Исследование эффективности выполнения DVMH-программ на кластерах с многоядерными процессорами и ускорителями

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

bakhtin@keldysh.ru

Институт прикладной математики им. М.В. Келдыша РАН

НСКФ-2014

План доклада

- DVM-модель параллельного программирования
- Принципы расширения DVM-модели
- Основные возможности DVMH (DVM for Heterogeneous systems)
- Поддержка сопроцессоров Intel Xeon PHI
- 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-DVMH
- Компилятор C-DVMH
- Библиотека поддержки LIB-DVMH
- DVM-отладчик
- Предсказатель производительности DVM-программ
- Анализатор производительности DVM-программ

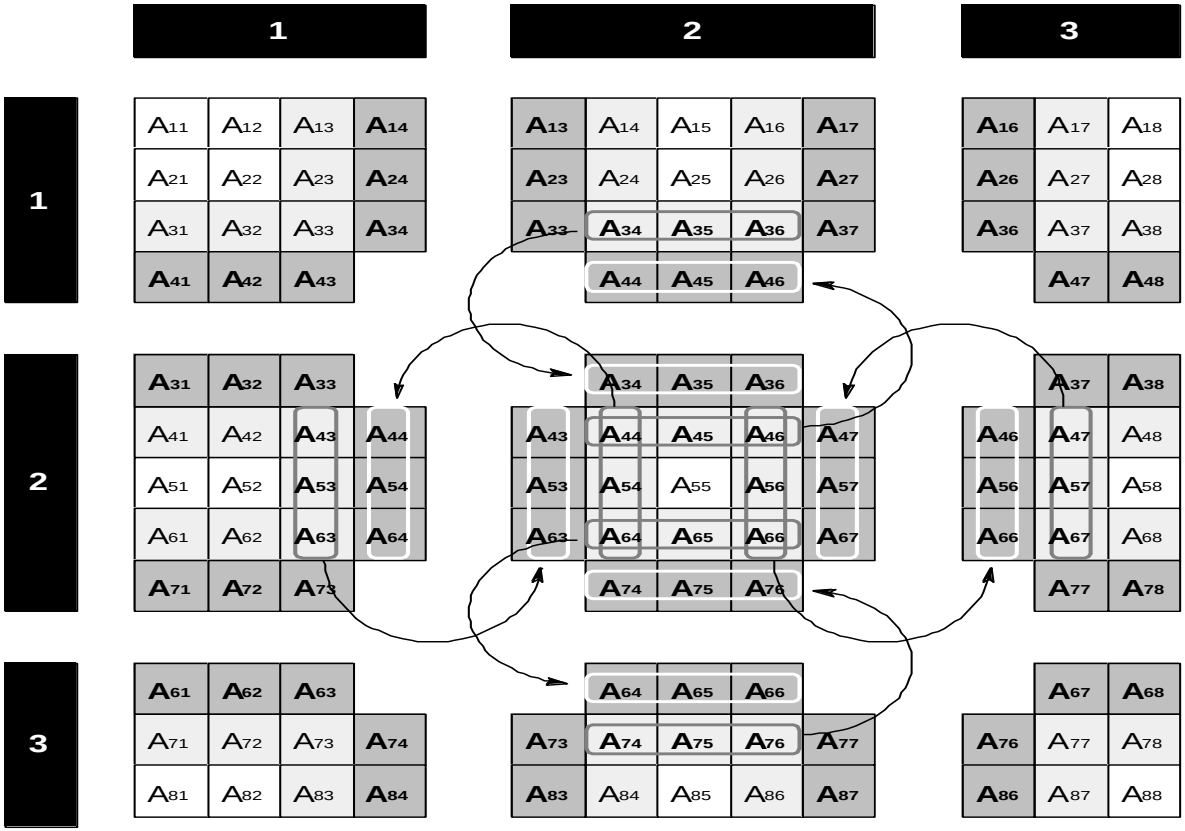
Спецификации параллельного выполнения программы

- 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)

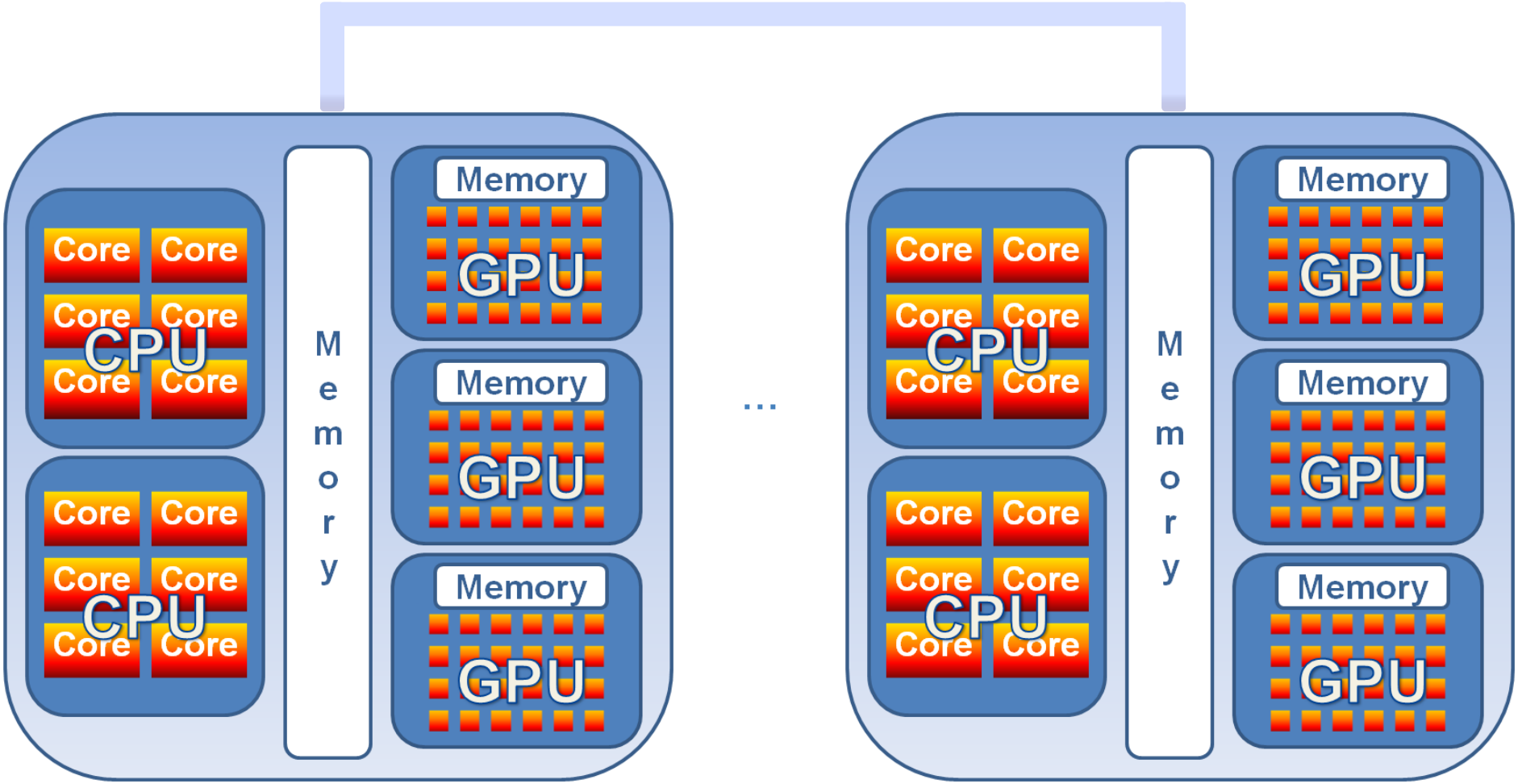
```
        PROGRAM    JACOBY_DVM
        PARAMETER    (L=4096,  ITMAX=100)
        REAL     A(L,L), B(L,L)
!DVM$   DISTRIBUTE    ( BLOCK,   BLOCK)   ::   A
!DVM$   ALIGN  B(I,J)  WITH  A(I,J)
        PRINT *,  '**********  TEST_JACOBI   **********'
        DO IT  =  1,  ITMAX
!DVM$         PARALLEL (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

Распределение массива



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

