Распараллеливание на графические процессоры тестов NAS NPB3.3.1 на языке Fortran DVMH

В.Ф. Алексахин, **В.А. Бахтин**, О.Ф. Жукова,
А.С. Колганов, В.А. Крюков, М.Н. Притула
Н.В. Поддерюгина, О.А. Савицкая, А.В. Шуберт
bakhtin@keldysh.ru

Институт прикладной математики им. М.В. Келдыша РАН

План доклада

- DVM-модель параллельного программирования
- Принципы расширения DVM-модели
- Основные возможности DVMH (DVM for Heterogeneous systems)
- Распараллеливание тестов NAS NPB
- Планы развития DVM-системы

DVM-модель параллельного программирования

- Объединяет достоинства модели параллелизма по данным и модели параллелизма по управлению (1993 г.)
- Базирующаяся на этих языках система разработки параллельных программ (DVM) создана в ИПМ им. М.В. Келдыша РАН
- Аббревиатура DVM (Distributed Virtual Memory, Distributed Virtual Machine) отражает поддержку виртуальной общей памяти на распределенных системах

Средства программирования

C-DVM = Язык Си + специальные прагмы

Fortran-DVM = Язык Фортран 95 + специальные комментарии

- Специальные комментарии и прагмы являются высокоуровневыми спецификациями параллелизма в терминах последовательной программы
- Отсутствуют низкоуровневые передачи данных и синхронизации
- Последовательный стиль программирования
- Спецификации параллелизма «невидимы» для стандартных компиляторов
- Существует только один экземпляр программы для последовательного и параллельного счета

Состав DVM-системы

DVM-система состоит из следующих компонент:

- Компилятор Fortran-DVM(H)
- Компилятор C-DVM(H)
- Библиотека поддержки LIB-DVM(H)
- DVM-отладчик
- Предсказатель выполнения DVM-программ
- Анализатор производительности DVM-программ

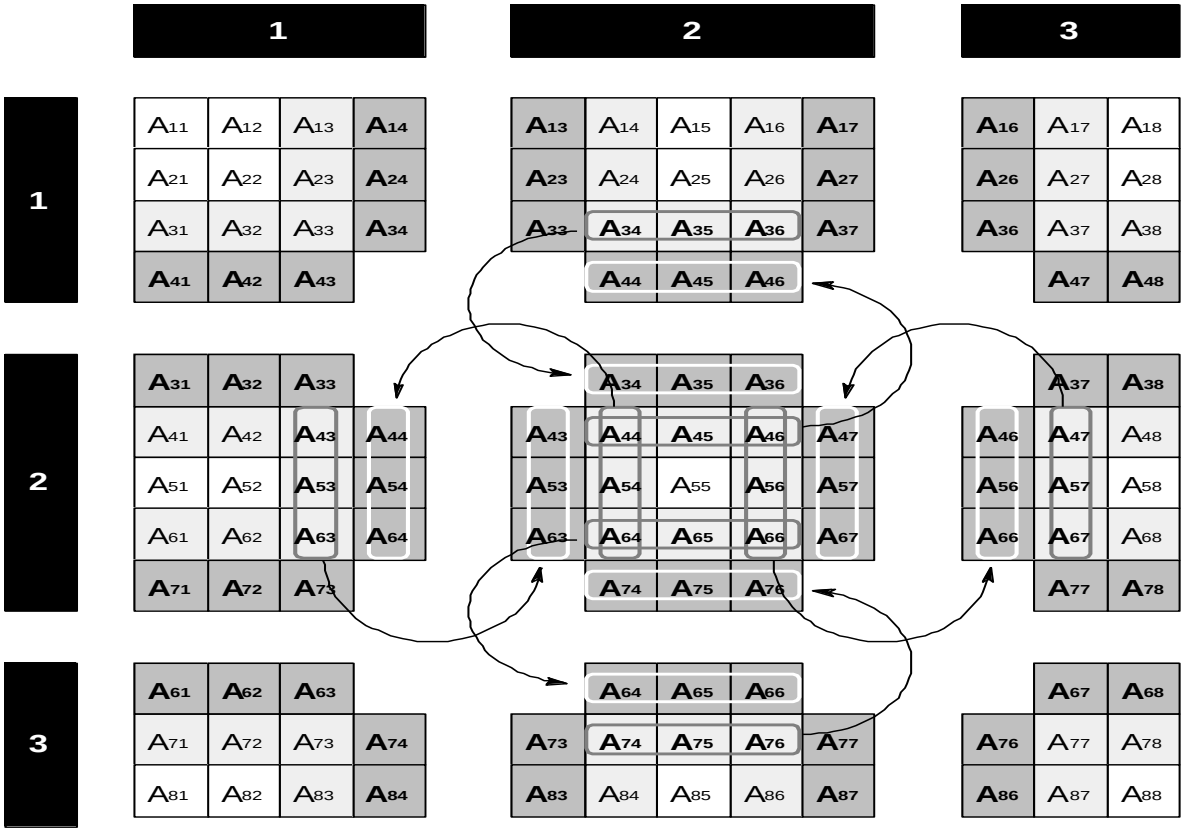
Спецификации параллельного выполнения программы

- Распределение элементов массива между процессорами
- Распределение витков цикла между процессорами
- Спецификация параллельно выполняющихся секций программы (параллельных задач) и отображение их на процессоры
- Организация эффективного доступа к удаленным (расположенным на других процессорах) данным
- Организация эффективного выполнения редукционных операций - глобальных операций с расположенными на различных процессорах данными (таких, как их суммирование или нахождение их максимального или минимального значения)

Алгоритм Якоби на языке Fortran

```
      PROGRAM    JACOB_SEQ
      PARAMETER    (L=4096,  ITMAX=100)
      REAL     A(L,L), B(L,L)
      PRINT *,  '**********  TEST_JACOBI   **********'
      DO IT  =  1,  ITMAX
          DO  J  =  2, L-1
              DO  I  =  2, L-1
                  A(I, J)  =  B(I, J)
              ENDDO
          ENDDO
          DO  J = 2,  L-1
              DO  I = 2,  L-1
                  B(I, J) =  (A(I-1, J) + A(I, J-1) +  A(I+1, J) +
     *                              A(I, J+1)) / 4
              ENDDO
           ENDDO
       ENDDO
       PRINT *,B
       END
```

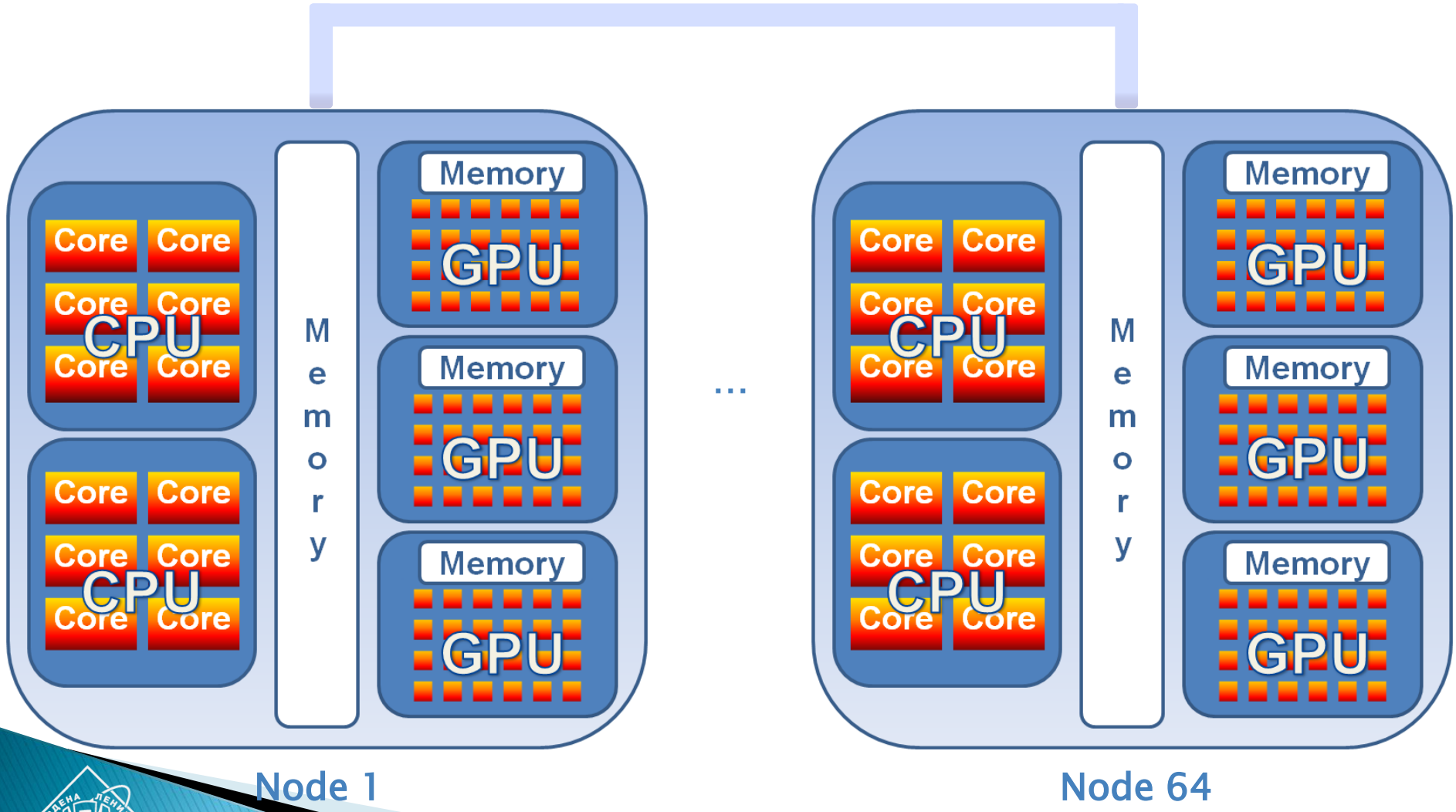
Распределение массива



```
      PROGRAM    JACOBY_DVM
      PARAMETER    (L=4096,  ITMAX=100)
      REAL     A(L,L), B(L,L)
!DVM$ DISTRIBUTE    ( BLOCK,   BLOCK)   ::   A
!DVM$ ALIGN  B(I,J)  WITH  A(I,J)
      PRINT *,  '**********  TEST_JACOBI   **********'
      DO IT  =  1,  ITMAX
!DVM$          PARALLEL (J,I) ON  A(I,  J)
               DO  J  =  2, L-1
                    DO  I  =  2, L-1
                         A(I, J)  =  B(I, J)
                    ENDDO
               ENDDO
!DVM$          PARALLEL (J,I) ON B(I,  J), SHADOW_RENEW (A)
               DO  J = 2,  L-1
                    DO  I = 2,  L-1
                         B(I, J) =  (A(I-1, J) + A(I, J-1) +  A(I+1, J) + A(I, J+1)) / 4
                    ENDDO
               ENDDO
      ENDDO
      PRINT *,B
      END
```

