Суперкомпьютерное моделирование и технологии

Отчет

Задача для трехмерного гиперболического уравнения в прямоугольном параллелепипеде. CUDA реализация.

Лазарев Владимир Александрович

2 вариант

25.12.2021

Оглавление

Математическая постановка задачи

В трехмерной замкнутой области

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

для $0 \le t \le T$ требуется найти решение $u(x, y, z, t)$ уравнения в частных производных $\frac{\partial^2 u}{\partial t^2} = \Delta u$ с начальными условиями

$$u\,(t = 0) = \phi(x, y, z)$$
$$\frac{\partial u}{\partial t}\,(t = 0) = 0$$
$$u(0, y, z, t) = 0$$
$$u(L_x, y, z, t) = 0$$
$$u(x, 0, z, t) = 0$$
$$u(x, L_y, z, t) = 0$$
$$u(x, y, 0, t) = u(x, y, L_z, t)$$
$$u_z(x, y, 0, t) = u_z(x, y, L_z, t)$$

Численный метод решения задачи

Введем на Ω сетку $\omega_{h\tau} = \overline{\omega_h} \times \omega_\tau$, где $T = T_0$,

$$L_x = L_{x0}, L_y = L_{y0}, L_z = L_{z0},$$

$$\overline{\omega_h} = \{(x_i = ih_x, y_j = jh_y, z_k = kh_z), i, j, k = \overline{0, N}, h_x N = L_x, h_y N = L_y, h_z N = L_z\},$$

$$\omega_\tau = \{t_n = n\tau, n = \overline{0, K}, \tau K = T\}$$

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

Для аппроксимации исходного уравнения воспользуемся следующей системой уравнений:

$$\frac{u_{i,j,k}^{n+1} - 2u_{i,j,k}^{n} + u_{i,j,k}^{n-1}}{\tau^2} = \Delta_h u^n, (x_i, y_i, z_i) \in \omega_h, n = \overline{1, K-1}$$

Здесь Δ_h – семиточечный разностный аналог оператора Лапласа:

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

Приведенная выше разностная схема является явной – значения $u_{i,j,k}^{n+1}$ на $(n+1)$-ом шаге можно явным образом выразить через значения на предыдущих слоях.

Для начала счета должны быть заданы значения $u_{i,j,k}^{0}, u_{i,j,k}^{1}, (x_i, y_i, z_i) \in \omega_h$:

$$u_{i,j,k}^{0} = \phi(x_i, y_i, z_i), (x_i, y_i, z_i) \in \omega_h$$

$$u_{i,j,k}^{1} = u_{i,j,k}^{0} + \frac{\tau^2}{2} \Delta_h \phi(x_i, y_i, z_i)$$

$$u_{i,j,0}^{n+1} = u_{i,j,N}^{n+1}$$

$$u_{i,j,1}^{n+1} = u_{i,j,N+1}^{n+1}$$

$$i, j, k = \overline{0, N}$$

Программная реализация

Реализована гибридная параллельная программа (MPI + OpenMP). Принимает входные данные в виде аргументов командной строки. Используются следующие аргументы:

- -*Lx*= – длина параллелепипеда вдоль оси X (по умолчанию 1);
- -*Ly*= – длина параллелепипеда вдоль оси Y (по умолчанию 1);
- -*Lz*= – длина параллелепипеда вдоль оси Z (по умолчанию 1);
- -*T*= – конечное время сетки (по умолчанию 1);
- -*N*= – количество точек пространственной сетки (по умолчанию 128);
- -*K*= – количество точек временной сетки (по умолчанию 2000);
- -*steps*= – количество шагов для решения (по умолчанию 5).

