Отчет по заданию №3

студента 620 группы Бугаевского Владимира.

Вариант 3

1 Постановка задачи

Рассмотрим трехмерную замкнутую область

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

Требуется для $t \in (0; T]$ найти численное решение $u(x, y, z, t)$ волнового уравнения

$$u_{tt} = \Delta u + f \tag{1}$$

с начальными условиями

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

$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = 0 \tag{3}$$

и заданными граничными условиями (см. вариант задания)

$$u(0, y, z, t) = 0 \qquad u(L_x, y, z, t) = 0 \tag{4}$$

$$u(x, 0, z, t) = u(L_x, y, z, t) \qquad u_y(x, 0, z, t) = u_y(x, L_y, z, t) \tag{5}$$

$$u(x, y, 0, t) = 0 \qquad u(x, y, L_z, t) = 0 \tag{6}$$

2 Алгоритма численного решения задачи

Для численного решения данной задачи введем дискретную сетку на Ω:

$$\bar{\omega}_h = \left\{(x = ih_x, y = jh_y; z = kh_z) \mid i, j, k = \overline{0, N}; L_x = Nh_x; L_y = Nh_y; L_z = Nh_z\right\}$$

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

Через ω_h обозначим множество внутренних, а через γ_h – множество граничных узлов сетки $\bar{\omega}_h$.

Для аппроксимации исходного уравнения воспользуемся следующей системой уравнений:

$$\frac{u^{n+1}_{ijk} + u^n_{ijk} + u^{n-1}_{ijk}}{\tau^2} = \Delta_h u^n + f(x_i, y_j, z_k, t_n) \qquad (x_i, y_j, z_k) \in \omega_h;\ n = \overline{1, K-1}$$

где Δ_h – 7-точечный оператор Лапласса:

$$\Delta_h u^n = \frac{u^n_{i-1jk} - 2u^n_{ijk} + u^n_{i+1jk}}{h_x^2} + \frac{u^n_{ij-1k} - 2u^n_{ijk} + u^n_{ij+1k}}{h_y^2} + \frac{u^n_{ijk-1} - 2u^n_{ijk} + u^n_{ijk+1}}{h_z^2}$$

Приведенная выше разностная схема является явной – значения u^{n+1}_{ijk} на $(n+1)$-м шаге можно явным образом выразить через значения на предыдущих шагах.

Из начальных условий дополнительно имеем:

$$u^0_{ijk} = \varphi(x_i, y_j, z_k)$$
$$u^1_{ijk} = u^0_{ijk} + \frac{\tau^2}{2}\left(\Delta_h u^0 + f(x_i, y_j, z_k, 0)\right)$$

Дополнительно нужно обеспечить выполнение граничных условий на каждой из итераций. Алгоритм для условий Дирихле (4), (6) очевиден. Разностная схема для периодических условий (5) выглядит следующим образом:

$$u^n_{i0k} = u^n_{iN-1k} \qquad u^n_{i1k} = u^n_{iNk}$$

Далее будем полагать, что сетка равномерная: $h_x = h_y = h_z = h$. В этом случае разностная схема будет **устойчива** при $\tau \leq h$.[1]

3 Разбиние области между процессами

В задании предлагается реализовать блочное разбиение между процессами, т.к. в этом случае предполагается меньшее число межпроцессных коммуникаций, чем в случае ленточного разбиения.

Предполагается, что наилучшим разбиением является наиболее равномерное разбиение: число узлов сетки, принадлежащих процессу, для почти всех процессов одинаково. Для осуществления этого предположения предлагается следующий алгоритм:

1. Найти все простые делители числа процессов `n_proc` (с учетом кратности). Это можно сделать эффективно, например, воспользовавшись решетом Эратосфена.

2. Упорядочить все простые делители числа `n_proc` в порядке убывания и поместить их в очередь `Q`.

3. Установить число разбиений `split[3]` для каждой из осей равным 1. Выполнять пока очередь `Q` не пуста:

   (a) Найти номер оси `i`, число разбиений для которой не превышает число разбиений по другим осям.

   (b) Взять очередной элемент `p` из очереди `Q`.

   (c) Обновить разбиение: `split[i] := split[i] * p`.

4 Создание гибридной реализации MPI и OpenMP

Каждый процесс выполняет описанный выше алгоритм численного решения уравнения в своей области, при этом пересылки сообщений между процессами выполняются средствами MPI в следующих случаях:

[1] Самарский А. А., Гулин А. В. "Численные методы".

