Методы динамической настройки DVMH-программ на кластеры с ускорителями

В.А. Бахтин, В.А. Крюков,
А.С. Колганов, Н.В. Поддерюгина, М.Н. Притула

bakhtin@keldysh.ru

Институт прикладной математики им. М.В. Келдыша РАН

План доклада

- DVMH-модель параллельного программирования
- Преимущества DVMH-модели
- Методы динамической настройки DVMH-программ
- Планы развития DVM-системы

```
FILE *f;
#pragma dvm array distribute[block][block] shadow[1:1][1:1]
double A[L][L];
#pragma dvm array 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 inout (A,B)
        {
            #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.;
        }
    }
    f = fopen("jacobi.dat", "wb");
    #pragma dvm get_actual(B)
    fwrite(B, sizeof(double), L * L, f);
    fclose(f);
    return 0;
}
```

Алгоритм Якоби в модели DVMH

Средства программирования

C-DVMH = Язык Си + специальные прагмы

Fortran-DVMH = 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 единственный экземпляр программы для последовательного и параллельного счета

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/ускорителях) данным

Спецификации параллельного выполнения программы

- Организация эффективного выполнения редукционных операций - глобальных операций с расположенными на различных процессорах/ускорителях данными (таких, как их суммирование или нахождение их максимального или минимального значения)
- Определение фрагментов программы (регионов) для возможного выполнения на ускорителях
- Управление перемещением данных между памятью ЦПУ и памятью ускорителей

Состав DVM-системы

Система состоит из следующих компонент:

- Компилятор Fortran-DVMH
- Компилятор C-DVMH
- Библиотека поддержки LIB-DVMH
- DVMH-отладчик
- Анализатор производительности DVMH-программ

XcalableACC=XcalableMP+OpenACC

RIKEN Advanced Institute for Computational Science
University of Tsukuba
The University of Tokyo
http://www.xcalablemp.org/download/publication/2014/WACCPD.pdf

Omni XMP Compiler http://omni-compiler.org

```
FILE *f;
#pragma  xmp nodes p(N, N)
#pragma  xmp template t(0:L−1,  0:L−1)
#pragma  xmp distribute t(block,  block)
#pragma  xmp align [i][j] with  t(i,j) :: A,B
#pragma  xmp shadow  A[1:1]
double A[L][L];
double B[L][L];
```

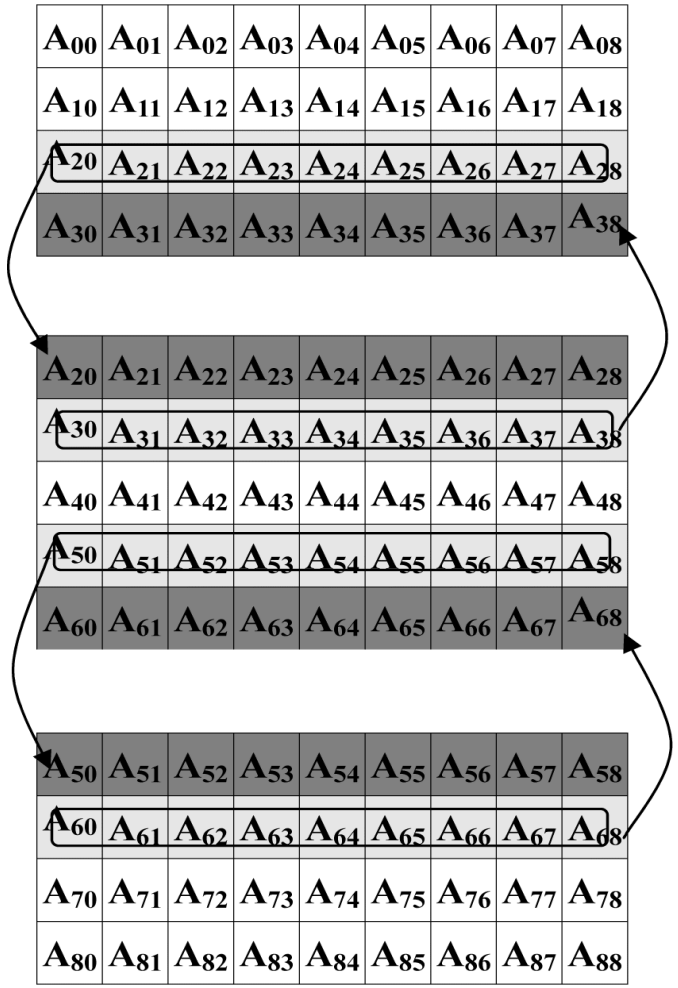
Алгоритм Якоби в модели XACC

```
int main(int argc, char *argv[]) {
  #pragma  acc  data copy(B) copyin(A)
  {
   for(int it = 0; it < ITMAX; it++) {
          #pragma  xmp  loop  (i,j)  on  t(i,j)
          #pragma  acc  parallel  loop  collapse(2)
         for (int i = 1; I < L - 1; i++)
           for (int j = 1; j < L-1; j++)  A[i][j] = B[i][j];
         #pragma  xmp  reflect (A) acc
         #pragma  xmp  loop  (i,j)  on  t(i,j)
         #pragma  acc  parallel  loop  collapse(2)
         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;
}
```

