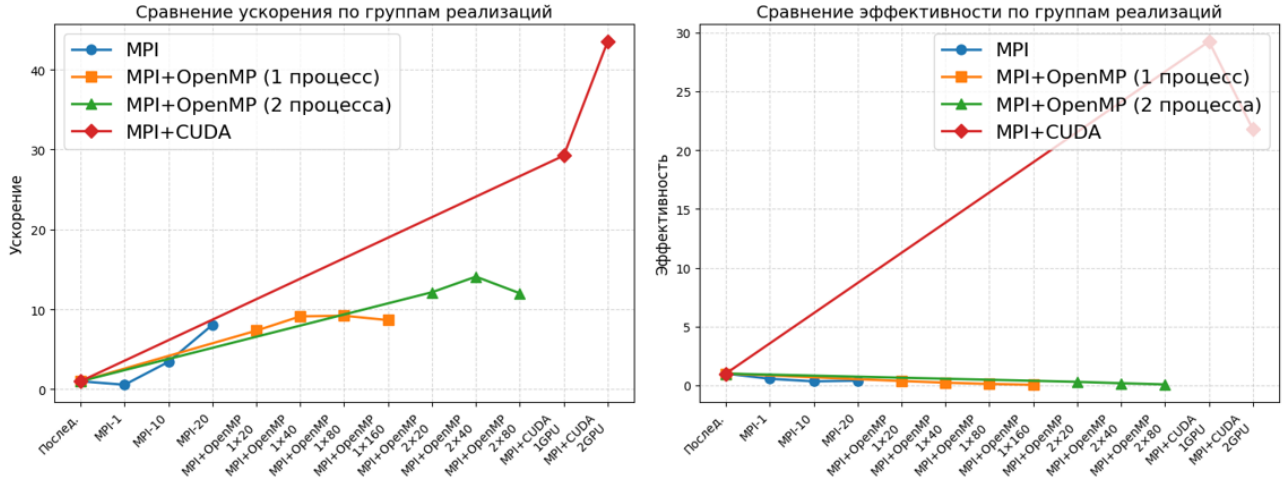
4.6. Сравнение производительности на основе времени вычислений

Версия программы	Конфигурация	Время вычислений (s)	Ускорение S	Эффективность E
Последовательная	1 процесс	21.55	1.00	1.00
MPI	1 процесс	31.71	0.68	0.68
MPI	10 процессов	5.17	4.17	0.42
MPI	20 процессов	1.88	11.46	0.57
MPI+OpenMP	1 MPI процесс × 20 OpenMP потоков	2.60	8.29	0.41
MPI+OpenMP	1 MPI процесс × 40 OpenMP потоков	2.01	10.72	0.27
MPI+OpenMP	1 MPI процесс × 80 OpenMP потоков	1.96	11.00	0.14
MPI+OpenMP	1 MPI процесс × 160 OpenMP потоков	2.04	10.56	0.07
MPI+OpenMP	2 MPI процесса × 20 OpenMP потоков	1.60	13.47	0.34
MPI+OpenMP	2 MPI процесса × 40 OpenMP потоков	1.33	16.20	0.20
MPI+OpenMP	2 MPI процесса × 80 OpenMP потоков	1.56	13.81	0.09
MPI+CUDA	1 MPI процесс + 1 GPU	0.66	32.65	32.65
MPI+CUDA	2 MPI процесса + 2 GPU	0.34	63.38	31.69

4.7. Анализ масштабируемости и эффективности параллельных реализаций



(a) Сравнение ускорения и эффективности для разных реализаций



(b) Сравнение ускорения и эффективности по группам реализаций

Рисунок 4.1. Анализ производительности параллельных реализаций

Анализ производительности различных параллельных реализаций показал существенные различия в эффективности и масштабируемости рассмотренных подходов.

MPI реализация: При использовании чистого MPI наблюдается значительное увеличение производительности с ростом числа процессов. На 20 процессах достигается ускорение в 11.46 раз при вычислительной эффективности 0.57, что демонстрирует хорошую масштабируемость для данного типа задачи. Однако, следует отметить снижение эффективности с увеличением числа процессов, что объясняется ростом накладных расходов на межпроцессное взаимодействие.

