Основы технологии CUDA

Семинар 5:Работа с разделяемой памятью

Автор курса:

- Казённов Андрей

Типы памяти в CUDA

Тип памяти	Доступ	Уровень выделения	Скорость работы
Регистры	R/W	Per-thread	Высокая(on-chip)
Локальная	R/W	Per-thread	Низкая (DRAM)
Shared	**R/W**	**Per-block**	**Высокая(on-chip)**
Глобальная	**R/W**	**Per-grid**	**Низкая (DRAM)**
Constant	R/O	Per-grid	Высокая(L1 cache)
Texture	R/O	Per-grid	Высокая(L1 cache)

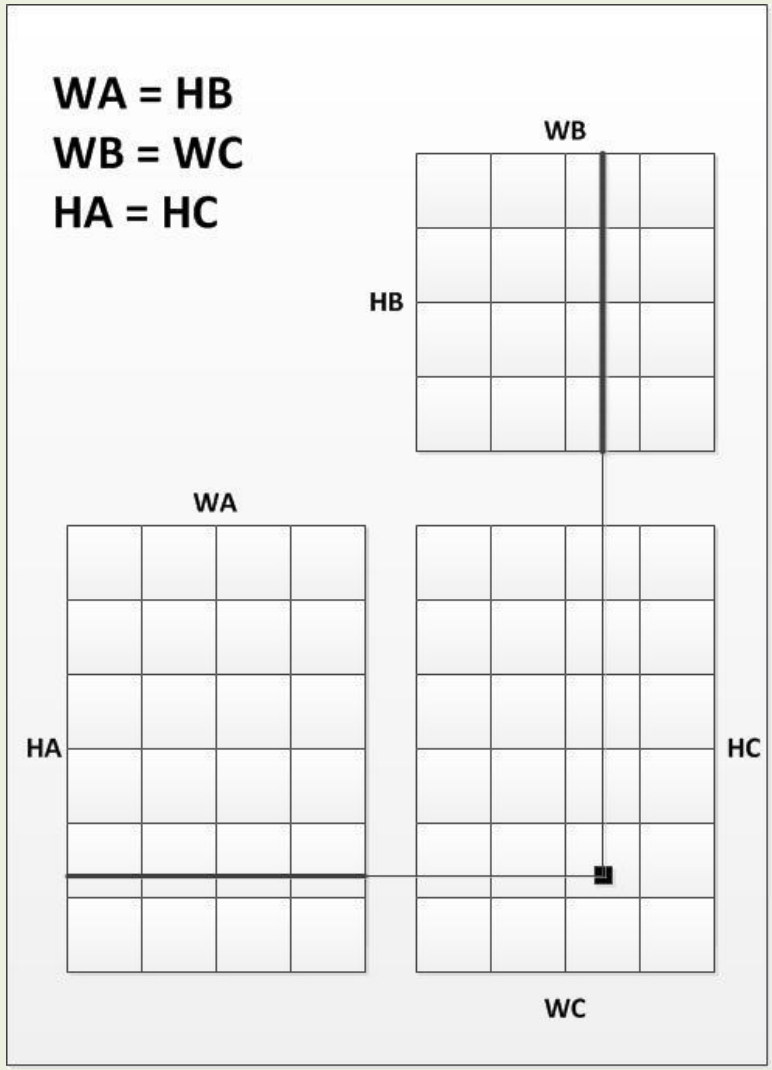
Пример: Задача перемножения матриц

В ядре нам будут доступны следующие переменные:

- WA, WB, HA
- BS – размер блока по любому измерению (блок квадратный)
- BX = blockIdx.x
- BY = blockIdx.y
- TX = threadIdx.x
- TY = threadIdx.y

Все матрицы хранятся в линейных массивах.

Каждый тред будет считать отдельный элемент матрицы C. Для этого ему понадобится строка из A и столбец из B.