- Обмен граничными областями между соседними процессами. Данный тип обмена необходим для вычисления оператора Лапласса внутри всех узлов решетки ω_h, принадлежащей процессу.
- Обмен граничными областями γ_h для поддержания выполнения периодических условий.
- Вычисление метрик для оценки качества работы программы.

Директивы OpenMP использовались в программе для распараллеливания циклов, так как значения на различных итерациях внутри циклов вычисляются независимо.

5 Создание гибридной реализации MPI и CUDA

По сравнению с гибридной реализации MPI/OpenMP вносятся следующие изменения:

- Почти все циклы в решении уравнения заменяются на вызовы соответсвующих ядровых функций CUDA.
- Поддержание граничных условий (копирование данных на GPU) осуществляется с помощью ядровых функций.
- Копирование данных с GPU в буферы на CPU для последующего обмена через средства MPI происходит через временный буфер на GPU, заполнение которого осуществляется с помощью вызовов ядровых функций.

Подсчет метрик реализован через дополнительный буфер на GPU, т.к. средствами библиотеки `thrust` не удалось получить значения численного решения только в рабочей области (без учета обменных областей), а использование разделяемой (shared) памяти запрещено.[2] Последующие вычисления норм реализованы с помощью функции `thrust::reduce`.

6 Проверка корректности результата

Возьмем $L_x = L_y = L_z = 16\pi$, шаг сетки τ по времени – 0.004, шаг сетки h в пространстве равен $\frac{16\pi}{N}$. Таким образом разностная схема при любых $N \leq 1536$ будет **устойчива**.

Рассмотрим волновое уравнение

$$u_{tt} = \Delta u$$

с граничными условиями (2) - (6). Легко проверить, что функция

$$u(x, y, z, t) = \cos(\sqrt{14}t)\sin(3x)\cos(2y)\sin(z) \tag{7}$$

[2] В этом случае можно было бы реализовать параллельную редукцию с помощью подхода, описанного в книге Боресков А.В. "Параллельные вычисления на GPU. Архитектура и программная модель CUDA".

является решением[3] волнового уравнения с заданными ограничениями, где

$$\varphi(x, y, z) = \sin(3x)\cos(2y)\sin(z)$$

Для оценки точности аппроксимации решения считались метрики:

- среднеквадратичное отклонение по всем узлам решетки $\bar{\omega}_h \times \omega_\tau$:[4]

$$RMSE = \sqrt{\frac{\sum\limits_{(x_i,y_j,z_k)\in\bar{\omega}_h;\ t_n\in\omega_\tau} [u(x_i, y_j, z_k, t_n) - u^n_{ijk}]^2}{N^3 K}}$$

- max-норма по всем узлам решетки $\bar{\omega}_h \times \omega_\tau$:

$$\|u - \hat{u}\|_\infty = \max_{(x_i,y_j,z_k)\in\bar{\omega}_h;\ t_n\in\omega_\tau} |u(x_i, y_j, z_k, t_n) - u^n_{ijk}|$$

Ниже в таблицах приведены результаты расчётов для приведенной выше задачи на вычислительных комплексах IBM Blue Gene/P и Polus. В замерах время на проверку корректности результата не учитывается. В таблице с подробными замерами время решения уравнения содержит в себе время, затрачиваемое на копирование и операции обмена.

Легко видеть, что для функции (7) ускорение программ пропорционально количеству запущенных процессов: увелечение количества процессов в 2 раза дает ускорение в 2 раза. Версия MPI/OpenMP дает ускорение в 3 раза по сравнению с версией MPI, т.к. в первой из них используется 3 потока в обработке циклов. Ошибка убывает в 4 раза, при увеличении числа узлов сетки N в 2 раза.

При создании гибридной версии MPI/CUDA совпадение показателей метрик с показателями для версии MPI служили индикатором верной реализации.

Гибридная реализация MPI/OpenMP запускалась на выделенном узле (эксклюзивно) на Polus:

```
#!/usr/bin/sh
bsub < $(echo "source /polusfs/setenv/setup.SMPI
#BSUB -n $4
#BSUB -x
#BSUB -W 6:00
#BSUB -q normal
#BSUB -o $1.%J.out
#BSUB -e $1.%J.err
OMP_NUM_THREADS=8 mpiexec $@ ")
```

[3] Подробнее о поиске аналитического решения трехмерного волнового уравнения в общей форме можно прочитать, например, в статье Мустафоевой А. Х., Шариповой М. А., Ортикова Б. Б., Кадирова Н. Х., Абдурашидова А. А. "*Метод разложения Адомиана и метод вариационных итераций решения начальной задачи для n-мерного волнового уравнения*" в журнале "Молодой учёный" № 27 (213) / 2018.