Гибридная реализация MPI+OpenMP: Использование гибридного подхода демонстрирует такие результаты:

- При одном MPI процессе увеличение числа OpenMP потоков с 20 до 80 приводит к росту ускорения с 8.29 до 11.00, однако дальнейшее увеличение числа потоков до 160 не даёт существенного прироста производительности.
- Конфигурация с двумя MPI процессами показывает лучшие результаты, достигая максимального ускорения в 16.20 раз при использовании 40 потоков на процесс.

Однако эффективность при этом снижается до 0.20, что указывает на значительные накладные расходы при большом количестве потоков.

Реализация MPI+CUDA: Наиболее эффективным оказалось использование графических ускорителей:

- На одном GPU достигается ускорение в 32.65 раз с эффективностью 32.65, что значительно превосходит результаты CPU-ориентированных реализаций.
- Использование двух GPU позволяет получить почти линейное масштабирование, увеличивая ускорение до 63.38 раз при сохранении высокой эффективности (31.69).

Сравнительный анализ: При сопоставлении различных подходов можно сделать следующие выводы:

- Для систем без GPU оптимальной является гибридная конфигурация с 2 MPI процессами и 40 OpenMP потоками на процесс.
- CUDA-реализация демонстрирует существенное преимущество над CPU-версиями, обеспечивая как высокое ускорение, так и эффективность использования вычислительных ресурсов.
- Масштабируемость MPI+CUDA решения близка к линейной, что указывает на эффективное распределение вычислительной нагрузки между GPU.

На основе проведённого анализа можно рекомендовать использование GPU-ускорителей как наиболее эффективного способа повышения производительности для данной задачи. При отсутствии доступа к GPU, оптимальным выбором является гибридная MPI+OpenMP реализация с умеренным количеством потоков на каждый MPI процесс.

4.8. Исходный код

Исходный код проекта доступен в репозитории: https://github.com/AICCer1/MPI-CUDA

Репозиторий содержит:

- Последовательную версию в ветке `sequential`
- MPI версию в ветке `mpi`
- MPI+OpenMP версию в ветке `mpi_omp`
- MPI+CUDA версию в ветке `mpi_cuda`
- Скрипты для запуска на кластере Polus

5. Компиляция и выполнение программы (на этапах 1 2 3)

5.1. Последовательная Вресия

5.1.1. Команда компиляции

›_ Команда компиляции

```
g++ -O2 wave.cpp -o wave
```

5.1.2. Команда выполнения

›_ Команда запуска

```
./wave 64 0.001 20
```

5.1.3. Параметры командной строки

- `64` --- размер сетки (N): количество точек в каждом измерении трехмерной сетки
- `0.001` --- шаг по времени (τ): временной интервал между итерациями
- `20` --- количество временных шагов (K): общее число итераций по времени

5.1.4. Скрипт для системы Polus

</> Скрипт OMP1.lsf

```
#!/bin/bash
#BSUB -J "Sequential"
#BSUB -o "Sequential_%J.out"
#BSUB -e "Sequential_%J.err"
#BSUB -n 1
#BSUB -W 0:30
#BSUB -R "span[hosts=1]"
#BSUB -R "affinity[core(1)]"
#BSUB -m "polus-c3-ib"
GRID_SIZES=(128 256 512)
for grid in "${GRID_SIZES[@]}"; do
 echo "Running test with grid size ${grid}^3"
 ./wave $grid 0.001 20
done
```

5.2. OpenMP Вресия

5.2.1. Команда компиляции

>_ Команда компиляции

```
g++ -fopenmp -O2 openmpwave.cpp -o openmpwave
```

5.2.2. Команда выполнения

>_ Команда запуска

```
./wave3dOMP 64 0.001 20 4
```

5.2.3. Параметры командной строки

- `64` --- размер сетки (N): количество точек в каждом измерении трехмерной сетки
- `0.001` --- шаг по времени (τ): временной интервал между итерациями
- `20` --- количество временных шагов (K): общее число итераций по времени
- `4` --- количество потоков OpenMP: число параллельных потоков для вычислений

5.2.4. Скрипт для системы Polus

Скрипт выполняет тестирование программы с различным количеством потоков (1, 2, 4, 8, 16, 32, 64) и различными размерами сетки (128, 256, 512).

