

Московский Государственный Университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Отчёт по практическому заданию 3
в рамках курса «Суперкомпьютерное моделирование и
технологии»

Вариант 8

Выполнил:
Шорошов Григорий Максимович,
622 группа

Москва, 2021

1. Математическая постановка задачи

В трехмерной замкнутой области

$$\Omega = [0 \leq x \leq L_x] \times [0 \leq y \leq L_y] \times [0 \leq z \leq L_z]$$

для $(0 < t \leq T)$ требуется найти решение $u(x, y, z, t)$ уравнения в частных производных

$$\frac{\partial^2 u}{\partial t^2} = \Delta u \quad (1)$$

с начальными условиями

$$u|_{t=0} = \phi(x, y, z) \quad (2)$$

$$\frac{\partial u}{\partial t}|_{t=0} = 0 \quad (3)$$

при условии, что на границах области заданы однородные граничные условия первого рода

$$u(0, y, z, t) = 0, \quad u(L_x, y, z, t) = 0 \quad (4)$$

$$u(x, 0, z, t) = 0, \quad u(x, L_y, z, t) = 0 \quad (5)$$

$$u(x, y, 0, t) = 0, \quad u(x, y, L_z, t) = 0 \quad (6)$$

либо периодические условия

$$u(0, y, z, t) = u(L_x, y, z, t) \quad u_x(0, y, z, t) = u_x(L_x, y, z, t) \quad (7)$$

$$u(x, 0, z, t) = u(x, L_y, z, t) \quad u_y(x, 0, z, t) = u_y(x, L_y, z, t) \quad (8)$$

$$u(x, y, 0, t) = u(x, y, L_z, t) \quad u_z(x, y, 0, t) = u_z(x, y, L_z, t) \quad (9)$$

Был реализован вариант 8 для функции

$$u(x, y, z, t) = \sin\left(\frac{2\pi}{L_x}x\right) \cdot \sin\left(\frac{4\pi}{L_y}y\right) \cdot \sin\left(\frac{6\pi}{L_z}z\right) \cdot \cos(a_t \cdot t), \quad a_t = \pi \sqrt{\frac{4}{L_x^2} + \frac{16}{L_y^2} + \frac{36}{L_z^2}}$$

и периодических граничных условий.

2. Численный метод решения задачи

Для численного решения задачи введем на Ω сетку $\omega_{h\tau} = \overline{\omega_h} \times \omega_\tau$, где

$$T = T_0,$$

$$L_x = L_{x_0}, \quad L_y = L_{y_0}, \quad L_z = L_{z_0},$$

$$\overline{\omega_h} = \{(x_i = ih_x, y_j = jh_y, z_k = kh_z), i, j, k = 0, 1, \dots, N, h_xN = L_x, h_yN = L_y, h_zN = L_z\},$$

$$\omega_\tau = \{t_n = n\tau, n = 0, 1, \dots, K, \tau K = T\}.$$

Через ω_h обозначим множество внутренних, а через γ_h — множество граничных узлов сетки $\overline{\omega_h}$.

Для аппроксимации исходного уравнения (1) воспользуемся следующей системой уравнений:

$$\frac{u_{ijk}^{n+1} - 2u_{ijk}^n + u_{ijk}^{n-1}}{\tau^2} = \Delta_h u^n, (x_i, y_j, z_k) \in \omega_h, n = 1, 2, \dots, K-1, \quad (10)$$

где Δ_h — разностный аналог оператора Лапласа:

$$\Delta_h u^n = \frac{u_{i-1,j,k}^n - 2u_{i,j,k}^n + u_{i+1,j,k}^n}{h^2} + \frac{u_{i,j-1,k}^n - 2u_{i,j,k}^n + u_{i,j+1,k}^n}{h^2} + \frac{u_{i,j,k-1}^n - 2u_{i,j,k}^n + u_{i,j,k+1}^n}{h^2}.$$

Значения $u_{i,j,k}^{n+1}$ можно явным образом выражаются через значения на n -м и $(n-1)$ -м шаге.

Зададим значения $u_{i,j,k}^0, u_{i,j,k}^1, (x_i, y_j, z_k) \in \omega_h$, исходя из условий (2) и (3):

$$u_{i,j,k}^0 = \phi(x_i, y_j, z_k), (x_i, y_j, z_k) \in \omega_h \quad (11)$$

$$u_{i,j,k}^1 = u_{i,j,k}^0 + \frac{\tau^2}{2} \Delta_h \phi(x_i, y_j, z_k) \quad (12)$$

Разностная аппроксимация для периодических граничных условий (7)–(9) выглядит следующим образом:

$$\begin{aligned} u_{0,j,k}^{n+1} &= u_{N,j,k}^{n+1} = \frac{u_{1,j,k}^{n+1} + u_{N-1,j,k}^{n+1}}{2} \\ u_{i,0,k}^{n+1} &= u_{i,N,k}^{n+1} = \frac{u_{i,1,k}^{n+1} + u_{i,N-1,k}^{n+1}}{2} \\ u_{i,j,0}^{n+1} &= u_{i,j,N}^{n+1} = \frac{u_{i,j,1}^{n+1} + u_{i,j,N-1}^{n+1}}{2} \end{aligned}$$

3. Описание программной реализации

Код решения можно найти по ссылке.