Алгоритм Якоби в модели DVM

Гибридная вычислительная система К-100



Принципы расширения DVM-модели

- Поддержка ускорителей различной архитектуры (GPU, Intel Xeon Phi(MIC)) в узлах кластера
- Гибкость в управлении распределением вычислений внутри узла кластера(между ускорителями и ядрами центрального процессора)
- (Полу-)автоматическое управление перемещением данных между оперативной памятью универсального процессора и памятями ускорителей

Проблема перемещения данных

- Два подхода
  - Ручное копирование
  - Указание входных и выходных данных
- Недостатки ручного копирования
  - Ориентированность на конкретный набор используемых вычислительных устройств
  - Проблема оптимизации копирований для разветвленных программ
    - Излишние перемещения
    - Сложно находимые ошибки

Основные возможности DVMH

- Определение *вычислительных регионов* (или просто регионов) - фрагментов программы, которые следует выполнять на том или ином ускорителе

!DVM$ REGION [clause {, clause}]

<region inner>

!DVM$ END REGION

Где <region inner>:

- Параллельный DVM-цикл
- Последовательная группа операторов
- Контрольная (хост) секция

Основные возможности DVMH

- Уточнение требуемых регионам данных и вида их использования (входные, выходные, локальные):

IN(subarray_or_scalar {, subarray_or_scalar})

OUT(subarray_or_scalar {, subarray_or_scalar})

INOUT(subarray_or_scalar {, subarray_or_scalar})

LOCAL(subarray_or_scalar {, subarray_or_scalar})

INLOCAL(subarray_or_scalar{,subarray_or_scalar})

Основные возможности DVMH

- Управление перемещением данных между оперативной памятью ЦПУ и памятью ускорителей при выполнении на ЦПУ фрагмента программы, не включенного в какой-либо регион

GET_ACTUAL[(subarray_or_scalar{,subarray_or_scalar})]
делает все необходимые обновления для того, чтобы на хост-памяти были самые новые данные в указанном подмассиве или скаляре.

ACTUAL[(subarray_or_scalar {, subarray_or_scalar})]
объявляет тот факт, что указанный подмассив или скаляр самую новую версию имеет в хост-памяти. При этом пересекающиеся части всех других представителей указанных переменных автоматически устаревают и перед использованием будут (по необходимости) обновлены.

Основные возможности DVMH

- Параллельный DVM-цикл - важнейшая часть вычислительного региона.

  !DVM$ PARALLEL clause {, clause}

  <DVM-loop nest>

  Кроме клауз DVM-цикла могут быть также заданы:

  PRIVATE(array_or_scalar {, array_or_scalar})

  Объявляет переменную приватной (локальной для каждого витка цикла).

  CUDA_BLOCK(X [, Y [, Z]])

  Указание размера блока нитей для вычислителя CUDA.

```
      PROGRAM    JACOBY_DVMH
      PARAMETER    (L=4096,  ITMAX=100)
      REAL     A(L,L), B(L,L)
!DVM$    DISTRIBUTE    ( BLOCK,   BLOCK)   ::   A
!DVM$    ALIGN  B(I,J)  WITH  A(I,J)
      PRINT *,  '**********  TEST_JACOBI   **********'
      DO IT  =  1,  ITMAX
!DVM$        REGION
!DVM$          PARALLEL (J,I) ON  A(I,  J)
               DO  J  =  2, L-1
                    DO  I  =  2, L-1
                        A(I, J)  =  B(I, J)
                    ENDDO
               ENDDO
!DVM$          PARALLEL (J,I) ON B(I,  J), SHADOW_RENEW (A)
               DO  J = 2,  L-1
                     DO  I = 2,  L-1
                          B(I, J) =  (A(I-1, J) + A(I, J-1) +  A(I+1, J) + A(I, J+1)) / 4
                     ENDDO
               ENDDO
!DVM$        END REGION
      ENDDO
!DVM$    GET_ACTUAL(B)
      PRINT *,B
      END
```

Алгоритм Якоби в модели DVMH

```
#define L 4096
#define ITMAX 100
#pragma dvm array A(distribute[block][block], shadow[1:1][1:1])
double A[L][L];
#pragma dvm array B(align([i][j] with A[i][j]))
double B[L][L];
int main(int argc, char *argv[]) {
   for(int it = 0; it < ITMAX; it++) {
      #pragma dvm region
      {
         #pragma dvm parallel([i][j] on A[i][j])
         for (int i = 1; I < L - 1; i++)
            for (int j = 1; j < L-1; j++)  A[i][j] = B[i][j];
         #pragma dvm parallel([i][j] on B[i][j]), shadow_renew(A)
         for (int i = 1; I < L - 1; i++)
            for (int j = 1; j < L - 1; j++)
               B[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i][j - 1] + A[i][j + 1]) / 4.;
      }
   }
   return 0;
}
```

Алгоритм Якоби в модели DVMH

Основные достоинства DVMH

- В качестве целевой архитектуры рассматривается кластер с гетерогенными узлами, а не отдельные узлы
- Перемещение информации между памятью ЦПУ и памятью ускорителей производится, в основном, не по директивам в программе, а автоматически в соответствии со спецификациями использования данных в регионах.

