Отчет по заданию курса «Суперкомпьютерное моделирование и технологии»

Михальцов Данила Алексеевич

Октябрь 2024 - Декабрь 2024

Содержание

1 Математическая постановка дифференциальной задачи

В трехмерной замкнутой области

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

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

$$\frac{\partial^2 u}{\partial t^2} = \Delta u \tag{1}$$

с начальными условиями

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

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

при условии, что на границах области заданы следующие граничные условия:

- Периодические граничные условия по осям x и y:

$$u(0, y, z, t) = u\left(L_x, y, z, t\right), u_x(0, y, z, t) = u_x\left(L_x, y, z, t\right),$$

$$u(x, 0, z, t) = u\left(x, L_y, z, t\right), u_y(x, 0, z, t) = u_y\left(x, L_y, z, t\right)$$

- Граничные условия первого рода по оси z:

$$u(x, y, 0, t) = 0, \quad u(x, y, L_z, t) = 0 \tag{4}$$

Аналитическое решение задачи имеет вид:

$$u_{\text{analytical}}(x, y, z, t) = \sin\left(\frac{2\pi x}{L_x} + 3\pi\right) \sin\left(\frac{2\pi y}{L_y} + 2\pi\right) \sin\left(\frac{\pi z}{L_z}\right) \cos(a_t \cdot t + \pi) \tag{5}$$

где

$$a_t = \pi \sqrt{\frac{4}{L_x^2} + \frac{4}{L_y^2} + \frac{1}{L_z^2}} \tag{6}$$

2 Численный метод решения задачи

Для численного решения задачи введем на Ω сетку $\omega_{h\tau} = \bar{\omega}_h \times \omega_\tau$, где

$$T = T_0$$
$$L_x = L_{x_0}, L_y = L_{y_0}, L_z = L_{z_0}$$
$$\bar{\omega}_h = \left\{\left(x_i = ih_x, y_j = jh_y, z_k = kh_z\right), i, j, k = 0, 1, \ldots, N, h_x N = L_x, h_y N = L_y, h_z N = L_z\right\},$$
$$\omega_\tau = \left\{t_n = n\tau, n = 0, 1, \ldots, K, \tau K = T\right\}$$

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

Для аппроксимации исходного уравнения (1) с однородными граничными условиями (4)-(6) и начальными условиями (2)-(3) воспользуемся следующей системой уравнений:

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

Здесь Δ_h - семиточечный разностный аналог оператора Лапласа:

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

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

Для начала счета (т.е. для нахождения u_{ijk}^2) должны быть заданы значения u_{ijk}^0, u_{ijk}^1, $(x_i, y_j, z_k) \in \omega_h$. Из условия (2) имеем

$$u_{ijk}^0 = \varphi(x_i, y_j, z_k), \quad (x_i, y_j, z_k) \in \omega_h \tag{10}$$

Аппроксимацию второго порядка по τ и h дает разностное уравнение

$$\begin{gathered} \frac{u_{ijk}^1 - u_{ijk}^0}{\tau} = \frac{\tau}{2}\Delta_h \varphi(x_i, y_j, z_k), \quad (x_i, y_j, z_k) \in \omega_h. \\ u_{ijk}^1 = u_{ijk}^0 + \frac{\tau^2}{2}\Delta_h \varphi(x_i, y_j, z_k) \end{gathered} \tag{12}$$

Разностная аппроксимация для периодических граничных условий по OX и OY выглядит следующим образом

$$\begin{gathered} u_{0jk}^{n+1} = u_{Njk}^{n+1}, \\ u_{i0k}^{n+1} = u_{iNk}^{n+1}, \\ u_{1jk}^{n+1} = u_{N+1jk}^{n+1}, \\ u_{ik}^{n+1} = u_{iN+k}^{n+1}, \end{gathered}$$

$i, j, k = 0, 1, \ldots, N$.

Для вычисления значений $u^0, u^1 \in \gamma_h$ допускается использование аналитического значения u, которое задается в программе еще для вычисления погрешности решения задачи.