Конфигурация запуска хранится в формате INI, для чтения конфигурации использовалась библиотека ini.h. Пример файла конфигурации:

```
[Solver]
L_x=1.0
L_y=1.0
L_z=1.0
N_x=128
N_y=128
N_z=128
periodic_x=true
```

```

periodic_y=true
periodic_z=true
K=20
T=0.025
print_layer_err=true
save_layers=false
save_step=5
layers_path=./layers/

```

Данные из файла считывает объект класса Config, который создается в конструкторе класса Solver. Далее в конструкторе Solver:

- выполняем команду

```
MPI_Init(&argc, &argv);
```

- запрашиваем количество запущенных MPI процессов и свой номер:

```

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

```

- определяем сбалансированное распределение процессов по решетке, создаем виртуальную топологию и определяем координаты процессора в решетке:

```

MPI_Dims_create(size, 3, dims);
MPI_Cart_create(MPI_COMM_WORLD, 3, dims,
                config.periodic, true, &comm);
MPI_Cart_coords(comm, rank, 3, coords);

```

- вычисляем размеры и координаты блока решетки $\overline{\omega_h}$, за который отвечает процессор:

```
compute_block_coords();
```

- выделяем память (вызывая метод resize) под блоки на шагах $n + 1$, n , $n - 1$ алгоритма, блоки для хранения погрешности и аналитического решения, а также под значения, пересылаемые из соседних процессов:

```

Block3D<double> blocks[3];
Block3D<double> errs;
Block3D<double> analytical;
Block3DBound<double> bound;

```

- создаем типы для передачи граней блока соседним процессам.

Для хранения блоков используются класс Block3D, который содержит размер блока, его координаты и узлы решетки, которые хранятся в объекте класса Grid3D. Элементы решетки хранятся в объекте класса std::vector<double> размера P_x * P_y * P_z. Доступ к элементам решетки происходит через вызов оператора operator()(int i, int j, int k).

Каждому процессу соответствует один блок. Расчет внутренних элементов блока не зависит от соседних блоков и векторизуется через pragma omp parallel for. Однако для расчета граничных элементов блока необходимо:

- получить граничные элементы соседних блоков u^n ;
- если граница блока совпадает с границей области Ω , получить $u_{1,j,k}^{n+1}, u_{N-1,j,k}^{n+1}, u_{i,1,k}^{n+1}, u_{1,N-1,k}^{n+1}, u_{i,j,1}^{n+1}, u_{i,j,N-1}^{n+1}$.

Пересылка граничных элементов блоков реализована в методах send_inner_values (для внутренних узлов сетки) и send_boundary_values (для граничных узлов сетки). Алгоритм пересылки следующий:

- по каждой оси выполняется асинхронная отправка граней своего блока соседним процессам;
- для каждой оси выполняется блокирующее получение граней блоков соседних процессов;
- процесс дожидается завершения отправок.

Вычисление разностной схемы происходит в методе run класса Solver. Алгоритм работы следующий:

- рассчитываются аналитические значения на 0 слое;
- рассчитывается 1 слой по формуле (12);
- для всех $t_n, 2 \leq n \leq K$:
 - рассчитывается 2 слой по формуле (10);
 - вычисляется максимальная погрешность на слое;
 - если в конфигурации задано сохранение слоя, то слой сохраняется на диск.

В деструкторе класса Solver:

- вычисляем наибольшее время выполнение среди всех процессов и выводим его;
- вызываем MPI_Finalize.

4. Графики аналитического, посчитанного решений и погрешности

Рис. 1: Аналитическое и посчитанное решения для $N = 256$, $T = 0.025$, $L_x = L_y = L_z = 1.0$

