Лабораторная работа №7 (ПОД №1)

Message Passing Interface (MPI)

Цель работы. Знакомство с технологией MPI. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода.

Математическая постановка:

$$\frac{d^2u(x,y,z)}{dx^2} + \frac{d^2u(x,y,z)}{dy^2} + \frac{d^2u(x,y,z)}{dz^2} = 0 ,$$

$u(x \le 0, y, z) = u_{left}$,

$u(x \ge l_x, y, z) = u_{right}$,

$u(x,\ y \le 0, z) = u_{front}$,

$u(x,\ y \ge l_y, z) = u_{back}$,

$u(x,\ y,\ z \le 0) = u_{down}$,

$u(x,\ y,\ z \ge l_z) = u_{up}$.

Над пространством строится регулярная сетка. С каждой ячейкой сопоставляется значение функции u в точке соответствующей центру ячейки. Граничные условия реализуются через виртуальные ячейки, которые окружают рассматриваемую область.

Поиск решения сводится к итерационному процессу:

$$u_{i,j,k}^{(n+1)} = \frac{\left(u_{i+1,j,k}^{(n)} + u_{i-1,j,k}^{(n)}\right)h_x^{-2} + \left(u_{i,j+1,k}^{(n)} + u_{i,j-1,k}^{(n)}\right)h_y^{-2} + \left(u_{i,j,k+1}^{(n)} + u_{i,j,k-1}^{(n)}\right)h_z^{-2}}{2\left(h_x^{-2} + h_y^{-2} + h_z^{-2}\right)} ,$$

где

$i = 1..n_x$, $j = 1..n_y$, $k = 1..n_z$,

$h_x = l_x n_x^{-1}$, $h_y = l_y n_y^{-1}$, $h_z = l_z n_z^{-1}$,

$u_{0,j,k}^{(n)} = u_{left}$, $u_{n_x+1,j,k}^{(n)} = u_{right}$,

$u_{i,0,k}^{(n)} = u_{front}$, $u_{i,n_y+1,k}^{(n)} = u_{back}$,

$u_{i,j,0}^{(n)} = u_{down}$, $u_{i,j,n_z+1}^{(n)} = u_{up}$,

$u_{i,j,k}^{(0)} = u^0$.

Процесс останавливается, как только

$$max_{i,j,k}\left|u_{i,j,k}^{(n+1)} - u_{i,j,k}^{(n)}\right| < \varepsilon$$

Рассмотрим распараллеливание на примере двумерной задачи. Пусть имеется сетка процессов 2x2. Исходная область разбивается на четыре одинаковые подобласти меньшего размера:

	u_{front}	u_{front}	u_{front}	u_{front}	
u_{left}	$u_{1,1}$	$u_{2,1}$	$u_{3,1}$	$u_{4,1}$	u_{right}
u_{left}	$u_{1,2}$	$u_{2,2}$	$u_{3,2}$	$u_{4,2}$	u_{right}
u_{left}	$u_{1,3}$	$u_{2,3}$	$u_{3,3}$	$u_{4,3}$	u_{right}
u_{left}	$u_{1,4}$	$u_{2,4}$	$u_{3,4}$	$u_{4,4}$	u_{right}
	u_{back}	u_{back}	u_{back}	u_{back}	

Каждый процесс отвечает за свой кусок сетки:

	u_{front}	u_{front}	
u_{left}	$u_{1,1}$	$u_{2,1}$	$u_{3,1}$
u_{left}	$u_{1,2}$	$u_{2,2}$	$u_{3,2}$
	$u_{1,3}$	$u_{2,3}$	

$\Leftrightarrow$

	u_{front}	u_{front}	
$u_{2,1}$	$u_{3,1}$	$u_{4,1}$	u_{right}
$u_{2,2}$	$u_{3,2}$	$u_{4,2}$	u_{right}
	$u_{3,3}$	$u_{4,3}$	

$\Updownarrow$ $\Updownarrow$

	$u_{1,2}$	$u_{2,2}$	
u_{left}	$u_{1,3}$	$u_{2,3}$	$u_{3,3}$
u_{left}	$u_{1,4}$	$u_{2,4}$	$u_{3,4}$
	u_{back}	u_{back}	

$\Leftrightarrow$

	$u_{3,2}$	$u_{4,2}$	
$u_{2,3}$	$u_{3,3}$	$u_{4,3}$	u_{right}
$u_{2,4}$	$u_{3,4}$	$u_{4,4}$	u_{right}
	u_{back}	u_{back}	

Серым цветом показаны виртуальные ячейки. Одна итерация решения исходной задачи состоит из трех этапов. На первом этапе происходит обмен граничными слоями между процессами. На втором этапе выполняется обновление значений во всех ячейках. И третий этап заключается в вычислении погрешности: сначала локально в рамках каждого процесса а потом через обмены и во всей области.