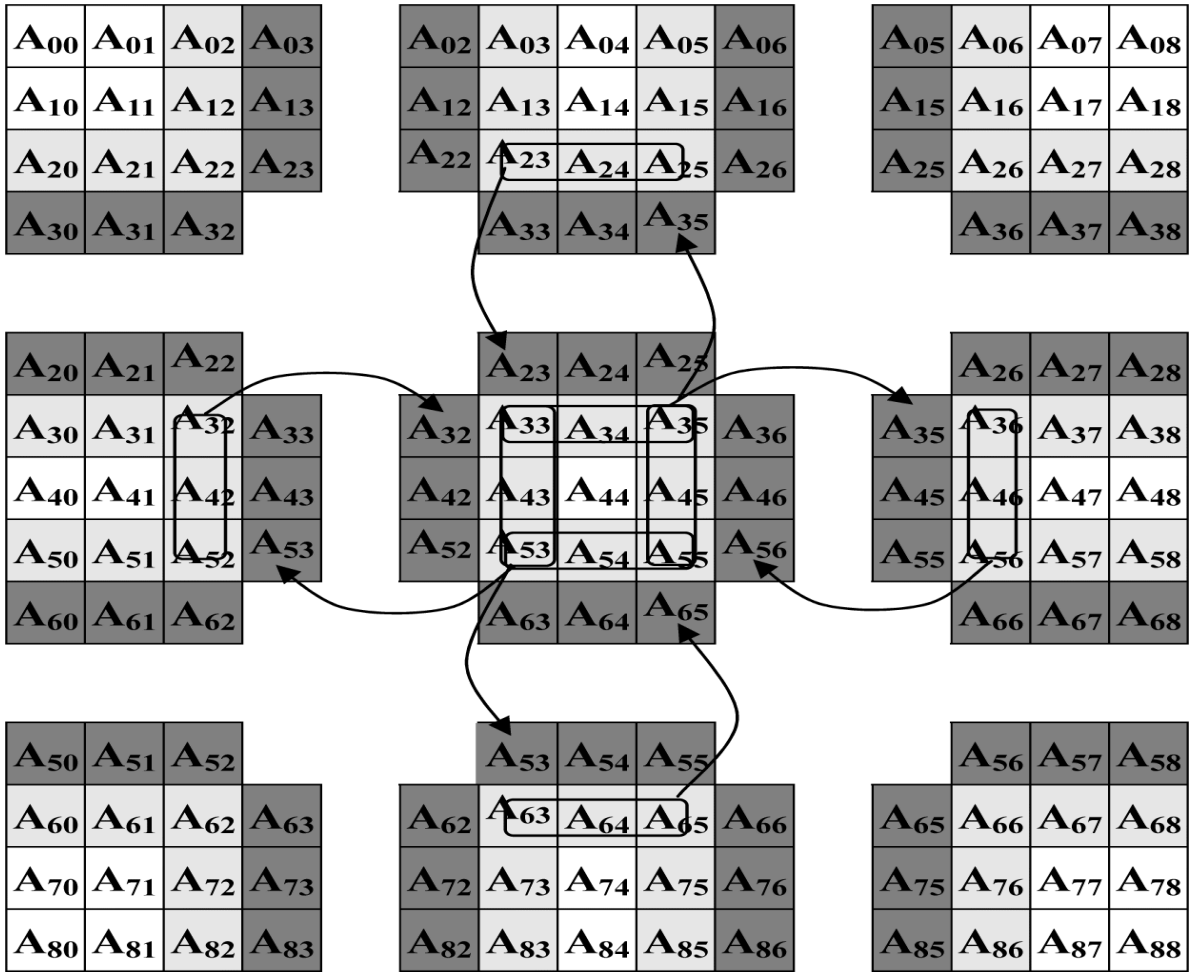
Алгоритм Якоби в модели ХАСС

Распределение массива

./dvm run 3 1 jac



./dvm run 3 3 jac



Отображение подзадач на узлы кластера в модели DVM

! описание массива виртуальных процессоров

!DVM$ PROCESSORS MVS10P(NUMBER_OF_PROCESSORS( ))

! описание массива задач

!DVM$ TASK MB (2)

! определение числа виртуальных процессоров

NPT = NUMBER_OF_PROCESSORS( ) / 2

! явное отображение задач по виртуальным процессорам

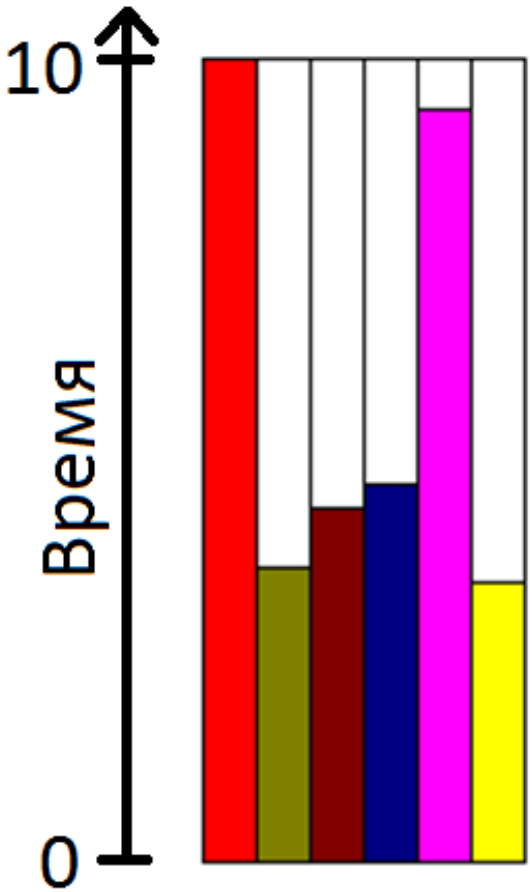
!DVM$ MAP MB(1) ONTO MVS10P(1 : NPT)

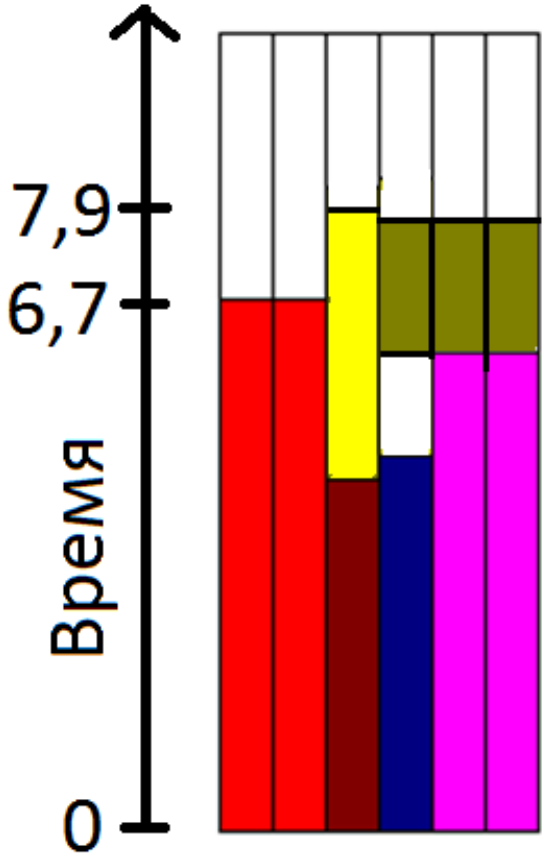
!DVM$ MAP MB(2) ONTO MVS10P(NPT+1 : 2*NPT)

