Московский государственный университет им. М.В. Ломоносова

Параллельное программирование для многопроцессорных систем с общей и распределенной памятью

Разработчики:
В.И. Лаева, e-mail: lvi@math.tsu.ru
А.А. Трунов, e-mail: trunov@math.tsu.ru
Томский государственный университет

Направление 010400.62
«Прикладная математика и информатика»

Проект комиссии Президента по модернизации и техническому развитию экономики России
«Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения»

Содержание курса

- Введение в параллельное программирование с использованием стандарта OpenMP. Параллельные области
- Параллельные циклы. Секции. Директивы master, single, workshare, threadprivate.
- Синхронизация в OpenMP
- Введение в параллельное программирование с использованием технологии CUDA
- Иерархия памяти ГПУ и работа с ней в CUDA
- Работа с разделяемой памятью. Синхронизация в CUDA

Содержание лекции

- Синхронизация: внутри блока потоков.
- Синхронизация памяти.
- Атомарные инструкции.
- Глобальная синхронизация.
- Решения двумерного нестационарного уравнения теплопроводности с применением явной разностной схемы типа "крест".

Синхронизация: внутри блока потоков

- Барьерная синхронизация потоков в пределах блока: **void __syncthreads()**.
- Барьер можно применять в условных конструкциях, но при условии, что все потоки блока достигнут его.
- Барьер не влияет на потоки из других блоков, т.е. осуществляет локальную синхронизацию.

Синхронизация: внутри блока потоков

- ГПУ архитектуры (compute capability) 2.x поддерживают ещё 3 варианта барьера:
  - **int __syncthreads_count(int predicate)** - барьер и подсчёт кол-ва потоков блока, для которых predicat != 0
  - **int __syncthreads_and(int predicate)** - барьер и возвращает !0 только если для всех потоков блока predicat != 0
  - **int __syncthreads_or(int predicate)** - барьер и возвращает !0 если хотя бы для одного потока блока predicat != 0

Синхронизация: внутри блока потоков

- Синхронизация памяти:
  - **`void __threadfence_block()` - ожидание до тех пор пока результаты предшествующих операций с глобальной и разделяемой памятью, совершаемых вызывающим потоком, не будут видны всем потокам в пределах блока**

Синхронизация: внутри блока потоков

- Синхронизация памяти:
  - **`void __threadfence()` - ожидание до тех пор пока результаты предшествующих операций с глобальной и разделяемой памятью, совершаемых вызывающим потоком, не будут видны следующим потокам:**
    - **для операций с разделяемой памятью - потокам в пределах блока**
    - **для операций с глобальной памятью - потокам в пределах ГПУ**

Синхронизация: внутри блока потоков

- Синхронизация памяти (для архитектуры 2.x):
  - **`void __threadfence_system()` - ожидание до тех пор пока результаты предшествующих операций с глобальной и разделяемой памятью, совершаемых вызывающим потоком, не будут видны следующим потокам:**
    - **для операций с разделяемой памятью - потокам в пределах блока**
    - **для операций с глобальной памятью - потокам в пределах ГПУ**
    - **для операций с page-locked памятью ЦПУ - потокам ЦПУ**

Атомарные инструкции

- Атомарные инструкции реализуются в CUDA через функции.
- Обычная последовательность действий с памятью: прочитать содержимое памяти, изменить его и записать обратно.
- Гарантируется, что при использовании атомарных инструкций потоки не будут взаимодействовать между собой.
- Например, после прочтения переменной 1-м потоком, 2-й поток не получит к ней доступ до записи результата 1-м потоком.

Атомарные инструкции

- Атомарные инструкции можно применять для доступа к разделяемой и глобальной памяти.
- Допустимые варианты атомарных инструкций достаточно сильно различаются для ГПУ разной архитектуры.
- В состав атомарных инструкций входят:
  - арифметические операции: atomicAdd, atomicSub, atomicExch, atomicMin, atomicMax, atomicInc, atomicDec, atomicCAS;
  - побитовые операции: atomicAnd, atomicOr, atomicXor.