```
#pragma omp target device(acc0) map(A,B)
#pragma omp parallel for
  for (i=0;i<N;i++)   // OpenMP 4.0
      A[i] += A[i]*B[i];
```

- 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.

```
        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 INOUT(A,B)
!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 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
      {
         #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.

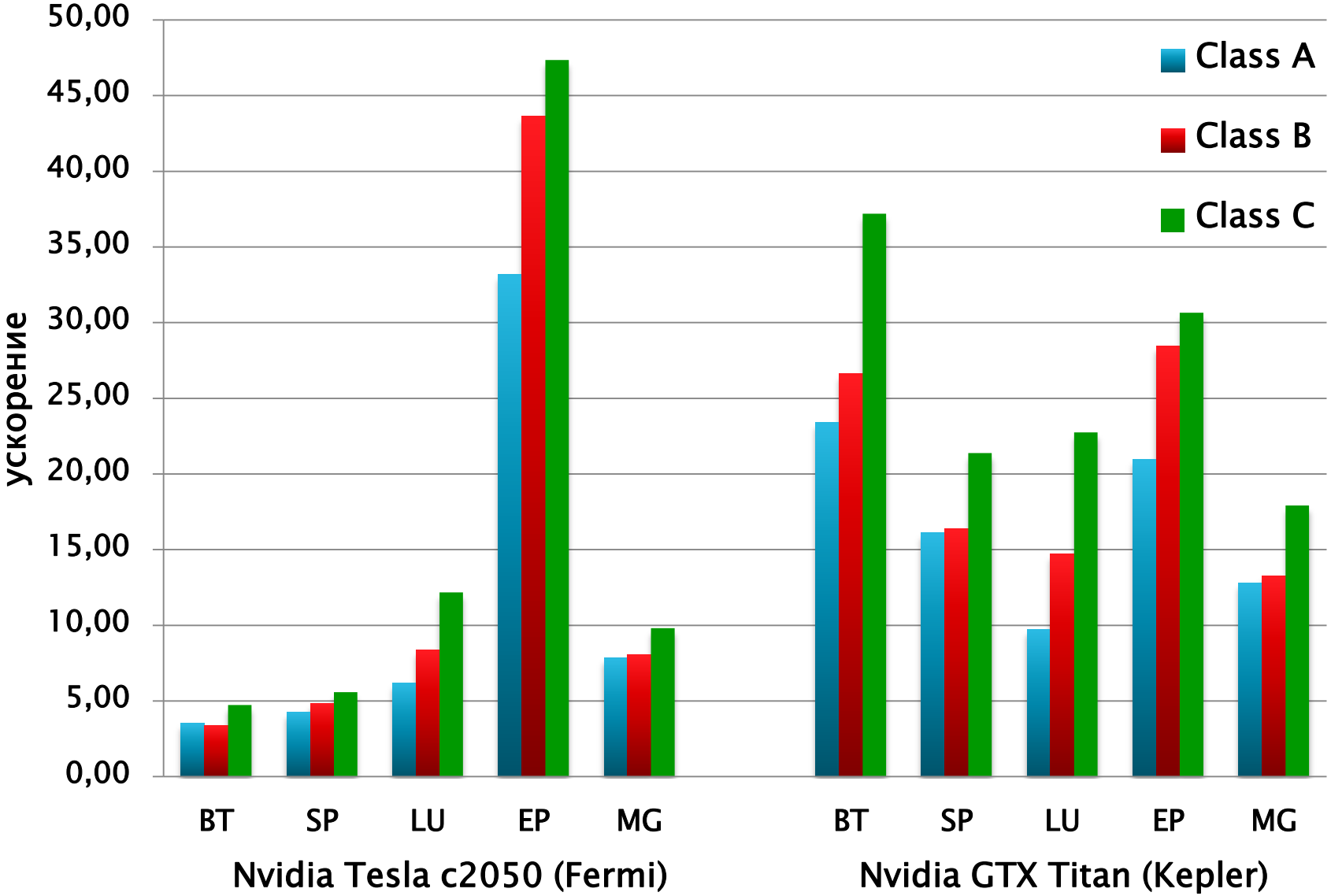
Это позволяет:

- 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

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



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
```

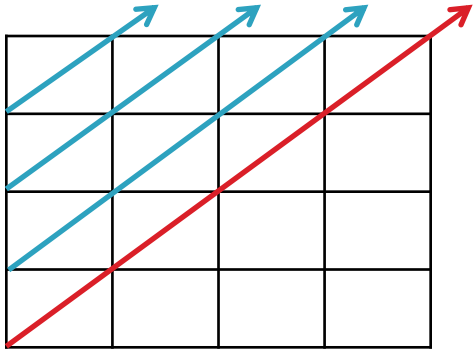
Использование приватных переменных для часто используемых элементов массивов, которые отображаются компилятором NVCC на регистры.

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
```

Выполнение гиперплоскостями

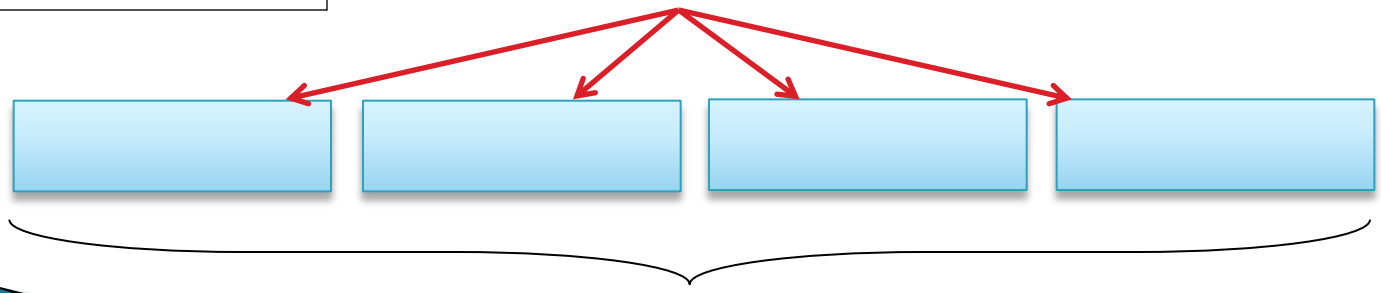
REAL A(4, 4)



```
!$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 массивов

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

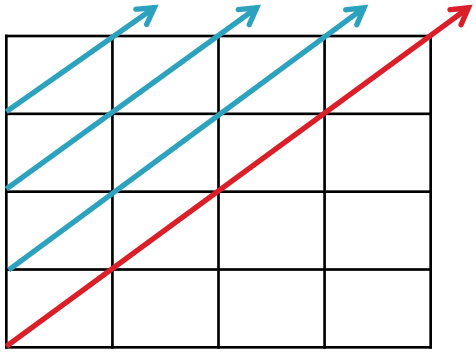
DVMH DR

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

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

Выполнение гиперплоскостями (-autoTfm)

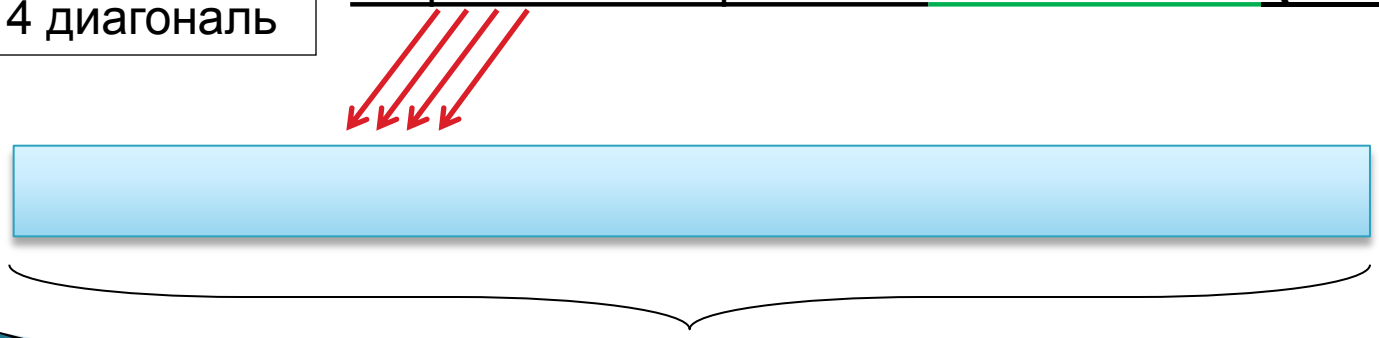
REAL A(4, 4)