Расчет дозвукового обтекания летательного аппарата

Задача 810 областей	средняя загрузка	Max загрузка
75 процессоров	19258	19296
128 процессоров	11284	11648
256 процессоров	5642	11648
384 процессоров	3761	11648
512 процессоров	2821	11648

Распределение подзадач в модели DVMH

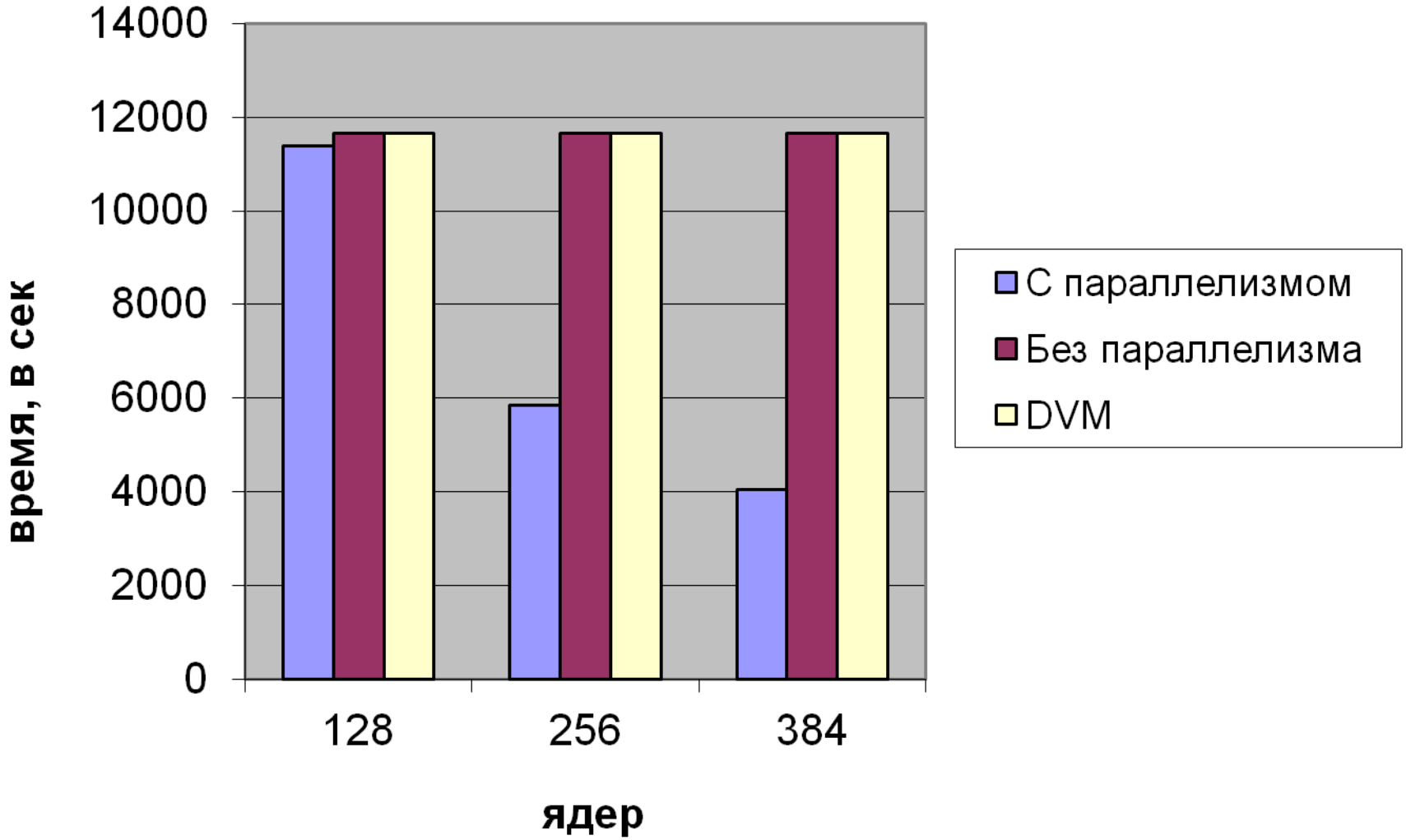




Автоматическое распределение подзадач в модели DVMH

```
!DVM$ TASK MB ( MXBL )
 REAL*8 PARMB(4, MXBL)        ! вспомогательный массив
 DO IB = 1, NBL
    PARMB(1, IB) = SIZE(1, IB)  ! сложность подзадачи
    PARMB(2, IB) = 1            ! MIN число процессоров
    PARMB(3, IB) = 0            ! MAX число процессоров
    PARMB(4, IB) = 0.9          ! доля параллельных
 END DO                         ! вычислений
!DVM$ MAP MB(:) BY PARMB ! автоматическое
                         ! распределение позадач
```

Расчет дозвукового обтекания летательного аппарата



Методы динамической настройки DVMH-программ

- Отображение массивов и циклов на узлы кластера с учетом их производительности
- Отображение массивов и циклов на устройства узла с учетом их производительности
- Трансформация массивов
- Использование динамической компиляции