Пример: редукция массива

```
__device__ unsigned int count = 0;
__shared__ bool isLastBlockDone;
__global__ void sum(const float *array, unsigned int N, float *result)
{
    float partialSum = calculatePartialSum(array, N);
    if (threadIdx.x == 0) {
       result[blockIdx.x] = partialSum;
       __threadfence();
       unsigned int value = atomicInc(&count, gridDim.x);
       isLastBlockDone = (value == (gridDim.x - 1));
    }
    __syncthreads();
    if (isLastBlockDone) {
       float totalSum = calculateTotalSum(result);
       if (threadIdx.x == 0) {
          result[0] = totalSum;
          count = 0;
       }
    }
}
```

возвращает предыдущее значение count

Глобальная синхронизация

- Использование атомарных инструкций (функций)
- Использование ЦПУ
- Вызов ядра (kernel) асинхронен!

```
for (t = 1; t <= Nt; t++)
{
    HeatTransfer_Step <<<dimGrid, dimBlock>>> (T0d, Td, Nx0, Ny0);
    cudaDeviceSynchronize(); // синхронизация между итерациями
    swap(T0d, Td);
}
```

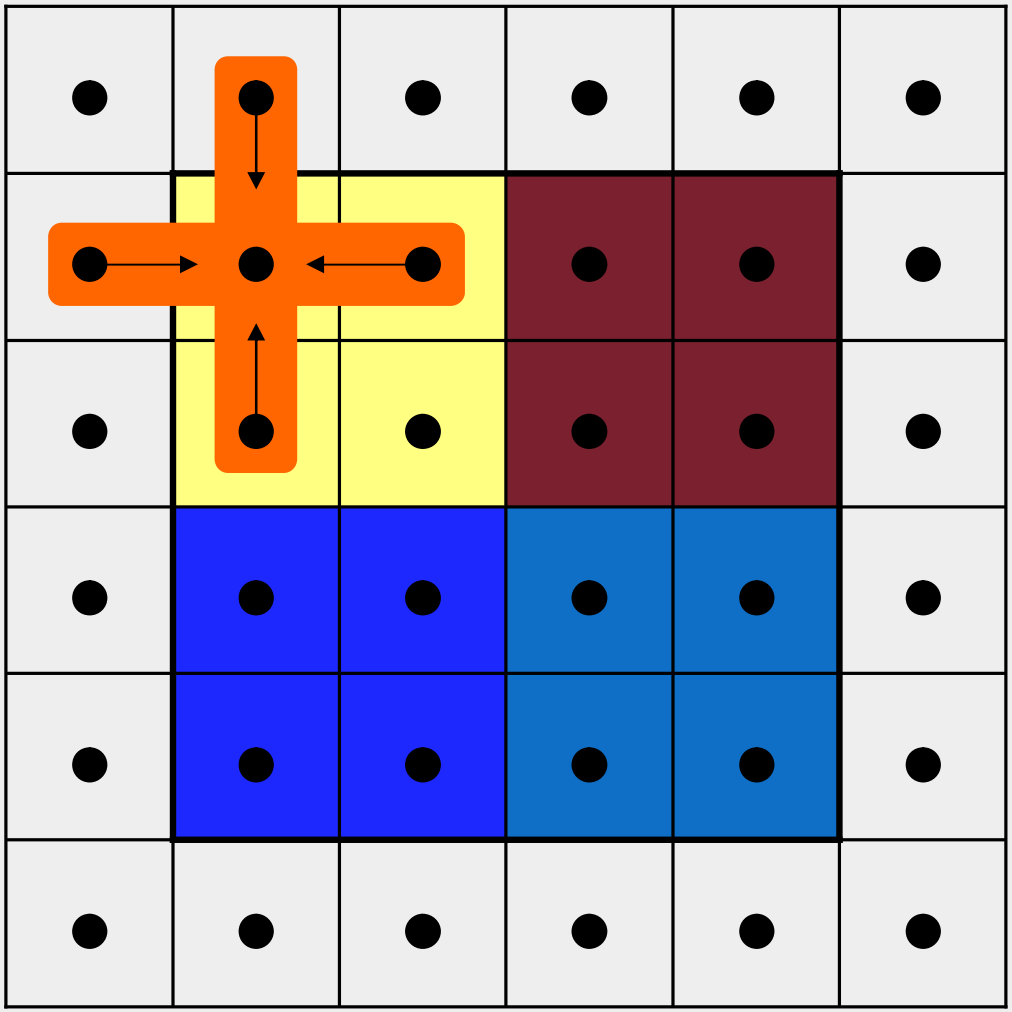
Модельная задача: 2D уравнение теплопроводности