Входные данные. На первой строке заданы три числа: размер сетки процессов. Гарантируется, что при запуске программы количество процессов будет равно произведению этих трех чисел. На второй строке задается размер блока, который будет обрабатываться одним процессом: три числа. Далее задается путь к выходному файлу, в который необходимо записать конечный результат работы программы и точность ε. На последующих строках описывается задача: задаются размеры области l_x, l_y и l_z, граничные условия: u_{down}, u_{up}, u_{left}, u_{right}, u_{front} и u_{back}, и начальное значение u^0.

Выходные данные. В файл, определенный во входных данных, необходимо напечатать построчно значения ($u_{1,1,1}, u_{2,1,1}, ..., u_{1,2,1}, u_{2,2,1}, ..., u_{n_x-1,n_y,n_z}, u_{n_x,n_y,n_z}$) в ячейках сетки в формате с плавающей запятой с семью знаками мантиссы.

Пример:

Входной файл	mpi.out
1 1 1 3 3 3 mpi.out 1e-10 1.0 1.0 1.0 7.0 7.0 7.0 7.0 7.0 7.0 0.0	7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00
1 1 2 3 3 3 mpi.out 1e-10 1.0 1.0 2.0 7.0 0.0 5.0 0.0 3.0 0.0 5.0	4.627978e+00 4.194414e+00 3.239955e+00 4.728116e+00 4.177304e+00 3.095109e+00 3.795164e+00 3.214610e+00 2.407141e+00 3.845336e+00 3.121249e+00 2.150205e+00 3.768252e+00 2.831573e+00 1.746256e+00 2.828257e+00 1.908052e+00 1.133126e+00 3.554538e+00 2.705966e+00 1.793770e+00 3.376228e+00 2.268328e+00 1.267522e+00 2.498077e+00 1.440742e+00 7.373100e-01 3.399697e+00 2.497911e+00 1.638930e+00 3.168173e+00 1.987935e+00 1.059467e+00 2.343237e+00 1.232688e+00 5.824692e-01 3.177560e+00 2.254941e+00 1.482429e+00 2.901943e+00 1.701042e+00 8.799475e-01 2.160481e+00 1.041743e+00 4.653501e-01 2.508778e+00 1.670702e+00 1.120755e+00 2.204404e+00 1.139741e+00 5.713973e-01 1.675964e+00 6.908981e-01 2.879409e-01

Запись результатов в файл должна осуществляться одним процессом. Необходимо использовать последовательную пересылку данных по частям на пишущий процесс.

Варианты заданий:

1. обмен граничными слоями через send/receive, контроль сходимости allgather;
2. обмен граничными слоями через bsend, контроль сходимости allgather;
3. обмен граничными слоями через sendrecv, контроль сходимости allgather;
4. обмен граничными слоями через isend/irecv, контроль сходимости allgather;
5. обмен граничными слоями через send/receive, контроль сходимости allreduce;
6. обмен граничными слоями через bsend, контроль сходимости allreduce;
7. обмен граничными слоями через sendrecv, контроль сходимости allreduce;
8. обмен граничными слоями через isend/irecv, контроль сходимости allreduce;

В программе допускается двойное использование памяти относительно размера блока обрабатываемого одним процессом.

Лабораторная работа №8 (ПОД №3)

Технология MPI и технология CUDA. MPI-IO

Цель работы. Совместное использование технологии MPI и технологии CUDA. Применение библиотеки алгоритмов для параллельных расчетов Thrust. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода. Использование механизмов MPI-IO и производных типов данных.

Требуется решить задачу описанную в лабораторной работе №7, используя возможности графических ускорителей установленных на машинах вычислительного кластера. Учесть возможность наличия нескольких GPU в рамках одной машины. На GPU необходимо реализовать основной расчет. Требуется использовать объединение запросов к глобальной памяти. На каждой итерации допустимо копировать только граничные элементы с GPU на CPU для последующей отправки их другим процессам. Библиотеку Thrust использовать только для вычисления погрешности в рамках одного процесса.

Все **входные-выходные данные** и **варианты заданий по межпроцессорному взаимодействию** совпадают с входными-выходными данными и вариантами заданий из лабораторной работы №7.

Запись результатов в файл должна осуществляться параллельно всеми процессами. Необходимо создать производный тип данных, определяющий шаблон записи данных в файл. **Варианты** конструкторов типов:

1. MPI_Type_create_subarray
2. MPI_Type_hvector
3. MPI_Type_hindexed

Допускается двойное использование графической памяти относительно размера блока обрабатываемого одним процессом.

Лабораторная работа №9 (ПОД №2)

Технология MPI и технология OpenMP

Цель работы. Совместное использование технологии MPI и технологии OpenMP. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода.

Требуется решить задачу описанную в лабораторной работе №7, с использованием стандарта распараллеливания openmp в рамках одного процесса.