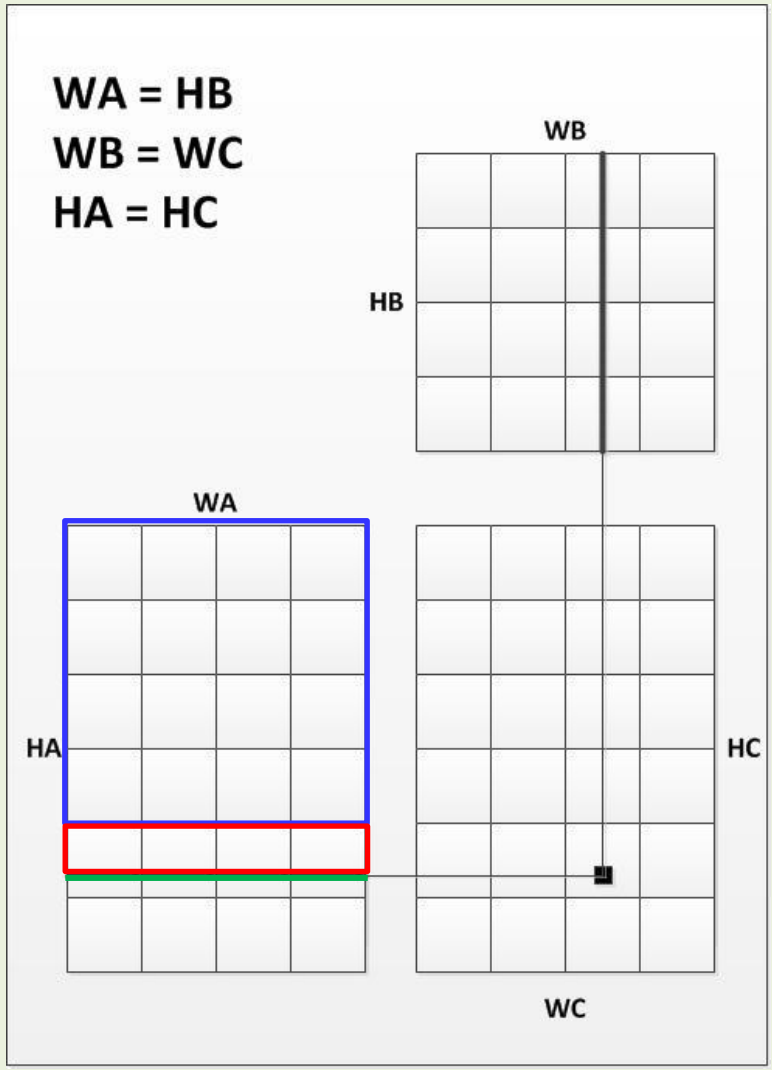


Пример: Задача перемножения матриц

Astart = WA * BS * BY + WA * TY

Astep = 1

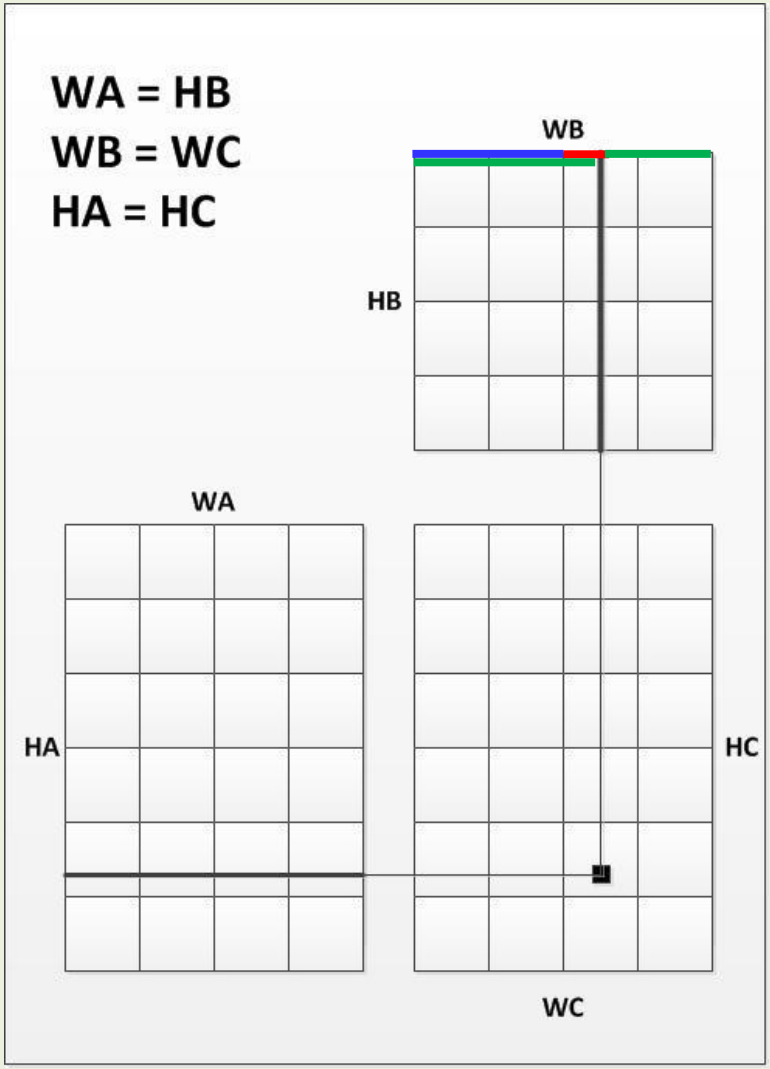
Astop = Astart + WA



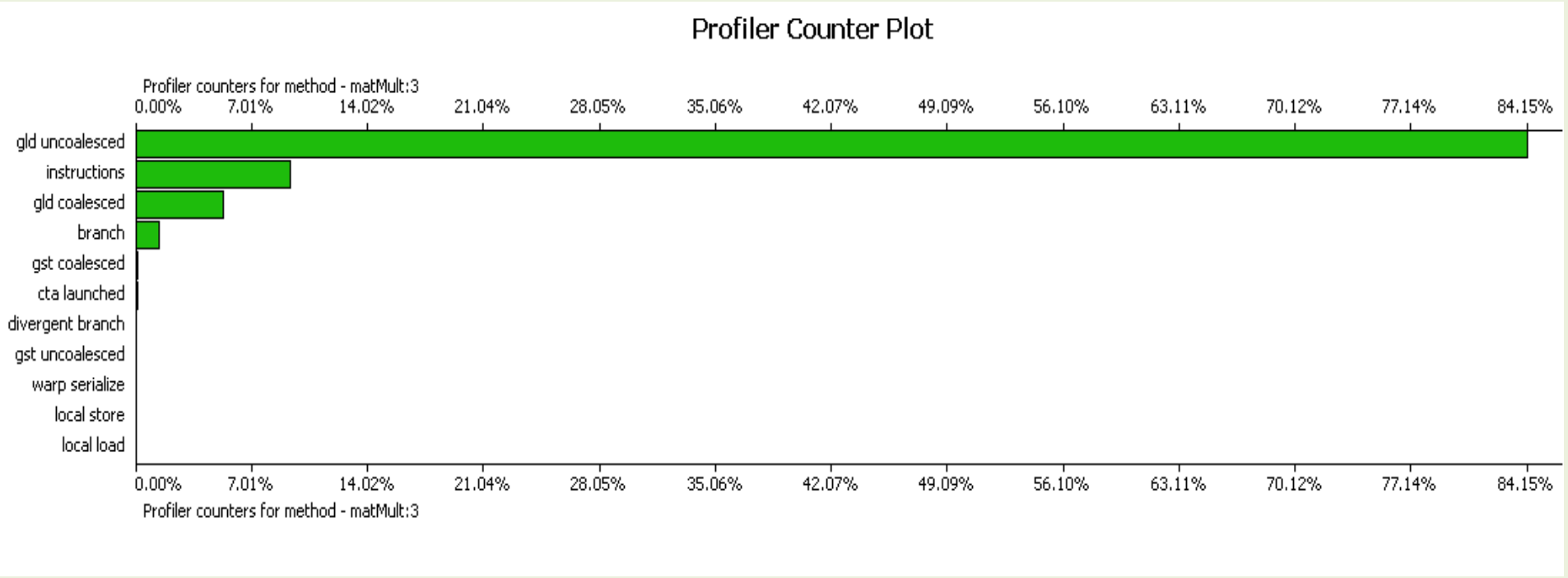
Пример: Задача перемножения матриц

Bstart = BS * BX + TX

Bstep = WB



Пример: Задача перемножения матриц: производительность



Применение разделяемой памяти

- Быстрая (*on-chip*)
- 16 или 48 Кб на каждый SM
- Используется совместно всеми тредами блока (нет понятия своей и чужой памяти)
- Отдельное обращение для каждой половины *варпа (как и для любой памяти в CUDA)*
- Как правило, требует явной синхронизации

Применение разделяемой памяти

- Для повышения пропускной способности вся *shared*-память разбита на 16 банков
- Каждый банк работает независимо от других
- Можно одновременно выполнить до 16 обращений к *shared*-памяти