- Явная разностная схема с 5-точечным шаблоном типа «крест»:

$$T_{i,j}^{n+1} = T_{i,j}^{n} + \alpha\Delta t\left(\frac{T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n}}{\Delta x^2} + \frac{T_{i,j+1}^{n} - 2T_{i,j}^{n} + T_{i,j-1}^{n}}{\Delta y^2}\right);$$

$$i = \overline{1,m_x};\ \ j = \overline{1,m_y};\ \ n = \overline{0,m_t};$$

1-й подход. Каждый поток копирует 5 чисел из ОЗУ независимо от других потоков.



Количество операций чтения/записи ОЗУ

Чтение из ОЗУ

0	1	1	1	1	0
1	3	4	4	3	1
1	4	5	5	4	1
1	4	5	5	4	1
1	3	4	4	3	1
0	1	1	1	1	0

Запись в ОЗУ

0	0	0	0	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

Суммарное количество операций чтения/записи ОЗУ

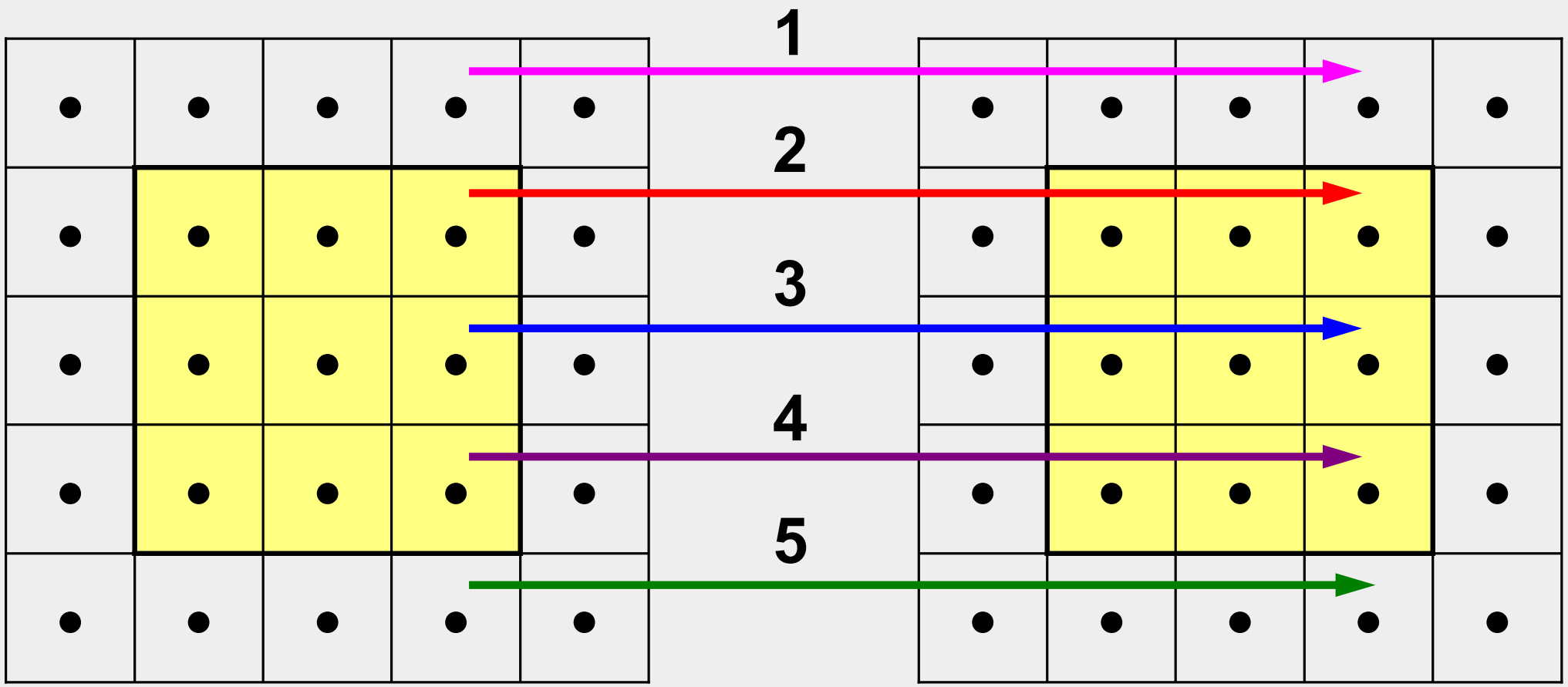
Чтение/запись ОЗУ

0	1	1	1	1	0
1	4	5	5	4	1
1	5	6	6	5	1
1	5	6	6	5	1
1	4	5	5	4	1
0	1	1	1	1	0

- Кол-во операций чтения/записи ≈ $6m_x \cdot m_y$
- min кол-во операций чтения/записи ≈ $2m_x \cdot m_y$
- Оценка потенциала оптимизации ≈ 3 раза

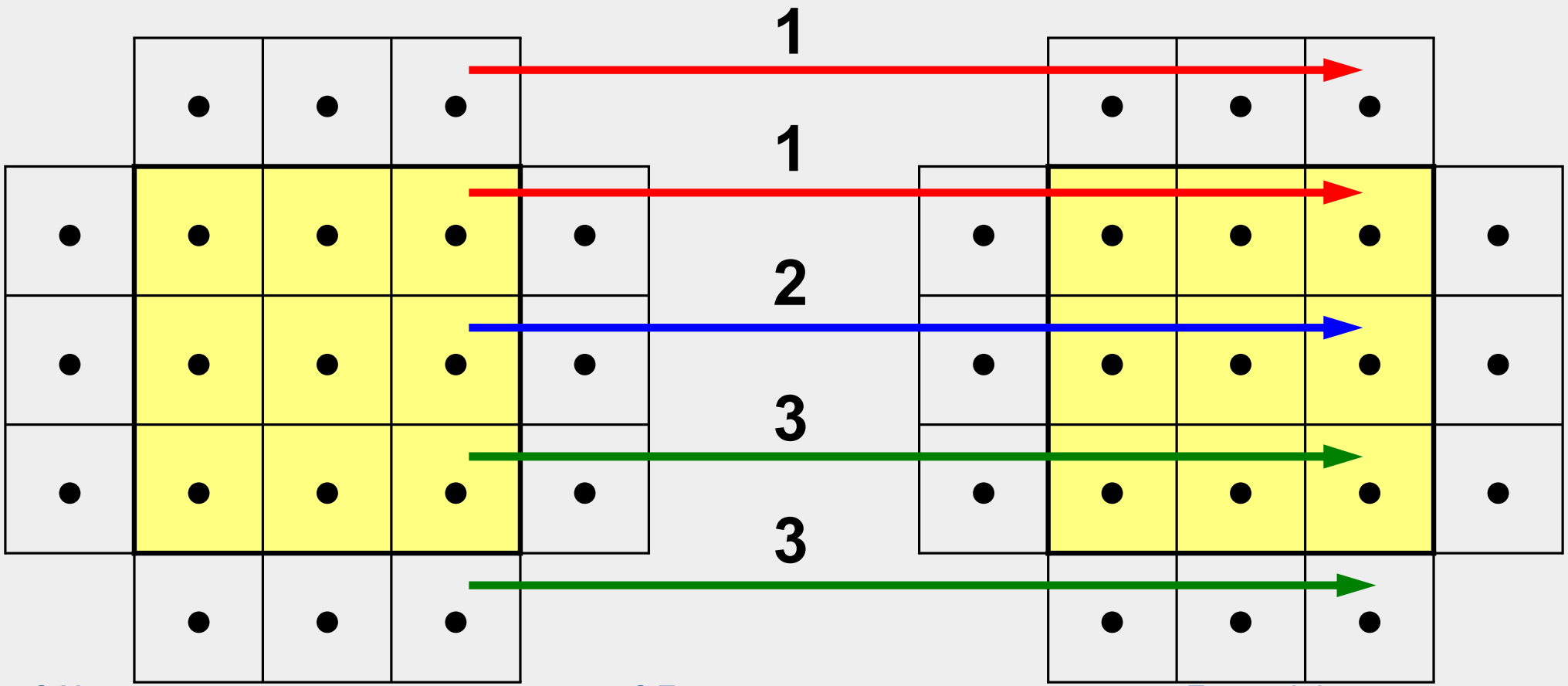
Подобласть для одного блока потоков. 2-й подход. 2D декомпозиция - 1