3 Программная реализация

Программа написана на языке C++ 11. Изначально была реализована последовательная версия программы, не использующая возможности OpenMP или MPI.

3.1 OpenMP

Для реализации параллельной версии программы с помощью интерфейса OpenMP в программу к основным вычисляющим циклам были добавлены следующие клаузы:

- `#pragma omp parallel for collapse(3)` — для основных циклов
- `#pragma omp parallel for collapse(3) reduction(max : error)` — для определения максимальной погрешности в конце выполнения программы

3.2 MPI

В MPI-версии программы для блочного разбиения используются функции `MPI_Dims_create` и `MPI_Cart_create`: создаётся декартова решетка процессов. Функция `compute_local_size` рассчитывает размер и начальный индекс локальной области сетки для каждого процесса, равномерно распределяя элементы оси с учетом остатка деления. Функция `exchange_boundaries` реализует передачу данных между соседними процессами по трем направлениям (x, y, z). Используются операции `MPI_Isend` и `MPI_Irecv` для асинхронной отправки и получения данных. `MPI_Reduce` используется для вычисления глобальной максимальной ошибки среди всех процессов.

Обмен граничными значениями

1. Соседи определяются с использованием функции `MPI_Cart_shift`, которая возвращает ранги соседних процессов в заданном направлении. Это позволяет связать процессы в декартовой топологии и определить, куда отправлять и откуда принимать данные.

2. Передача данных между процессами осуществляется через `MPI_Isend` (асинхронная отправка) и `MPI_Irecv` (асинхронный приём). Это позволяет перекрывать время на упаковку данных.

3. Все асинхронные операции синхронизируются вызовом `MPI_Waitall`, который завершает обмен данными и гарантирует, что все данные корректно переданы и приняты.

4. Принятые данные записываются в специальные слои-призраки локальных массивов, которые используются для расчётов на границах текущего процесса.

Компиляция программ

Последовательная программа и версия с OpenMP компилировались с помощью свободно распространяемого компилятора g++, для MPI-версии использовался mpic++. Для компиляции параллельной версии программы был добавлен флаг `-fopenmp`. Для повышения производительности программы при компиляции использовался флаг оптимизации `-O2`.

4 Результаты и их анализ

Результаты численных расчетов приведены в таблицах ниже.

Во всех таблицах при $N_{\text{threads}} = 1$ и $N_{\text{processes}}$ использовалась последовательная версия программы (без прагм OpenMP и без использования функций MPI).

4.1 Результаты для OpenMP-версии

Полученные результаты для $L = 1$:

N_{threads}	N^3	T (время решения в секундах)	S (ускорение)	δ (погрешность)
1	128^3	7.75161	1.00	9.78588×10^{-5}
2	128^3	3.87850	2.00	9.78588×10^{-5}
4	128^3	1.97013	3.93	9.78588×10^{-5}
8	128^3	1.06273	7.29	9.78588×10^{-5}
1	256^3	57.4123	1.00	4.78022×10^{-5}
2	256^3	28.5211	2.01	4.78022×10^{-5}
4	256^3	14.4583	3.97	4.78022×10^{-5}
8	256^3	7.70675	7.45	4.78022×10^{-5}
1	512^3	440.478	1.00	2.3903×10^{-5}
2	512^3	218.594	2.02	2.3903×10^{-5}
4	512^3	111.082	3.96	2.3903×10^{-5}
8	512^3	57.2771	7.69	2.3903×10^{-5}

Полученные результаты для $L = \pi$:

N_{threads}	N^3	T (время решения в секундах)	S (ускорение)	δ (погрешность)
1	128^3	7.7516	1.00	9.78588×10^{-5}
2	128^3	3.8829	2.00	9.78588×10^{-5}
4	128^3	1.9888	3.90	9.78588×10^{-5}
8	128^3	1.1054	7.01	9.78588×10^{-5}
1	256^3	57.4123	1.00	4.78022×10^{-5}
2	256^3	28.5332	2.01	4.78022×10^{-5}
4	256^3	14.5339	3.95	4.78022×10^{-5}
8	256^3	7.7635	7.39	4.78022×10^{-5}
1	512^3	440.4780	1.00	2.3903×10^{-5}
2	512^3	219.4330	2.01	2.3903×10^{-5}
4	512^3	110.2730	4.00	2.3903×10^{-5}
8	512^3	57.1360	7.71	2.3903×10^{-5}

