Задание 1

Реализуйте параллельный алгоритм умножения матриц, использующий не менее двух потоков.

На вход подаются матрицы произвольного размера (не обязательно квадратные). Полученная в результате умножения матрица выводится на печать.

Оценка задания

1. Умножение произвольных матриц с фиксированным числом потоков (два потока, число потоков всегда равно числу строк матрицы и т.п.) - 70 %.
2. Число потоков может быть произвольным, задаётся в начале работы программы - 30 %.

Рекомендованный порядок выполнения задания

Реализуйте обычный (однопоточный) алгоритм умножения матриц. Если знакомы с библиотекой STL, используйте vector<vector<double>> для представления матрицы. Проверьте корректность работы алгоритма.

Обычный алгоритм умножения матриц содержит три вложенных цикла. Естественный способ распараллелить его – разделить внешний цикл на несколько частей, и выполнять каждую часть в отдельном потоке. Сначала реализуйте один из простых вариантов разделения:

a) можно разбить цикл на две части и выполнять в двух потоках: первый будет считать первую половину строк, второй – вторую (если число строк матрицы нечётное, один поток будет считать на одну строку больше, это не критично);
b) можно сделать по одному потоку на каждую итерацию внешнего цикла, чтобы каждый поток вычислял одну строку результирующей матрицы.

Для того чтобы работать, каждый поток должен иметь доступ к входным матрицам, выходной матрице и номеру потока (по которому он будет определять, какую часть работы он должен сделать). Организовать передачу информации в потоки можно двумя способами:

1) простой но «не красивый»: объявить все матрицы (вход и выход) глобальными переменными, тогда потоки будут иметь к ним прямой доступ без дополнительных действий; номер потока передать через аргумент потока, выполнив преобразование в указатель и обратно (как было показано на лекции, смотрите пример http://prog.tversu.ru/os/thread_example.cpp);
2) более сложный и более корректный:
    i) объявить структуру, содержащую необходимые поля (указатели или ссылки на матрицы, номер потока или номер/диапазон строк, который нужно обработать) – задание на работу для конкретного потока;
    ii) перед запуском потока создать и заполнить для каждого из них свою отдельную структуру, указатель на которую передать функции _beginthreadex;
    iii) в потоке привести указатель к типу указателя на структуру и использовать её поля.

Обратите внимание, что делать для каждого потока отдельную копию входных и выходной матриц не имеет смысла (это приведёт к напрасному перерасходу памяти), достаточно передать указатели или ссылки на единственные экземпляры этих матриц, созданных в функции main. А вот структуры с заданием для каждого потока (вариант 2) должны быть свои – они отличаются номером потока, и если использовать одну структуру на всех (меняя номер потока), то вполне

возможна ситуация, что поток не успеет прочитать из структуры свой номер, перед тем, как его заменят следующим. В таком случае вполне возможно ожидать, что часть работы будет сделана несколько раз, а часть – не сделана вообще.

Реализуйте один из способов разделения, для этого:

1) вынесите код двух внутренних циклов умножения матрицы в отдельную функцию, необходимые данные передайте как её параметры – это функция вычисления одной строки;
2) создайте функцию потока, в этой функции реализуйте половину внешнего цикла, вызывая функцию вычисления строки (если делаете вариант a) или просто вызовите функцию вычисления для одной строки (вариант b);
3) обеспечьте передачу необходимой информации в функцию потока;
4) в main() создайте два потока (вариант a) или по одному потоку на каждую строку (вариант b).

Проверьте, что программа работает корректно.

Полученная таким образом программа решает поставленную задачу в минимальном виде. Однако оба варианта не являются эффективными: вариант a использует не все ресурсы процессора (если у процессора больше двух ядер), а вариант b может напротив, создать слишком много потоков.

Слишком много потоков это тоже плохо. Во-первых, если у процессора меньше ядер, чем потоков, то потоки не будут выполняться одновременно, и никак не могут дать ускорения программы. Во-вторых, обычные (не графические) процессоры и операционные системы не рассчитаны на создание огромного числа потоков, и, если сделать несколько тысяч потоков, то накладные расходы на их создание и переключение скорее всего сделают программу гораздо более медленной, чем даже однопоточный вариант.

Надо заметить, что организация кэш-памяти многоядерных процессоров устроена так, что для обеспечения максимальной производительности желательно, чтобы одно ядро записывало значения в ячейки памяти, расположенные близко к друг другу, а разные ядра – в удалённые друг от друга. Так как матрицы обычно хранят в памяти построчно[1], то это требование приводит к выводу, что желательно, чтобы с одну строку результирующей матрицы вычислял один поток. Конечно, всё зависит от размеров матрицы – если строки очень большие, то, скорее всего, не будет никакой беды, если строка будет разбита, например, пополам, и обработана в разных потоках. Чтобы понять, что будет происходить в конкретной ситуации, надо как минимум знать размеры кэш-линий конкретного процессора, учесть размеры матрицы и её расположение в памяти. Лучше всего провести эксперименты, чтобы понять, какой вариант будет наиболее эффективным. Учтите, что результаты такого эксперимента будут разными на разных моделях процессоров. Из общих соображений понятно, что варианты, когда последовательно расположенные значения вычисляются в разных потоках, будут не эффективными. Вариант «одна строка матрицы в одном потоке» - скорее всего, будет работать быстро (кроме, может быть случая, когда строка очень длинная, а строк очень мало – меньше чем ядер).

[1] Это стандартный порядок для многомерных массивов в языке C. Обратный вариант, когда матрицы хранятся по столбцам, используется в языке Fortran. Если использовать для хранения матрицы vector<vector<double>> и в качестве первого индекса использовать номер столбца («икс») а в качестве второго – номер строки («игрек»), как это привычно делать, то получится как раз C-порядок хранения.

Таким образом, разумное число потоков для умножения матрицы (если не требуется одновременно делать что-то ещё) это минимум между числом строк матрицы и числом ядер процессора. Поэтому, хороший алгоритм умножения должен иметь возможность настраиваться на разное число потоков в момент выполнения.

Измените программу так, чтобы для каждого потока можно было задать диапазон строк, которые он должен вычислить, и имелась возможность запустить 2, 3, 4 или любое другое число потоков. Для этого надо в функции потока сделать цикл по диапазону строк, а сам диапазон или вычислять в потоке по его номеру и общему числу потоков, или вычислять в main()'е и передавать через структуры – задания.

Можно найти функцию Windows, позволяющую узнать число ядер, и запускать столько потоков, сколько имеется ядер в системе.