[4] Данная метрика лучше характеризует качество найденного численного решения, чем $l2$-норма, т.к. последняя неограниченно растет с увеличением размерности сетки.

Число процессов, Np	**Размер сетки, N**	**Время работы (сек.), T**	**Ускорение (отн.), S**	**Ошибка, RMSE**	**Ошибка, MAX**
128		8.703534	1.000000		
256	512	4.407454	1.974731	0.000037	0.000224
512		2.218194	3.923703		
128		67.879660	1.000000		
256	1024	34.111406	1.989940	0.000009	0.000055
512		17.094844	3.970768		
128		207.064139	1.000000		
256	1536	103.666751	1.997402	0.000004	0.000024
512		51.944190	3.986281		

Таблица 1: IBM Blue Gene/P, программа с MPI для функции (7), 20 шагов по времени

Число процессов, Np	**Размер сетки, N**	**Время работы (сек.), T**	**Ускорение (отн.), S**	**Ошибка, RMSE**	**Ошибка, MAX**
128		2.737822	1.000000		
256	512	1.421451	1.926076	0.000037	0.000224
512		0.687664	3.981337		
128		20.983482	1.000000		
256	1024	10.583806	1.982603	0.000009	0.000055
512		5.329957	3.936895		
128		70.118795	1.000000		
256	1536	35.283875	1.987276	0.000004	0.000024
512		17.774926	3.944815		

Таблица 2: IBM Blue Gene/P, программа с MPI/OpenMP для функции (7), 20 шагов по времени

Число процессов, Np	**Размер сетки, N**	**Время работы (сек.), T**	**Ускорение (отн.), S**	**Ошибка, RMSE**	**Ошибка, MAX**
1	128	1.519646	1.000000	0.000574	0.003495
2		0.786214	1.932866		
4		0.378663	4.013189		
8		0.231214	6.572465		
16		0.172600	8.804438		
32		0.088042	17.260467		
1	256	12.687070	1.000000	0.000147	0.000894
2		7.137305	1.777572		
4		4.131959	3.070473		
8		1.972973	6.430433		
16		1.347463	9.415524		
32		0.749208	16.933976		
1	512	99.741427	1.000000	0.000037	0.000224
2		50.437589	1.977522		
4		23.256076	4.288833		
8		14.380635	6.935815		
16		6.929573	14.393589		
32		3.929446	25.383076		

Таблица 3: Polus, программа с MPI для функции (7), 20 шагов по времени

Число процессов, Np	**Размер сетки, N**	**Время работы, T (сек.)**	**Ускорение (отн.), S**	**Ошибка, RMSE**	**Ошибка, MAX**
1		0.223418	1.000000		
2		2.891371	0.077271		
4	128	3.848822	0.058048	0.000574	0.003495
8		13.148239	0.016992		
16		24.091810	0.009274		
1		1.793154	1.000000		
2		4.183501	0.428625		
4	256	4.271179	0.419826	0.000147	0.000894
8		13.487228	0.132952		
16		24.169538	0.074191		
1		13.660726	1.000000		
2		15.885040	0.859974		
4	512	10.137706	1.347516	0.000037	0.000224
8		20.006924	0.682800		
16		30.605441	0.446350		

Таблица 4: Polus, программа с MPI/OpenMP для функции (7), 20 шагов по времени

Размер сетки, N	**Ошибка, RMSE**	**Ошибка, MAX**	**Скорость убывания, RMSE**	**Скорость убывания, MAX**
128	0.000574	0.003495	1.000000	1.000000
256	0.000147	0.000894	3.904762	3.909396
512	0.000037	0.000224	15.513514	15.602679
1024	0.000009	0.000055	63.777778	63.545455

Таблица 5: Скорость убывания ошибки для функции (7)

Число процессов, Np	**Размер сетки, N**	**Время работы, T (сек.)**	**Ускорение (отн.), S**	**Ошибка, RMSE**	**Ошибка, MAX**
1	128	0.006437	1.000000	0.000574	0.003495
2		0.028653	0.224654		
4		0.096063	0.067008		
8		0.271496	0.023709		
16		0.605260	0.010635		
32		0.763887	0.008427		
1	256	0.033931	1.000000	0.000147	0.000894
2		0.039865	0.851148		
4		0.067685	0.501308		
8		0.309095	0.109775		
16		0.588771	0.057630		
32		0.878527	0.038623		
1	512	0.219215	1.000000	0.000037	0.000224
2		0.181180	1.209929		
4		0.266179	0.823562		
8		0.242433	0.904229		
16		0.271025	0.808837		
32		0.474010	0.462469		