- Если идет несколько обращений к одному банку, то они выполняются по очереди (конфликт банков)

Применение разделяемой памяти

- Банки строятся из 32-битовых слов
- Подряд идущие 32-битовые слова попадают в подряд идущие банки
- Конфликт банков– несколько нитей из одного полуварпа обращаются к одному и тому же банку
- Конфликта не происходит если все 16 нитей обращаются к одному слову (рассылка)

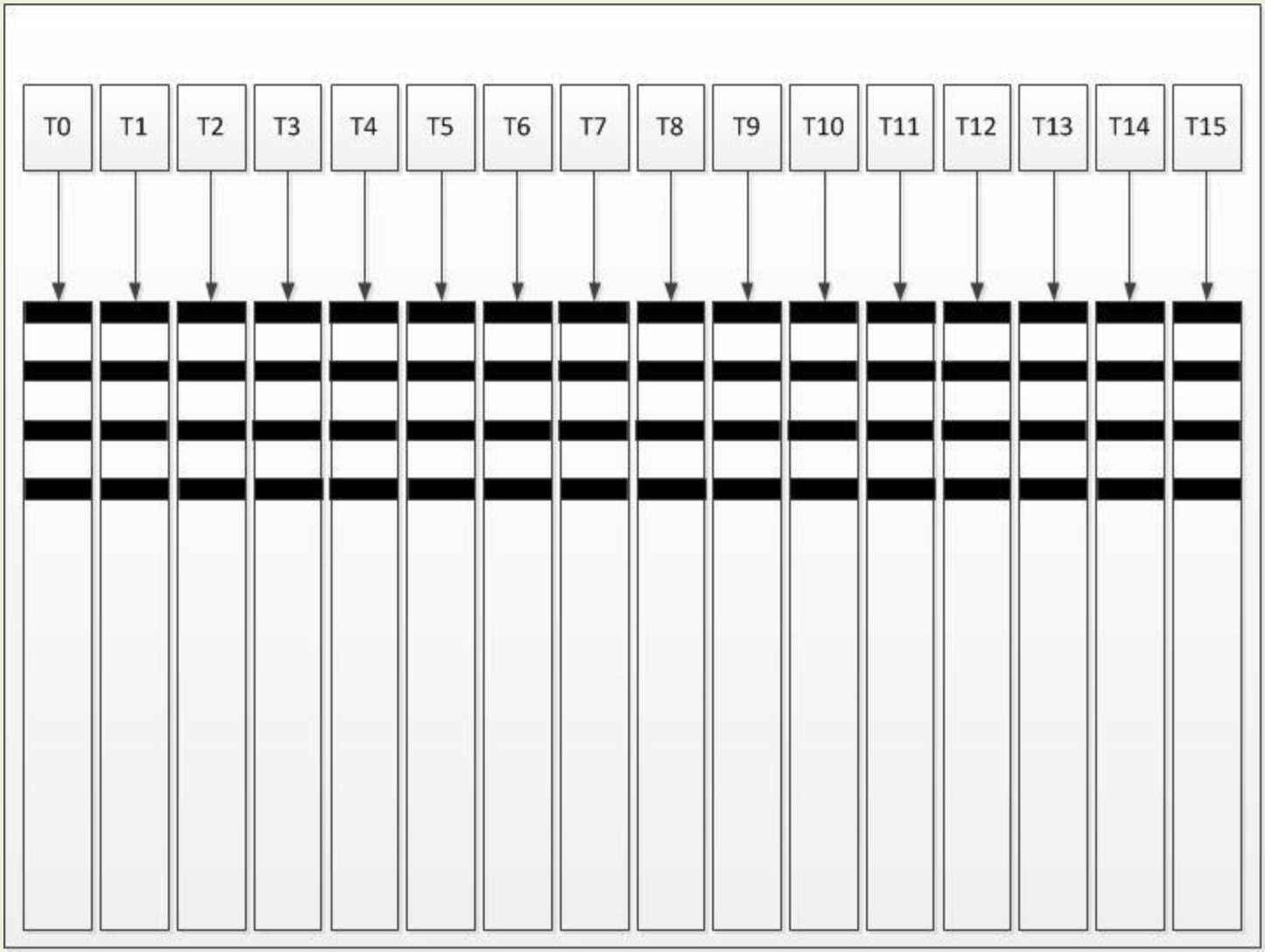
Применение разделяемой памяти

```
__global__ void shared_kern(CUDA_FLOAT *A)
{
      int IDG = threadIdx.x + blockIdx.x * BLOCK_SIZE;
      int IDL = threadIdx.x;

      __shared__ CUDA_FLOAT AS[BS]; // Задание разделяемой памяти

      AS[IDL]=A[IDG]; // Копирование из глобальной в разделяемую
      __syncthreads(); // Синхронизация

      AS[IDL] = AS[IDL] + 1;
      __syncthreads(); // Синхронизация

      A[IDG]=AS[IDL]; // Копирование из разделяемой в глобальную

}
```