Все **входные-выходные данные** и **варианты заданий** по технологии MPI совпадают с входными-выходными данными и вариантами заданий из лабораторной работы №8. **Обмен граничными слоями** организовать без использования дополнительных буферов, через производный тип данных в соответствии с вариантом лабораторной работы №8.

По технологии OpenMP вводятся **два варианта**:

1. Распараллеливание основных циклов через parallel for (+директива reduction для вычисления погрешности);
2. Распараллеливание в общем виде с разделением работы между нитями вручную (“в стиле CUDA”).

Кластер infway

Технические характеристики машин в кластере, на котором будет происходить проверка работ:

1. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 16 Gb, GeForce GTX 1050, 2 Gb
2. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 16 Gb, GeForce GT 545, 3 Gb
3. Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 16 Gb, GeForce GTX 650, 2 Gb
4. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 12 Gb, GeForce GT 530, 2 Gb
5. Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 8 Gb, GeForce GT 530, 2 Gb

Все машины соединены гигабатным ethernet и находятся в подсети 10.10.1.1/24. В качестве операционной системы установлена Ubuntu 16.04.6 LTS. Версии софта:

mpirun 1.10.2, g++ 4.8.4, nvcc 7.0

При компиляции используется одна из команд:

```
mpic++ --std=c++11 -fopenmp -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm
nvcc -ccbin=mpic++ --std=c++11 -Werror cross-execution-space-call -lm
```

Для запуска:

```
mpirun -np $np ./a.out
```

где $np - количество процессов

Программный код необходимо отправить на чекер с темой письма pgp:7, pgp:8 или pgp:9 соответственно.

Вариант на “два”. Решение двумерной задачи.

Для лабораторных работ №7, №8 и №9 упрощенный вариант заключается в замене трехмерной задачи Дирихле на двумерную.

Математическая постановка:

$$\frac{d^2u(x,y)}{dx^2} + \frac{d^2u(x,y)}{dy^2} = 0,$$

$u(x \le 0, y) = u_{left}$,
$u(x \ge l_x, y) = u_{right}$,
$u(x,\ y \le 0) = u_{front}$,
$u(x,\ y \ge l_y) = u_{back}$.

Над пространством строится регулярная сетка. С каждой ячейкой сопоставляется значение функции u в точке соответствующей центру ячейки. Граничные условия реализуются через виртуальные ячейки, которые окружают рассматриваемую область.

Поиск решения сводится к итерационному процессу:

$$u_{i,j}^{(k+1)} = \frac{\left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)}\right)h_x^{-2} + \left(u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)}\right)h_y^{-2}}{2\left(h_x^{-2} + h_y^{-2}\right)},$$

где

$i = 1..n_x$, $j = 1..n_y$,
$h_x = l_x n_x^{-1}$, $h_y = l_y n_y^{-1}$,
$u_{0,j}^{(k)} = u_{left}$, $u_{n_x+1,j}^{(k)} = u_{right}$,
$u_{i,0}^{(k)} = u_{front}$, $u_{i,n_y+1}^{(k)} = u_{back}$,
$u_{i,j}^{(0)} = u^0$.

Процесс останавливается, как только

$$max_{i,j}\left|u_{i,j}^{(k+1)} - u_{i,j}^{(k)}\right| < \varepsilon$$

Входные данные. На первой строке заданы два числа: размер сетки процессов. Гарантируется, что при запуске программы количество процессов будет равно произведению этих двух чисел. На второй строке задается размер блока, который будет обрабатываться одним процессом: два числа. Далее задается путь к выходному файлу, в который необходимо записать конечный результат работы программы и точность ε. На последующих строках описывается задача: задаются размеры области l_x и l_y, граничные условия: u_{left}, u_{right}, u_{front} и u_{back}, и начальное значение u^0.

Выходные данные. В файл, определенный во входных данных, необходимо напечатать построчно значения ($u_{1,1}, u_{2,1}, ..., u_{n_x,1}, u_{1,2}, u_{2,2}, ..., u_{n_x-1,n_y}, u_{n_x,n_y}$) в ячейках сетки в формате с плавающей запятой с семью знаками мантиссы.

Пример:

Входной файл	mpi.out
1 1 3 3 mpi.out 1e-10 1.0 1.0 7.0 7.0 7.0 7.0 0.0	7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00 7.000000e+00
1 2 3 3 mpi.out 1e-10 1.0 2.0 7.0 0.0 5.0 0.0 5.0	5.402480e+00 4.260734e+00 2.841504e+00 5.349185e+00 3.798951e+00 2.105283e+00 5.195309e+00 3.480605e+00 1.780675e+00 4.951445e+00 3.147484e+00 1.536811e+00 4.462988e+00 2.621075e+00 1.219086e+00 3.279432e+00 1.654741e+00 7.184566e-01

В ЛР №7 выполнить вариант №6.

В ЛР №8 выполнить вариант №2.

В ЛР №9 выполнить вариант №1.