Таблица 6: Polus, программа с MPI/CUDA для функции (7), 20 шагов по времени

Np	**N**	**Время работы, T (сек.)**			
		Последовательная	**MPI**	**MPI/OpenMP**	**MPI/CUDA**
1	128	0.640927	1.519646	0.223418	0.006437
2			0.786214	2.891371	0.028653
4			0.378663	3.848822	0.096063
8			0.231214	13.148239	0.271496
16			0.172600	24.091810	0.605260
32			0.088042	–	0.763887
1	256	6.249384	12.687070	1.793154	0.033931
2			7.137305	4.183501	0.039865
4			4.131959	4.271179	0.067685
8			1.972973	13.487228	0.309095
16			1.347463	24.169538	0.588771
32			0.749208	–	0.878527
1	512	56.555289	99.741427	13.660726	0.219215
2			50.437589	15.885040	0.181180
4			23.256076	10.137706	0.266179
8			14.380635	20.006924	0.242433
16			6.929573	30.605441	0.271025
32			3.929446	–	0.474010

Таблица 7: Polus, Сравнительная таблица времени работы, $K = 20$

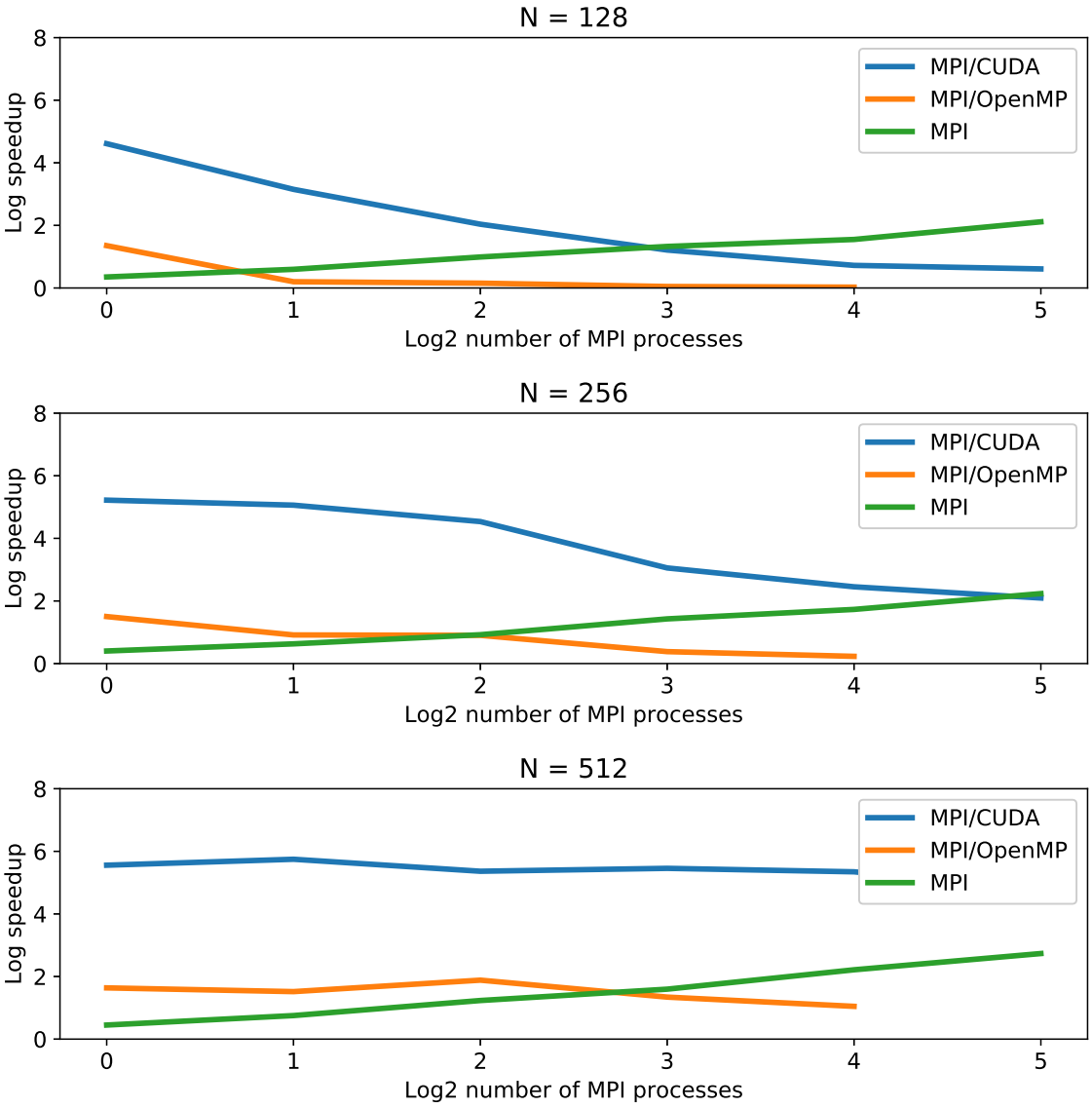

Рис. 1: Ускорение по сравнению с последовательной программой

Np	Время работы, T (сек.)	Послед.	MPI	MPI/OpenMP	MPI/CUDA
1	Инициализация	0.000058	0.041208	0.040529	0.089722
	Копирование	0.124117	0.382080	0.101173	0.008483
	Операции обмена	–	0.000000	0.000000	0.000000
	Решение уравнения	42.145533	97.924752	13.740415	0.219565
	Освобождение ресурсов	0.025072	0.032888	0.030439	0.006878
	Общее время	42.170667	97.998852	13.811390	0.316174
2	Инициализация	0.000058	0.056582	0.043866	0.098678
	Копирование	0.124117	0.348692	0.707532	0.025944
	Операции обмена	–	1.578209	1.871879	0.017404
	Решение уравнения	42.145533	49.213889	15.787442	0.166470
	Освобождение ресурсов	0.025072	0.025388	0.022763	0.003609
	Общее время	42.170667	49.295865	15.854079	0.268765
4	Инициализация	0.000058	0.062120	0.055118	0.110599
	Копирование	0.124117	0.194580	1.037105	0.020070
	Операции обмена	–	0.154491	2.606831	0.035752
	Решение уравнения	42.145533	22.980075	10.132057	0.150749
	Освобождение ресурсов	0.025072	0.020302	0.017665	0.003188
	Общее время	42.170667	23.062503	10.204847	0.264544
8	Инициализация	0.000058	0.100981	0.090078	0.153120
	Копирование	0.124117	0.095475	3.738840	0.020325