Конфликтов нет

float A[size];

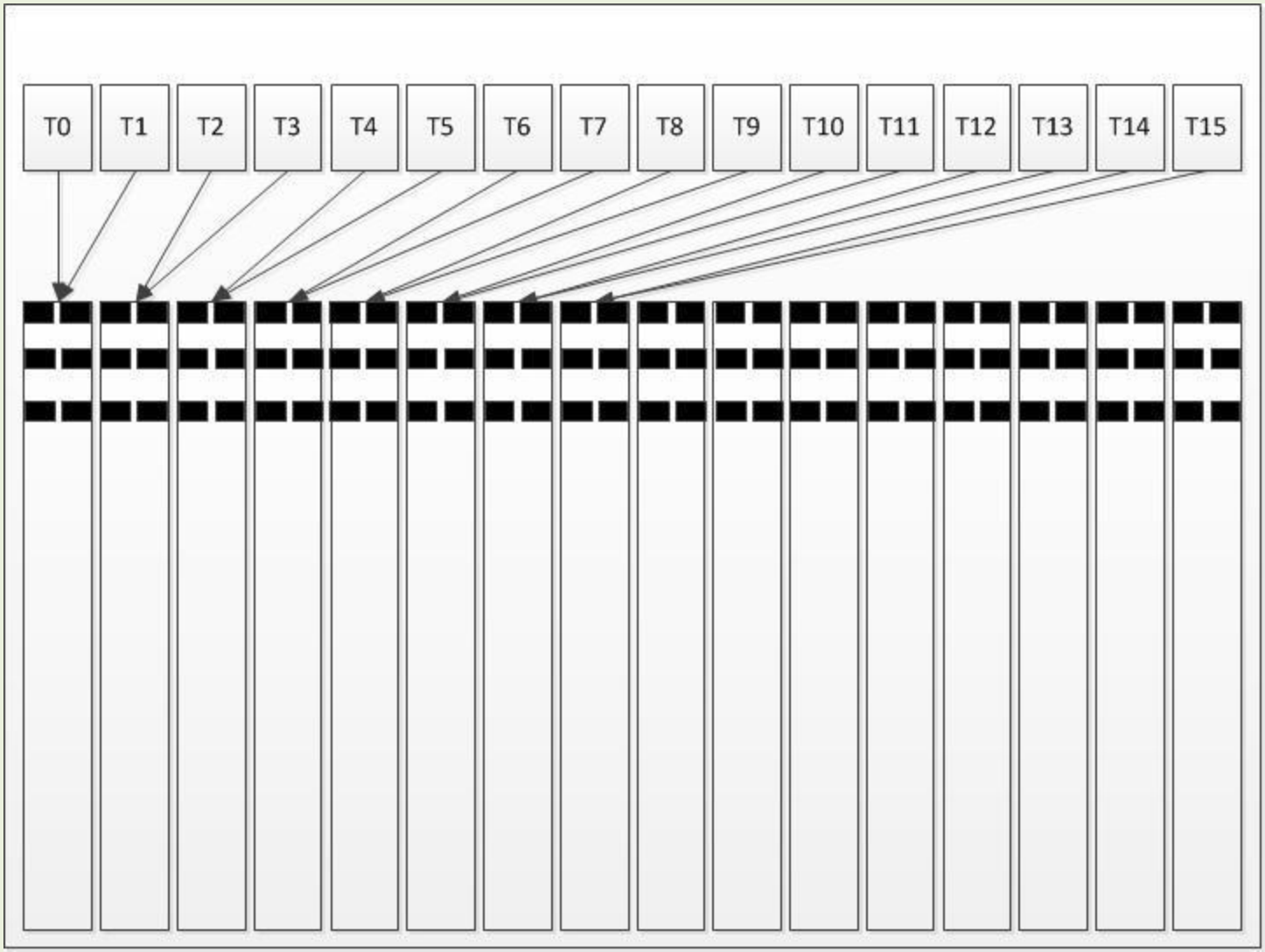
a = A[TID];