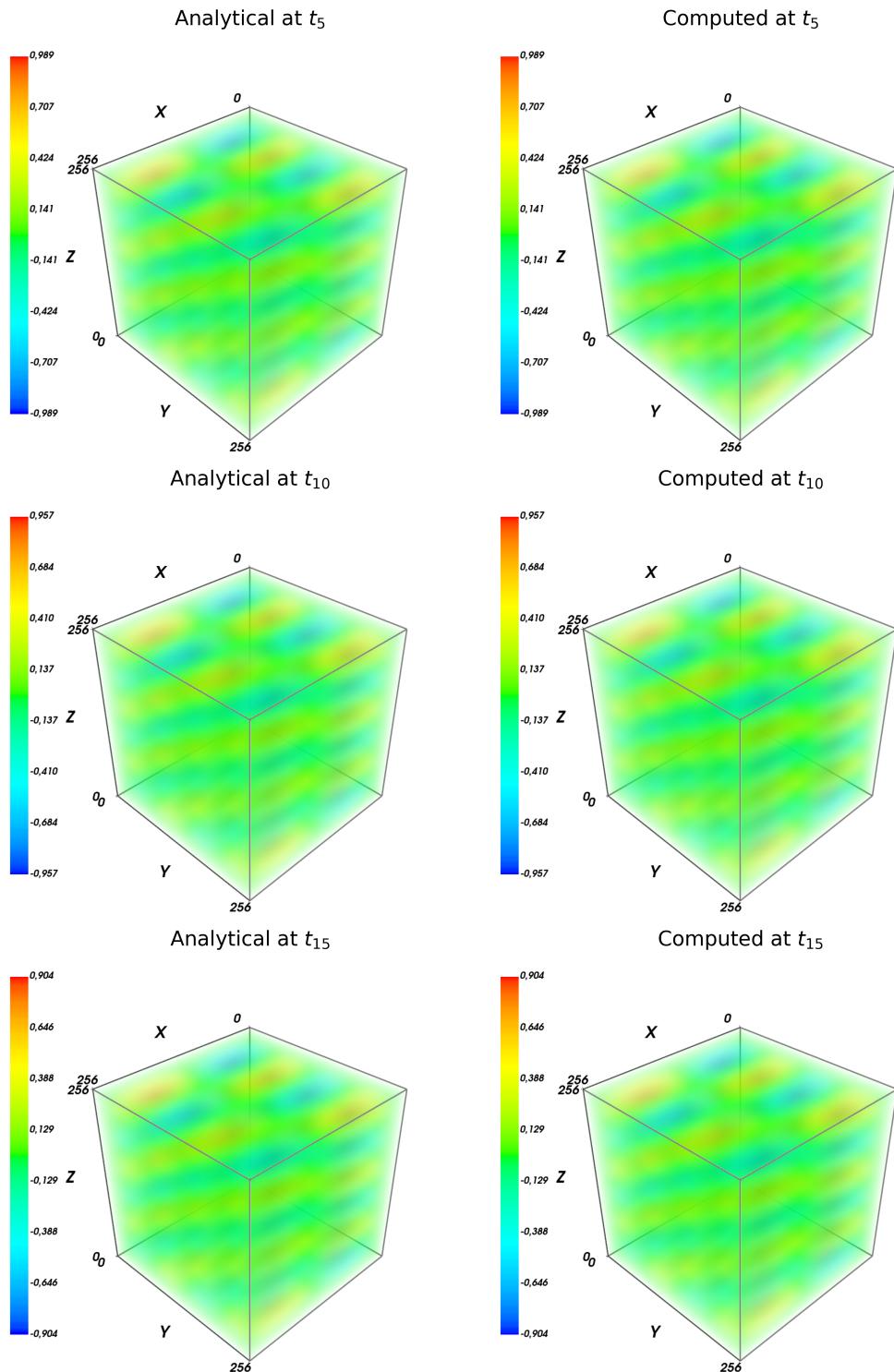
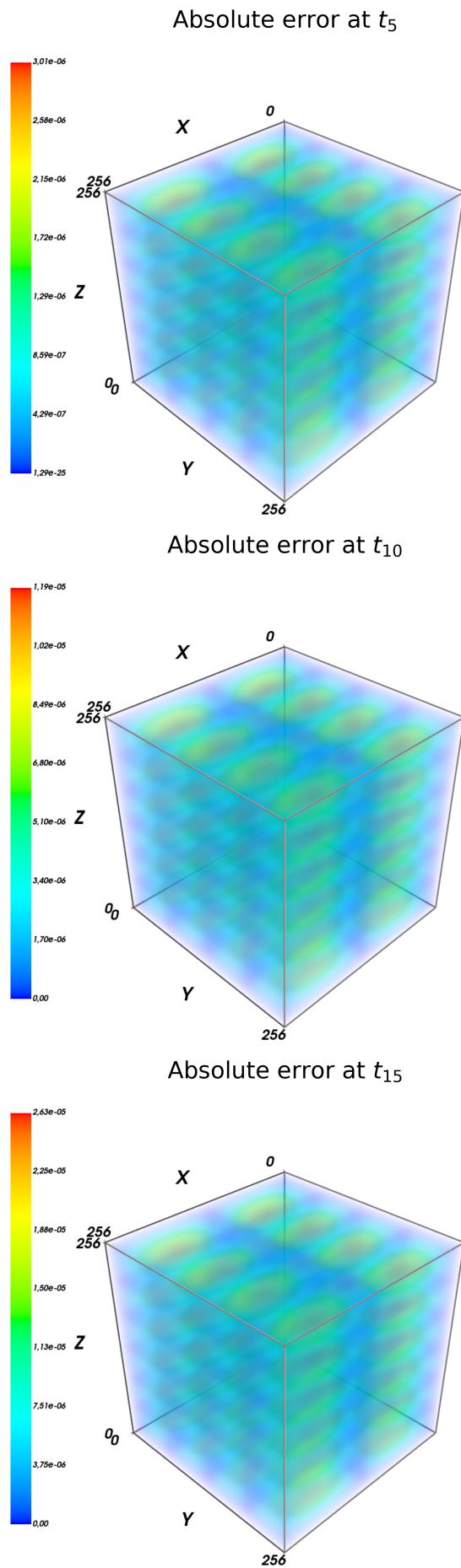


