

ЛАБОРАТОРНАЯ №1

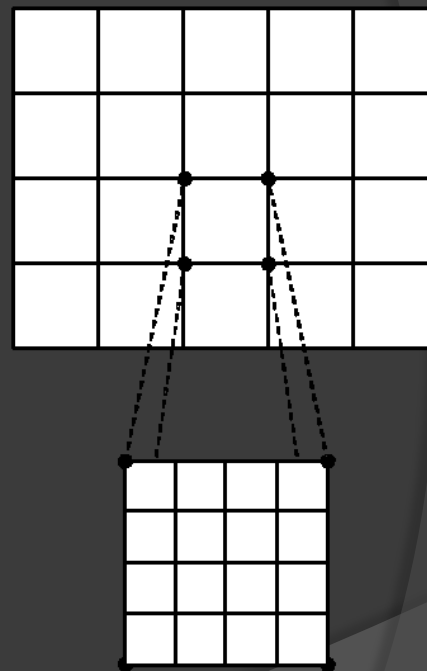
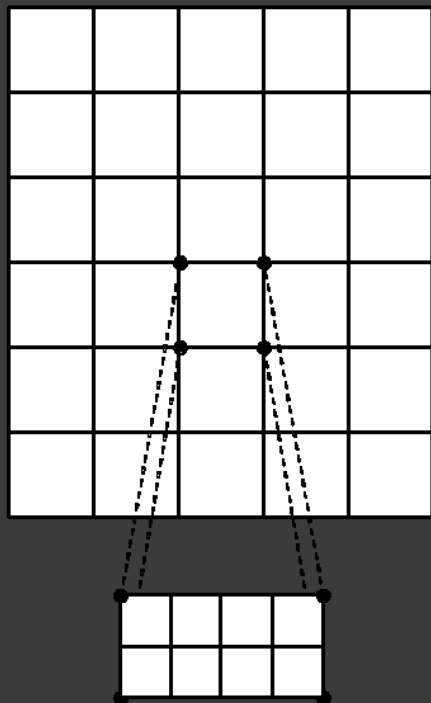
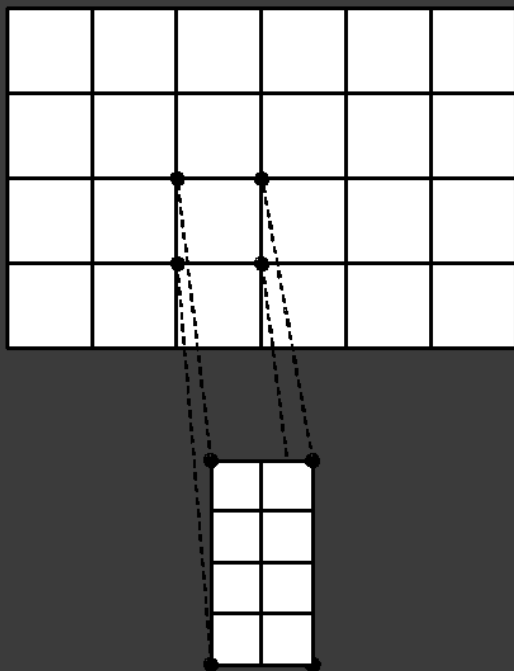
«ВЕКТОРИЗАЦИЯ»

Задание

- ① перемножить 2 матрицы следующими способами:
 - результат – матрица C1:
 - с включенной векторизацией
 - с выключенной векторизацией
 - результат – матрица C2:
 - ручная векторизация с использованием SSE2-инструкций (либо новее)
 - ассемблер
 - intrinsics (альтернатива ассемблеру)
- ① элементы матрицы
 - своя матрица меньшего размера (см. сл.слайд)
- ① размер внешней матрицы подбирается самостоятельно
 - время получения матрица C1 от нескольких секунд
- ① размер матрицы / тип входных данных:
 - **определяется преподавателем**

Задание

■ ■



Задание

● Обязательные условия:

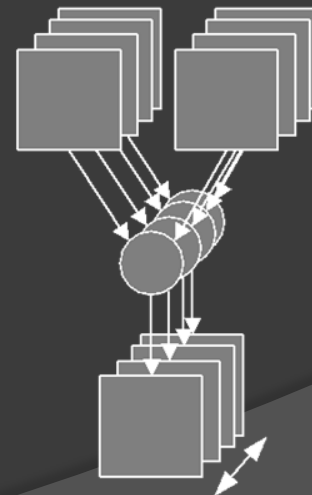
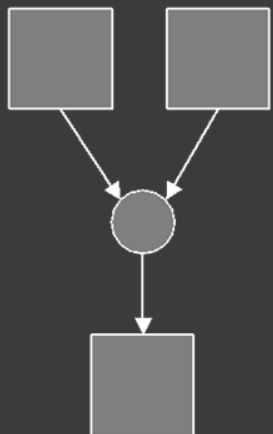
- в одном проекте минимум (!) 2 функции:
 - с автоматической векторизацией
 - с ручной векторизацией
- результат матриц C1 и C2 должны **ПОЛНОСТЬЮ** совпадать
 - никакого вывода фрагментов матрицы НЕ делать
- время работы SSE2-версии *не* медленнее версии с автоматической векторизацией