Это позволяет:

- динамически решать, где выгоднее выполнять тот или иной регион
- многократно выполнять регион для нахождения оптимального отображения вычислений на ГПУ
- сравнивать результаты выполнения региона на ЦПУ и ГПУ с целью обнаружения расхождений в результатах выполнения

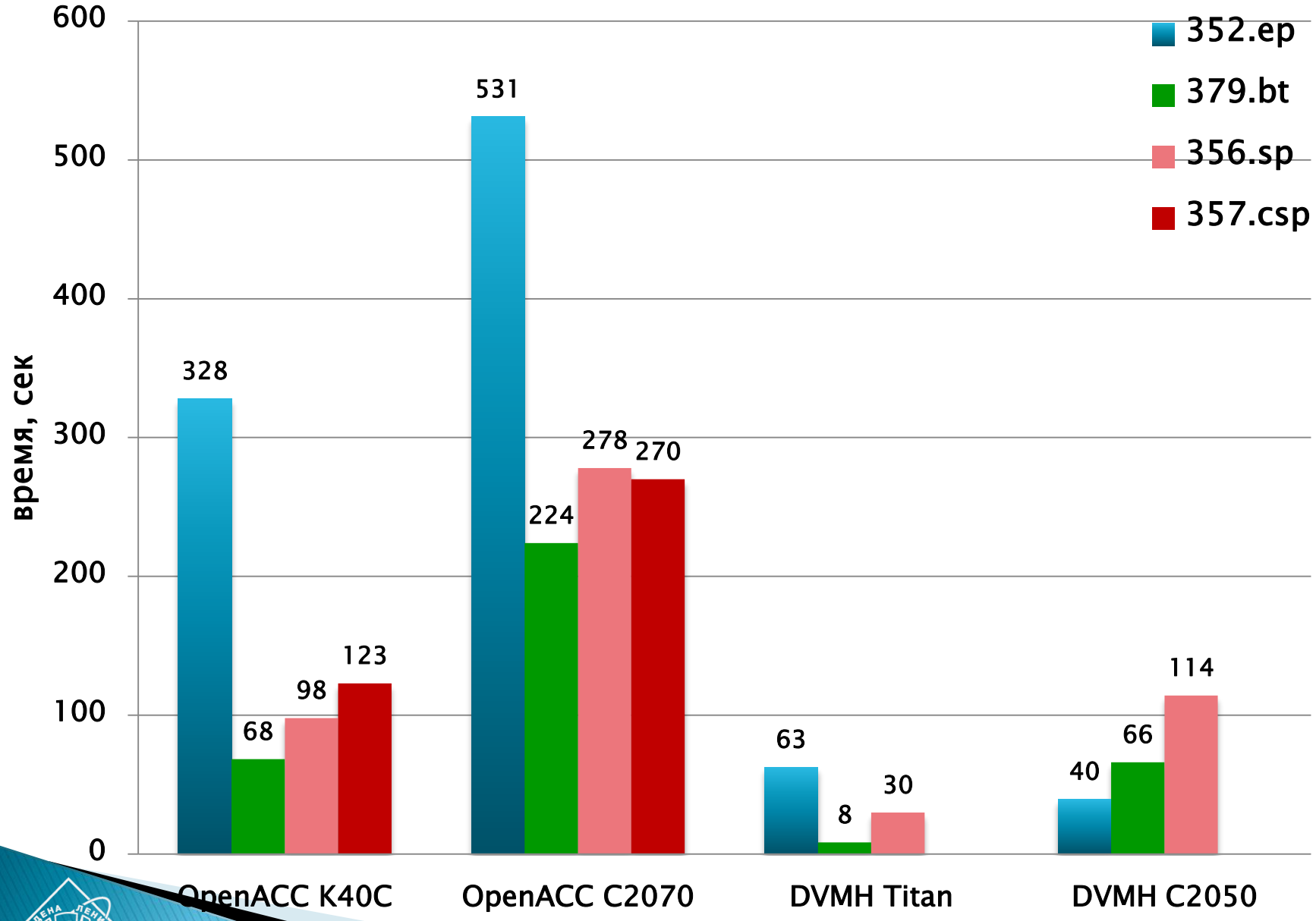
Распараллеливание тестов NAS

- **EP** - генерация пар случайных чисел Гаусса
- **MG** - приближенное решение трехмерного уравнения Пуассона. Метод MultiGrid
- **BT** - 3D Навье-Стокс, блочная трехдиагональная схема. Метод переменных направлений
- **LU** - 3D Навье-Стокс. Метод последовательной верхней релаксации
- **SP** - 3D Навье-Стокс. Скалярная пятидиагональная схема. Beam-Warning approximate factorization

Тесты NAS для GPU

- OpenCL-версии
  - Center for Manycore Programming at Seoul National University (SNU NPB Suite)
- CUDA-версии
  - Chemnitz University of Technology (BT, LU, SP)
  - Laercio Lima Pilla from the Federal University of Rio Grande do Sul (EP, CG, FT)
- OpenACC-версии
  - SPEC ACCEL V1.0 (BT, EP, CG, SP)

Времена выполнения OpenACC и DVMH-версий тестов NAS NPB



Оптимизация работы с памятью

- Вынесение часто используемых элементов массива в теле цикла в скалярные переменные
- Сокращение операций чтения из глобальной памяти GPU за счет избыточных вычислений
- Динамическое переупорядочивание массивов в памяти GPU

```
!DVM$ PARALLEL (K,J,I) ON U(I,J,K,*), PRIVATE(M1,M2,M)
DO K = 2,NZ-1
  DO J = 2,NY-1
     DO I = 2,NX-1
        M1 = 2
        M2 = 3
        DO M = 1,5
           U(I,J,K,M) = U(I,J,K,M1) + U(I,J,K,M2)
        ENDDO
        DO M = 1,5
           U(I,J,K,M) = U(I,J,K,M1+1) + U(I,J,K,M2+1)
        ENDDO
        DO M = 1,5
           U(I,J,K,M) = U(I,J,K,M1-1) + U(I,J,K,M2-1)
        ENDDO
     ENDDO
  ENDDO
ENDDO
```

Фрагмент программы LU

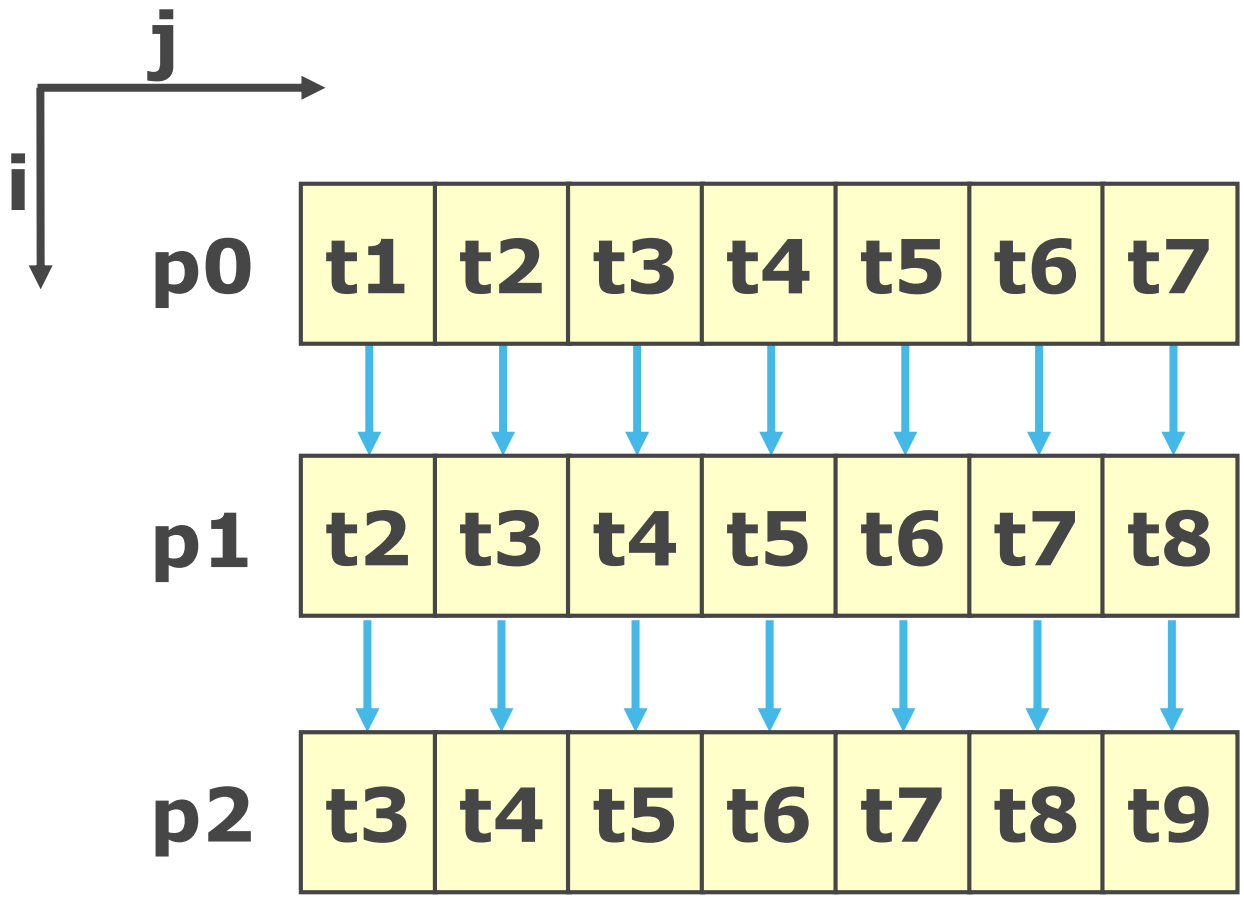
```
!DVM$ PARALLEL (K,J,I) ON U(I,J,K,*), PRIVATE(M1,M2,M,U_)
DO K = 2,NZ-1
  DO J = 2,NY-1
    DO I = 2,NX-1
      DO M=1,5
        U_(M) = U(I,J,K,M)
      ENDDO
      M1 = 2; M2 = 3
      DO M = 1,5
        U_(M) = U_(M1) + U_(M2)
      ENDDO
      DO M = 1,5
        U_(M) = U_(M1+1) + U_(M2+1)
      ENDDO
      DO M = 1,5
        U_(M) = U_(M1-1) + U_(M2-1)
      ENDDO
      DO M=1,5
        U(I,J,K,M) = U_(M)
       ENDDO
    ENDDO
  ENDDO
ENDDO
```

Использование приватных переменных для часто используемых элементов массивов, которые отображаются компилятором NVCC на регистры.