Конфликты 2го порядка

short B[size];

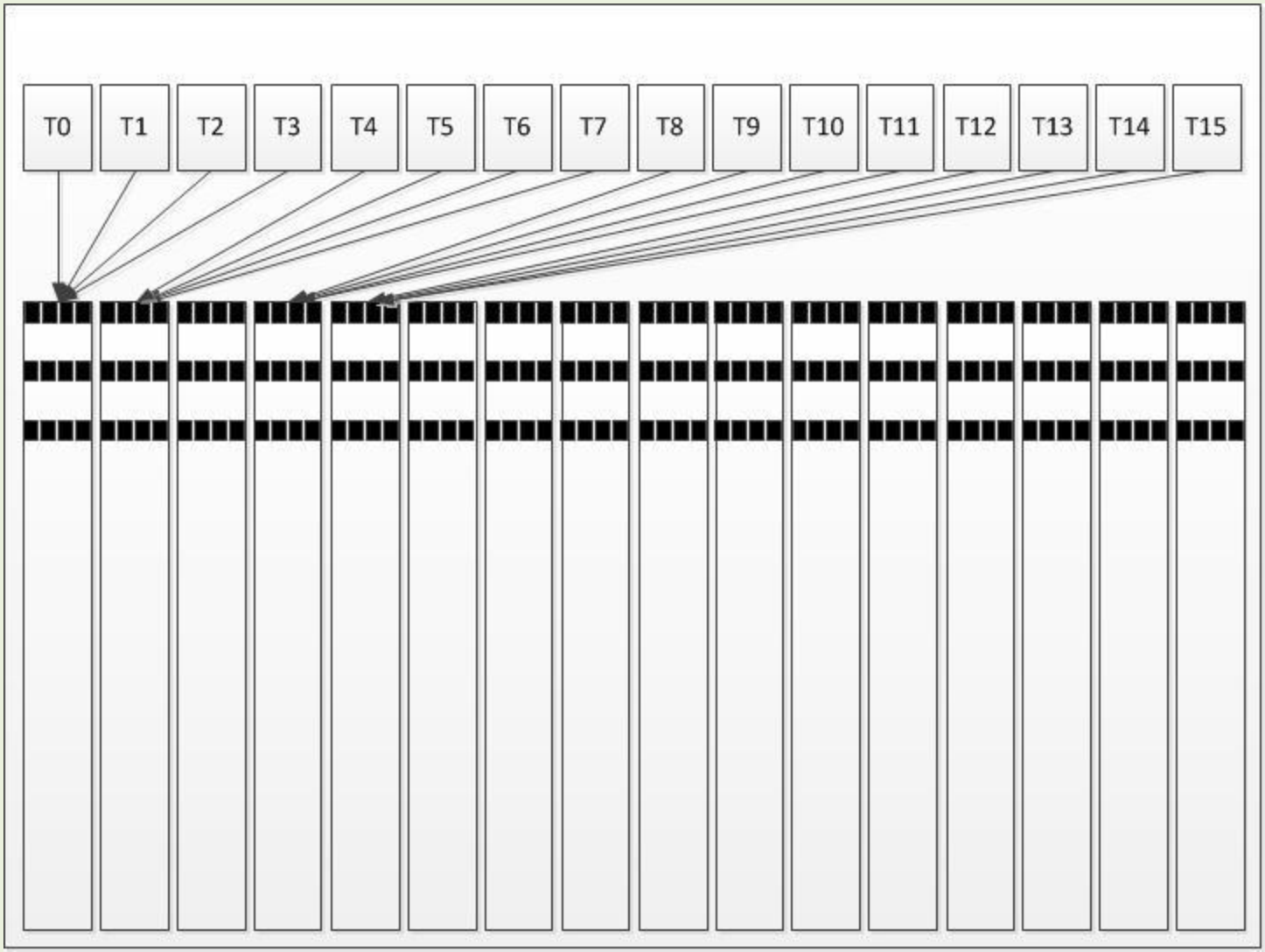
b = B[TID];



Конфликты 4го порядка

char C[size];

c = C[TID];



