*Государственное образовательное учреждение высшего профессионального образования*

*«Московский государственный технический университет имени Н. Э. Баумана»*

ОТЧЕТ

Домашнее задание

по дисциплине «Программирование параллельных процессов»

Вариант 8

|  |  |
| --- | --- |
| Студент: | Лавренова Е.А. |
| Группа: | ИУ7-31м |
| Преподаватель: | Ковтушенко А.П. |

Москва, 2018

**Теоретическая часть**

1. Постановка задачи.

Разработать программу для решения задачи коммивояжера. Построить на заданном графе оптимальный путь, проходящий через все вершины по одному разу и приходящий в исходную вершину. Исходные данные: массив вершин, массив ребер с приписанными весами. Оптимальный путь имеет оптимальный суммарный вес включенных в него рёбер.

Обосновать проектное решение (выбор алгоритма). Обеспечить равномерную загрузку процессоров. Результат вывести в текстовый файл. Исследовать зависимость времени счета от размерности задачи и количества процессоров.

1. Задача коммивояжера.

Задача коммивояжера является одной из самых известных задач комбинаторной оптимизации. Эта задача состоит в поиске наилучшего (выгодного, дешевого, краткого и т.п.) маршрута, проходящего через всех указанные города хотя бы по одному разу (чаще всего - с точностью по одному разу) с возвращением в начальный город.

Данная задача относится к классу **NP-трудных задач** и решение переборными методами (вычисление длин всех возможных маршрутов и их сравнение) практически невозможно (для nn городов существует (n−1)! /2(n−1)! /2 маршрутов). Например, если взять 10 городов - найдем 181440 путей, а для 20 - уже 6,1⋅10166,1⋅1016.

На практике чаще всего изучают **метод ветвей и границ** (и его модификации, в том числе алгоритм Литтла), который позволяет на каждом шаге отбрасывать целую группу неоптимальных маршрутов.

1. Метод ветвей и границ

Задачи дискретной оптимизации имеют конечное множество допустимых решений, которые теоретически можно перебрать и выбрать наилучшее (дающее минимум или максимум целевой функции). Практически же зачастую это бывает неосуществимо даже для задач небольшой размерности.

В методах неявного перебора делается попытка так организовать перебор, используя свойства рассматриваемой задачи, чтобы отбросить часть допустимых решений. Наибольшее распространение среди схем неявного перебора получил метод ветвей и границ, в основе которого лежит идея последовательного разбиения множества допустимых решений. На каждом шаге метода элементы разбиения (подмножества) подвергаются анализу – содержит ли данное подмножество оптимальное решение или нет. Если рассматривается задача на минимум, то проверка осуществляется путем сравнения нижней оценки значения целевой функции на данном подмножестве с верхней оценкой функционала. В качестве оценки сверху используется значение целевой функции на некотором допустимом решении. Допустимое решение, дающее наименьшую верхнюю оценку, называют рекордом. Если оценка снизу целевой функции на данном подмножестве не меньше оценки сверху, то рассматриваемое подмножество не содержит решения лучше рекорда и может быть отброшено. Если значение целевой функции на очередном решении меньше рекордного, то происходит смена рекорда. Будем говорить, что подмножество решений просмотрено, если установлено, что оно не содержит решения лучше рекорда.

Если просмотрены все элементы разбиения, алгоритм завершает работу, а текущий рекорд является оптимальным решением. В противном случае среди нерассмотренных элементов разбиения выбирается множество, являющееся в определенном смысле перспективным. Оно подвергается разбиению (ветвлению). Новые подмножества анализируются по описанной выше схеме. Процесс продолжается до тех пор, пока не будут просмотрены все элементы разбиения.

**Практическая часть**

1. Модели для параллельных вычислений

* Модель OpenMP – в рамках данной модели для решения задачи между потоками существует общая очередь, содержащая частично развернутые ветви. Различные потоки продолжают удалять их из очереди, оценивая, могут ли они привести к оптимальным решениям или нет, и если да, расширяя их в дополнительные подзадачи и добавляя их в очередь.
* Модель MPI - имеется главный узел, который поддерживает информацию обо всех других (ведомых) процессорах и отслеживает, какие из них простаивают. Ведомые процессоры обрабатывают подзадачи; и когда им нужно создать новые ведомые процессоры, они главный узел о выделении новых процессоров. Главный узел также периодически обновляет у ведомых узлов информацию о лучшем решении на текущий момент.
* Гибридная модель – данная модель сочетает в себе преимущества обеих вышеупомянутых стратегий. На каждом подчиненном узле параллельно выполняются различные потоки (на нескольких ядрах этого узла). Узел имеет общую очередь среди потоков, и потоки работают параллельно с помощью openMP. Как и в случае модели MPI, ведущий узел отслеживает простаивающие ведомые узлы и позволяет другим ведомым узлам создавать их по мере необходимости.

1. Алгоритм распределения столбцов

Как можно увидеть, пересчет очередного столбца матрицы зависит лишь от основного столбца, но не от значений иных столбцов, т.е. обновление неосновных столбцов не зависит от других неосновных столбцов. Следовательно, логично определить для очередного процессора некоторый набор вычисляемых столбцов/подматриц. В связи с чем была реализована передача каждому процессору k последовательных столбцов. Данный подход обеспечивает равномерную загрузку процессоров.

1. Результаты исследования

Для проверки работы программы было подсчитано время в секундах, требующееся для решения задачи размером 5, 10 и 15 ребер на 1, 2, 4 и 8 узлах, а также две метрики: отношение времени работы на нескольких узлах к времени работы на одном узле и отношение времени работы к числу узлов. Результаты этих исследований представлены таблицами 1, 2 и 3, соответственно.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число ребер / число процессов | 1 | 2 | 4 | 8 |
| 4 | 0,007 | 8,83683 | 7,43283 | 5,72831 |
| 8 | 0,786 | 58,38534 | 30,49550 | 22,97714 |
| 10 | 57,189 | 485,40879 | 431,52489 | 242,75265 |

Таблица 1. Зависимость времени работы программы от числа процессоров в секундах.

|  |  |  |  |
| --- | --- | --- | --- |
| Число ребер/ число процессов | 2 | 4 | 8 |
| 5 | 1,3118 | 1,5596 | 2,02 |
| 10 | 2,1927 | 4,1922 | 5,5118 |
| 15 | 1,5810 | 1,7784 | 3,1614 |

Таблица 2. Отношение времени выполнения программы на одном узле к времени выполнения на нескольких узлах.

График 1. Зависимость времени работы программы от числа процессоров в секундах.

Также для ускорения работы была добавлена опция OpenMP #pragma parallel for для арифметических операций со столбцами матрицы. Получившиеся результаты представлены таблицей 3. Эксперимент проводился для 4 потоков.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер матрицы/ число процессов | 1 | 2 | 4 | 8 |
| 1000 | 28,33725 | 27,98104 | 27,03618 | 25,88276 |
| 2000 | 40,21271 | 39,54756 | 38,35069 | 37,26128 |

Таблица 3. Время работы программы (в секундах) с использованием OpenMP.

1. Выводы

В результате проведенной работы был реализован параллельный алгоритм симплекс-метода. Использованный алгоритм распределения столбцов позволил добиться равномерной загрузки всех процессоров и ускорения программы более чем в 4 разa. Добавление OpenMP-директив pragma также ускорило работу программы более чем в 4 раз при тех же параметрах на 8 потоках.