Для распараллеливания вся сетка разбивается на области в количестве используемых процессов по следующему алгоритму:

- Начинается разбиение с параллелепипеда $[0, N] \times [0, N] \times [0, N]$, выбирается X как начальная ось;
- Если оставшееся количество областей для разбиения равно 1, возвращается обрабатываемый параллелепипед;

- Если размер нечетный, то по текущей оси выбирается область $\frac{1}{countOfProcesses}$ и делается из нее параллелепипед, также продолжается разбиваться область $1 - \frac{1}{countOfProcesses}$;
- По выбранной оси делим область пополам и рекурсивно запускаем для этих подобластей переходя на следующую ось $X \to Y, Y \to Z, Z \to X$).

Операции, производимые с получаемыми параллелепипедами, обрабатываются с помощью CUDA ядер.

Компиляция производилась посредством на Polus посредством следующего набора команд:

- `module load SpectrumMPI`
- `nvcc main.cu -O3 -std=c++11 -I/opt/ibm/spectrum_mpi/include -L/opt/ibm/spectrum_mpi/lib -lmpiprofilesupport -lmpi_ibm`

Постановка в очередь производилась командой:

```
bsub -n 6 -gpu "num=2" -R "span[ptile=2]" -o stdout.txt -e
error.txt OMP_NUM_THREADS=1 mpiexec ./a.out -N=128 -K=2000 -
steps=20
```

Код доступен по ссылке https://github.com/vlazarew/parallel_2021-22/tree/main/task4