Управление выполнением DVMH-программы

Более 20 переменных окружения:

export DVMH_PPN=‘2,1,1’ # Number of process per node

export DVMH_NUM_THREADS=‘8,240,240’ # Number of CPU threads per process

export DVMH_NUM_CUDAS=‘3’ # Number of GPUs per process

export DVMH_CPU_PERF=‘’ # Performance of all cores of CPU per process

export DVMH_CUDAS_PERF=‘’ # Performance of each GPU per device

export DVMH_SCHED_TECH=‘dynamic1’ # Schedule mode

export DVMH_SET_AFFINITY=‘enable’ # Thread affinity control

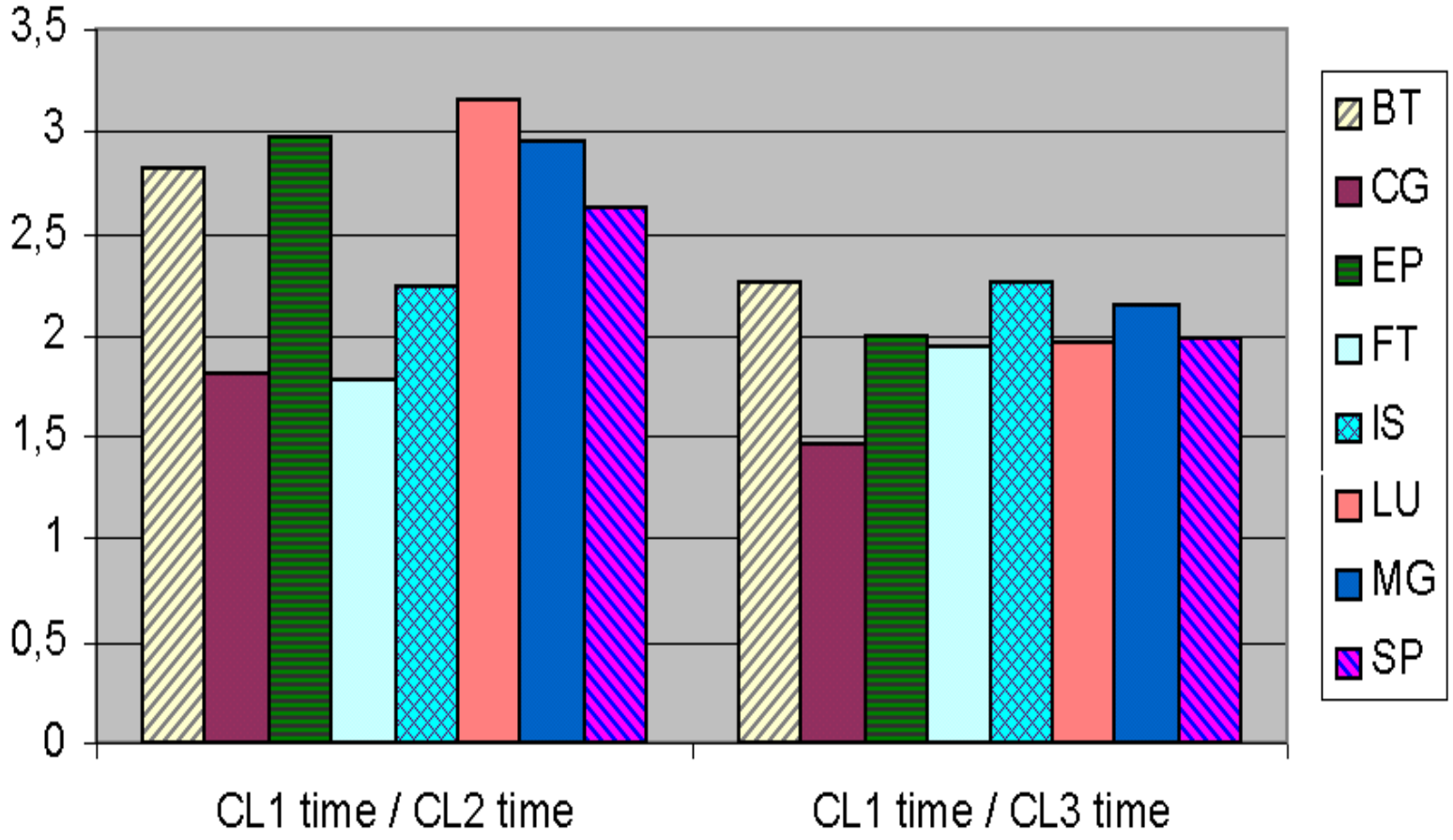
export DVMH_NO_DIRECT_COPY=‘1’ # Don’t use GPUDirect transfers

export DVMH_IO_BUF_SIZE=‘10485760’ # Size of input/output buffer

Методы динамической настройки DVMH-программ

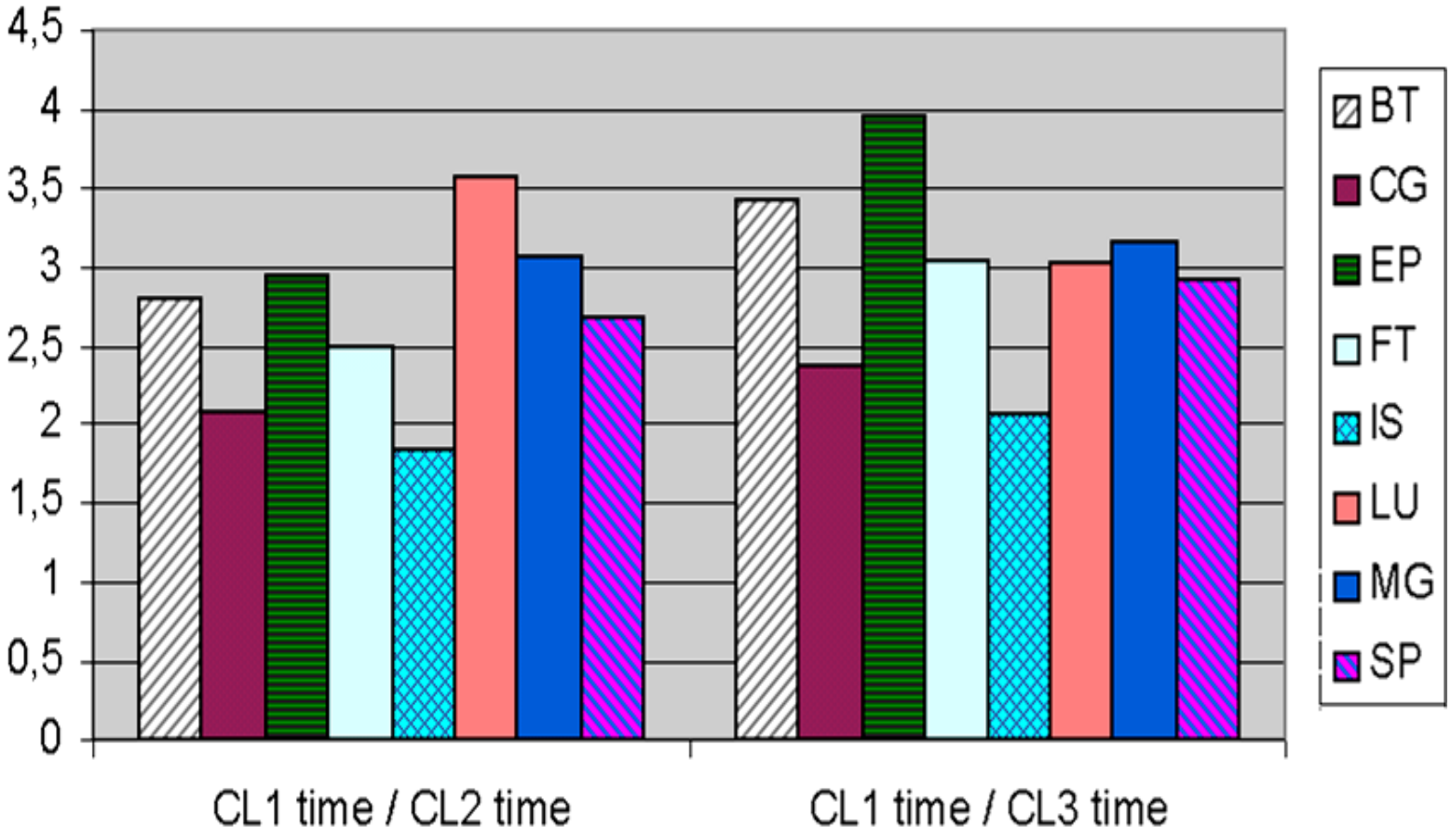
- Отображение массивов и циклов на узлы кластера с учетом их производительности
- Отображение массивов и циклов на устройства узла с учетом их производительности
- Трансформация массивов
- Использование динамической компиляции