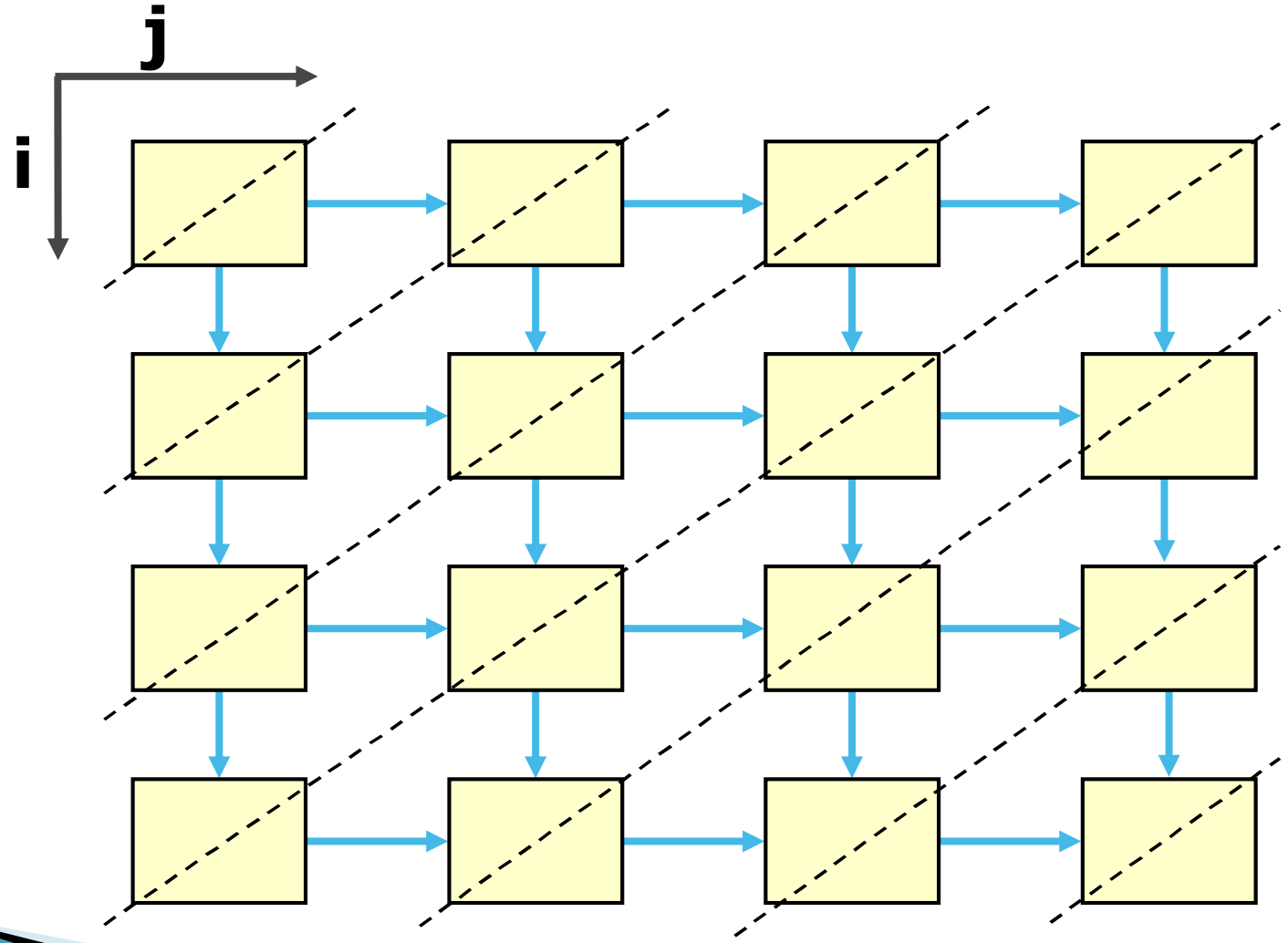
Алгоритм SOR на языке Fortran

```
      PROGRAM    SOR_DVMH
      PARAMETER    (L=1000,  ITMAX=20, W = 0.5)
      REAL  A(L,L) ,  EPS, S
!DVM$   DISTRIBUTE A(BLOCK,BLOCK)
      PRINT *,  '**********  TEST_SOR   **********'
      DO IT = 1, ITMAX
        EPS = 0.
!DVM$    ACTUAL(EPS)
!DVM$    REGION
!DVM$    PARALLEL(J, I) ON A(I, J), ACROSS(A(1:1,1:1)),
!DVM$&   REDUCTION(MAX(EPS)), PRIVATE(S)
        DO J = 2, L-1
          DO I = 2, L-1
            S = A(I, J)
            A(I, J) = (W / 4) * (A(I-1, J) + A(I+1, J) + A(I, J-1) +
     &          A(I, J+1)) + ( 1-W ) * A( I, J)
            EPS = MAX ( EPS,  ABS( S - A( I, J )))
          ENDDO
        ENDDO
!DVM$    END_REGION
!DVM$    GET_ACTUAL(EPS)
        PRINT 200,  IT, EPS
200     FORMAT(' IT = ',I4, '   EPS = ', E14.7)
      ENDDO
      END
```

Конвейрный параллелизм (dvm run 3 1 sor)



Параллелизм по гиперплоскостям (dvm run 4 4 sor)



Реализация для GPU

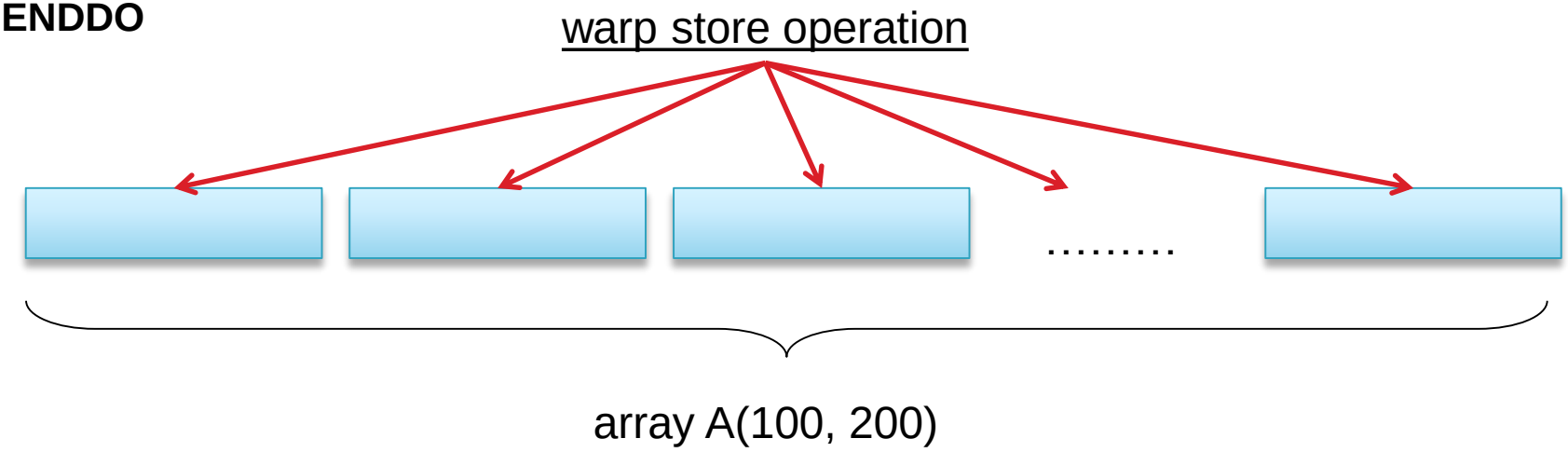
- Зависимость по 1 измерению - его сериализация
- Зависимость по 2,3 измерениям - выполнение гиперплоскостями
- Зависимость по 4 и более - комбинация методов
- Изменение порядка выполнения витков - изменение шаблона доступа в память
- Переупорядочивание элементов массивов