Пример: Задача перемножения матриц

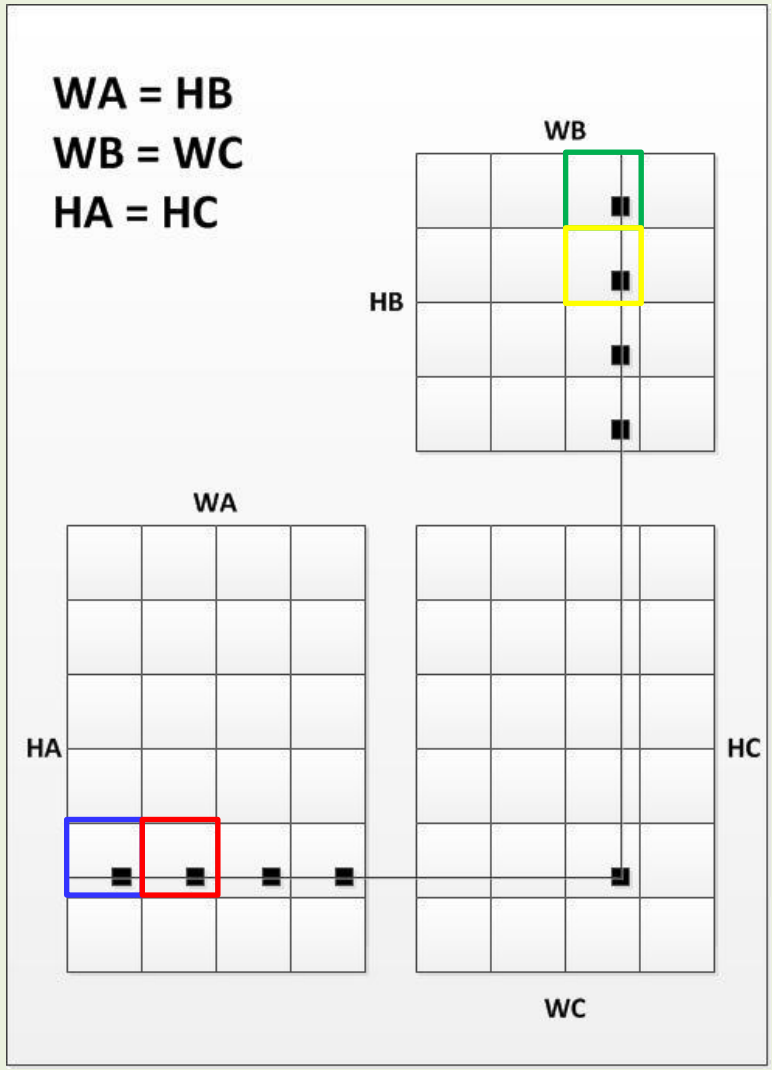
В ядре нам будут доступны следующие переменные:

- WA, WB, HA
- BS – размер блока по любому измерению (блок квадратный)
- BX = blockIdx.x
- BY = blockIdx.y
- TX = threadIdx.x
- TY = threadIdx.y

Все матрицы хранятся в линейных массивах.

Будем считать по блочно.

C = A1 * B1 + A2 * B2



Пример: Задача перемножения матриц

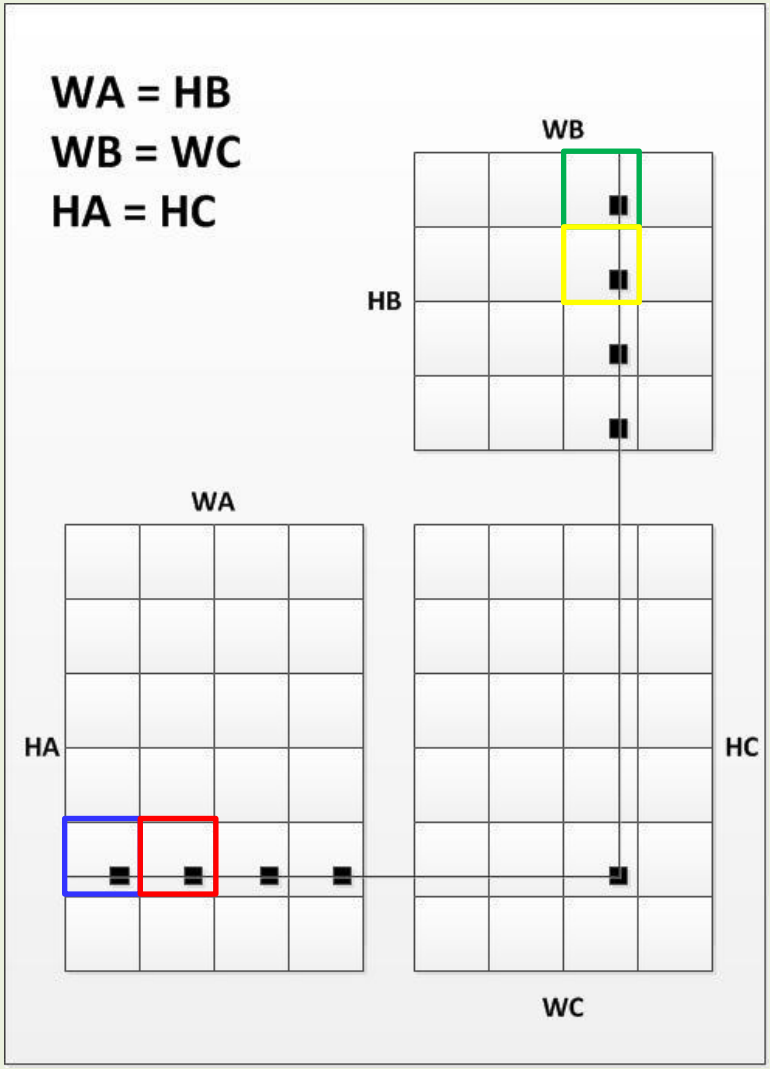
В ядре нам будут доступны следующие переменные:

- WA, WB, HA
- BS – размер блока по любому измерению (блок квадратный)
- BX = blockIdx.x
- BY = blockIdx.y
- TX = threadIdx.x
- TY = threadIdx.y

Все матрицы хранятся в линейных массивах.

Будем считать по блочно.

C = A1 * B1 + A2 * B2 + ….



