DVM-технология разработки параллельных программ для вычислительных кластеров

Бахтин Владимир Александрович
к.ф.-м.н., ведущий научный сотрудник Института прикладной математики им М.В.Келдыша РАН
bakhtin@keldysh.ru

Содержание

- DVMH-модель параллельного программирования
- Преимущества DVMH-модели
- Новые возможности DVM-системы:
  - новая версия компилятора и языка C-DVMH;
  - параллельный ввод-вывод.
- Функциональная отладка и анализ эффективности DVMH-программ.
- Система САПФОР

Рейтинг. Top-500

PRESENTED BY

TOP 500 The List.

ICL INNOVATIVE COMPUTING LABORATORY

BERKELEY LAB Lawrence Berkeley National Laboratory

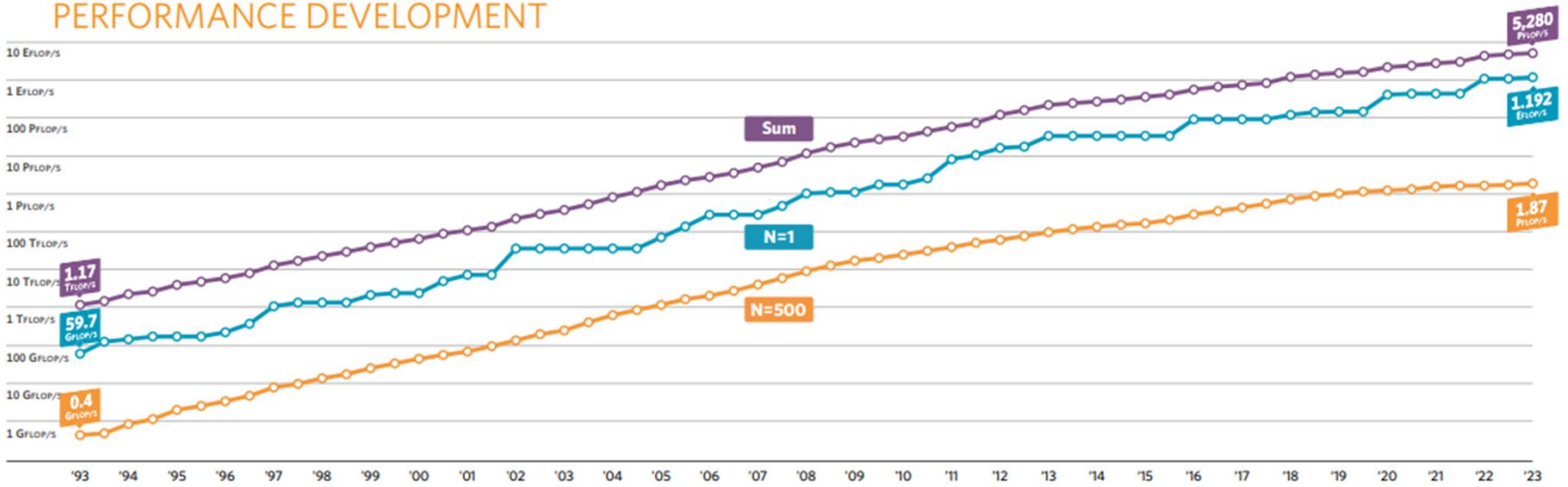
ISC GROUP Moving Forward.