Зависимость по 1-му измерению

```
REAL  A(100, 200)
!DVM$ DISTRIBUTE A(BLOCK,BLOCK)

!DVM$ PARALLEL(K,I) on A(I,K),
!DVM$& ACROSS( A(1:1, 0:0) )
DO K = 1, 200
   DO I = 1, 100
      A(I, K) = …A(I-1,K)…A(I+1,K)
    ENDDO
ENDDO
```

```
DO I = 1, 100
   A(I, GLOBAL_IDX_X) = …
ENDDO
```



Преобразование цикла в DVMH-программе

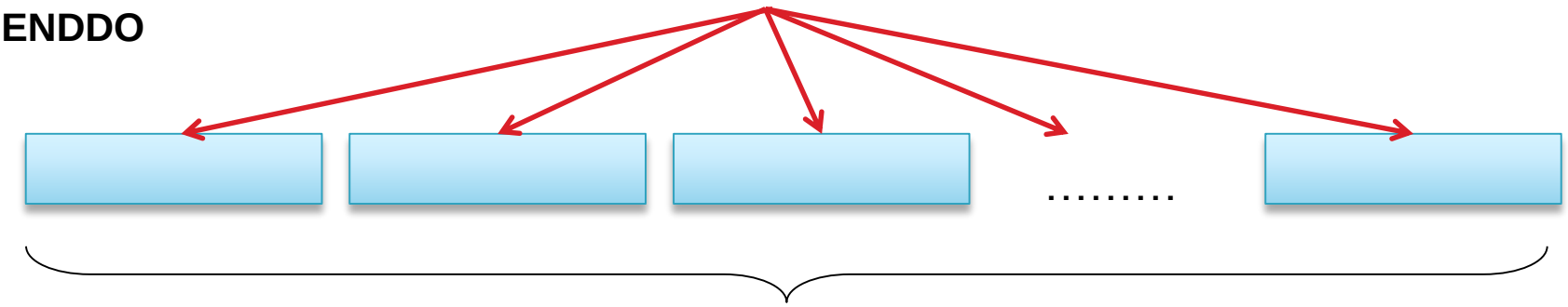
```
REAL  A(100, 200)
!DVM$ DISTRIBUTE A(BLOCK,BLOCK)

!DVM$ PARALLEL(K,I) on A(I,K),
!DVM$& ACROSS( A(1:1, 0:0) )
DO K = 1, 200
   DO I = 1, 100
      A(I, K) = …A(I-1,K)…A(I+1,K)
   ENDDO
ENDDO
```

DVMH cuda kernel

```
...
A(I * A_1 +  GLOBAL_IDX_X * A_2) =
…
```

warp store operation



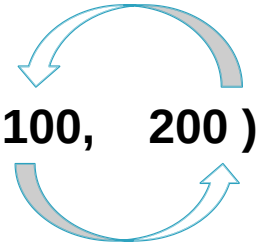
array A(100, 200)

Оптимизация обращений в память GPU в DVMH

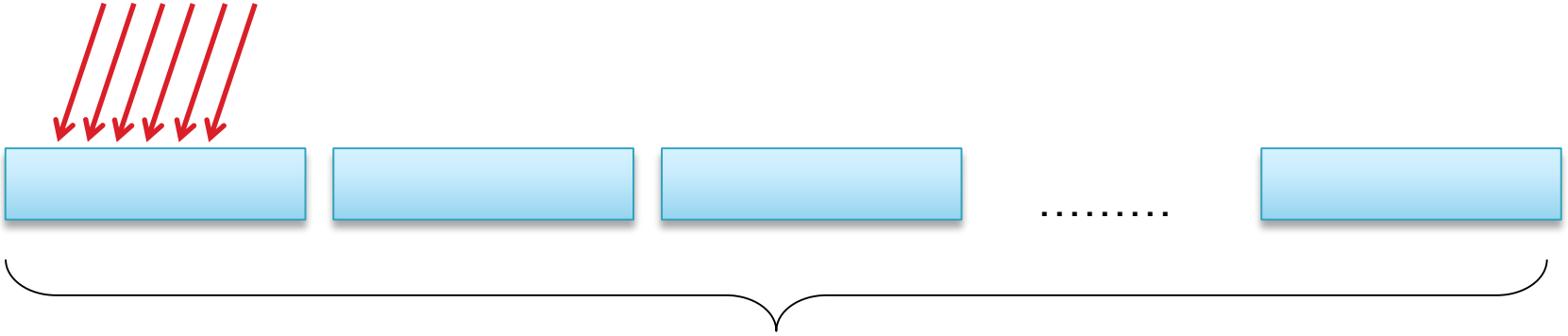
DVMH CUDA kernel

```
...
A(I * A_1 +  GLOBAL_IDX_X * A_2) =
…
...
```

DVMH Dynamic reordering

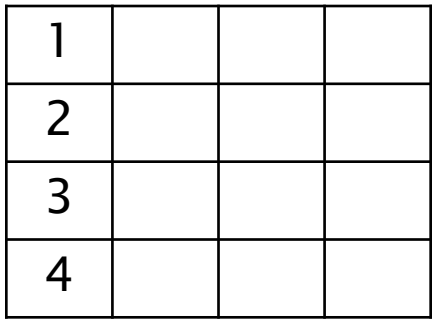


warp store operation – необходима одна транзакция на 32 нити



Зависимость по 2,3 измерениям

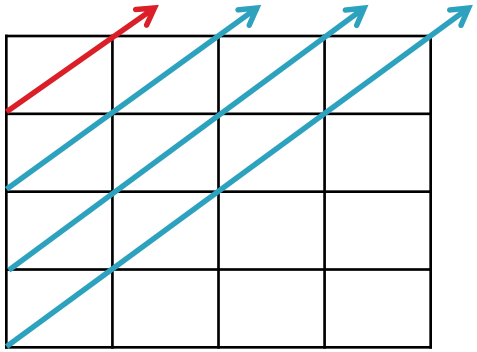
REAL A(4, 4)



```
!$DVM PARALLEL (K,I) ON A(I,K),
            ACROSS(A(1:1, 1:1))
DO K = 1,4
  DO I = 1,4
     A(I, K) = A(I+1,K) + A(I-1,K) + A(I,K+1)+ …
  ENDDO
ENDDO
```

Выполнение гиперплоскостями (Шаг 1)

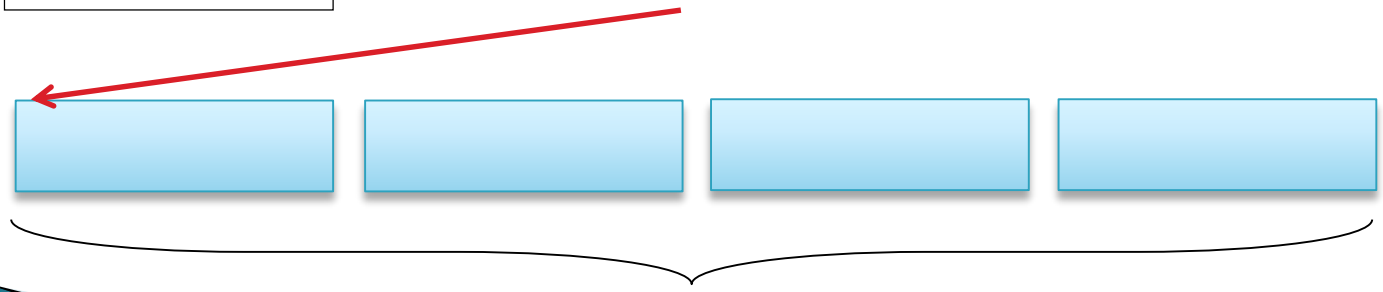
REAL A(4, 4)



```
!$DVM PARALLEL (K,I) ON A(I,K),
          ACROSS(A(1:1, 1:1))
DO K = 1,4
  DO I = 1,4
    A(I, K) = A(I+1,K) + A(I-1,K) + A(I,K+1)+ …
  ENDDO
ENDDO
```

1 диагональ

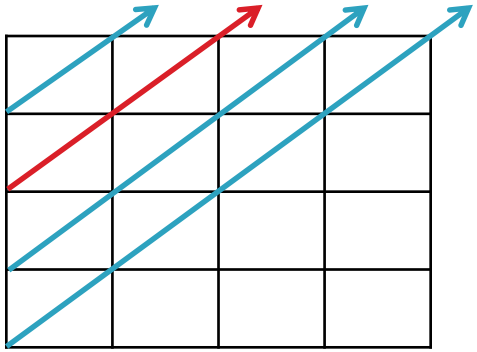
warp load/store operations



Array A(4, 4)