Ускорение выполнения MPI-версий тестов NAS NPB на неоднородном кластере



CL1 – 128 процессоров со скоростью выполнения P,
CL2 – 128 процессоров со скоростью выполнения 3P,
CL3 – 128 процессоров со скоростью выполнения P и 128 процессоров со скоростью выполнения 3P.

Ускорение выполнения DVMH-версий тестов NAS NPB на неоднородном кластере



CL1 – 128 процессоров со скоростью выполнения P,
CL2 – 128 процессоров со скоростью выполнения 3P,
CL3 – 128 процессоров со скоростью выполнения P и 128 процессоров со скоростью выполнения 3P.

Методы динамической настройки DVMH-программ

- Отображение массивов и циклов на узлы кластера с учетом их производительности
- Отображение массивов и циклов на устройства узла с учетом их производительности
- Трансформация массивов
- Использование динамической компиляции

Режимы распределения данных и вычислений по вычислительным устройствам

- Простой статический режим
- Динамический режим с подбором схемы распределения
- Динамический режим с использованием подобранной схемы распределения

Простой статический режим

- В каждом регионе одинаковое распределение по устройствам
- Задается пользователем в виде вектора относительных производительностей устройств

  export DVMH_CPU_PERF='0.35016‘

  export DVMH_CUDAS_PERF='0.64984‘
- Не учитывается различное соотношение производительности в разных регионах