FIND OUT MORE AT top500.org

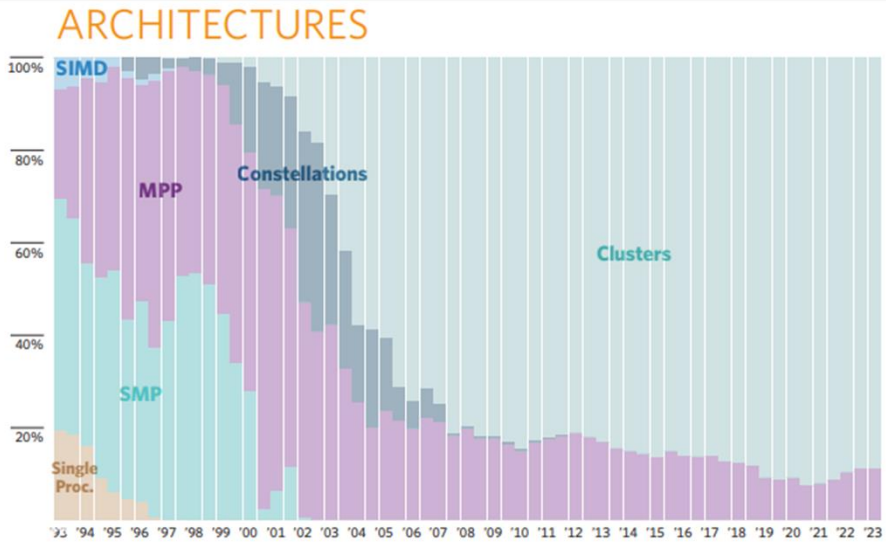
JUNE 2023

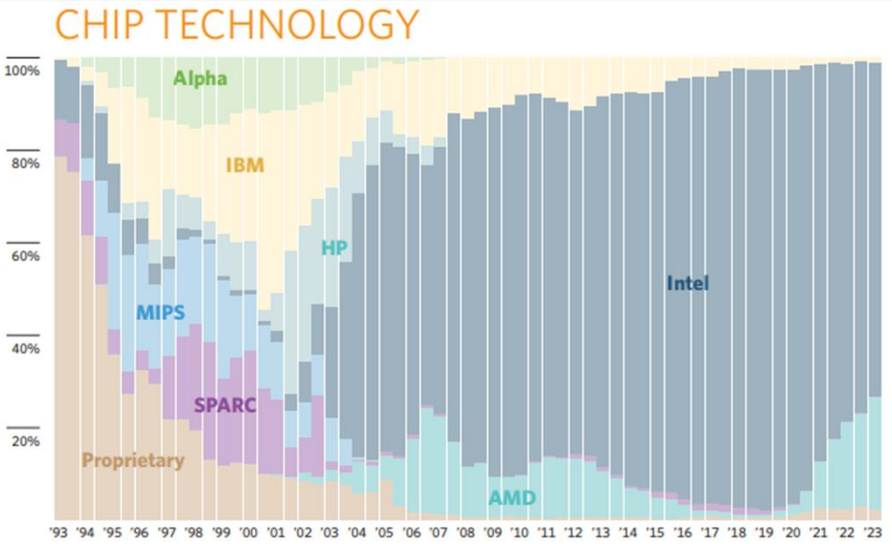
			SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	**Frontier**	HPE Cray EX235a, AMD Opt 3rd Gen EPYC (64C 2GHz), AMD Instinct MI250X, Slingshot-11	DOE/SC/ORNL	USA	8,699,904	1,194.0	22.7
2	**Fugaku**	Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D	RIKEN R-CCS	Japan	7,630,848	442.0	29.9
3	**LUMI**	HPE Cray EX235a, AMD Opt 3rd Gen EPYC (64C 2GHz), AMD Instinct MI250X, Slingshot-11	EuroHPC/CSC	Finland	2,220,288	309.0	6.01
4	**Leonardo**	Atos Bullsequana intelXeon (32C, 2.6 GHz), NVIDIA A100 quad-rail NVIDIA HDR100 Infiniband	EuroHPC/CINEC	Italy	1,824,768	238.7	7.40
5	**Summit**	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,414,592	148.6	10.1

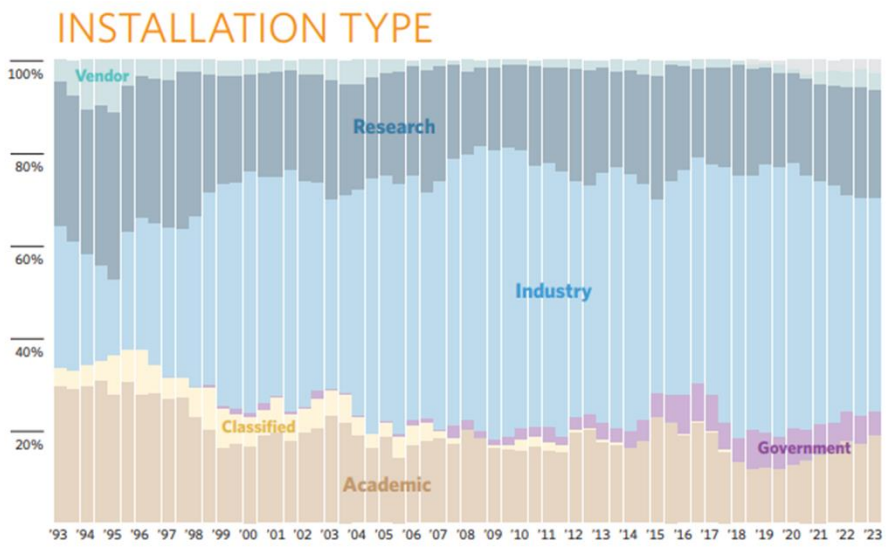
PERFORMANCE DEVELOPMENT

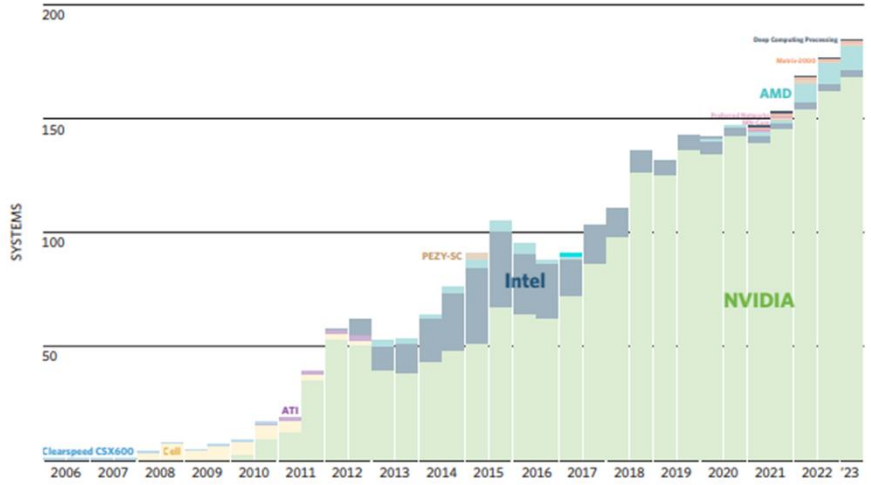


Рейтинг. Top-500









HPLINPACK A Portable Implementation of the High Performance Linpack Benchmark for Distributed Memory Computers FIND OUT MORE AT https://icl.utk.edu/hpl/

```
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