Выполнение гиперплоскостями (Шаг 2)

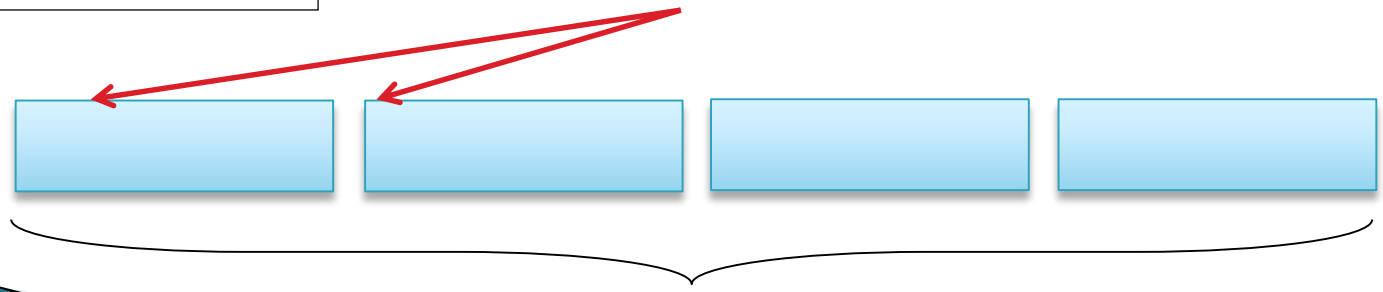
REAL A(4, 4)



```
!$DVM PARALLEL (K,I) ON A(I,K),
          ACROSS(A(1:1, 1:1))
DO K = 1,4
  DO I = 1,4
    A(I, K) = A(I+1,K) + A(I-1,K) + A(I,K+1)+ …
  ENDDO
ENDDO
```

2 диагональ

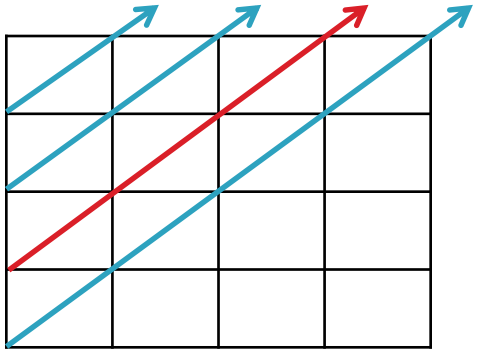
warp load/store operations



Array A(4, 4)

Выполнение гиперплоскостями (Шаг 3)

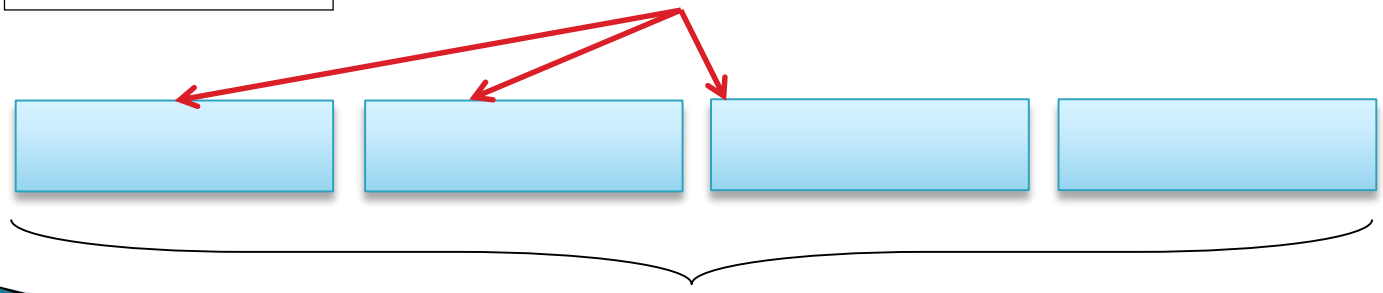
REAL A(4, 4)



```
!$DVM PARALLEL (K,I) ON A(I,K),
          ACROSS(A(1:1, 1:1))
DO K = 1,4
  DO I = 1,4
    A(I, K) = A(I+1,K) + A(I-1,K) + A(I,K+1)+ …
  ENDDO
ENDDO
```

3 диагональ

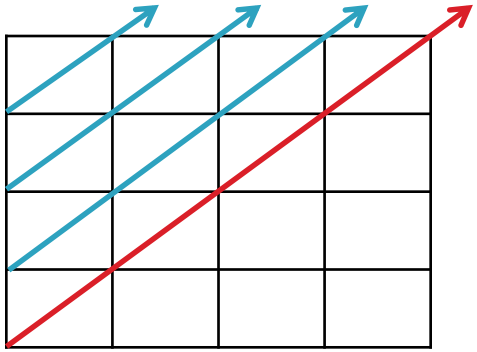
warp load/store operations



Array A(4, 4)

Выполнение гиперплоскостями (Шаг 4)

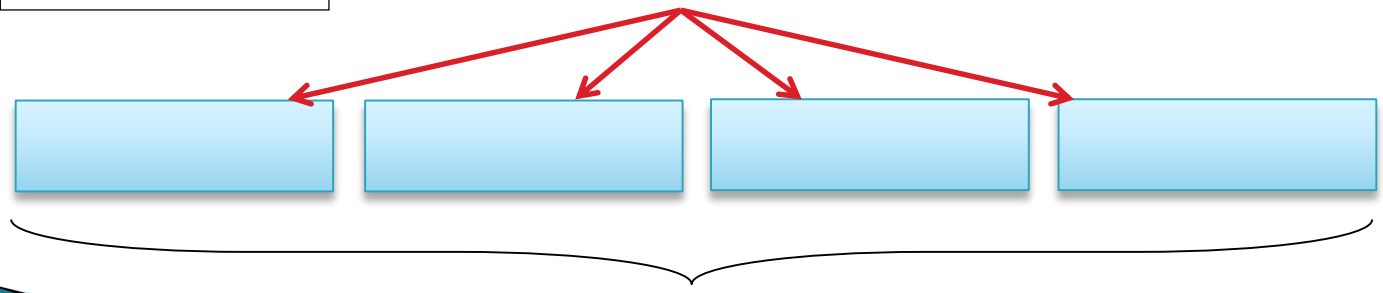
REAL A(4, 4)



```
!$DVM PARALLEL (K,I) ON A(I,K),
          ACROSS(A(1:1, 1:1))
DO K = 1,4
  DO I = 1,4
    A(I, K) = A(I+1,K) + A(I-1,K) + A(I,K+1)+ …
  ENDDO
ENDDO
```

4 диагональ

warp load/store operations



Array A(4, 4)

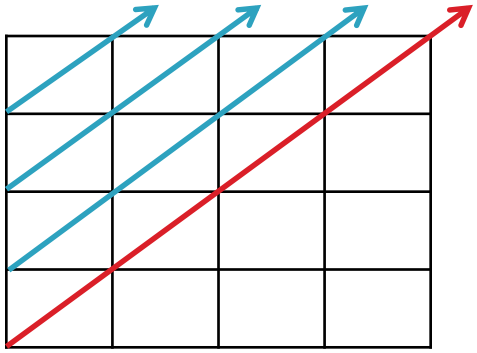
Динамическое переупорядочивание для 2x и 3x зависимых измерений

1			
2			
3			
4			

1	3	6	
2	5		
4			

Выполнение гиперплоскостями (Шаг 4)

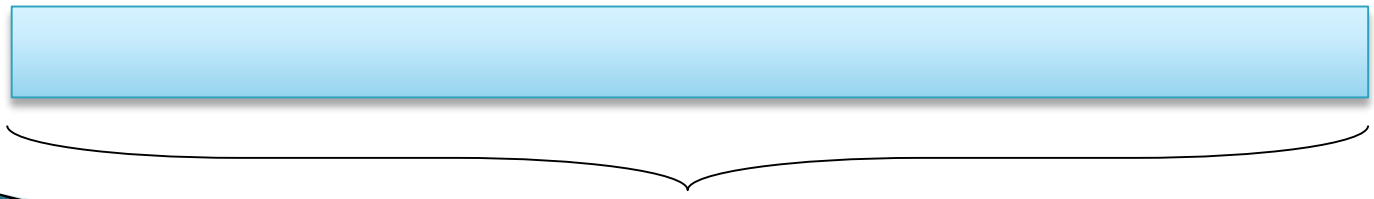
REAL A(4, 4)



```
!$DVM PARALLEL (K,I) ON A(I,K),
          ACROSS(A(1:1, 1:1))
DO K = 1,4
  DO I = 1,4
    A(I, K) = A(I+1,K) + A(I-1,K) + A(I,K+1)+ …
  ENDDO
ENDDO
```