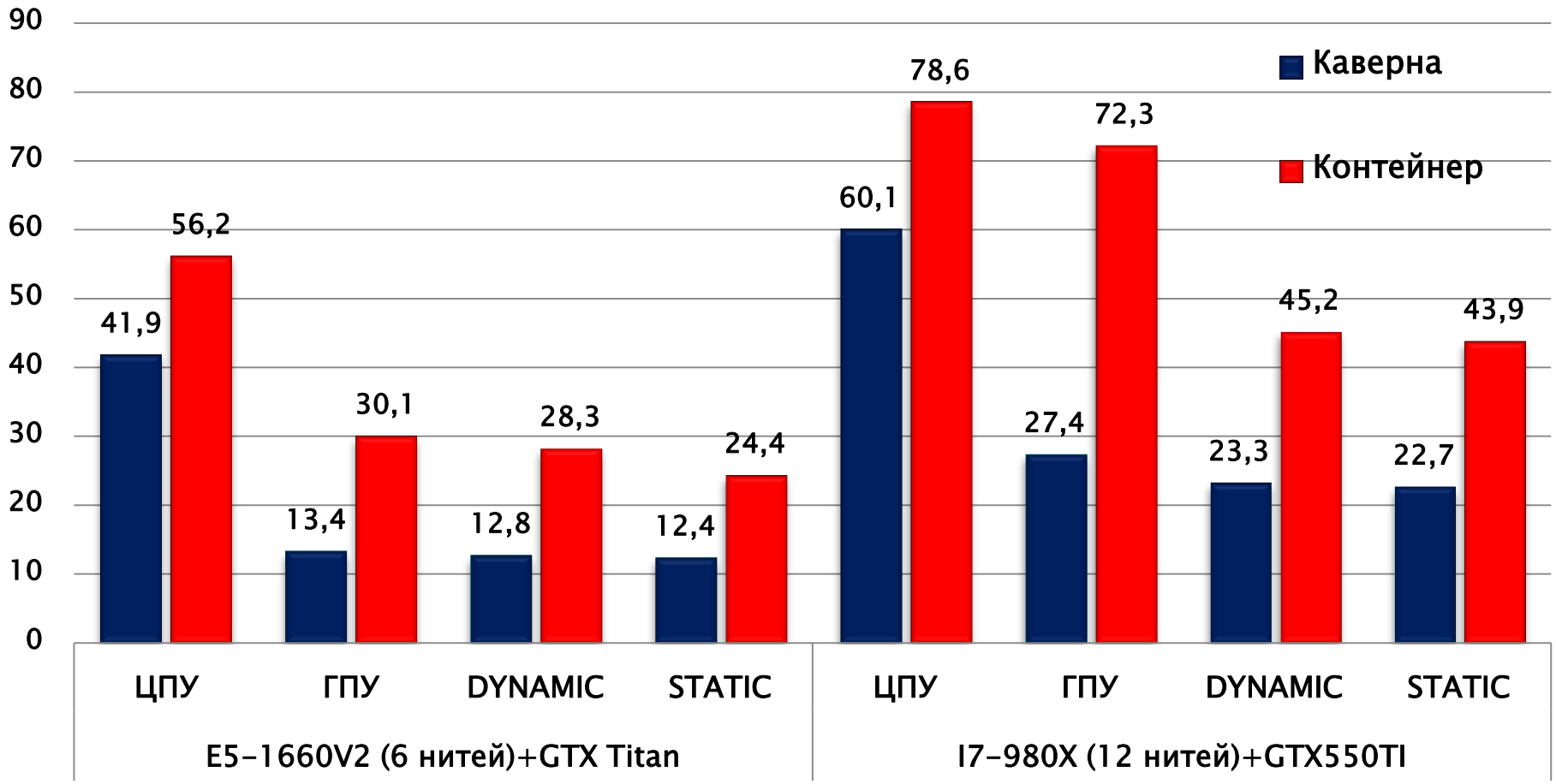
Динамический режим

- Для регионов и параллельных циклов строится зависимость их времени работы от распределения
- Строится зависимость затрат на перемещения данных от распределения
- Производится планирование выполнения с целью минимизации общего времени
- При завершении – вывод результатов планирования

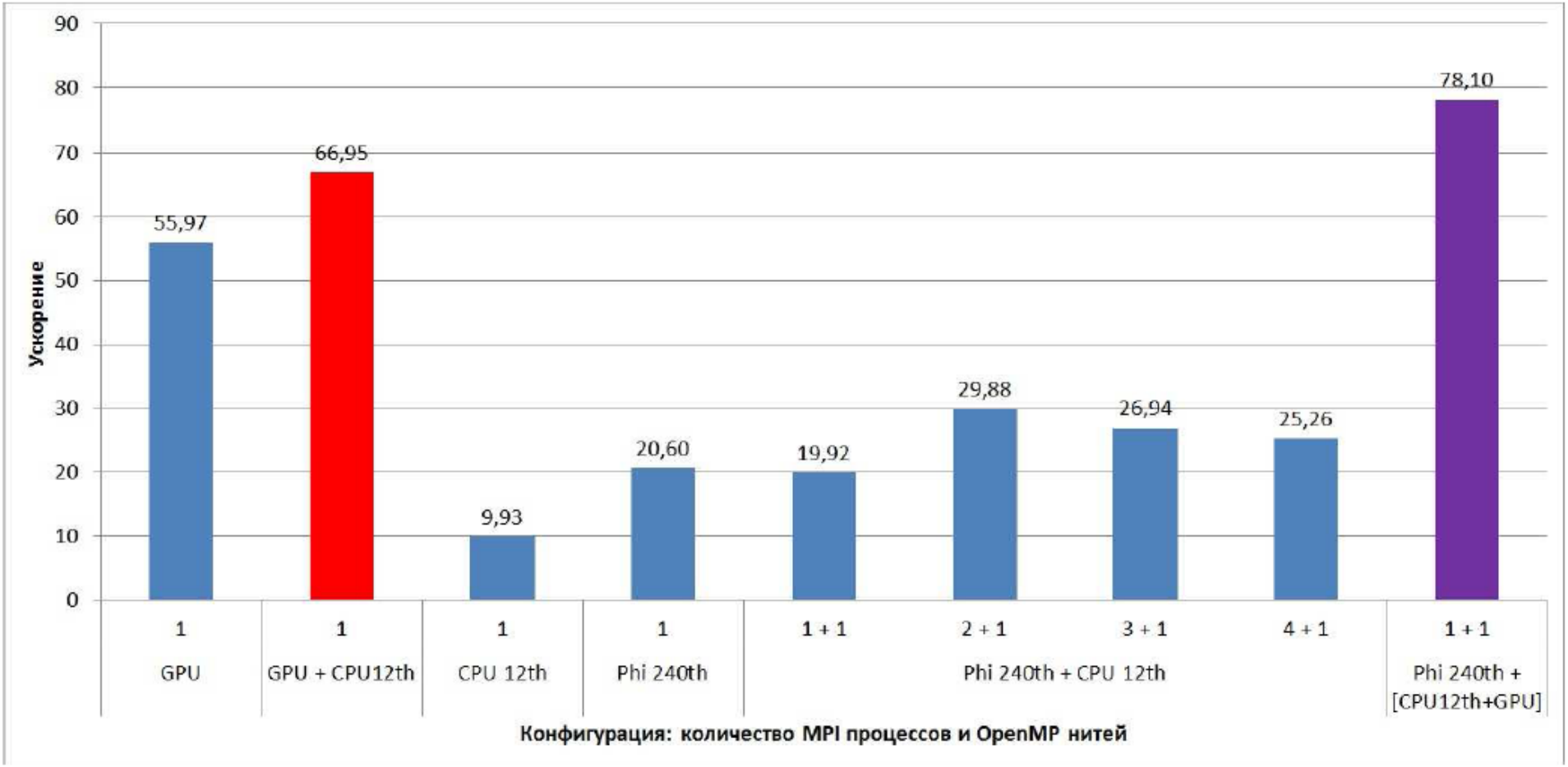
Зависимость производительности от объема вычислений

```
Performance statistics for parallel loop at cavity.fdv(601):
   Device #0:
      Handler #0:
         Best parameters: threads=11
         Table function (iterations => performance):
                 894400 => 2.21928e+08
                  1e+06 => 2.20961e+08
               2.56e+06 => 2.22738e+08
   Device #1:
      Handler #0:
         Best parameters: thread-block=(32,8,1)
         Table function (iterations => performance):
               1.56e+06 => 6.3996e+08
             1.6656e+06 => 6.44597e+08
               2.56e+06 => 6.74815e+08
Simple dynamic run performances:
DVMH_CPU_PERF='0.35016'
DVMH_CUDAS_PERF='0.64984'
```

Автоматическое распределение работы между устройствами узла



Выполнение теста EP при одновременном использовании ЦПУ, ГПУ и сопроцессора Xeon PHI