4.2 Результаты для MPI-версии

Полученные результаты для $L = 1$:

$N_{\text{processes}}$	N^3	T (время решения в секундах)	S (ускорение)	δ (погрешность)
1	128^3	7.75161	1.00	9.78588×10^{-5}
4	128^3	4.92677	1.57	9.7158×10^{-4}
8	128^3	4.53628	1.71	9.7158×10^{-4}
16	128^3	7.72939	1.00	9.7158×10^{-4}
32	128^3	9.20474	0.84	9.7158×10^{-4}
1	256^3	57.4123	1.00	4.78022×10^{-5}
4	256^3	31.2190	1.84	4.85947×10^{-4}
8	256^3	33.2043	1.73	4.85947×10^{-4}
16	256^3	37.4448	1.53	4.85947×10^{-4}
32	256^3	28.4587	2.02	4.85947×10^{-4}
1	512^3	440.478	1.00	2.3903×10^{-5}
4	512^3	235.206	1.87	2.42993×10^{-4}
8	512^3	231.367	1.90	2.42993×10^{-4}
16	512^3	195.181	2.26	2.42993×10^{-4}
32	512^3	116.538	3.78	2.42993×10^{-4}

Полученные результаты для $L = \pi$:

$N_{\text{процессов}}$	N^3	T (время решения в секундах)	S (ускорение)	δ (погрешность)
1	128^3	7.75161	1.00	9.78588×10^{-5}
4	128^3	4.90805	1.58	9.7158×10^{-4}
8	128^3	6.68562	1.16	9.7158×10^{-4}
16	128^3	10.2063	0.76	9.7158×10^{-4}
32	128^3	9.99473	0.78	9.7158×10^{-4}
1	256^3	57.4123	1.00	4.78022×10^{-5}
4	256^3	31.2203	1.84	4.85947×10^{-4}
8	256^3	33.4433	1.72	4.85947×10^{-4}
16	256^3	37.1163	1.55	4.85947×10^{-4}
32	256^3	28.7275	2.00	4.85947×10^{-4}
1	512^3	440.478	1.00	2.3903×10^{-5}
4	512^3	235.137	1.87	2.42993×10^{-4}
8	512^3	231.873	1.90	2.42993×10^{-4}
16	512^3	205.28	2.15	2.42993×10^{-4}
32	512^3	116.397	3.78	2.42993×10^{-4}

4.3 Результаты для версии OpenMP + MPI

Полученные результаты для $L = 1$:

$N_{\text{процессов}}$	$N_{\text{потоков}}$	N^3	T (секунды)	S (ускорение)	δ (погрешность)
1	1	128^3	7.75161	1.00	9.78588×10^{-5}
2	4	128^3	2.91323	2.66	9.7158×10^{-4}
4	4	128^3	2.76536	2.80	9.7158×10^{-4}
8	4	128^3	5.05592	1.53	9.7158×10^{-4}
16	4	128^3	10.162	0.76	9.7158×10^{-4}
32	4	128^3	12.8685	0.60	9.7158×10^{-4}
1	1	256^3	57.4123	1.00	4.78022×10^{-5}
2	4	256^3	15.7445	3.65	4.85947×10^{-4}
4	4	256^3	8.64876	6.64	4.85947×10^{-4}
8	4	256^3	11.3902	5.04	4.85947×10^{-4}
16	4	256^3	16.3104	3.52	4.85947×10^{-4}
32	4	256^3	15.8975	3.61	4.85947×10^{-4}
1	1	512^3	440.478	1.00	2.3903×10^{-5}
2	4	512^3	114.071	3.86	2.42993×10^{-4}
4	4	512^3	59.6326	7.39	2.42993×10^{-4}
8	4	512^3	61.3528	7.18	2.42993×10^{-4}
16	4	512^3	66.0439	6.67	2.42993×10^{-4}
32	4	512^3	48.4313	9.09	2.42993×10^{-4}