Глобальная память ГПУ → **Разделяемая память мультипроцессора**

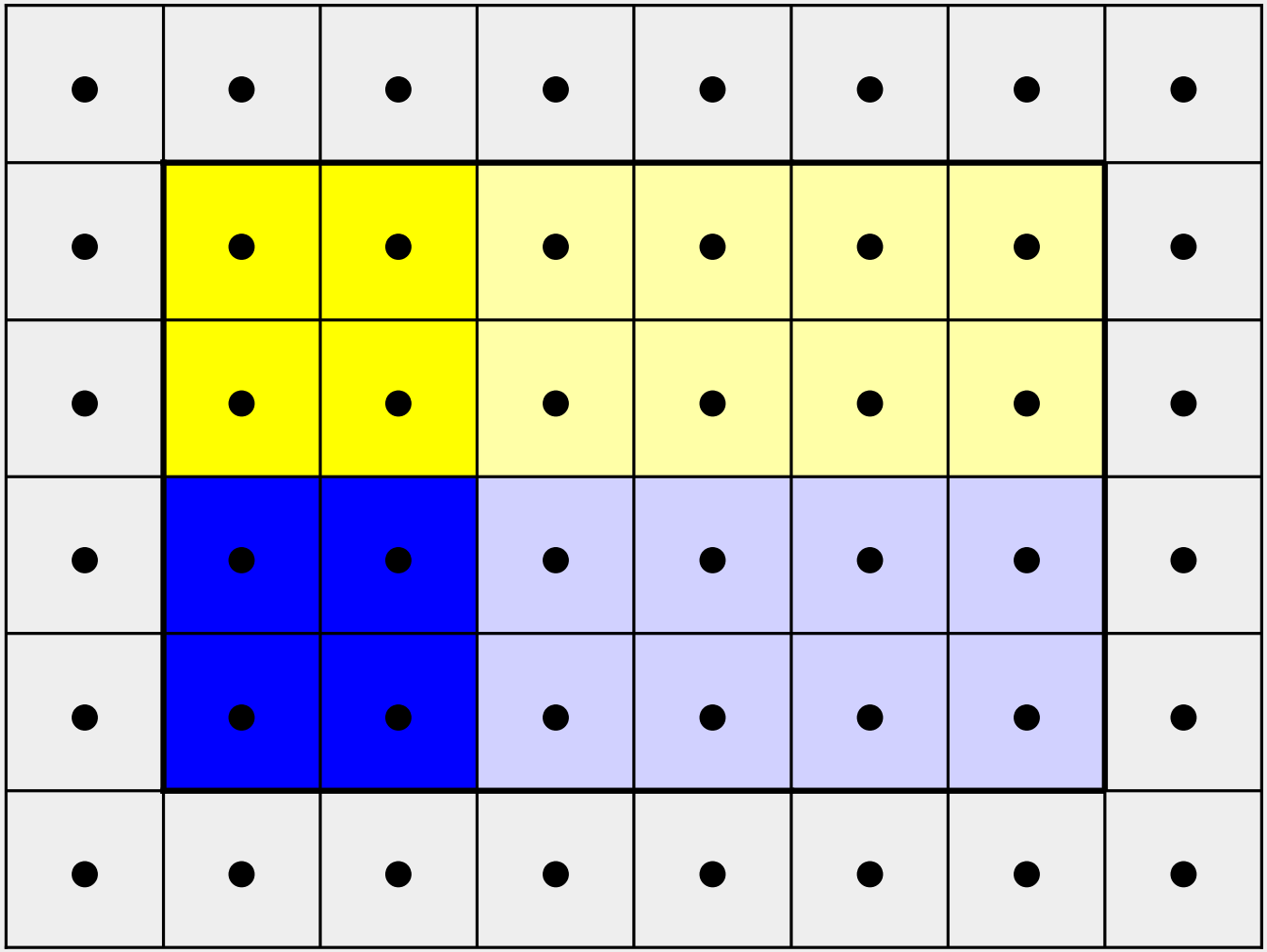


Подобласть для одного блока потоков. 