4 диагональ

warp load/store operations **with DVMH DR** **(-autoTfm)**

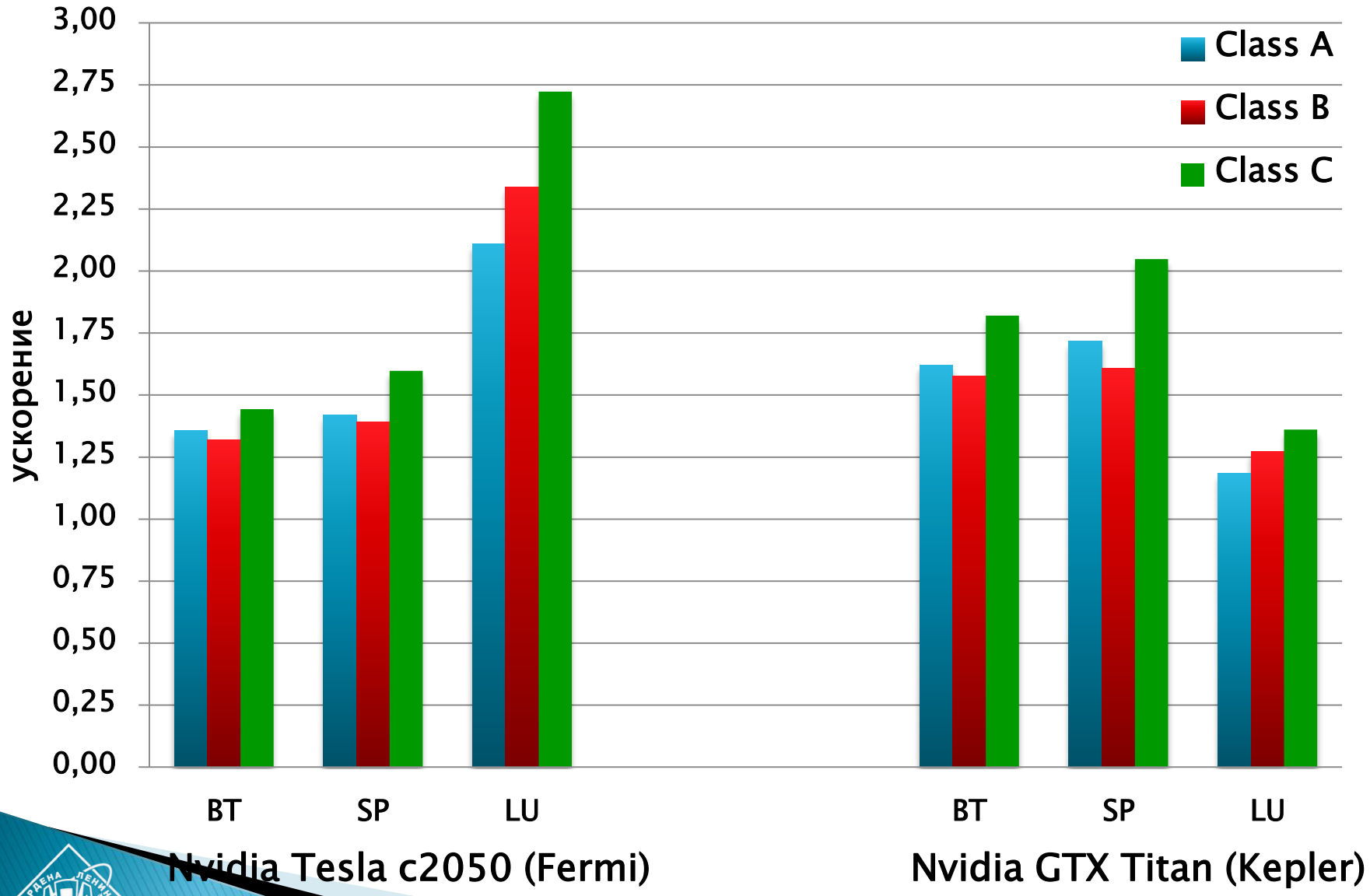


Array A(4, 4)

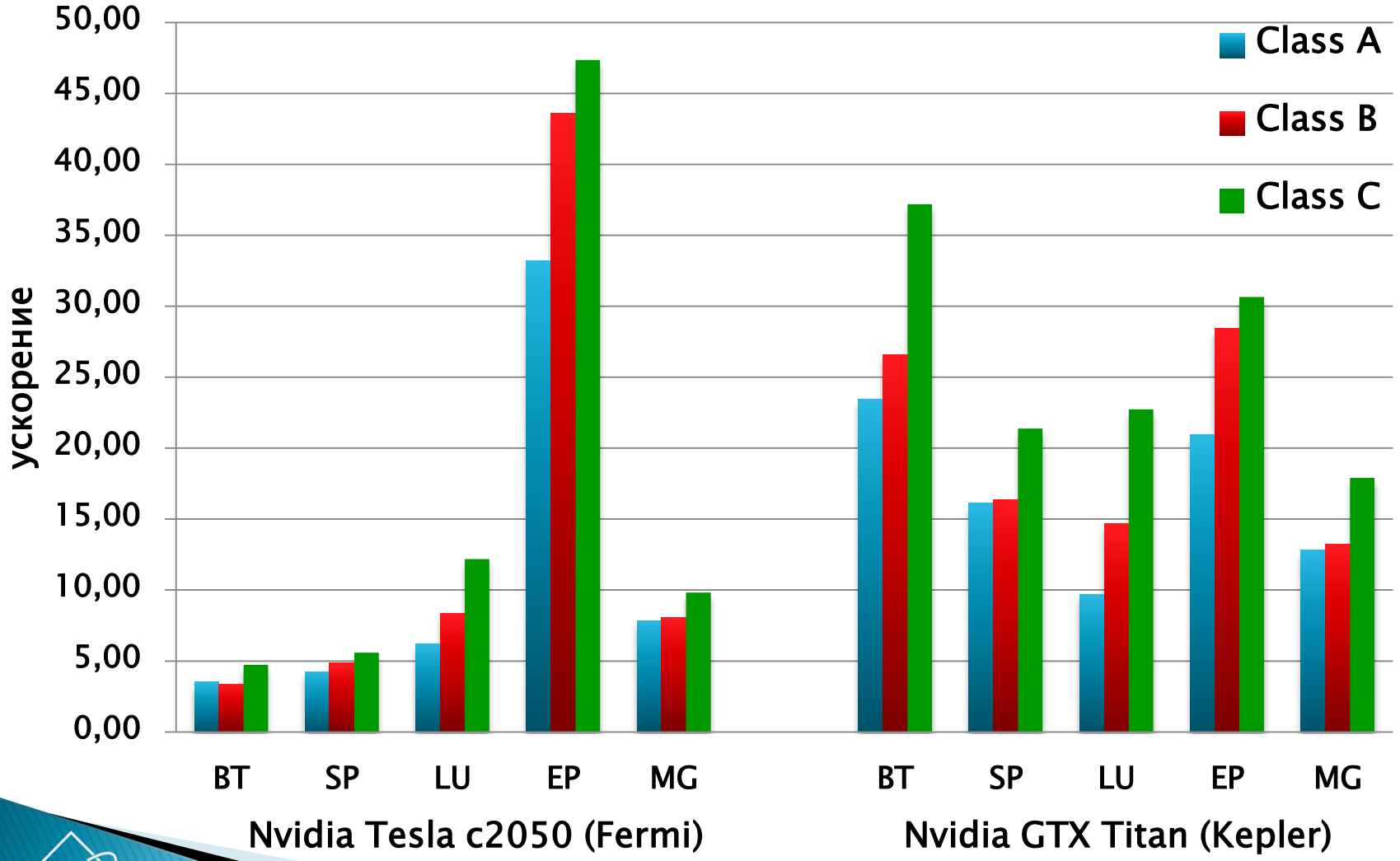
Переупорядочивание массивов

- Никаких дополнительных указаний в DVMH-программе
- Работает в динамике
- Для каждого цикла для каждого массива выбирает лучший порядок элементов
- Поддержка диагонализированных представлений
- Не происходит возврата состояния в конце цикла, только переход в требуемое

