Умножение матриц с использованием OpenMP

Вариант 28

Галицкий К.В.

4 курс 2 группа

**Введение**

Необходимо реализовать программу блочного умножения матриц, первая матрица симметричная, вторая нижнее-треугольная, обе они хранятся в виде ономерного массива (в вытянутом виде). Программа будет реализована в трех исполнениях:  
**Последовательная  
Паралельная "Блоки"  
Паралельная "Вычисл Ядра"**

После чего проведём сравнение, обсуждение и построим диаграммы наглядно дающие понять о результатах.

**Реализация программы**

Реализация будет проводиться на языке программирования C++ версии v141, в среде разработки Visual Studio 17 версии 15.9, на операционной системе Windows 10 версии 1803. Вставлять сюда скриншоты программы не совсем корректно, так как программа объемная, и они будут не информативны, в общем программа будет во вложениях.  
Но обсудим основные тезисы этой программы. Для начала хочется сказать, что будем дробить матрицы на блоки, что бы в дальнейшем можно было проще распараллелить, а так же для облегчения проверки программы на верность (отладки). И так, в программе реализована система которая подбирает размер блока относительно величины матрицы, единственное, что нам необходимо указать, это минимальный размер блока, от которого он будет расширяться. После определения оптимального размера блока для данной матрицы, мы дробим матрицу на блоки. И приступаем к перемножению. В дальнейшем эти блоки помогут нас с добавлением OpenMP. В данной программе, так как тестировалась она на одном моём ПК, было подобранно оптимальное количество потоков, для обработки всех типов матриц, которые участвовали в тестах.

**Тесты**

Тесты проводились на машине:  
**Процессор: Intel core i7 2600k 4700 МГц  
 Кеш 1-го уровня 64 Кб  
 Кеш 2-го уровня 1024 Кб  
 Кеш 3-го уровня 5 Мб  
 4 Ядра 8 Потока  
ОЗУ: 12 Гигабайта 1333 МГц  
Устройство памяти: HDD накопитель 4 Терабайта**

Тесты выполнялись на матрицах 5 размеров, с рандомными элементами, в замер времени входила только процедура умножения матриц, потому будем считать эти значения эталонными.

|  |  |  |  |
| --- | --- | --- | --- |
| Размер матрицы | Последовательная | Паралельная "Блоки" | Паралельная "Вычисл Ядра" |
| 10х10 | 0,000284 | 0,00142 | 0,0042 |
| 100х100 | 0,002413 | 0,0003737 | 0,0001202 |
| 500х500 | 0,213296 | 0,187546 | 0,098513 |
| 1000х1000 | 1,2456 | 0,98543 | 0,7856 |
| 50000х50000 | 57,48324 | 48,564423 | 34,2431 |
|  |  |  |  |

Пришлось разбить на две диаграммы, так как на маленьких матрицах результатов было вовсе не видно, далее приведены две матрицы 10х10 и 100х100, на диаграмме(чем меньше тем лучше), и так можем заметить, что на маленькой матрице 10х10 последовательная программа работает быстрее, но что мы заметим на 100х100, абсолютно противоположные показания, это объясняется тем, что на малых объёмах, программа больше тратит время на создание потоков, чем на вычисление, следственно параллелить данные объёмы глупо. Во втором же случае увеличив объём данных в 10 раз видим, что потоки делят работу равномерно и справляются в 4 раза быстрее, чем последовательная, теперь разберём с блоками и вычислительными ядрами, когда мы используем блоки рано или поздно мы встанем в очередь вычисления, так как не все вычисляются одинаково быстро, во втором же случае мы не создаем эту очередь, а наоборот параллельно раскидываем её.

Это очень заметно на огромных объемах данных, когда параллельные аналоги работают в два раза быстрее,

**Вывод**

Каждый метод будет отлично работать в той ситуации которая подходит ему больше всего, не всегда потоки помогают улучшить работу программы сделав её быстрее, зачастую, как раз наоборот, потому, к созданию параллельного аналога программы надо подходить с полным осознанием логики и работы программы…