Рис. 2: Погрешность решения для $N = 256$, $T = 0.025$, $L_x = L_y = L_z = 1.0$



5. Результаты расчетов

Запуски программы проведены на системах Blue Gene/P и Polus для:

- $L_x = L_y = L_z = 1.0, K = 20, T = 0.025$
- $L_x = L_y = L_z = \pi, K = 20, T = 0.025$

Запуски на Blue Gene/P проводились в режиме SMP. Гибридная версия программы MPI/OpenMP запускалась на BlueGene/P с OMP_NUM_THREADS=4.

Таблица 1: Таблица с результатами расчётов для системы Polus, $L_x = L_y = L_z = 1.0, K = 20, T = 0.025$

Число точек сетки N^3	Число процессоров N_p	Время решения T_s	Ускорение S	Погрешность δ
128^3	10	0.500	1.000	0.000217197
	20	0.481	1.041	
	40	0.305	1.640	
256^3	10	3.214	1.000	4.55295e-05
	20	2.211	1.454	
	40	1.520	1.920	
512^3	10	25.770	1.000	2.58835e-06
	20	16.955	1.520	
	40	10.276	2.741	

Таблица 2: Таблица с результатами расчётов для системы Polus, $L_x = L_y = L_z = \pi, K = 20, T = 0.025$

Число точек сетки N^3	Число процессоров N_p	Время решения T_s	Ускорение S	Погрешность δ
128^3	10	0.503	1.000	2.43122e-05
	20	0.458	1.046	
	40	0.380	1.649	
256^3	10	3.216	1.000	5.98586e-06
	20	2.540	1.454	
	40	1.674	2.115	
512^3	10	25.325	1.000	1.4015e-06
	20	16.852	1.520	
	40	9.401	2.508	

Таблица 3: Таблица с результатами расчётов для системы Blue Gene/P, $L_x = L_y = L_z = 1.0$, $K = 20$, $T = 0.025$

N^3	N_p	Время решения T_{MPI}	Ускорение S_{MPI}	Время решения T_{MPI_OpenMP}	Ускорение S_{MPI_OpenMP}	$\frac{T_{MPI}}{T_{MPI_OpenMP}}$	Погрешность δ
128	64	1.243	1.000	0.403	3.084	3.084	0.000217197
	128	0.663	1.874	0.240	5.174	2.761	
	512	0.189	6.564	0.078	16.014	2.440	
256	64	9.248	1.000	2.812	3.288	3.288	4.55295e-05
	128	4.777	1.936	1.514	6.110	3.156	
	512	1.252	7.389	0.407	22.735	3.077	
512	64	71.109	1.000	21.007	3.385	3.385	2.58822e-06
	128	36.101	1.970	10.842	6.559	3.330	
	512	9.225	7.709	2.789	25.495	3.307	

Таблица 4: Таблица с результатами расчётов для системы Blue Gene/P, $L_x = L_y = L_z = \pi$, $K = 20$, $T = 0.025$

N^3	N_p	Время решения T_{MPI}	Ускорение S_{MPI}	Время решения T_{MPI_OpenMP}	Ускорение S_{MPI_OpenMP}	$\frac{T_{MPI}}{T_{MPI_OpenMP}}$	Погрешность δ
128	64	1.208	1.000	0.394	3.070	3.070	2.43122e-05
	128	0.644	1.877	0.235	5.136	2.735	
	512	0.185	6.522	0.077	15.742	2.414	
256	64	8.938	1.000	2.739	3.263	3.263	5.98586e-06
	128	4.619	1.935	1.477	6.050	3.126	
	512	1.217	7.346	0.396	22.556	3.070	
512	64	68.650	1.000	20.394	3.366	3.366	1.4015e-06
	128	34.872	1.969	10.544	6.511	3.307	
	512	8.927	7.690	2.711	25.319	3.292	