	Операции обмена	–	0.541761	9.932141	0.227529
	Решение уравнения	42.145533	12.129515	19.393700	0.280711
	Освобождение ресурсов	0.025072	0.024988	0.032329	0.002207
	Общее время	42.170667	12.255489	19.516115	0.436044
16	Инициализация	0.000058	0.204947	0.156190	0.726385
	Копирование	0.124117	0.084995	3.902944	0.037003
	Операции обмена	–	0.891466	19.853067	0.192001
	Решение уравнения	42.145533	6.823599	29.844663	0.330344
	Освобождение ресурсов	0.025072	0.030794	0.057797	0.007119
	Общее время	42.170667	7.059344	30.058657	1.063856

Таблица 8: Polus, Подробные замеры времени для функции (7), $K = 20$, $N = 512$

7 Выводы

Для MPI-программы наблюдается ожидаемое ускорение: при увеличении числа процессов в 2 раза, время работы падает в 2 раза. Однако, MPI-программа на 1 процессе существенно проигрывает последовательной. Данное явление связано с наличием больших накладных расходов на инициализацию окружения для процессов. Начиная с некоторого момента накладные расходы на инициализацию становятся менее существенными, и MPI-программа дает ожидаемое ускорение.

Поведение для программы MPI/OpenMP не вполне ожидаемо. Наблюдается достаточно быстрый рост времени накладных расходов: операции копировани и операций обмена, – при этом чистое время, затраченное на решение уравнения, убывает. В связи с этим пик ускорения на больших данных достигается на 4-х процессах из 16-и. Вероятнее всего причиной этого являются большие издержки на управление потоками.

Время работы гибридной MPI/CUDA-программы существенно меньше времени работы остальных реализаций. Данный факт обусловлен большим потенциалом векторных вычислений. Операция копирования данных между CPU и GPU достаточно дорогая, поэтому при большом числе процессов заметно замедление (из-за увелечения количества операций копирования), и программа на MPI/CUDA начинает проигрывать программе на MPI. Однако при больших размерах матрицы данный эффект не сильно заметен, более того ускорение относительно последовательной программы будет только расти с увеличением данных.