Министерство образования и науки Российской Федерации Федеральное агентство по образованию Федеральное государственное бюджетное образовательное учреждение высшего образования «Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Лабораторная работа №3

по курсу «Параллельное программирование»

Выполнил студент группы ИВТ-31\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Птахова А.М/

Проверил доцент кафедры ЭВМ \_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Долженкова М.Л./

Киров 2023

**1. Цель**

Знакомство со стандартом OpenMP, получение навыков реализации многопоточных SPMD-приложений с применением библиотеки OpenMP.

**2. Задание**

- Выполнить разбиение исследованного в ходе первой лабораторной работы алгоритма на независимо выполняемые фрагменты;

- Реализовать многопоточную версию алгоритм с помощью языка С++ и библиотеки OpenMP, используя при этом необходимые примитивы синхронизации;

- Показать корректность полученной реализации путем осуществления тестирования на построенном в ходе первой лабораторной работы наборе тестов;

- Провести доказательную оценку эффективности многопоточной реализации алгоритма.

**3. Алгоритм**

3.1. Описание алгоритма

Алгоритм выполнения LU – разложения матрицы следующий:

1. Формирование верхней и нижней матрицы выделить в отдельные блоки при помощи фигурных скобок

2. Для каждого блока написать директиву parallel, необходимую для создания потоков

3. Перед циклом for, отвечающим за вычисление новых элементов строки или столбца для матриц U и L соответственно, написать директиву for schedule (guided, 5), способную распараллелить цикл for.

3.2. Листинг программы

void LU\_Decomposition(vector <vector<double>>& a, long n)

{

double time = 0;

for (long i = 0; i < n; i++)

{

#pragma omp parallel

{

#pragma omp for schedule(guided, 5)

//upper triangular

for (long k = i; k < n; k++)

{

double sum = 0;

for (long j = 0; j < i; j++)

{

sum += (a[i][j] \* a[j][k]);

}

a[i][k] = a[i][k] - sum;

}

}

#pragma omp parallel

{

#pragma omp for schedule(guided, 5)

//lower tiangular

for (long k = i; k < n; k++)

{

if (i != k)

{

int sum = 0;

for (long j = 0; j < i; j++)

{

sum += (a[k][j] \* a[j][i]);

}

a[k][i] = (a[k][i] - sum) / a[i][i];

}

}

}

}

}

5. Сравнительные результаты

Результаты сравнения простого и многопоточного алгоритма приведены в таблице 1.

Таблица 1 – Сравнительные результаты

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Размерность | Послед. | Потоки | OpenMP |
| 1 | 10 | time: 0.019 ms | time: 9.851 ms | time: 7.197 ms |
| 2 | 50 | time: 2.04 ms | time: 18.373 ms | time: 8.262 ms |
| 3 | 100 | time: 15.798 ms | time: 37.944 ms | time: 10.88 ms |
| 4 | 500 | time: 1890.83 ms | time: 577.73 ms | time: 277.44 ms |
| 5 | 1000 | time: 15242 ms | time: 3399.6 ms | time: 2262.8 ms |
| 6 | 2000 | time: 148965 ms | time: 31956.3 ms | time: 19494.4 ms |
| 7 | 5000 | time: 1.93568e+06 ms | time: 412307 ms | time: 281279 ms |
| 8 | 7500 | time: 6.65623e+06 ms | time: 1.55738e+06 ms | time: 1.16805e+06 ms |
| 9 | 13000 | time: 2.31827e+07 ms | time: 4.9538e+06 ms | time: 4.27352e+06 ms |
| 10 | 15000 | time: 1.414706e+08 ms | time: 3.15265e+07 ms | time: 2.81277e+07 ms |

**6. Доказательство эффективности алгоритма**

6.1 Эффективность расписания

При использовании директивы parallel for без параметров произойдет распараллеливании программы, но оно будет не эффективным. Это связано с неодинаковой загрузкой потоков. То есть пока одни потоки работают, другие простаивают. Для решения этой проблемы возможно использование расписания, позволяющая управлять загруженностью потоков.

6.2. Эффективность параметров расписания

У расписания возможно задавать параметры, указывающие правила распределения нагрузки между потоками.

6.2.1 Static

Статический метод распределения подразумевает выделению каждому потоку одинаковое кол-во задач. Можно предположить, что данное распределение эффективно для задач, имеющих одинаковое время выполнения.

При выполнении алгоритма Краута, вычисление каждого элемента матрицы будет занимать разное время. На рисунке 2 показано время вычисления каждого элемента матрицы U и L для исходной матрицы 3х3.

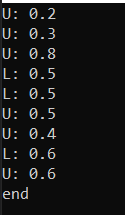


Рисунок 2 – Время вычисления элемента матрицы

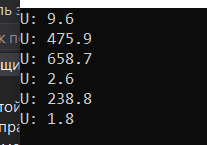
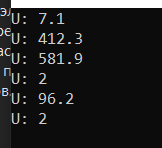
Так как время вычисления каждого элемента матрицы различно, следовательно, можно предположить, что время вычисление строк так же будет различным. Из этого предположения делается вывод о неэффективности использования параметра static.

6.2.2. Dynamic и guided

Особенность dynamic заключается в том, что распределение задач происходит в зависимости от свободности потока. Следовательно, такое распределение хорошо использовать для несбалансированных задача.

Что касается параметра guided, то он, как и dynamic делает распределение задач, отличием является то, что блоки делятся не от кол-ва, а от не пройденных итераций.

На рисунке 3а и 3б показано время вычисления каждого элемента матрицы U при использовании параметра dynamic и guided соответственно.

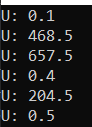
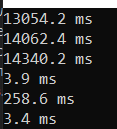
3а 3б

Рисунок 3а – dynamic, 3б – guided

Из полученных результатов можно сделать вывод об эффективности использования параметра guided

6.3. Затраты

При работе с потоками необходимо учитывать временные затраты на создание потоков, их управление и их синхронизацию, при наличии последней. На рисунке 4а представлено время выполнения при последовательном алгоритме, 4б представлено время выполнения для параллельного.

4а 4б

Рисунок 4а – последовательный алгоритм; 4б – параллельный алгоритм

Первоначально, при запуске подпрограммы, происходит создание потоков, за счет чего происходит увеличение времени на выполнение операции. В последующем, потоки не создаются, и время затрачивается только на вычисление элементов, а так вычисление происходит параллельно, то это позволяет уменьшить время выполнения. Но для того, чтобы был выигрыш с последовательном алгоритмом необходимо перекрыть расходы, возникшие при первоначальном создании потоков. Это возможно при размерности матрицы близкой к 100.

Посчитать?

7. Вывод

В ходе выполнения лабораторной работы был реализован алгоритм LU – разложения матрицы при помощи библиотеки openMP. Для достижения наибольшей загрузки потоков было использовано расписание с параметром guided. Для оценки эффективности результаты времени выполнения подпрограммы были помещены в таблицу. Сравнивая данные с уже имеющимся, можно заметить несколько моментов:

1) алгоритм проигрывает последовательному на размерности матрицы меньше 100. Это можно объяснить дополнительными временными затратами на создание потоков и их управлением.

2) алгоритм с иcпользованием openMP работает быстрее, чем алгоритм с использованием потоков. Можно предположить, что это связано с загрузкой потоков: в потоках операционной системы не было управления распределения нагрузки, в то время, как в программе с openMP это было реализовано. И в итоге получается, что простоев нет, а это в значительной степени снижает время выполнения подпрограммы.