3-й подход. 2D декомпозиция - 2

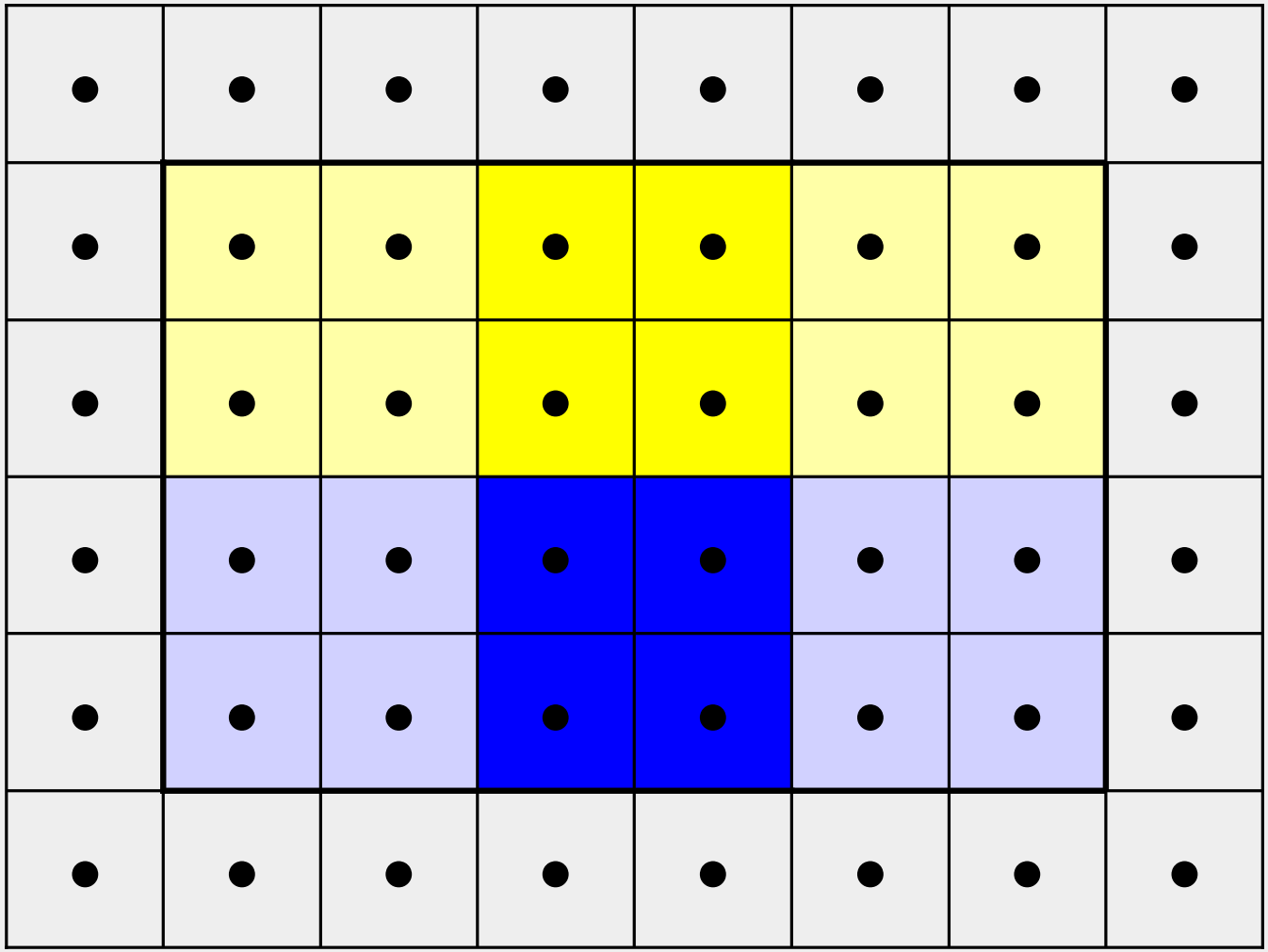
Глобальная память ГПУ → **Разделяемая память мультипроцессора**