График аналитического и полученного решений

Решение при $L_x = L_y = L_z = 1$

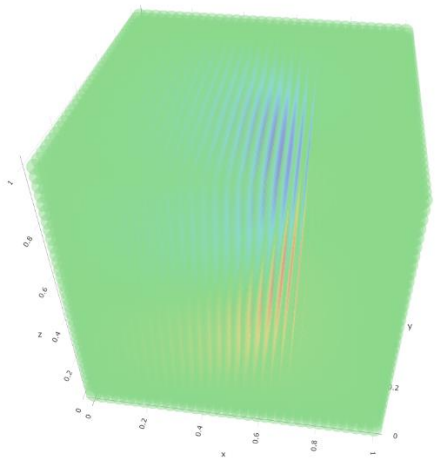

Рисунок 1. Аналитическое решение

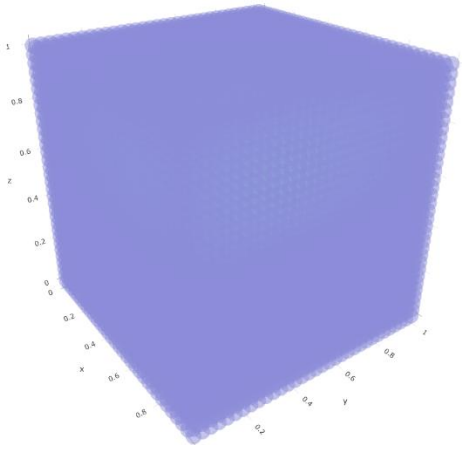

Рисунок 2 Погрешность

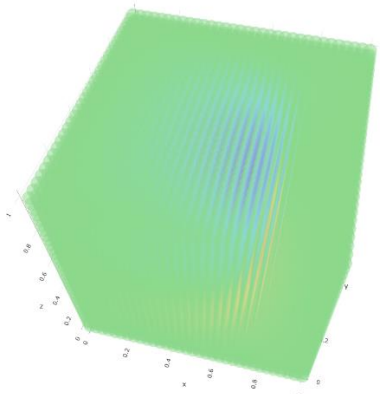

Рисунок 3 Полученное решение

Решение при $L_x = L_y = L_z = \pi$

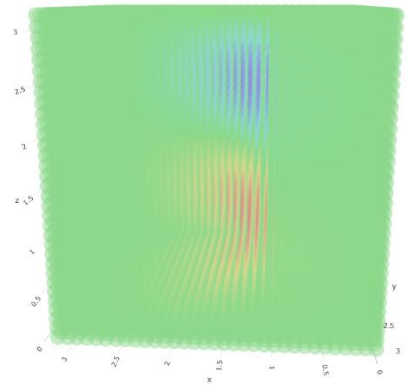

Рисунок 4 Аналитическое решение

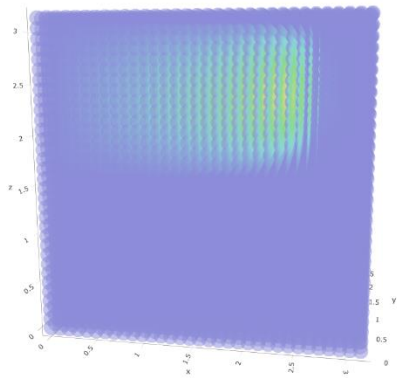

Рисунок 5 Погрешность

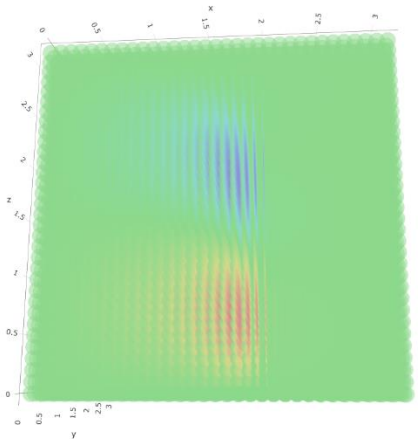

Рисунок 6 Полученное решение

Результаты запусков программ на различных кластерах

Таблица 1. Результаты расчетов на Polus

Lx = Ly = Lz = 1, N = 128, K = 2000

Число MPI-процессов	MPI			MPI + CUDA			Ускорение
	Время решения (с)	Ускорение	Погрешность	Время решения (с)	Ускорение	Погрешность	
1	0.511036	1	2.7637e-08	0.179019	1	2.7637e-08	2,85465
2	0.314435	1,62525	2.7637e-08	0.181736	0,98505	2.7637e-08	1,73017
4	0.201344	2,53812	2.7637e-08	0.19022	0,94112	2.7637e-08	1,05848
6	0.221924	2,30275	2.7637e-08	0.217649	0,82251	2.7637e-08	1,01964

Lx = Ly = Lz = 1, N = 256, K = 2000

Число MPI-процессов	MPI			MPI + CUDA			Ускорение
	Время решения (с)	Ускорение	Погрешность	Время решения (с)	Ускорение	Погрешность	
1	3.93388	1	6.73841e-09	0.112484	1	6.73841e-09	34,97280
2	2.45262	1,60395	6.73841e-09	0.0944015	1,19155	6.73841e-09	25,98073
4	1.63391	2,40765	6.73841e-09	1.3997	0,08035	6.73841e-09	1,16710
6	1.35657	2,89987	6.73841e-09	0.98688	0,11398	6.73841e-09	1,37460

Lx = Ly = Lz = 1, N = 512, K = 2000

Число MPI-процессов	MPI			MPI + CUDA			Ускорение
	Время решения (с)	Ускорение	Погрешность	Время решения (с)	Ускорение	Погрешность	
1	30.5387	1	1.51341e-09	0.69225	1	1.51341e-09	44,11513
2	19.2748	1,58438	1.51341e-09	0.416098	1,66367	1.51341e-09	46,32274
4	12.5462	2,43410	1.51341e-09	0.41664	1,66151	1.51341e-09	30,11281
6	9.93361	3,07428	1.51341e-09	0.403805	1,71432	1.51341e-09	24,60002

Выводы

Как следует из приведенных выше таблиц, задача для трехмерного гиперболического уравнения отлично подходит для распараллеливания. В результате получены программные средства, решающую поставленную задачами гибридным способом при использовании средств MPI и CUDA. Важную роль в MPI при распараллеливании задачи сеточного метода играет способ разбиения на блоки. Использование CUDA для распараллеливания полученной MPI программы более чем целесообразно, о чем свидетельствуют полученные значения ускорений по сравнению с версией без использования CUDA вызовов.