Методы динамической настройки DVMH-программ

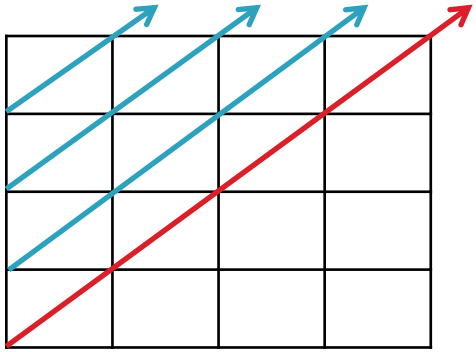
- Отображение массивов и циклов на узлы кластера с учетом их производительности
- Отображение массивов и циклов на устройства узла с учетом их производительности
- Трансформация массивов
- Использование динамической компиляции

```
#pragma dvm array distribute[block][block]
float A[L][L];
int main(int argc, char *argv[]) {
   float MAXEPS = 0.5E-5f;
   float w = 0.5f;
   for(int it = 0; it < ITMAX; it++) {
     float eps = 0.f;
     #pragma dvm actual(eps)
     #pragma dvm region
       {
         #pragma dvm parallel([i][j] on A[i][j]) across(A[1:1][1:1]), reduction(max(eps))
          for (int i = 1; i < L - 1; i++)
              for (int j = 1; j < L - 1; j++) {
                  float s = A[i][j];
                  A[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i][j - 1] + A[i][j + 1]) / 4.f;
                  eps = Max(fabs(s - A[i][j]), eps);
              }
       }
      #pragma dvm get_actual(eps)
       printf("it=%4i   eps=%e\n", it, eps);
       if (eps < MAXEPS) break;
   }
return 0;
}
```

Алгоритм SOR в модели DVMH

Выполнение гиперплоскостями

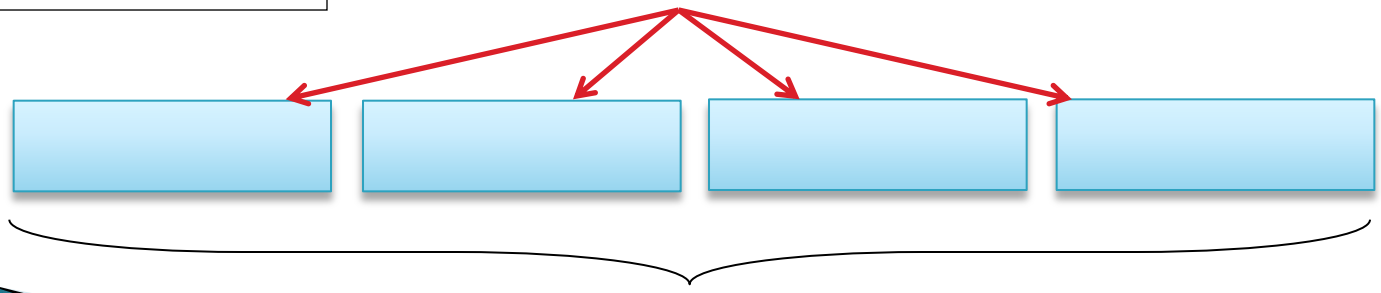
double A[N][N];



```
#pragma dvm parallel([i][j] on A[i][j]), across(A[1:1][1:1])
 for (i = 1; i < N - 1; i++)
      for (j = 1; j < N - 1; j++)
          A[i][j] = A[i - 1][j] + A[i + 1][j] + A[i][j - 1] …;
```

4 диагональ

warp load/store operations



Array A(4, 4)

Динамическая транформация данных

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

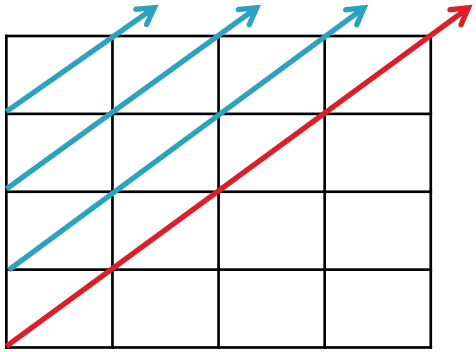
DVMH DR

1	5	2	9
6	3	13	10
7	4	14	11
8	15	12	16

Выполняется подиагональная трансформация матрицы - соседние элементы на диагоналях располагаются в соседних ячейках памяти

Выполнение гиперплоскостями

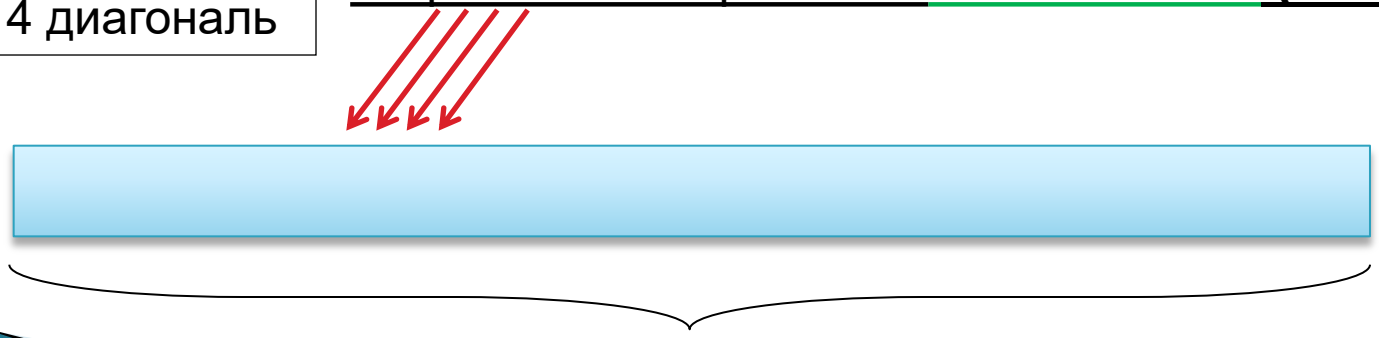
double A[N][N];