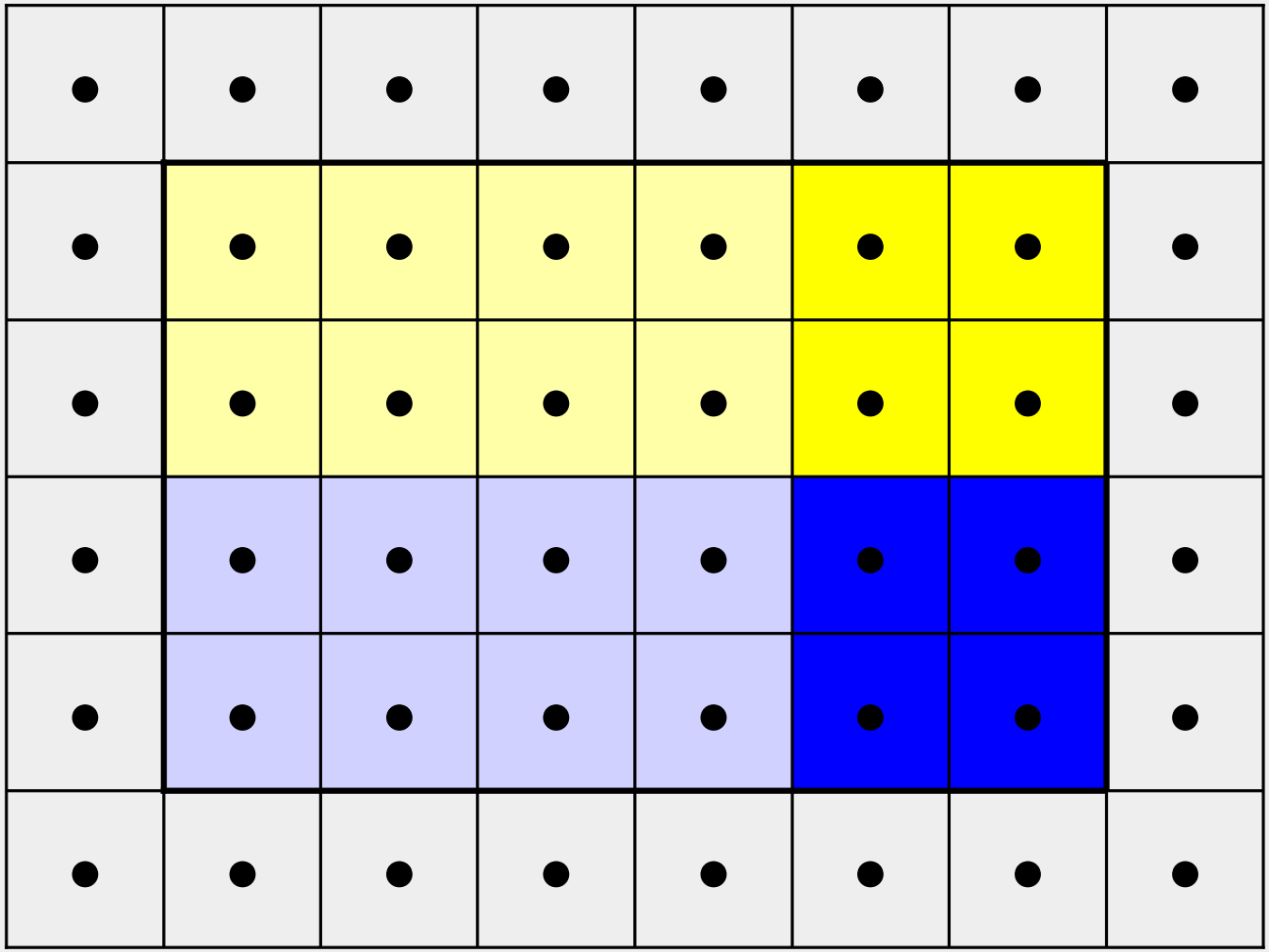
Подобласть для одного блока потоков. 4-й подход. 1D декомпозиция Устранение повторного копирования столбцов (C)