```
!$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)**

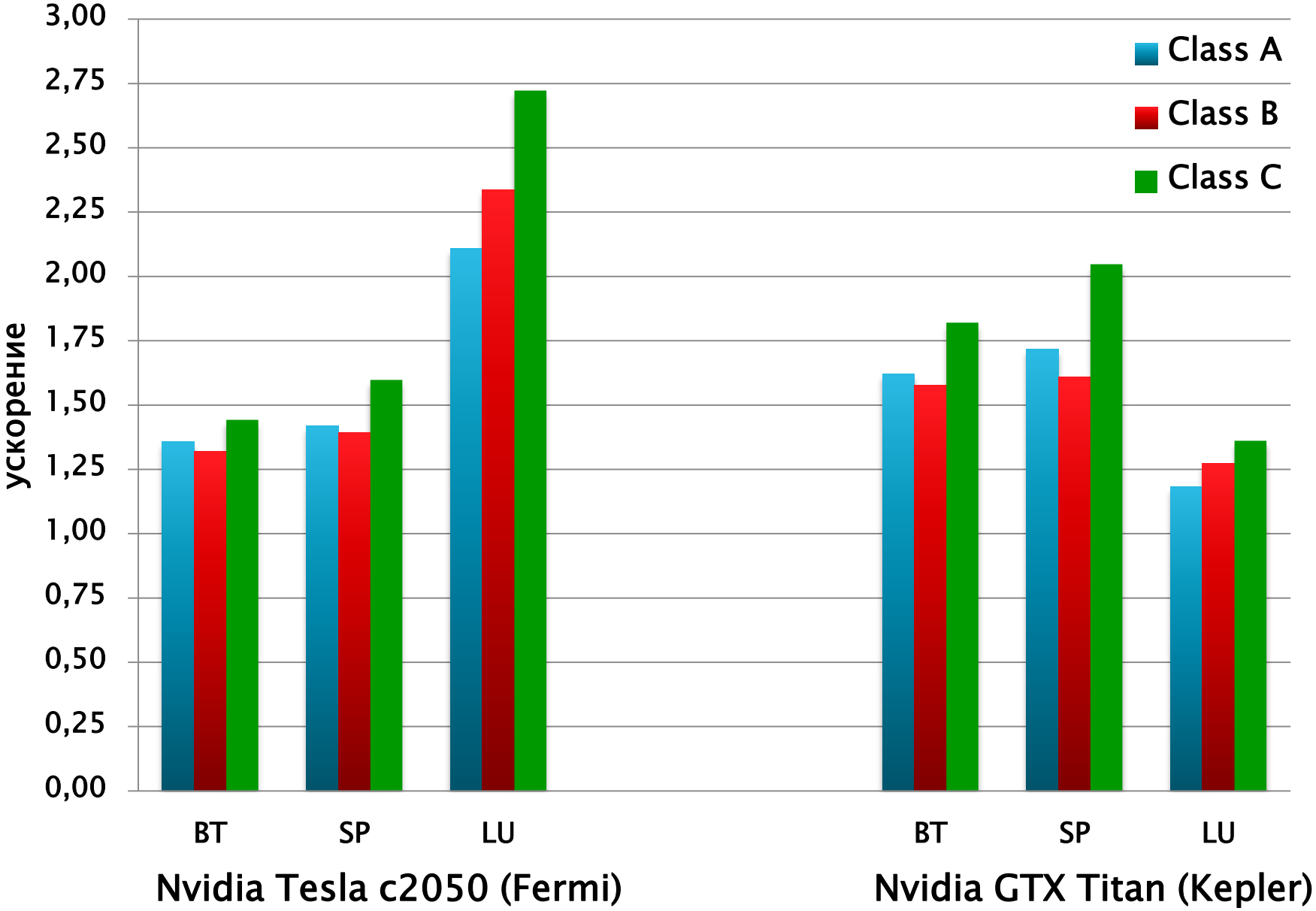


Array A(4, 4)

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



Поддержка сопроцессоров Intel Xeon PHI

- Симметричный режим
- Балансировка нагрузки
  - setenv DVMH_PPN 1 # number of processors per node
  - setenv DVMH_USE_OMP 1
  - setenv DVMH_NUM_THREADS 240
  - setenv DVMH_NUM_CUDAS 0 # number of GPUs per one proc
  - setenv DVMH_CPU_PERF 1 # performance of all cores of CPU
  - setenv DVMH_CUDAS_PERF 1# performance of each GPU

Оптимизации OpenMP-обработчиков

- Оптимизация работы с распределенными массивами
- Использование клаузы COLLAPSE
- Оптимизация выполнения циклов с зависимостью
- Использование директивы SIMD
- Использование AVX-инструкций

Оптимизация работы с распределенными массивами

DOUBLE PRECISION A(L,L,L)

!DVM$ DISTRIBUTE A(BLOCK,BLOCK, BLOCK)

- **Старая версия компилятора**

A(I,J,K) => BASE_ARRAY(ARRAY_A_OFFSET + I + COEFF_ARRAY_1 * J + COEFF_ARRAY_A_2 * K)

- **Новая версия компилятора**

DOUBLE PRECISION A(0:A_HEAD(3) - 1,0:A_HEAD(2) / A_HEAD(3) - 1,0:*)

A(I,J,K) => A(I,J,K)

Использование клаузы COLLAPSE

```
!DVM$ PARALLEL (i3,i2)  ON  u(1,i2,i3)
    do i3=2,n3-1
      do i2=2,n2-1
         u(1,i2,i3) = u(n1-1,i2,i3)
         u(n1,i2,i3) = u(2,i2,i3)
      enddo
    enddo

!$OMP PARALLEL DO COLLAPSE (2), PRIVATE (i3,i2)…
    do  i3 = boundsLow(1),boundsHigh(1)
       do  i2 = boundsLow(2),boundsHigh(2)
         u(1,i2,i3) = u(n1 - 1,i2,i3)
         u(n1,i2,i3) = u(2,i2,i3)
       enddo
    enddo
```

Оптимизация выполнения циклов с зависимостью

```
!DVM$ PARALLEL(J, I) ON B(I, J), ACROSS(A(1:1,0:0))
 DO  J  =  2, L-1
       DO  I  =  2, L-1
             B(I,J) = …
```

```
!DVM$  ALIGN B(I,J) WITH A(I,J)
!$OMP PARALLEL DO PRIVATE (J,I),SCHEDULE (runtime)…
DO  J  =  boundsLow(1),boundsHigh(1)
      DO  I  =  boundsLow(2),boundsHigh(2)
            B(I,J) = …
```

```
!DVM$  ALIGN B(I,J) WITH A(J,I)
!$OMP PARALLEL PRIVATE (J,I)…
DO  J  =  boundsLow(1),boundsHigh(1)
      !$OMP DO SCHEDULE (runtime)
      DO  I  =  boundsLow(2),boundsHigh(2)
            B(I,J) = …
     !$OMP ENDDO NOWAIT
!$OMP END PARALLEL
```

Новая версия компилятора генерирует несколько OpenMP-обработчиков.

В момент выполнения программы система поддержки определяет обработчик, который будет выполнять цикл.

Использование директивы SIMD

```
!$OMP PARALLEL DO NUM_THREADS(lgsc),REDUCTION (max:rnmu),
!$OMP* REDUCTION (+:s),PRIVATE (a),SCHEDULE (runtime)
    do  i3 = boundsLow(1),boundsHigh(1)
    !$OMP SIMD REDUCTION (max:rnmu),REDUCTION (+:s),PRIVATE (a)
        do  i2 = boundsLow(2),boundsHigh(2)
          do  i1 = boundsLow(3),boundsHigh(3)
              s = s + r(i1,i2,i3)** 2
              a = abs (r(i1,i2,i3))
              rnmu = dmax1 (rnmu,a)
          enddo
       enddo
    enddo
```

Ускорение теста MG при использовании директивы SIMD

Класс	Intel Xeon PHI	Intel Xeon PHI + SIMD	Ускорение
A	0,7 сек	0,61 сек	1,15
B	3,8ек	2,89 сек	1,31
C	35,5 сек	15,51 сек	2,28