Полученные результаты для $L = \pi$:

$N_{\text{processes}}$	N_{threads}	N^3	T (секунды)	S (ускорение)	δ (погрешность)
1	1	128^3	7.75161	1.00	9.78588×10^{-5}
2	4	128^3	3.18958	2.43	9.7158×10^{-4}
4	4	128^3	2.70394	2.87	9.7158×10^{-4}
8	4	128^3	4.91475	1.58	9.7158×10^{-4}
16	4	128^3	10.2256	0.76	9.7158×10^{-4}
32	4	128^3	12.5563	0.62	9.7158×10^{-4}
1	1	256^3	57.4123	1.00	4.78022×10^{-5}
2	4	256^3	15.6375	3.67	4.85947×10^{-4}
4	4	256^3	8.67156	6.62	4.85947×10^{-4}
8	4	256^3	11.3301	5.07	4.85947×10^{-4}
16	4	256^3	16.4405	3.49	4.85947×10^{-4}
32	4	256^3	16.058	3.57	4.85947×10^{-4}
1	1	512^3	440.478	1.00	2.3903×10^{-5}
2	4	512^3	113.539	3.88	2.42993×10^{-4}
4	4	512^3	59.7976	7.36	2.42993×10^{-4}
8	4	512^3	61.7325	7.13	2.42993×10^{-4}
16	4	512^3	53.214	8.28	2.42993×10^{-4}
32	4	512^3	49.6448	8.87	2.42993×10^{-4}

Порядок проведения экспериментов

Для получения более стабильных результатов было проведено по 5 запусков для каждого набора параметров, и в качестве времени выполнения для этого набора параметров было взято минимальное из значений этих запусков. Результаты по времени сильно различались от запуска к запуску из-за разной загруженности IBM Polus задачами пользователей, однако взятие минимума помогло получить более объективную картину.

Также, вне рамок постановки задания, были проведены эксперименты для $N = 1024$, однако для такого размера последовательная программа превышает максимально доступный на IBM Polus лимит по времени для задач пользователя-студента (30 минут), поэтому посчитать ускорение для такого N не представляется возможным.

Для наглядности все значения в таблице округлены до двух знаков после запятой в ускорении и до четырех знаков после запятой во времени выполнения.

5 Заключение

На основании полученных данных можно сделать вывод о том, что увеличение количества потоков приводит к значительному сокращению времени вычислений, что демонстрирует эффективность параллелизации задачи с использованием OpenMP и MPI.

Полученное ускорение не равно количеству используемых тредов, и это вполне ожидаемо. Самым главным фактором является пропускная способность памяти. Влияние этого фактора подтвердилось с помощью запуска одной и той же программы в разное время: при разной загрузке суперкомпьютера задачами других пользователей получались сильно разные результаты по времени, что свидетельствует о том, что, скорее всего, на работу программы влияет интенсивность использования памяти. Также, для параллельных вычислений неизбежны накладные расходы, связанные с управлением потоками, в том числе для подсчёта максимальной погрешности. В программе присутствуют непараллельные области, что ограничивает теоретически возможное ускорение.

Для MPI версии видно, что при увеличении размерности задачи ускорение для одинакового числа процессов увеличивается. При текущих выбранных значениях N накладные расходы на обмен данными между процессами достаточно велики по сравнению с арифметическими операциями. При большем размере задачи ($N > 1000$ и далее), скорее всего, можно будет получить большее ускорение, повышая эффективность распараллеливания.

Таким образом, проведенные расчеты показывают, что распараллеливание программ с помощью OpenMP и MPI позволяет добиться значительного ускорения. Для небольших размеров задачи, когда массив задачи может поместиться в оперативную память целиком, и пропускной способности памяти хватает, разумно использовать распараллеливание с применением OpenMP. Для больших размеров задач будет эффективнее использовать MPI-версию, в том числе с применением OpenMP.