Санкт-Петербургский Политехнический Университет Петра Великого

Институт прикладной математики и механики

Кафедра Телематики при ЦНИИ РТК

Курсовая работа

по дисциплине «Параллельные вычисления и алгоритмы»

на тему «Параллельное программирование при решении СЛАУ общего вида»

Преподаватель Лукашин А.А.

Студент гр.13643/2 Крутских А.О.

Санкт-Петербург

2019 г.

Оглавление

[1. Алгоритм решения задачи 2](#_Toc12099830)

[2. Принцип распараллеливания задачи 3](#_Toc12099831)

[3. Важные моменты реализации распараллеливания на pthreads 3](#_Toc12099832)

[4. Важные моменты реализации распараллеливания на MPI 3](#_Toc12099833)

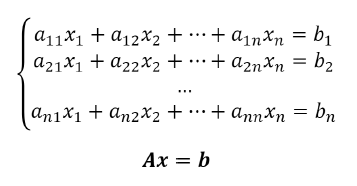
[5. Важные моменты реализации распараллеливания на OpenMP 3](#_Toc12099834)

[6. Требуемое окружение на СКЦ и методика запуска задач 3](#_Toc12099835)

[7. Исследование эффекта от распараллеливания 3](#_Toc12099836)

**1. Алгоритм решения задачи**

Дана система линейных алгебраических уравнений:



Требуется найти решение – неизвестные X1, X2, …, Xn.

Данная задача имеет единственное решение, когда количество уравнений больше или равно количеству переменных. Наиболее простым для ее решения является метод Гаусса.

Метод Гаусса решения системы линейных уравнений включает в себя 2 стадии:

* последовательное (прямое) исключение;
* обратная подстановка.

Последовательное исключение

Исключения Гаусса основаны на идее последовательного исключения переменных по одной до тех пор, пока не останется только одно уравнение с одной переменной в левой части. Затем это уравнение решается относительно единственной переменной. Таким образом, систему уравнений приводят к треугольной (ступенчатой) форме. Для этого среди элементов первого столбца матрицы выбирают ненулевой (а чаще максимальный) элемент и перемещают его на крайнее верхнее положение перестановкой строк. Затем нормируют все уравнения, разделив его на коэффициент ai1, где i– номер столбца.

Затем вычитают получившуюся после перестановки первую строку из остальных строк. Получают новую систему уравнений, в которой заменены соответствующие коэффициенты. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают указанный процесс для всех последующих уравнений пока не останется уравнение с одной неизвестной.

Обратная подстановка

Обратная подстановка предполагает подстановку полученного на предыдущем шаге значения переменной Хn в предыдущие уравнения.

**2. Принцип распараллеливания задачи**

В основе распараллеливания будет распределение строк матрицы (уравнений) по исполняющим потокам/процессам. Таким образом каждый поток/процесс обрабатывает n/p строк, где n – общее количество строк, p – количество используемых потоков/процессов.

Описание алгоритма

1. Обрабатывается строка i
2. Все потоки/процессы, больше чем i, вычитают ее из своей строки, так чтобы коэффициент i стал нулевым.
3. Все потоки/процессы присваивают своему значению Х значение измененной переменной b.
4. Последняя строка передает наверх свое значение Хn, после чего значение уточняется делением на коэффициент An.
5. Все остальные строки отнимают это значение от своего Xi.
6. Цепочка продолжается до 1 строки.

**3. Важные моменты реализации распараллеливания на pthreads**

Суть метода в разделение основного потока на несколько в определенном месте программы с целью ускорения его обработки. При реализации параллельной программы на pthreds создается функция, которая и содержит участок кода, выбранный и подходящий для распараллеливания.

double \* thread\_func(void \*arg)

{…}

В самой программе объявляется thread:

pthread\_t thread;

После, в необходимом месте программы тред создается, передавая управление функции и принимая аргументы.

result = pthread\_create(&thread, NULL, &thread\_func, &data);

result – отличный от нуля сигнализирует об ошибке.