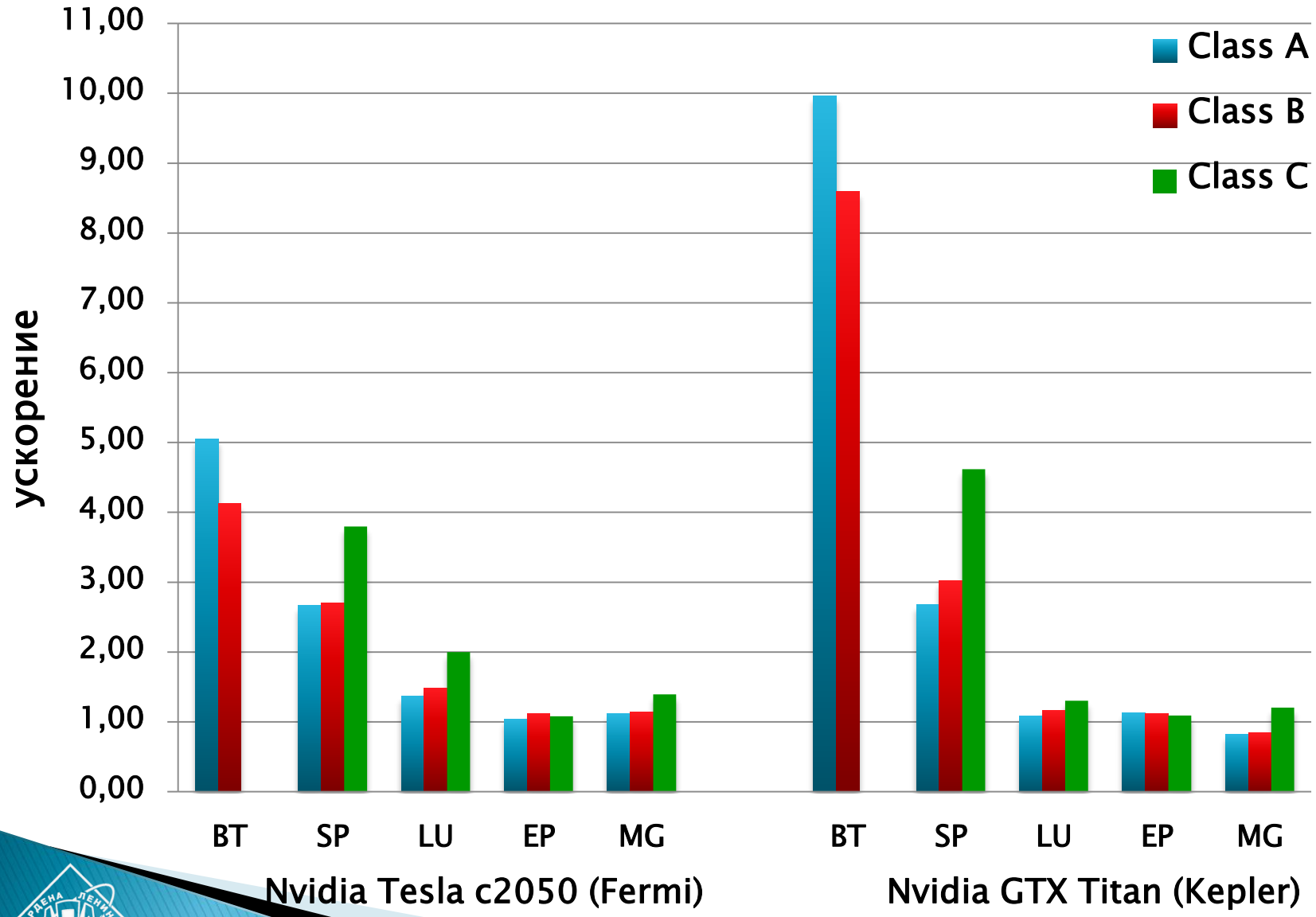
Ускорение выполнения DVMH-версий программ в результате выполнения динамического переупорядочивания массивов



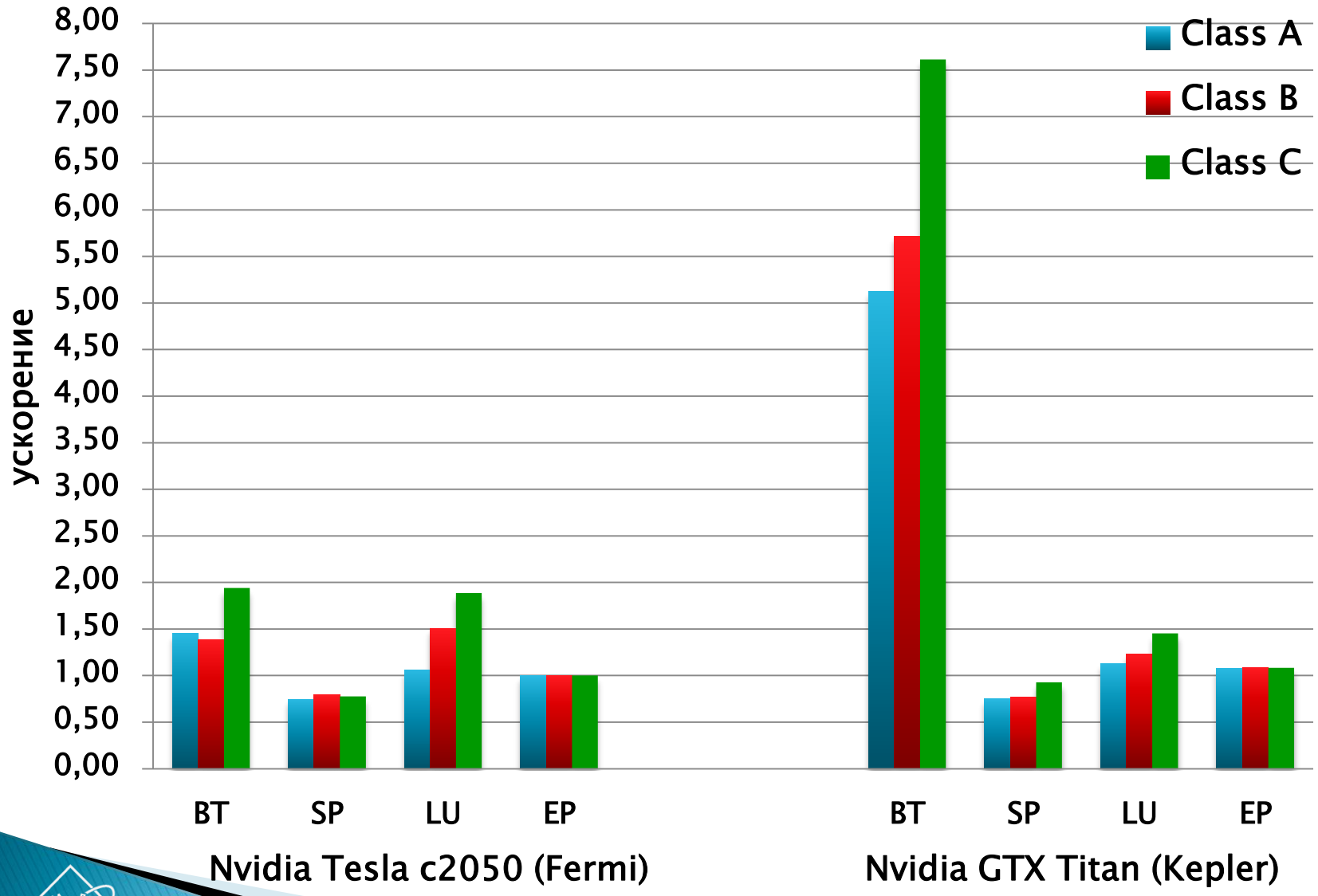
Ускорение выполнения DVMH-версий тестов NAS NPB по сравнению с последовательными версиями тестов на CPU Intel Xeon 5670



Ускорение выполнения DVMH-версий тестов NAS NPB по сравнению с OpenCL-версиями



Ускорение выполнения DVMH-версий тестов NAS NPB по сравнению с CUDA-версиями



Выводы

- Эффективность программ, написанных в высокоуровневой модели DVMH, близка к эффективности программ, написанных с использованием низкоуровневой модели CUDA, и превосходит эффективность программ в моделях OpenCL и OpenACC
- Проблема эффективного отображения параллельных программ на вычислительные системы различной архитектуры требует дальнейших исследований

Планы развития DVM-системы

- Поддержка неструктурных сеток и разреженных матриц
- Исследование эффективности отображения DVMH-программ на сопроцессоры Intel Xeon Phi
- Доработка системы отладки эффективности для DVMH-программ

Вопросы, замечания?

СПАСИБО !

http://www.keldysh.ru/dvm
dvm@keldysh.ru

Алгоритм Якоби

- Двумерный 5002x5002, два цикла в регионе + редукция
  - 1 ГПУ – 57 с.
  - 1 ГПУ + 4 ядра ЦПУ – 49 с.
- Трехмерный 502x252x252, один цикл в регионе
  - 1 ГПУ – 14,7 с.
  - 1 ГПУ + 4 ядра ЦПУ – 12,6 с.