Подобласть для одного блока потоков. 4-й подход. 1D декомпозиция Устранение повторного копирования столбцов (C)



Подобласть для одного блока потоков. 4-й подход. 1D декомпозиция Устранение повторного копирования столбцов (C)



Результаты расчётов. 2D уравнение теплопроводности

- Размерность области 14400 x 14400
- 100 шагов по времени
- Расчёты с одинарной точностью

GTX 260

	Подход 1	Подход 2	Подход 3	Подход 4
Время вычислений на ГПУ (без обмена ЦПУ-ГПУ), с	**7,4**	**6,3**	**6**	**4**
Ускорение	**1**	**1,17**	**1,23**	**1,85**

Вопросы для обсуждения

- Какие имеются способы синхронизации всех потоков ГПУ?
- Как можно использовать асинхронность вызова ядра для повышения производительности программы?
- Какие недостатки присущи атомарным инструкциям?
- Для чего применяются барьеры памяти?
- Что произойдёт, если какой-либо поток из блока, выполняющего __syncthreads(), не выполнит данной функции?
- Какой метод использовался при оптимизации решения уравнения теплопроводности?

Темы заданий для самостоятельной работы

- Параллельная реализация итерационного метода Якоби для решения систем линейных алгебраических уравнений
- Параллельная реализация численного интегрирования
- Параллельная реализация явной разностной схемы для решения одномерного волнового уравнения
- Параллельная реализация явной разностной схемы для решения двумерного уравнения теплопроводности

Литература

- http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
- http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf
- http://www.steps3d.narod.ru/tutorials/cuda-tutorial.html
- Сандерс Дж., Кэндрот Э. Технология CUDA в примерах. Введение в программирование графических процессоров. М.: ДМК Пресс, 2011 г., ISBN 978-5-94074-504-4, 978-0-13-138768-3