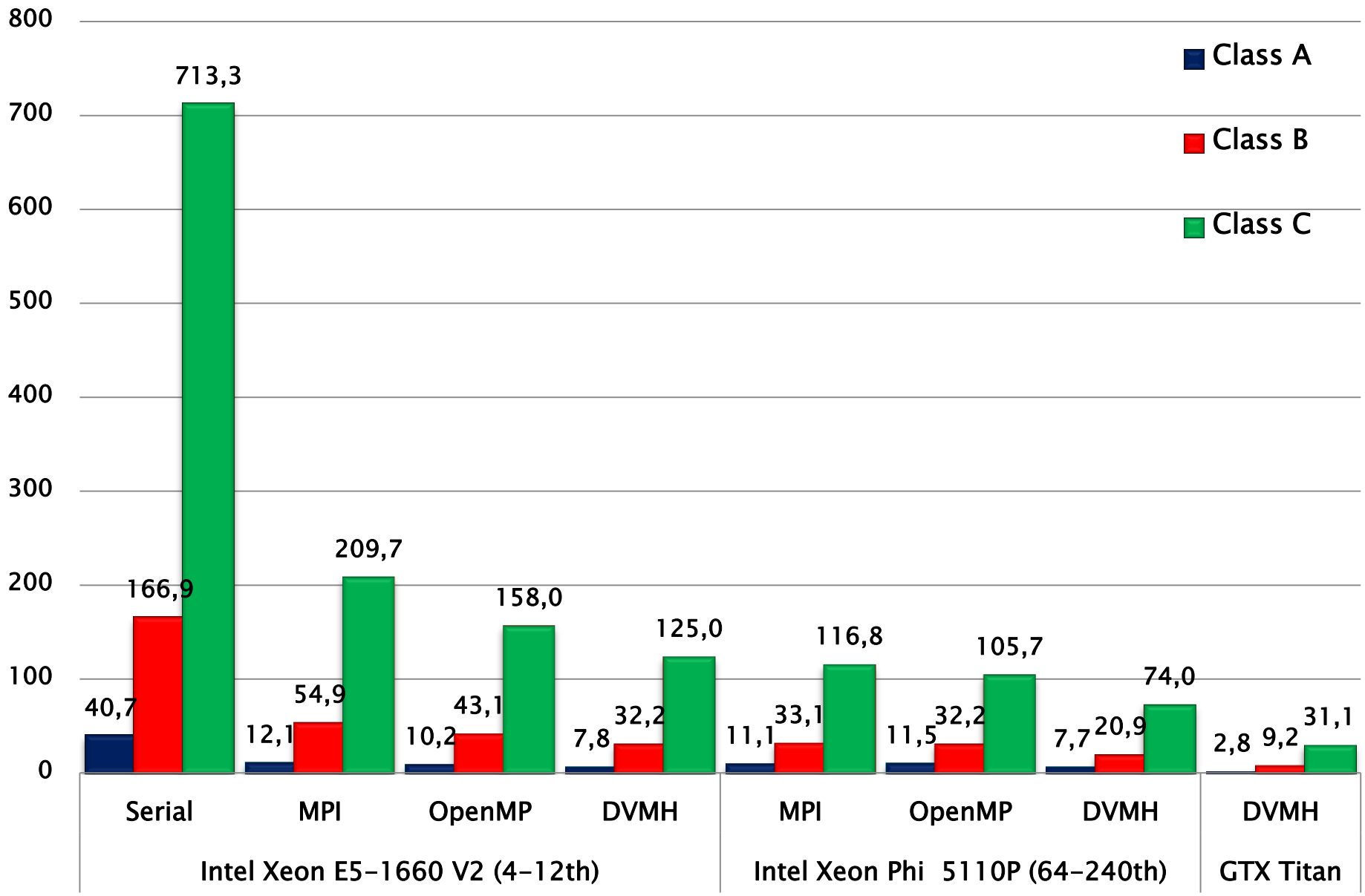
Использование AVX-инструкций

```
double rhs [5][162][162][162] , u [5][162][162][162];
# pragma omp parallel for
for (int i = 0; i < Ni; i++) {
   for (int j = 0; j < Nj; j++) {
      /*** изменение индексного пространства цикла ***/
      for (int k = 0; i < Nk; k +=8) {
         /*** rhs [0][ i][j][k] = u [0][i][j][k] + u [0][i +1][j][k]; ***/
         _mm512_store_pd (& rhs [0][i][j][k],
            _mm512_add_pd (
               _mm512_load_pd (u [0][i][j][k]),
               _mm512_load_pd (u [0][i +1][j][k]));
      }
   }
}
```