Пример: Задача перемножения матриц

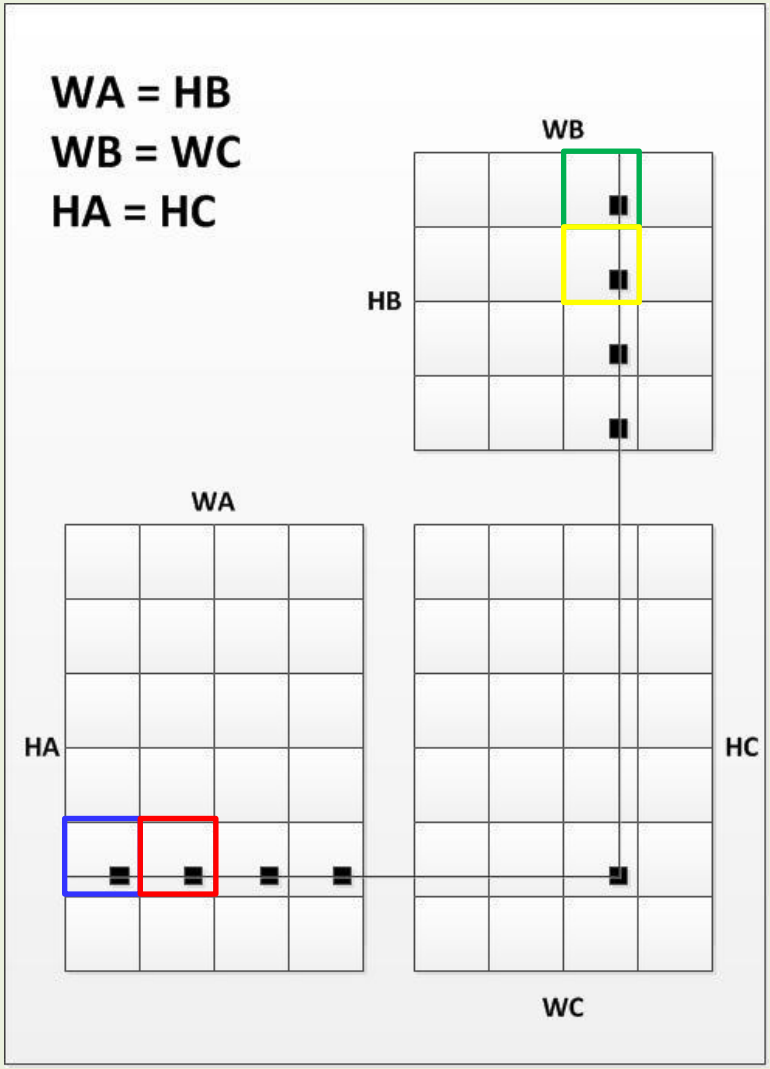
В ядре нам будут доступны следующие переменные:

- WA, WB, HA
- BS – размер блока по любому измерению (блок квадратный)
- BX = blockIdx.x
- BY = blockIdx.y
- TX = threadIdx.x
- TY = threadIdx.y

Все матрицы хранятся в линейных массивах.

Будем считать по блочно.

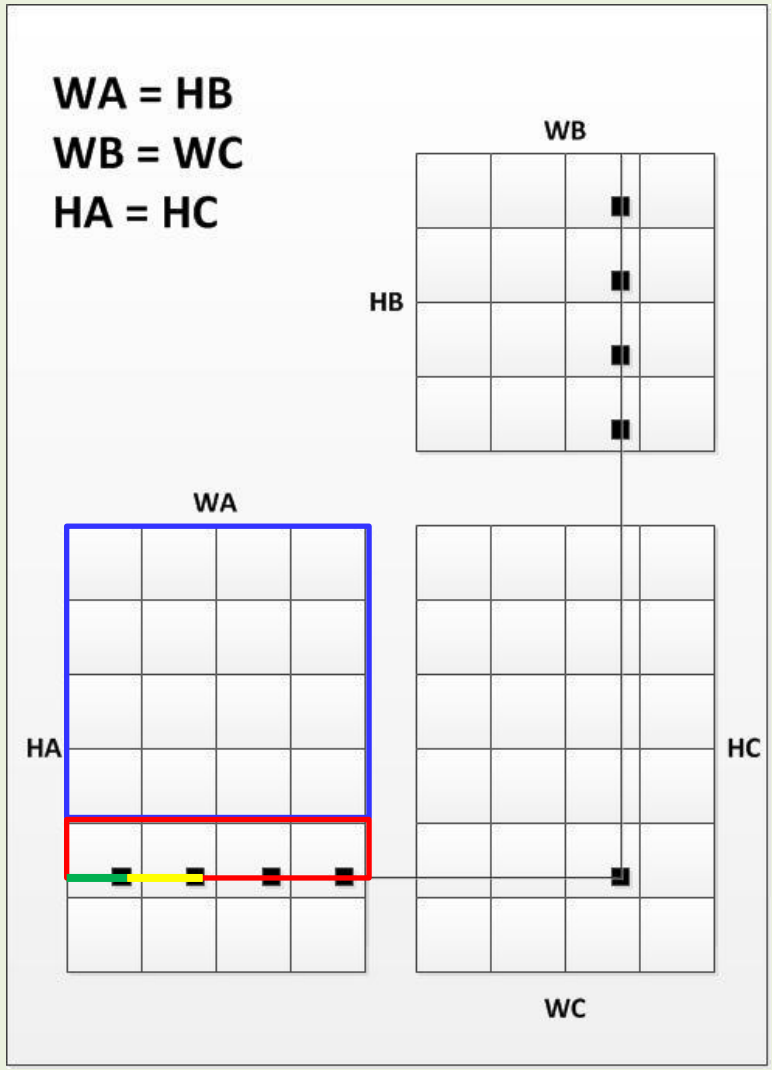
C = A1 * B1 + A2 * B2 + ….