Замечания по векторизации

- ◎ Зависит от среды разработки:
 - Visual Studio 2012 и новее
 - GCC (версия неизвестна, точно с 2009 года)
 - ПРОБЛЕМА: как отключить векторизацию ???
- ◎ способы доказать:
 - дизассемблер
 - спорный способ есть на MSDN
 - принимается в определенных условиях 😊
 - Intel vTune
- ◎ Ускорение векторизованной версии:
 - float: ~3-3,5 раза
 - double: ~1,2-1,5 раза

Векторизация

- выполнение одной инструкции над вектором данных
- в лаб.работе транспонировать матрицы запрещено



Векторизация

⦿ типы данных:

- int (не во всех SIMD-наборах)
- double, float (кроме MMX)

⦿ наборы SIMD-инструкции:

- MMX (Multi-media extension, 1997 г.)
 - 64 бит, регистры mmx
- SSE (Streaming SIMD Extension, 1999 г.)
 - 128 бит, регистры xmm
- AVX (Advanced Vector Extension, 2011 г.)
 - 256 бит, регистры ymm
- AVX-512 (2016 г.)
 - 512 бит, регистры zmm

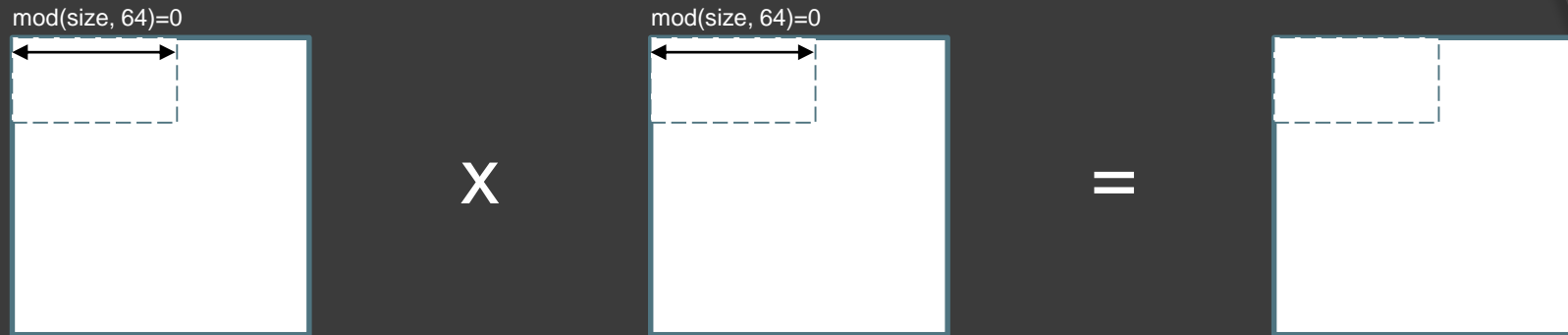
Требования к коду

- итерации цикла должны быть независимыми
- код в цикле «просчитываемый»
- нарушение потока команд
- внутренний цикл

Способы заработать бонусы

- ① универсальная версия с произвольными размерами матрицы
 - для ручной векторизации
- ② доказательство векторизации с помощью vTune
- ③ выявить ВСЕ «узкие» места производительности (аппаратная часть) с помощью vTune
- ④ существенное ускорение ручной векторизации по отношению к автоматической
- ⑤ качественное, УНИКАЛЬНОЕ (!) и интересное решение *(на усмотрение преподавателя)*

Оптимизация доступа в кэш (доп.задание)



- ⦿ умножение выполняется поблочно
 - размер блока:
 - $L3_{size}$ – размер кэша третьего уровня
 - $block_{size} = \left\lfloor \frac{L3_{size}}{3} \cdot 0,9 \right\rfloor$ – максимально допустимый размер одного блока
 - как распределяется размер $block_{size}$ по X и Y – решаете самостоятельно. Ограничения:
 - ширина подматриц A и B в байтах ДОЛЖНА БЫТЬ кратна 64 байтам (!!!!!)
- ⦿ алгоритм:
 - блок одной из матриц остается без изменений до тех пор, пока не будут обработаны ВСЕ блоки второй матрицы

Бонусные баллы

- оптимизация не только под L3, но и под L2, L1
- расширенная версия алгоритма (в соответствии с рекомендациями документа [cpumemory.pdf](#), [matrixmult.pdf](#))
- работа с Intel vTune

Рекомендуемое чтение

- ◎ [24504501.pdf](#) – то, как рекомендуется перемножать матрицы от Intel
 - P.S. Версия для тех, кому скучно 😊