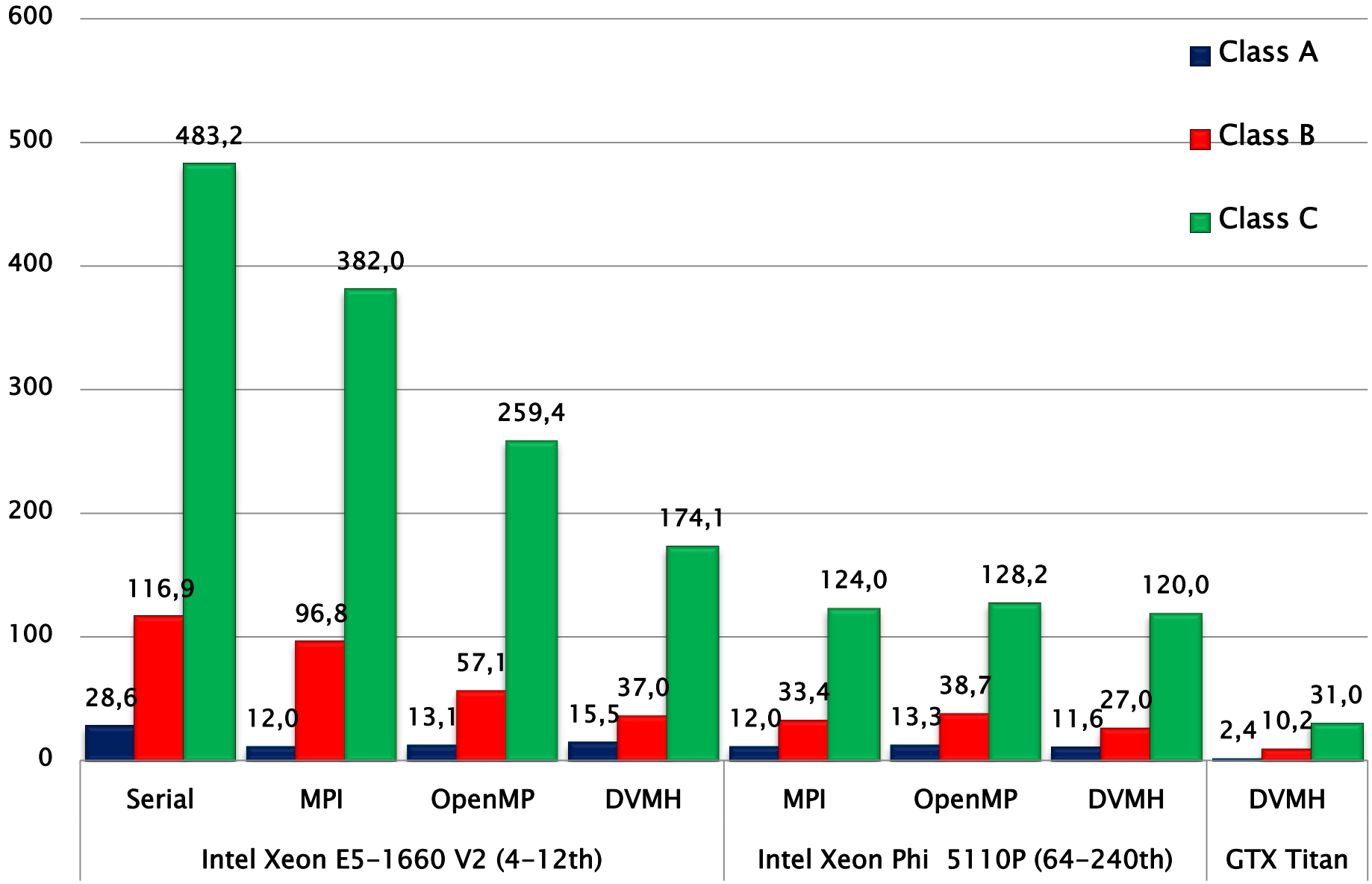
Ускорение процедуры compute_rhs теста SP при использовании AVX-инструкций

Класс	Intel Xeon PHI	Intel Xeon PHI + AVX512	Ускорение
A	2,9 сек	1,8 сек	1,55
B	13,4 сек	4,8 сек	2,76
C	118,1 сек	18,7 сек	6,31