Тред завершается.

result = pthread\_join(thread, NULL);

**4. Важные моменты реализации распараллеливания на MPI**

MPI в отличие от других методов работает не с потоками, а с процессами, что позволяет использовать его, например, в python, так как последний не работает с потоками.

Суть метода заключается, в создании нескольких процессов, которые все время выполнения программы работают одновременно.

MPI принимает в качестве аргумента количество процессов. Rank и Size определяют номер процесса и их общее количество, соответственно.

MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &commsize);

Эти процессы могут обмениваться данными между собой. В данном случае использована общая рассылка:

MPI\_Bcast(&a[row \* (n+1)], n+1, MPI\_DOUBLE, rank, MPI\_COMM\_WORLD);

где 1 атрибут - адрес начала расположения в памяти рассылаемых данных, второй – число передаваемых элементов, 3 – тип посылаемых элементов, 4 – номер процесса-отправителя, 5 – «вселенная».

По завершения программы, завершается и работа MPI

MPI\_Finalize();

Для измерения времени работы программы в данном случае использовалась функция t = MPI\_Wtime() - t;

**5. Важные моменты реализации распараллеливания на OpenMP**

OpenMP, на мой взгляд, является самым простым из исследуемых в работе методов.

Для его реализации необходимо выделить участок кода, подходящий для распараллеливания с переменными не зависимыми друг от друга.

В моей работе основную вычислительную сложность представляет трижды вложенный цикл прямого распределения. Он подходит для этого метода, так как нет разницы в каком порядке строки вычитаются друг из друга.

Реализация метода

#pragma omp parallel //num\_threads(5)

{

# pragma omp for

……………

}

Функция num\_threads(5) позволяет в ручную управлять количеством потоков.

**6. Требуемое окружение на СКЦ и методика запуска задач**

Подключение к СКЦ

Для резервирования узла используется команда salloc -N 1 -p tornado.

Подключение к кластеру осуществляется командой ssh №узла, который можно посмотреть командой squeue.

Компиляция программ требует подгрузки окружения. Это делается командой module load «путь к необходимому модулю». Ниже описаны необходимые пакеты для реализации различных методов:

C + Pthread:

Module: mpi/openmpi/3.0.0/gcc/7.2.0

Компиляция: gcc -l pthread \*.c

C + MPI:

Module: mpi/openmpi/3.0.0/gcc/7.2.0

Компиляция: mpicc \*.c

Запуск: mpirun -np \* ./a.out, где \* - число потоков

C + openmp:

Module: mpi/openmpi/3.0.0/gcc/7.2.0

Компиляция: gcc -fopenmp openmp\_mult.c

Python + MPI:

Установка окружения для python не столь тривиальна. Пункты 1-5 выполняйте не на кластере:

1. module load mpi/openmpi/3.0.0/gcc/7.2.0

2. module load python/3.6.5

3. python3 -m venv test

4. source test/bin/activate

5. pip install «необходимые пакеты»

6. ssh n01p034

7. Выполнить пункты 1,2,4 на узле.

Запуск: mpiexec python3 \*.py

**7. Исследование эффекта от распараллеливания**

Для исследования эффекта засечено время выполнения последовательной программы. Испытания проводились с размерностью матрицы 1000х1000, для получения минимальной наглядности.

Время работы последовательной программы: 3,4 c.

Далее на графиках представлено изменения вмени исполнения программы от числа процессов/потоков. Время работ различных методов не корректно сравнивать между собой, так как оно получено разными способами.

MPI на языке С (рис. 7.1). Максимальная скорость выполнения при 28 процессах 0,2 с.

Рис. 7.1 MPI на языке С

OpenMP на языке С (рис. 7.2). Максимальная скорость выполнения при 28 процессах 0,3 с.

Рис 7.2 OpenMP на языке С

Вывод: Время исполнения программ при использовании методов параллельного программирования значительно уменьшается в зависимости от метода и количества используемых потоков/процессов в 10 и более раз.

При выборе метода в будущих работах важным фактором для меня станет удобство применения и по возможности я отдам предпочтение openMP.