Пример: Задача перемножения матриц

Astart = WA * BS * BY + WA * TY + TX

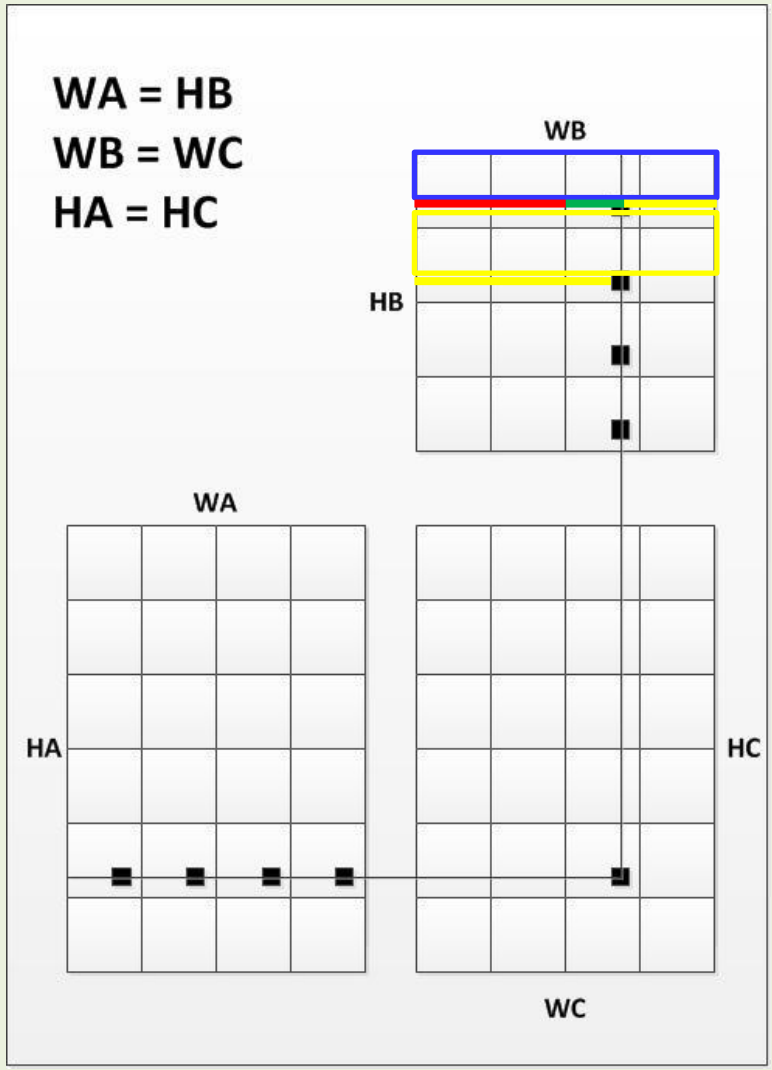
Astep = BS



Пример: Задача перемножения матриц

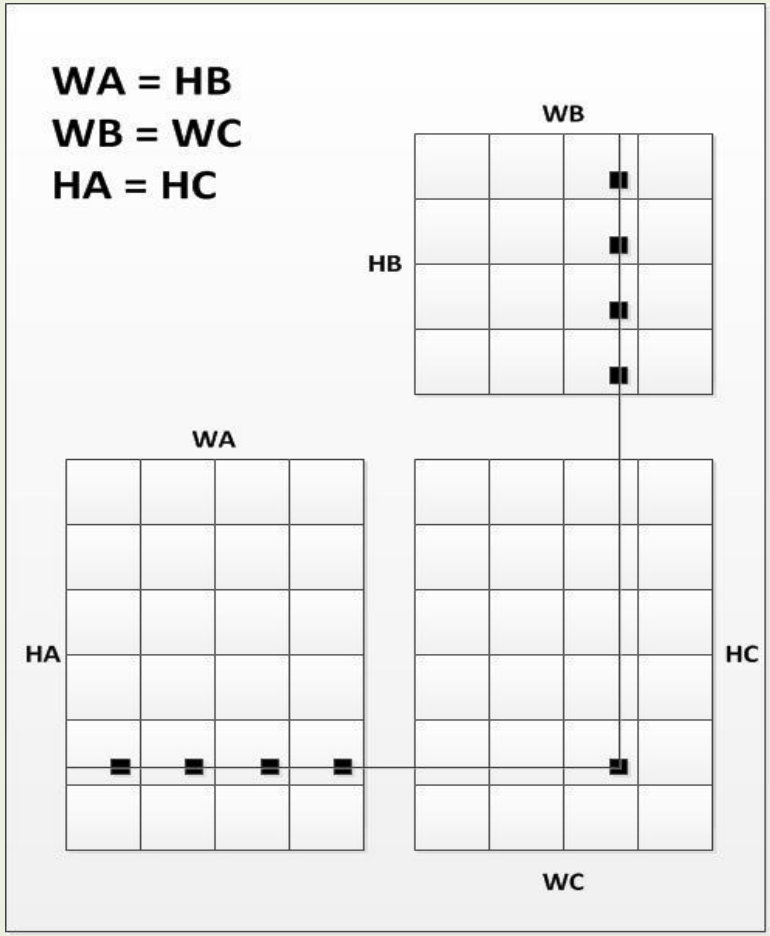
Bstart = WA * TY + BS * BX + TX

Bstep = BS *WA



Основы технологии CUDA

Задача: Реализовать перемножение матриц с использованием разделяемой памяти



Ресурсы курса

Сайт: HPC.MIPT.RU

Раздел образование

Автор курса:

Казённов Андрей

- E-mail: kazennov@gmail.com
- ICQ: 622774