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** 3](#_Toc91362244)

[**Численный метод решения задачи** 3](#_Toc91362245)

[**Программная реализация** 5](#_Toc91362246)

[**Результаты запусков программ на различных кластерах** 9](#_Toc91362247)

[**Выводы** 10](#_Toc91362248)

**Математическая постановка задачи**

В трехмерной замкнутой области

для требуется найти решение уравнения в частных производных с начальными условиями

**Численный метод решения задачи**

Введем на сетку , где ,

Через обозначим множество внутренних, а через – множество граничных узлов сетки .

Для аппроксимации исходного уравнения воспользуемся следующей системой уравнений:

Здесь – семиточечный разностный аналог оператора Лапласа:

Приведенная выше разностная схема является явной – значения на -ом шаге можно явным образом выразить через значения на предыдущих слоях.

Для начала счета должны быть заданы значения :

# **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 , 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 и делается из нее параллелепипед, также продолжается разбиваться область;
* По выбранной оси делим область пополам и рекурсивно запускаем для этих подобластей переходя на следующую ось

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

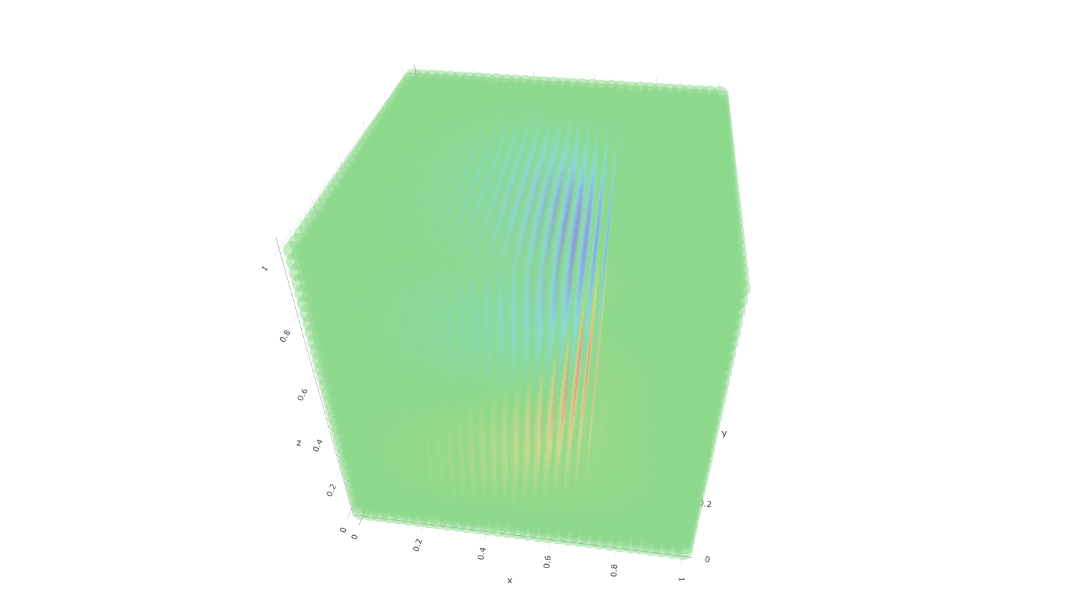


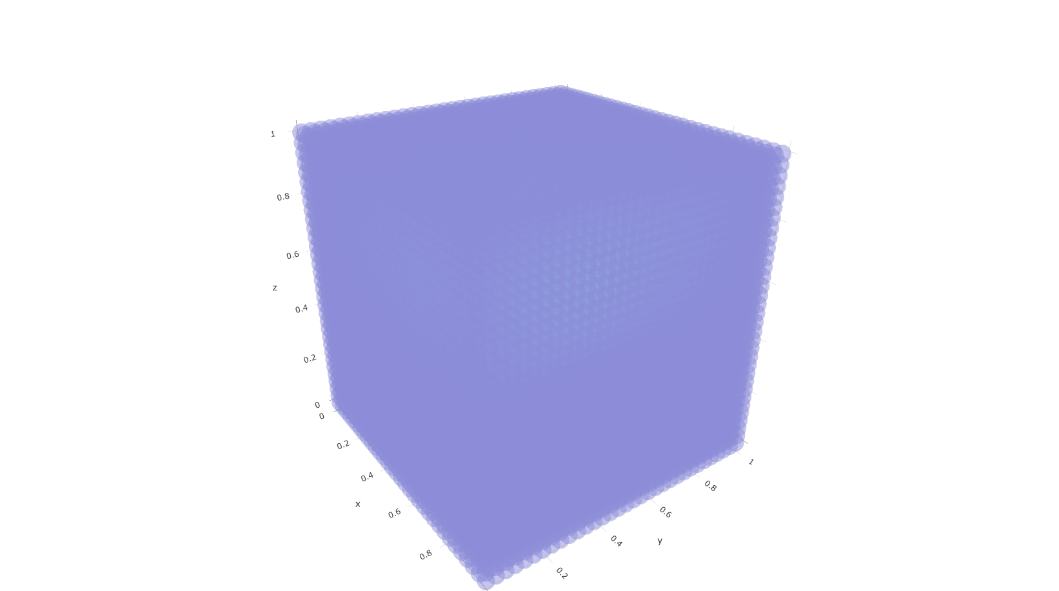
Рисунок 1. Аналитическое решение

Рисунок 2 Погрешность

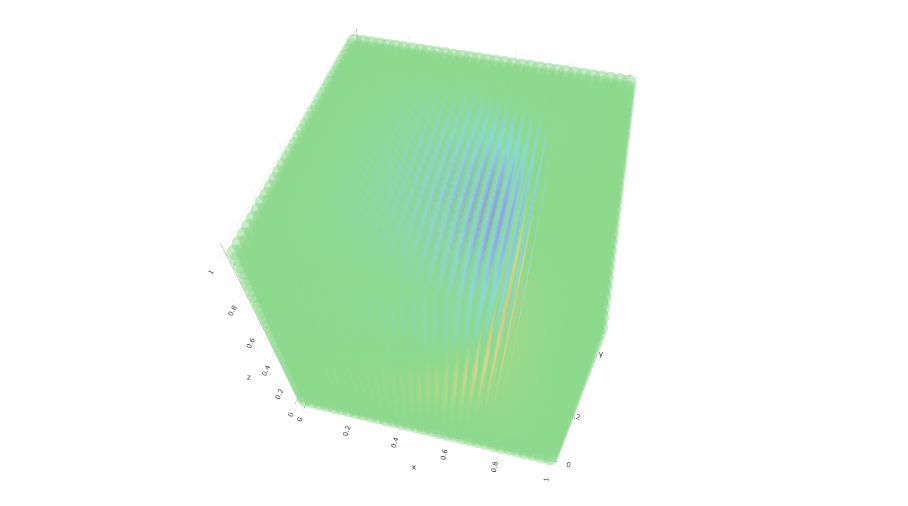


Рисунок 3 Полученное решение

Решение при

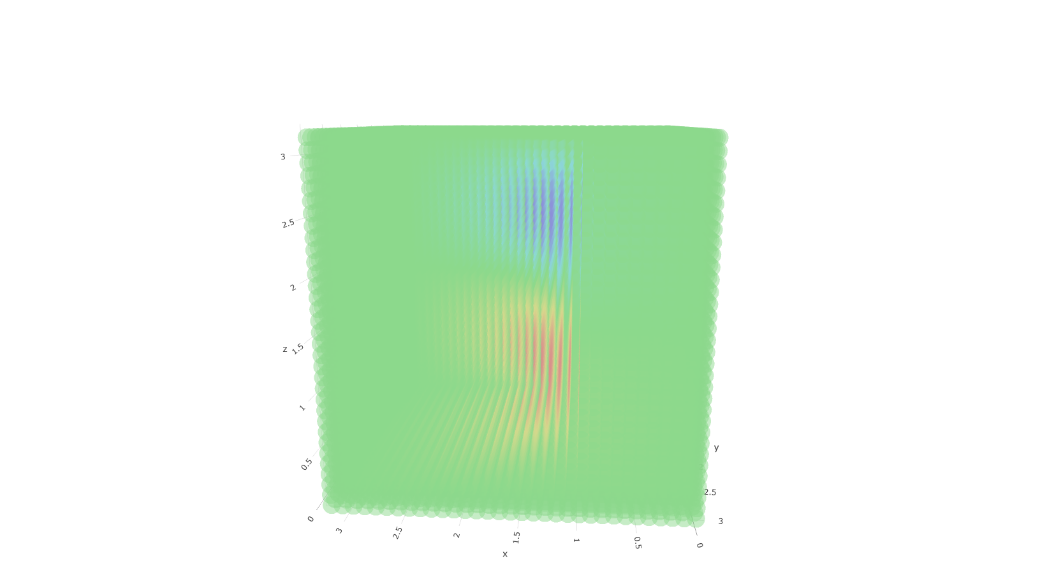


Рисунок 4 Аналитическое решение

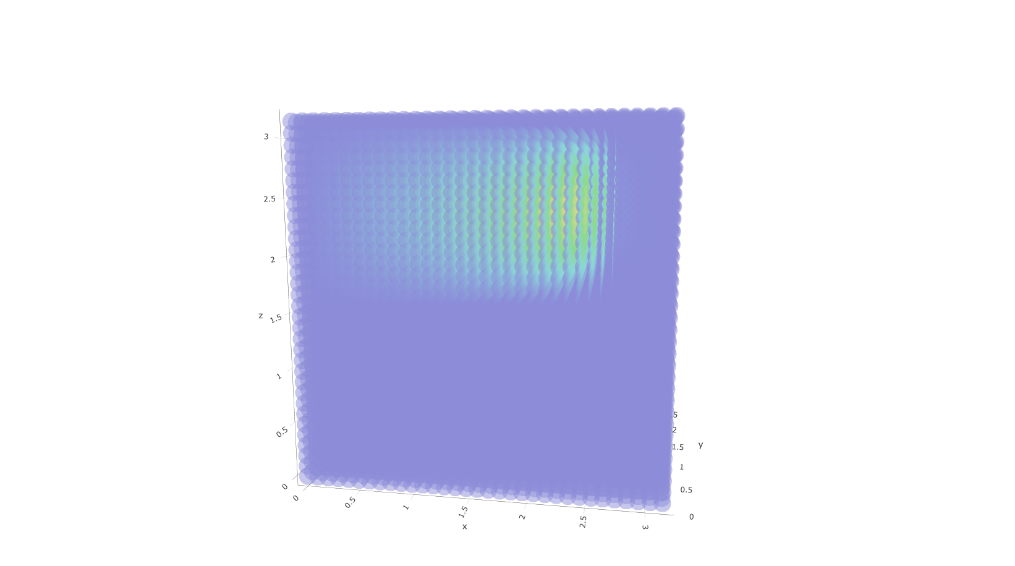


Рисунок 5 Погрешность

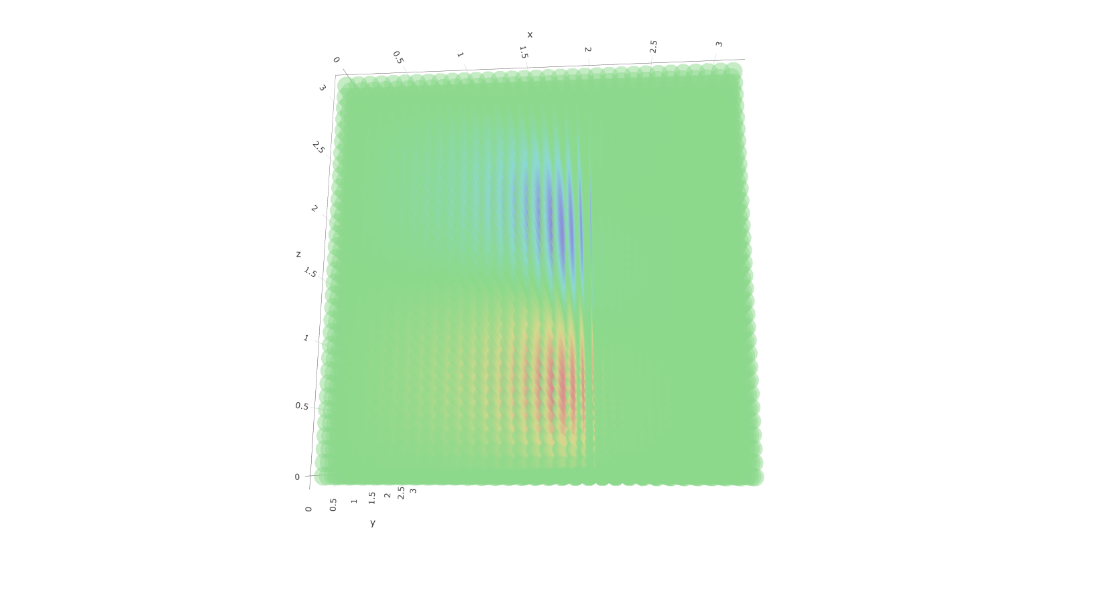


Рисунок 6 Полученное решение

# **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.