Отчёт

по Лабораторной работе:

«Параллельный Алгоритм Флойда на Fortran с OpenMP»

Преподаватель: Кулябов Дмитрий Сергеевич

Выполнил: Кремер Илья

Группа: НК-401

Москва, РУДН – 2011

Оглавление

Постановка задачи

Алгоритм Флойда (очень часто называют Флойда – Уоршелла) – один из нескольких алгоритмов, высчитывающих кратчайшие расстояния между всеми вершинами взвешенного ориентированного графа на ряду с алгоритмами Дейкстры, Джонсона и Данцига.

Требуется написать программу на языке Fortran, демонстрирующую работу указанного алгоритма, применив к ней распараллеливание с помощью технологии OpenMP. Для выполнения задания воспользуемся прикреплёнными к нему материалами – презентацией проф. Гергеля из НГУ «Высокопроизводительные вычисления для многопроцессорных многоядерных систем».

В данной работе графы будут представляться с помощью матрицы смежности. Вместо бесконечности будем использовать отрицательный вес: `(-1.0)`, все элементы `i = j` будут иметь нулевой вес. Напомним, что на месте пересечения `i`-ой строки с `j`-ым столбцом стоит число, отвечающее за вес ребра между вершинами `i` и `j`[1].

Решение

Рассмотрим псевдокод алгоритма:

```
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            W[i][j] = min(W[i][j], W[i][k] + W[k][j])
```

Мы перезаполняем ту же матрицу, в которой содержится информация об оригинальном графе (`W`). По прохождению цикла по `k`, в этой матрице будет содержаться информация о минимальном пути из `i`-ой вершины в `j`-ую – пересечение строчки и столбца будет говорить о то, какой минимальный вес придётся преодолеть, проходя из одной вершины в другую.

Очевидно, что непосредственный переход итераций из одной в другую и обращение к элементам заняли бы едва различимое время даже при триллионных `n`, из чего следует вывод, что единственная операция, требующая ресурсов процессора – это нахождение минимума (выполнение функции `min`). *Однако данная операция является достаточно простой и её распараллеливание не приведёт к заметному ускорению работы алгоритма.*

[1] Более подробно теория изложена в прикреплённой к лабораторной материале.

Более эффективное распараллеливание – это одновременное выполнение нескольких операций обновления значений матрицы[2].

В прикреплённом материале доказывается, что матрица `W` может быть общей для всех процессов, т.е. что гонки данных гарантированно не возникнет[3]. Далее разбираются причины, по которым следует представить матрицу, изображающую граф в виде массива, разложив её по столбцам, либо по строкам[4].

Программная реализация

На страницах 17-20 презентации представлены коды на C++ (явное использование C++ выдаёт использования ссылок наряду с указателями), которые, исходя из задания лабораторной работы, основной работой (помимо разбора мат. теории и технологий программирования) будет перевод этих кодов на Fortran. Напишем программку на основе этих кодов с тем, чтобы убедиться в правильности алгоритма.

Для начала добавим недостающую функцию заполняющую массив случайным образом (напомним, матрица представлена в виде массива) – `RandomDataInitialization(double *&pMatrix, int Size)`:

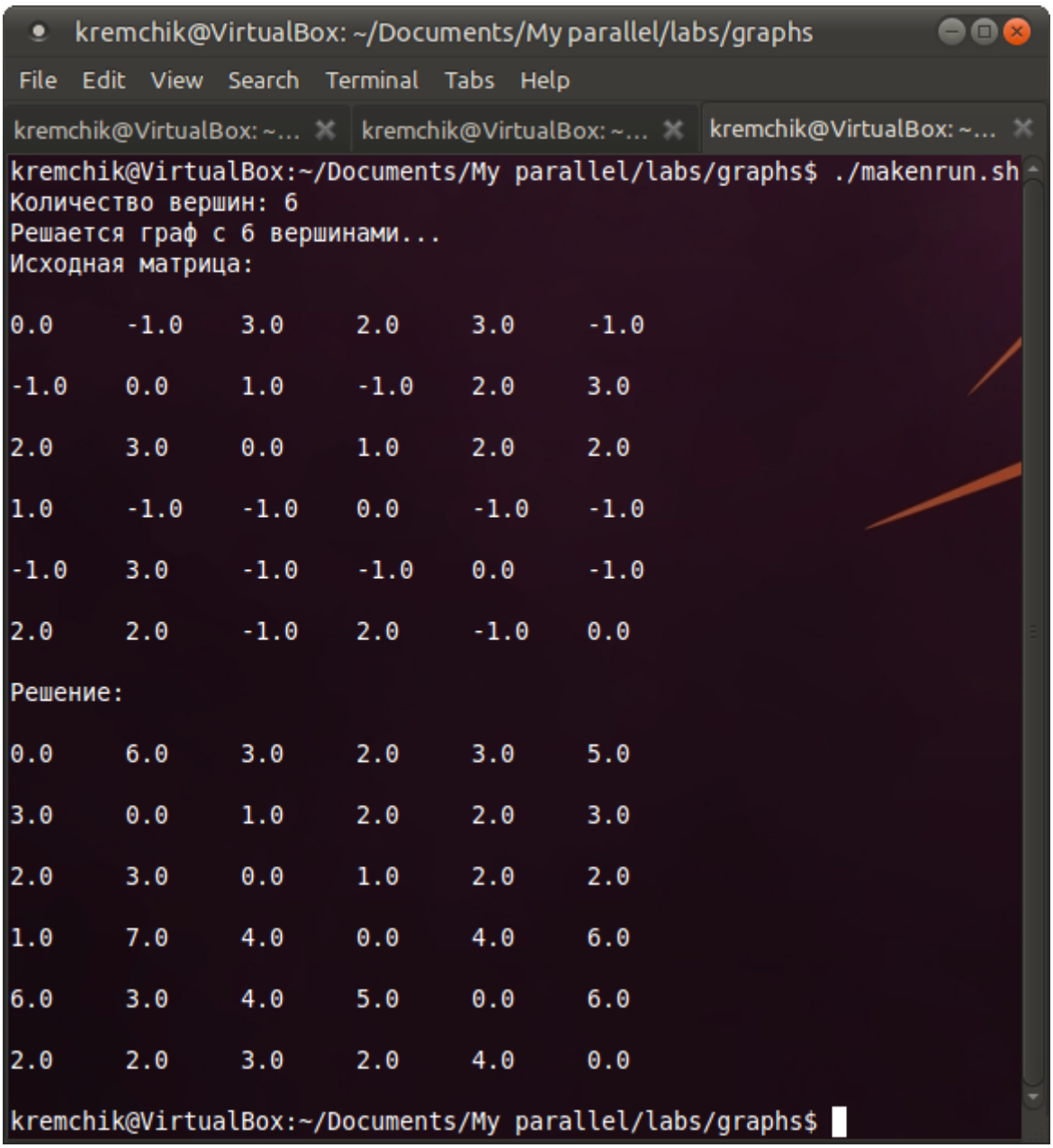
```
void RandomDataInitialization(double *&pMatrix, int Size) {
    srand(time(NULL)); // избавляемся от "сценарного" рандома
    for (int i = 0; i < Size * Size; i++) {
        double random = rand() % 10 - 1;
        if ((i) % (Size + 1) == 0)
            pMatrix[i] = 0;
        else if (random == 0)
            pMatrix[i] = random - 1;
        else pMatrix[i] = random;
    }
}
```

Первое условие служит для того, чтобы элементы по диагонали были нулями. В случае, если случайное число оказалось нулём, то для всех недиагональных элементов меняем его на `-1` для удобочитаемости матрицы. Во всех остальных случаях оставляем выпавшее число (от `1` до `8`).

А также функцию распечатки матрицы.

---

[2] 11-ая страница презентации Гергеля (прикреплённый материал)
[3] 12-ая страница, там же
[4] Страницы 13-16, там же



Коды этой программы приложены к лабораторной работе.

Перевод данного кода на Fortran в данном случае не является сложной задачей, однако перевод функции `ParallelFloyd` в отдельности не является тривиальным. Проблему вызывает различное исчисление индексов массивов: с нуля в Си и с единицы в Фортране. Решение этой трудности следующее:

`W[i * Size + k]` превращается в `W((i-1) * Size + (k-1) + 1)`

Все остальные строки преобразуются точно так же.

Эффективность

В приложенном материале представлены графики результатов распараллеливания, полученных при запуске программы на компьютере с современным мощным 4-ядерном процессором Intel Xeon E5320, с установленной ОС Windows HPC Server 2008. Для компиляции использовался компилятор Intel. Запуск на значительно менее мощном процессоре Intel i7 860 при 2 процессах даёт похожие результаты, однако на 4 процессах сильно уступает старшему брату: матрицу 600x600 i7 обрабатывает за прибл. 4,5 секунды, в то время как по графику из презентации видно, что Xeon проделывает эту работу примерно за 2,8 сек.

Как и во второй части лабораторной работы, «Решето Эратосфена», эксперименты показали преимущество выбора автоматического распределения `schedule(runtime)` (в оригинальных кодах презентации `schedule` опущен). Так, при 8 процессах (логический максимум для 4-ядерных процессоров) i7 обсчитывает граф из 400 вершин без указания способа распределения в среднем за 1,2 секунды, а с указанием `schedule(runtime)` – за 1,1.

ОС и инструменты компиляции этой программы те же, что и в работе с простыми числами: Windows 7; gfortran из MinGW.

Прикреплённая к работе программа на Fortran, как и программа «Решето Эратосфена» понимает аргумент командной строки "`-q`", который запрещает вывод матриц на экран. Его удобно использовать в случае запуска программы только с целью замера времени.