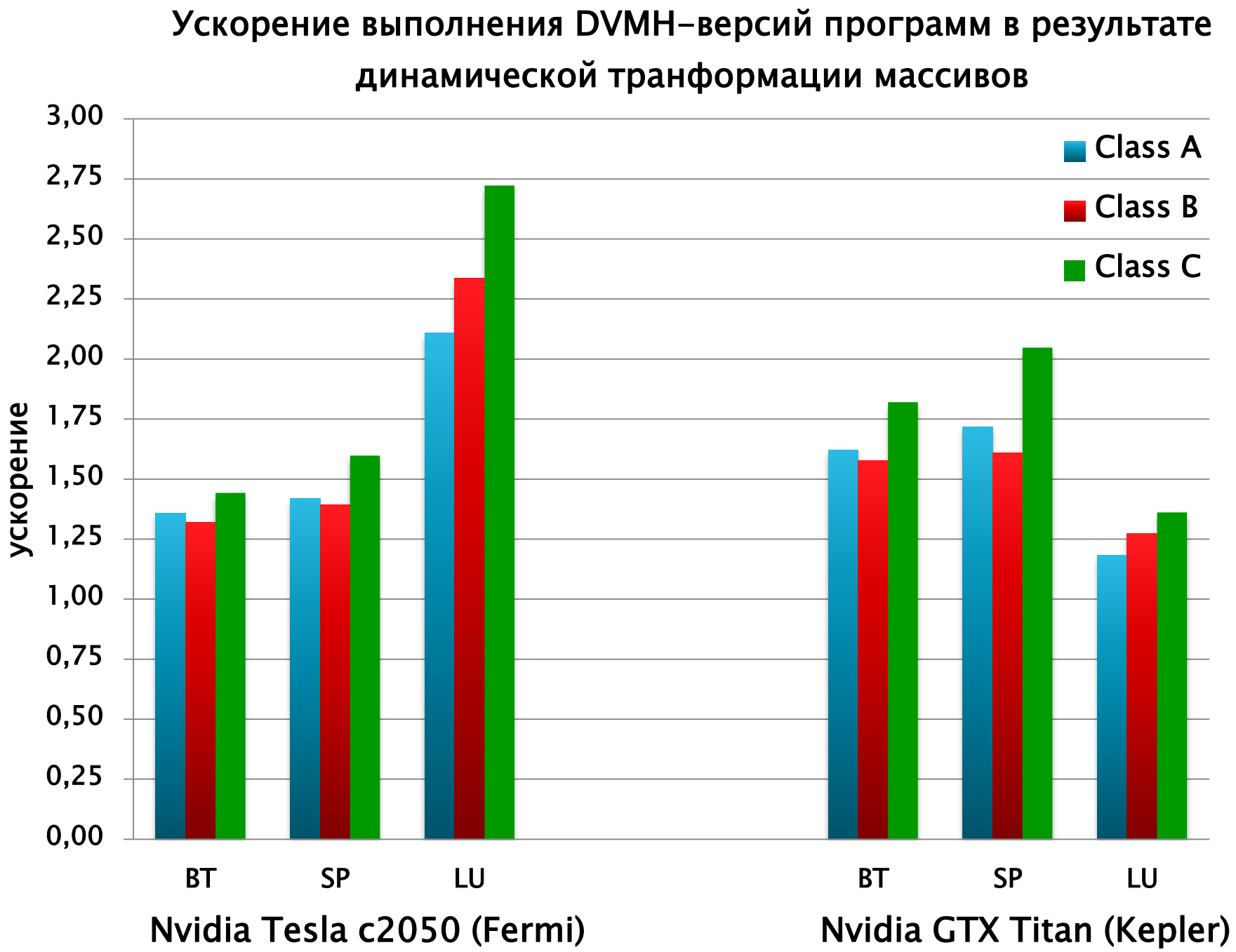
```
#pragma dvm parallel([i][j] on A[i][j]), across(A[1:1][1:1])
 for (i = 1; i < N - 1; i++)
      for (j = 1; j < N - 1; j++)
          A[i][j] = A[i - 1][j] + A[i + 1][j] + A[i][j - 1] …;
```

4 диагональ

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



Array A(4, 4)


Ускорение выполнения DVMH-версий программ в результате динамической транформации массивов
ускорение
3,00
2,75
2,50
2,25
2,00
1,75
1,50
1,25
1,00
0,75
0,50
0,25
0,00
Class A
Class B
Class C
BT
SP
LU
BT
SP
LU
Nvidia Tesla c2050 (Fermi)
Nvidia GTX Titan (Kepler)

Динамическая транформация данных

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

Методы динамической настройки DVMH-программ

- Отображение массивов и циклов на узлы кластера с учетом их производительности
- Отображение массивов и циклов на устройства узла с учетом их производительности
- Трансформация массивов
- Использование динамической компиляции

Динамическая компиляция CUDA-обработчиков во время выполнения программы

- CUDA Toolkit 7.0
- Тест SP из пакета NAS NPB

	Количество регистров			Время выполнения, ms		
	Было	Стало	Сокращение	Было	Стало	Ускорение
compute_rhs	125	82	52%	74	66.5	11%
x_solve	153	112	37%	64.5	42.9	50%
y_solve	132	107	23%	64.4	42.6	51%
z_xolve	128	107	20%	53	48	10%

- Общая производительность всей программы увеличилась на 28%

Выводы

- DVM-система существенно упрощает процесс разработки параллельных программ для гибридных вычислительных кластеров.
- Получаемые DVMH-программы без каких-либо изменений могут эффективно выполняться на кластерах различной архитектуры, использующих многоядерные универсальные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi. Это достигается как за счет различных оптимизаций, которые выполняются статически, при компиляции DVMH-программ, так и за счет динамических оптимизаций, которые были рассмотрены в данном докладе.
- Параллельные программы могут динамически настраиваться при запуске на выделенные для их выполнения ресурсы (количество узлов кластера, ядер, ускорителей и их производительность).

Планы развития DVM-системы

- Поддержка неструктурированных сеток и разреженных матриц.
- Расширение языка C-DVMH для поддержки распараллеливания современных программных комплексов на C++.

Вопросы, замечания?

СПАСИБО !

http://dvm-system.org
dvm@keldysh.ru