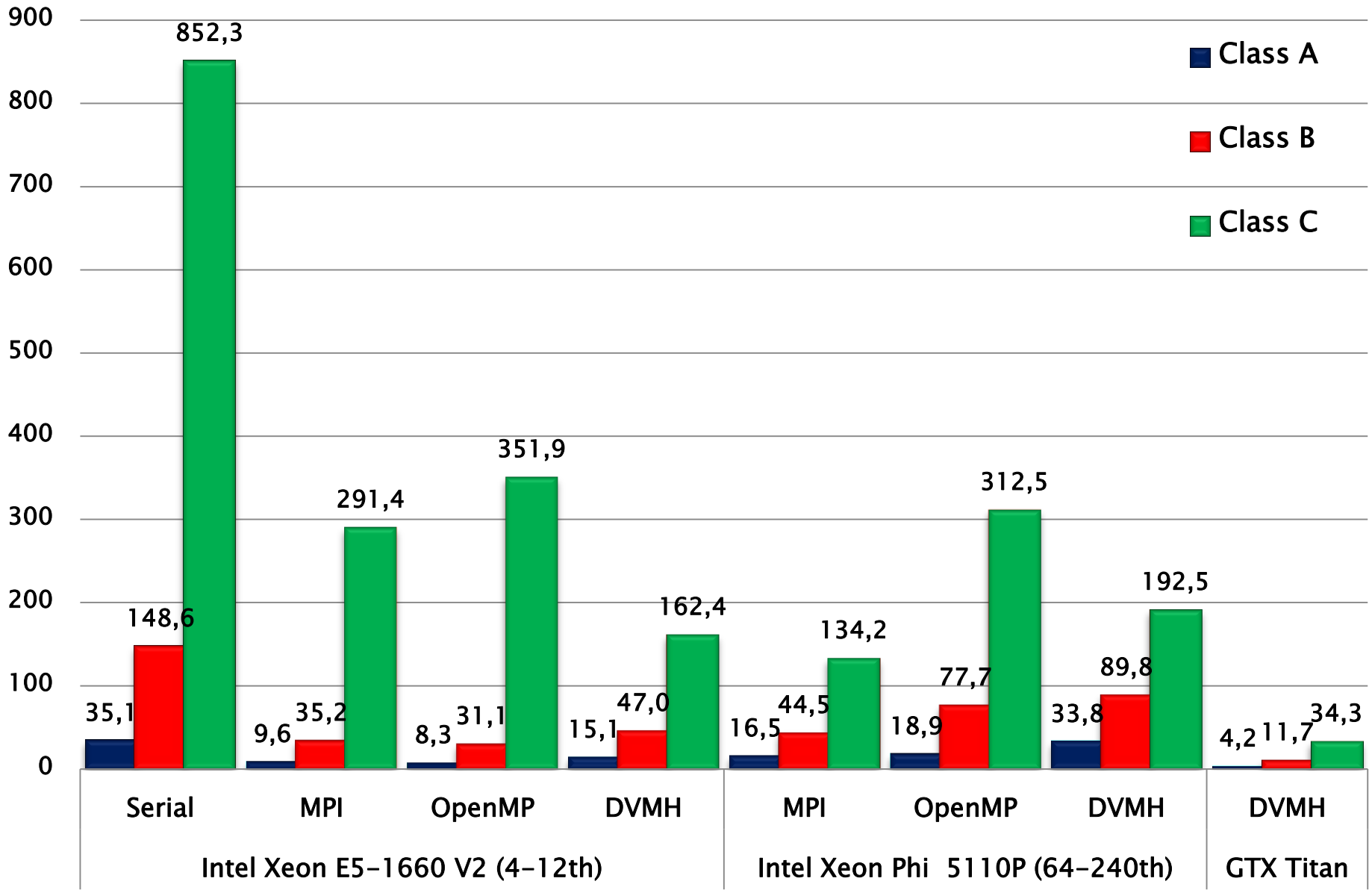
Времена выполнения теста BT, в секундах



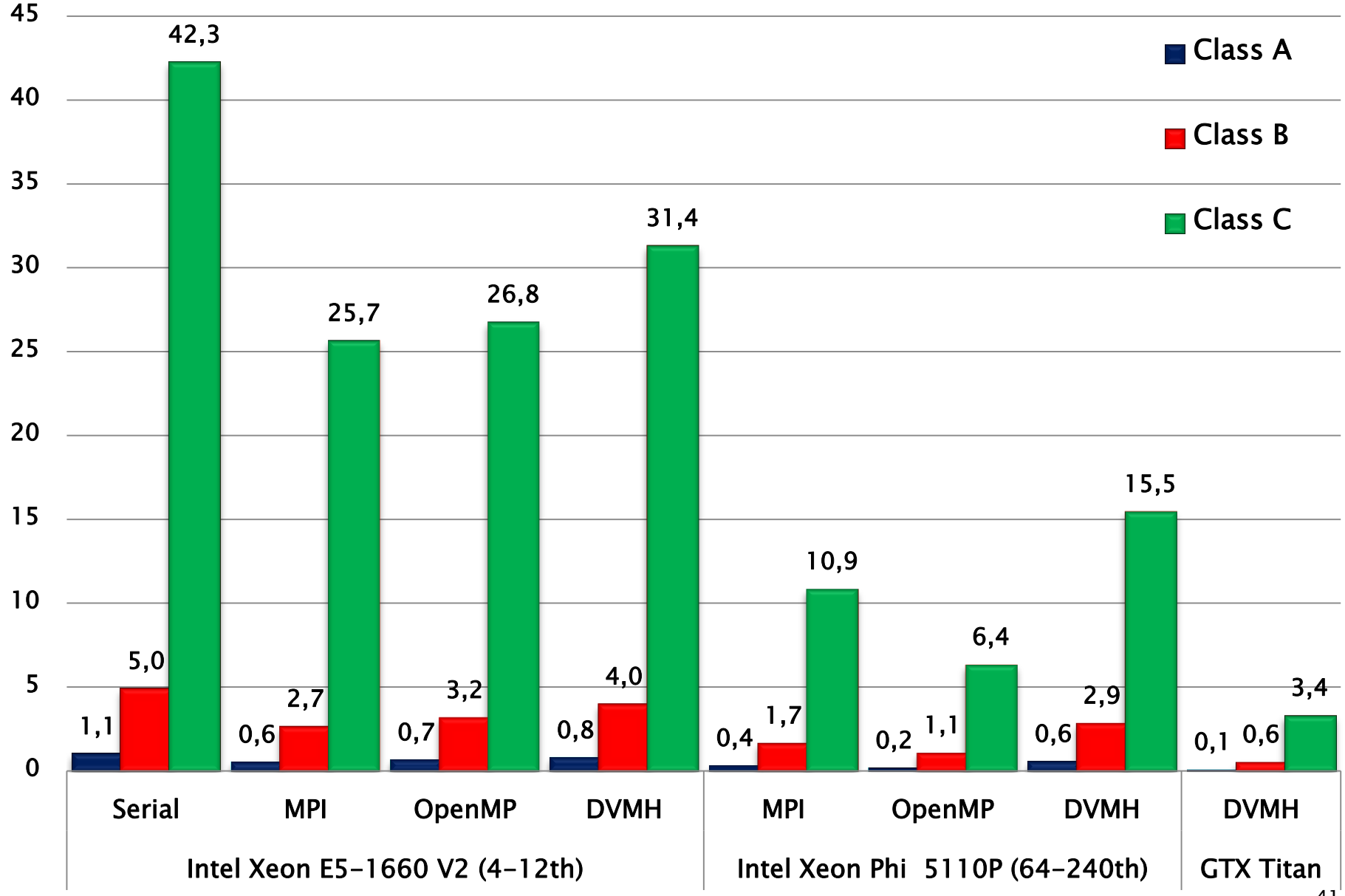
Времена выполнения теста SP, в секундах



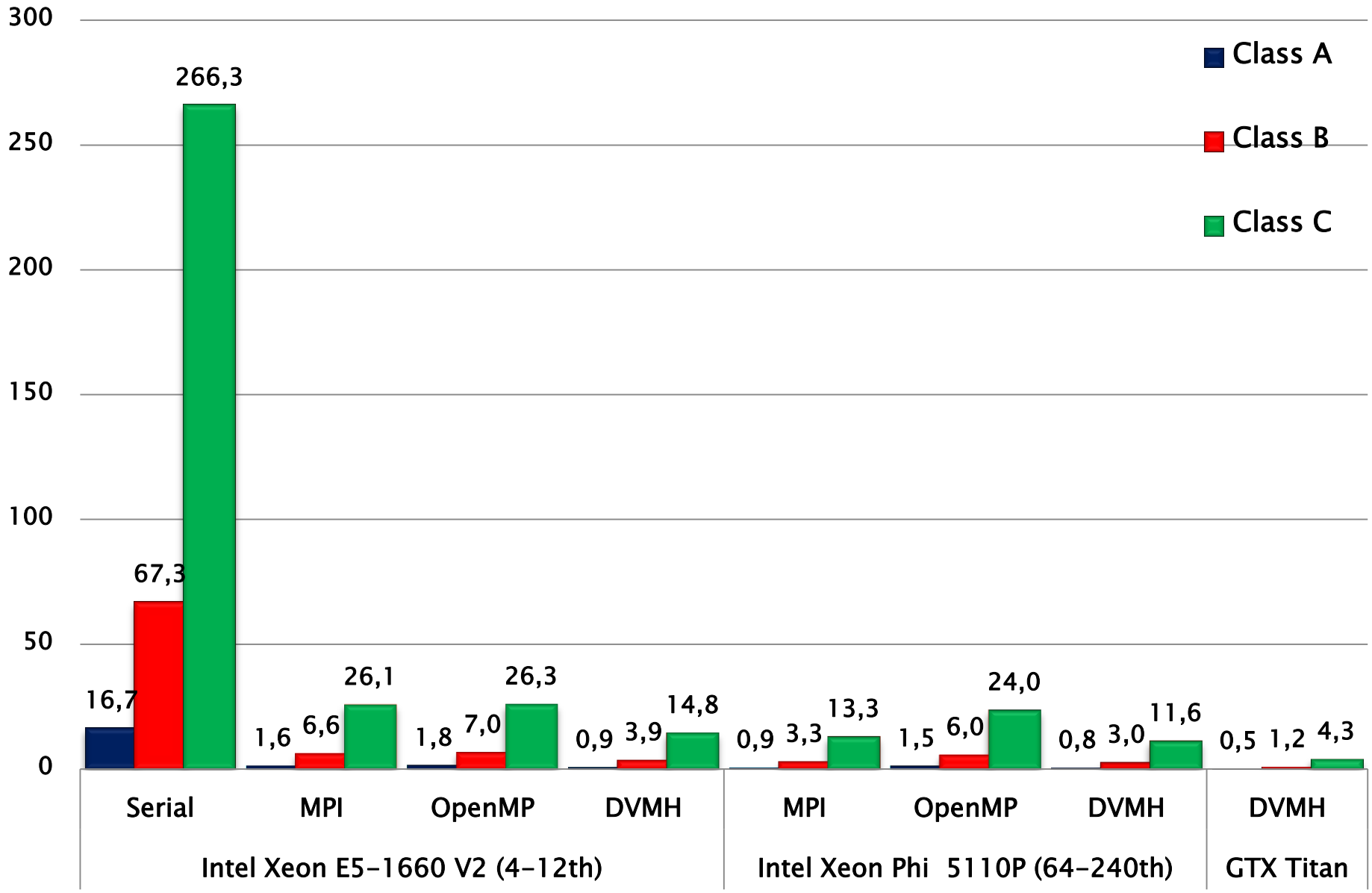
Времена выполнения теста LU, в секундах



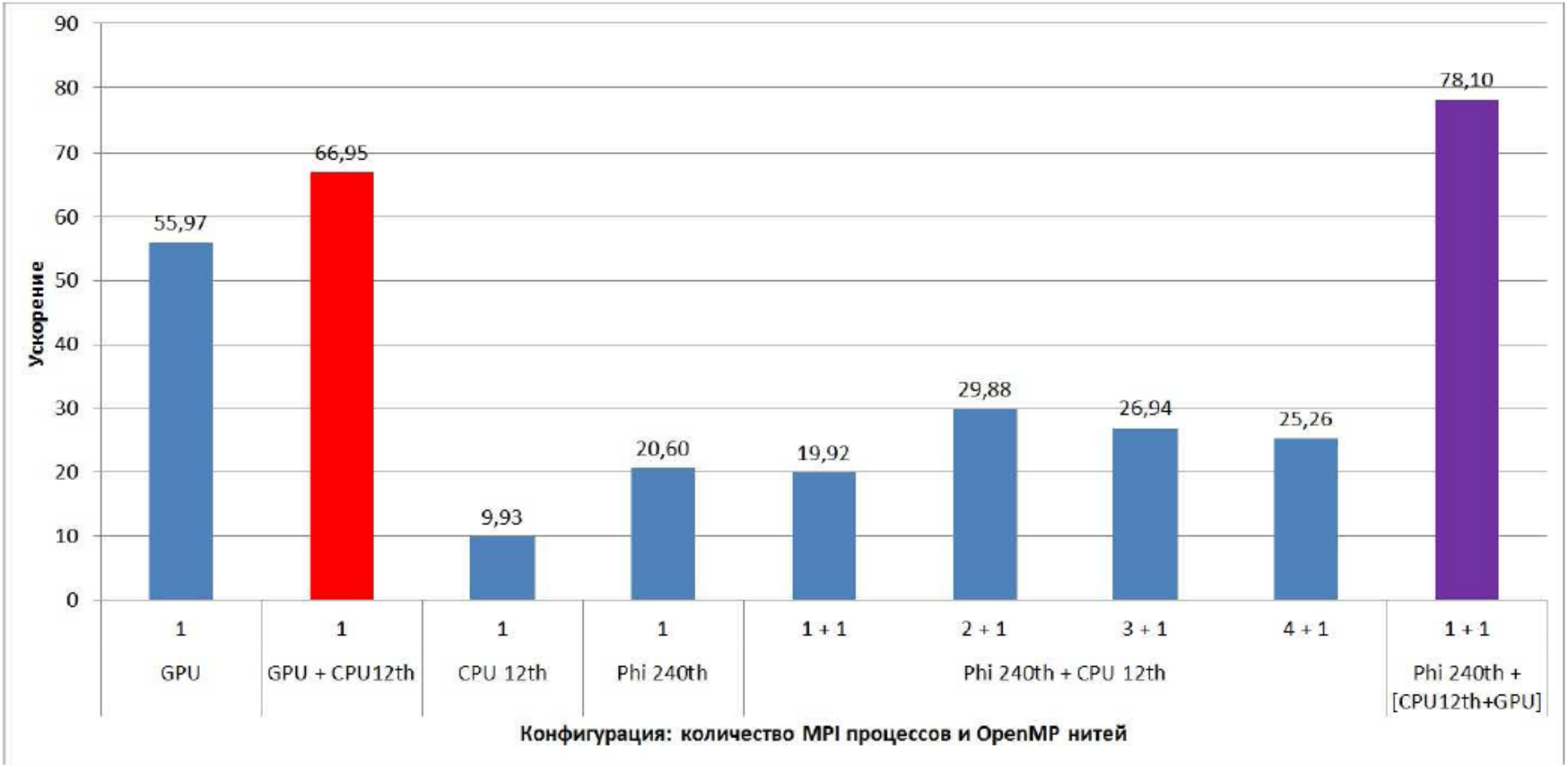
Времена выполнения теста MG, в секундах



Времена выполнения теста EP, в секундах



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



Выводы

- Разработанный подход к созданию прикладного программного обеспечения (DVM-подход) существенно упрощает создание прикладных программ для суперкомпьютерных систем с ускорителями.
- Эффективность программ, написанных в высокоуровневой модели DVMH, близка к эффективности программ, написанных с использованием низкоуровневых моделей MPI и CUDA.
- Высокоуровневые языки C-DVMH и Fortran-DVMH позволяют обеспечить эффективное отображение одной и той же DVMH-программы на вычислительные системы различной архитектуры.

Планы развития DVM-системы

- Поддержка неструктурных сеток и разреженных матриц
- Доработка системы отладки эффективности для DVMH-программ

Вопросы, замечания?

СПАСИБО !

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