</> Скрипт OMP1.lsf

```
#BSUB -J "OpenMP_1_32t"
#BSUB -o "OpenMP_1_32t_%J.out"
#BSUB -e "OpenMP_1_32t_%J.err"
#BSUB -R "affinity[core(16)]"
for threads in 1 2 4 8 16 32 64; do
  OMP_NUM_THREADS=$threads
  for size in 128 256 512; do
   /polusfs/lsf/openmp/launchOpenMP.py ./openmpwave $size 0.001 20 $threads
  done
done
```

5.3. MPI Версия

5.3.1. Команда компиляции

>_ Команда компиляции

```
module load SpectrumMPI/10.1.0
mpicxx -O2 onlyMPI.cpp -o onlyMPI -std=c++11
```

5.3.2. Команда выполнения

>_ Команда запуска

```
mpirun -np 16 ./onlyMPI 512 0.001 20
```

5.3.3. Параметры командной строки

- `512` --- размер сетки (N): количество точек в каждом измерении трехмерной сетки
- `0.001` --- шаг по времени (τ): временной интервал между итерациями
- `20` --- количество временных шагов (K): общее число итераций по времени

5.3.4. Скрипт для системы Polus

При проведении тестирования использовались два узла Polus (polus-c3-ib и polus-c4-ib). Процессы равномерно распределялись между узлами для обеспечения сбалансированной нагрузки.

Для версии с MPI использовались следующие скрипты:

Для 2 процессов (по 1 процессу на узел):

```
#!/bin/bash
#BSUB -n 2
#BSUB -W 0:30
#BSUB -o n2.%J.out
#BSUB -e n2.%J.err
#BSUB -R "span[ptile=1]"
#BSUB -m "polus-c3-ib polus-c4-ib"

GRID_SIZES=(128 256 512)

for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 2 processes"
    mpirun -np 2 ./onlyMPI $grid 0.001 20
done
```

Для 4 процессов (по 2 процесса на узел):

```
#!/bin/bash
#BSUB -n 4
#BSUB -W 0:30
#BSUB -o n4.%J.out
#BSUB -e n4.%J.err
#BSUB -R "span[ptile=2]"
#BSUB -m "polus-c3-ib polus-c4-ib"

GRID_SIZES=(128 256 512)

for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 4 processes"
    mpirun -np 4 ./onlyMPI $grid 0.001 20
done
```

Для 8 процессов (по 4 процесса на узел):

```
#!/bin/bash
#BSUB -n 8
#BSUB -W 0:30
#BSUB -o n8.%J.out
#BSUB -e n8.%J.err
#BSUB -R "span[ptile=4]"
#BSUB -m "polus-c3-ib polus-c4-ib"

GRID_SIZES=(128 256 512)

for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 8 processes"
    mpirun -np 8 ./onlyMPI $grid 0.001 20
done
```

Для 10 процессов (по 5 процесса на узел):

```
#!/bin/bash
#BSUB -n 10
#BSUB -W 0:30
#BSUB -o n10.%J.out
#BSUB -e n10.%J.err
#BSUB -R "span[ptile=5]"
#BSUB -m "polus-c3-ib polus-c4-ib"

GRID_SIZES=(128 256 512)

for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 10 processes"
    mpirun -np 10 ./onlyMPI $grid 0.001 20
done
```

Для 16 процессов (по 8 процессов на узел):

```
#!/bin/bash
#BSUB -n 16
#BSUB -W 0:30
#BSUB -o n16.%J.out
#BSUB -e n16.%J.err
#BSUB -R "span[ptile=8]"
#BSUB -m "polus-c3-ib polus-c4-ib"

GRID_SIZES=(128 256 512)

for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 16 processes"
    mpirun -np 16 ./onlyMPI $grid 0.001 20
done
```

