Задача 3: Умножение последовательности матриц

Реализуйте алгоритм вычисления произведения последовательности матриц вида $A_1 * A_2 * ... * A_n$, использующий задаваемое пользователем число потоков.

Все матрицы в последовательности могут иметь разные размеры, но так, чтобы произведение было корректно определено. Например, размеры матриц могут быть $[3 \times 4] * [4 \times 5] * [5 \times 2]$, результат будет иметь размер $[3 \times 2]$.

Обратите внимание, что произведение матриц коммутативно, т.е. вычисление произведения можно проводить в любом порядке: $\big((A_1 * A_2) * (A_3 * A_4)\big) = \Big(\big((A_1 * A_2) * A_3\big) * A_4\Big) = \cdots$.

В этом задании необходимо распараллелить вычисления **попарных произведений в последовательности** в отличие от задачи 1, где распараллеливалось вычисление произведения пары матриц.

Например, вычисление произведения 4-х матриц $A = A_1 * A_2 * A_3 * A_4$ двумя потоками можно организовать следующим образом:

main	*Поток 1*	*Поток 2*
Ждёт завершения потоков	$T_1 = A_1 * A_2$	$T_2 = A_3 * A_4$
…	Ждёт завершения вычислений в потоке 2	Завершает работу
…	$A = T_1 * T_2$	
…	Завершает работу	
Печать A		

Обратите внимание, что вычисление каждого конкретного произведения может занимать существенно различающееся время, в зависимости от размеров матриц: для вычисления произведения матриц размером $[n_1 \times n_2] * [n_2 \times n_3]$ необходимо выполнить $O(n_1 * n_2 * n_3)$ операций. Из этого следует, что потоки будут заканчивать работу в разные моменты времени.

Решение должно удовлетворять следующим требованиям:

1) число потоков может быть произвольным: оно задаётся при запуске программы пользователем или определяется по числу доступных в системе ядер;
2) потоки должны создаваться один раз в начале вычисления и использоваться до его полного завершения, или пока не станет понятно, что данный поток уже не будет востребован (т.е. потоки не должны завершаться в конце каждого умножения и перезапускаться в начале следующего).
   При этом должны быть исключены необоснованные простои потоков: если есть готовая для вычисления пара исходных или промежуточных матриц и есть свободный поток, он должен начать выполнять это вычисление.

Для того чтобы не вводить с клавиатуры большое количество чисел при отладке программы, можно использовать заполнение матриц случайными числами. Достаточно, чтобы пользователь вводил только число матриц, размеры матриц и число потоков.

Для тестирования программы обязательно выполняйте вычисление этого же произведения с помощью однопоточного алгоритма и сравнивайте результаты.

Усложнение задания для самых любопытных

Разный порядок расстановки скобок при умножении матриц даёт разное время вычислений. На курсе построения оптимальных алгоритмов изучается алгоритм, который определяет оптимальный порядок вычислений для однопоточного умножения цепочки матриц. С параллельным умножением эта задача становится более сложной. Поищите в web (скорее всего в его английской части) статьи на эту тему, и подумайте, сможете ли вы оптимизировать выполнение параллельного алгоритма умножения цепочки матриц по времени.

Срок сдачи задания

Окончание второго модуля.

За сдачу **чужой программы** по заданию 3 будет наложен **штраф** в размере **50 баллов**.

Оценка задания

Баллы за задачи начисляются следующим образом:

Задача 2, не удовлетворяющая требованиям 1 и 2	20%.
Задача 2, реализовано только требование 1	40%
Задача 2, реализовано только требование 2	60%
Полностью выполненная задача 2	100%
Оптимизированный алгоритм	Дополнительные баллы в зависимости от сложности доработки.