Для 20 процессов (по 10 процесса на узел):

```
#!/bin/bash
#BSUB -n 20
#BSUB -W 0:30
#BSUB -o n20.%J.out
#BSUB -e n20.%J.err
#BSUB -R "span[ptile=10]"
#BSUB -m "polus-c3-ib polus-c4-ib"

GRID_SIZES=(128 256 512)

for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 20 processes"
    mpirun -np 20 ./onlyMPI $grid 0.001 20
done
```

Для 32 процессов (по 16 процессов на узел):

```
#!/bin/bash
#BSUB -n 32
#BSUB -W 0:30
#BSUB -o n32.%J.out
#BSUB -e n32.%J.err
#BSUB -R "span[ptile=16]"
#BSUB -m "polus-c3-ib polus-c4-ib"

GRID_SIZES=(128 256 512)

for grid in "${GRID_SIZES[@]}"; do
    echo "Running test with grid size ${grid}^3 and 32 processes"
    mpirun -np 32 ./onlyMPI $grid 0.001 20
done
```

5.4. MPI + OpenMP Версия

5.4.1. Команда компиляции

›_ Команда компиляции

```
module load SpectrumMPI/10.1.0
mpicxx -fopenmp -O2 mpiomp.cpp -o mpiomp -std=c++11
```

5.4.2. Команда выполнения

›_ Команда запуска

```
mpirun -np 4 ./mpiomp 256 0.001 20 4
```

5.4.3. Скрипт для системы Polus

Для гибридной версии MPI+OpenMP использовались следующие скрипты: Для 1 MPI процесса с 4 потоками OpenMP:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 0:30
#BSUB -o mpiomp.%J.out
#BSUB -e mpiomp.%J.err
#BSUB -R "affinity[core(4)]"
#BSUB -R "span[ptile=1]"
#BSUB -m "polus-c3-ib"

export OMP_NUM_THREADS=4

for grid in 128 256 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 4
done
```

Для 2 MPI процессов с 4 потоками OpenMP каждый (по 1 процессу на узел):

```
#!/bin/bash
#BSUB -n 2
#BSUB -W 0:30
#BSUB -o mpiomp.%J.out
#BSUB -e mpiomp.%J.err
#BSUB -R "affinity[core(4)]"
#BSUB -R "span[ptile=1]"
#BSUB -m "polus-c3-ib polus-c4-ib"

export OMP_NUM_THREADS=4

for grid in 128 256 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 4
done
```

Для 4 MPI процессов с 4 потоками OpenMP каждый (по 2 процесса на узел):

```
#!/bin/bash
#BSUB -n 4
#BSUB -W 0:30
#BSUB -o mpiomp.%J.out
#BSUB -e mpiomp.%J.err
#BSUB -R "affinity[core(4)]"
#BSUB -R "span[ptile=2]"
#BSUB -m "polus-c3-ib polus-c4-ib"

export OMP_NUM_THREADS=4

for grid in 128 256 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 4
done
```

Для 8 MPI процессов с 4 потоками OpenMP каждый (по 4 процесса на узел):

```
#!/bin/bash
#BSUB -n 8
#BSUB -W 0:30
#BSUB -o mpiomp.%J.out
#BSUB -e mpiomp.%J.err
#BSUB -R "affinity[core(4)]"
#BSUB -R "span[ptile=4]"
#BSUB -m "polus-c3-ib polus-c4-ib"

export OMP_NUM_THREADS=4

for grid in 128 256 512; do
    echo "Running test with grid size ${grid}^3"
    mpiexec ./mpiomp $grid 0.001 20 4
done
```

Параметр `span[ptile=N]` в скриптах обеспечивает равномерное распределение процессов между узлами, где N - количество процессов на один узел. Для гибридной версии дополнительно используется параметр `affinity[core(4)]`, который выделяет 4 ядра для каждого MPI процесса под потоки OpenMP.

Список литературы

[1] Задание №1 3-й поток по курсу «Суперкомпьютерное моделирование и технологии», октябрь 2024 - декабрь 2024.

[2] Суперкомпьютер IBM Polus. [Электронный ресурс]. URL: http://hpc.cs.msu.su/polus

[3] Самарский А.А., Гулин А.В. Численные методы. --- М.: Наука. Гл. ред. физ-мат. лит., 1989.