```
      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
      OPEN(3, FILE='JAC.DAT', ACCESS='STREAM')
!DVM$ GET_ACTUAL(B)
      WRITE(3, *) B
      CLOSE(3)
      END
```

Алгоритм Якоби в модели DVMH

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
- Управление параллельным вводом-выводом

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

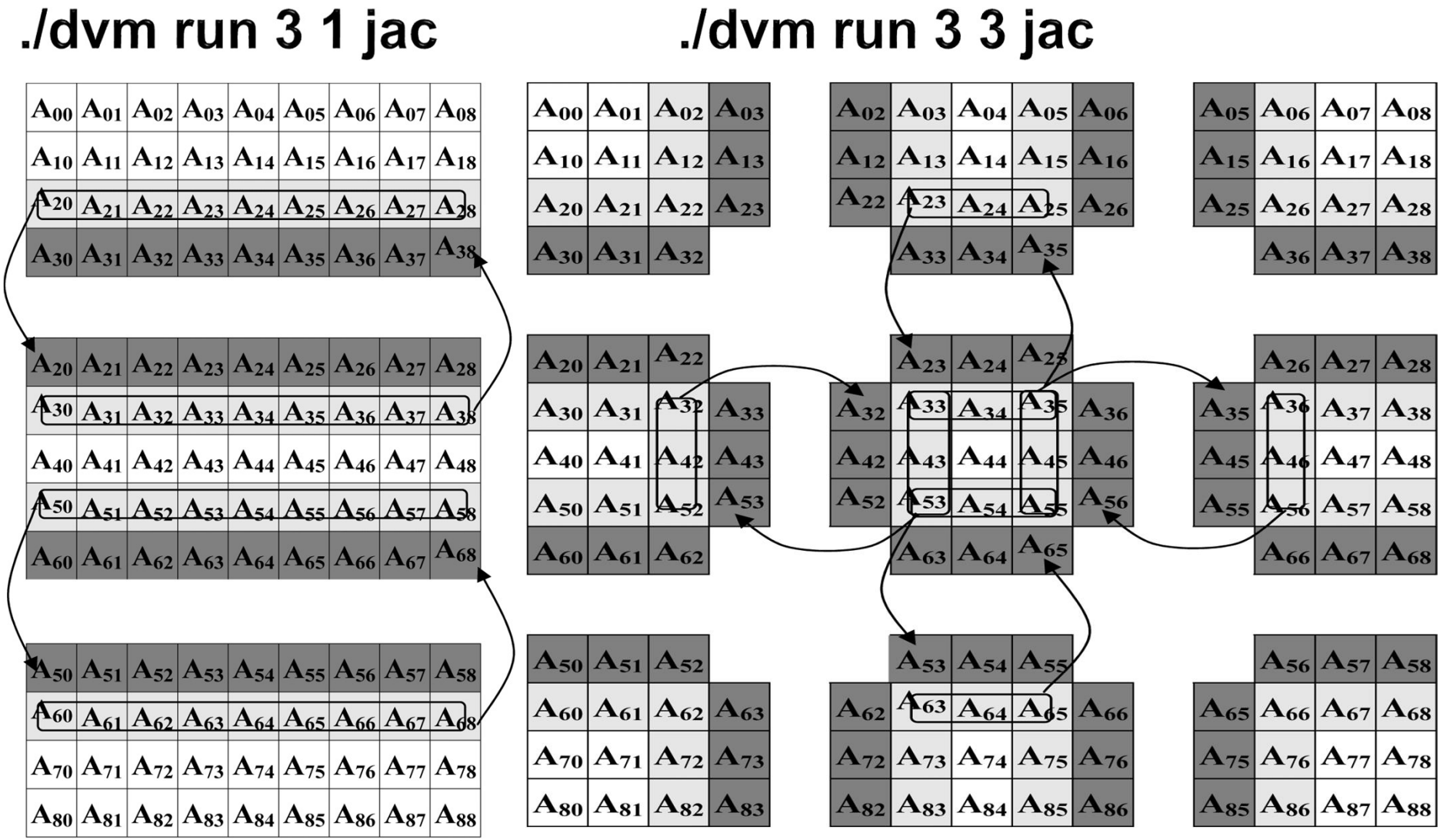
```
FILE *f;
#pragma  xmp nodes p(N, N)
#pragma  xmp template t(0:L−1,  0:L−1)
#pragma  xmp distribute t(block,  block)
#pragma  xmp align [i][j] with  t(i,j) :: A,B
#pragma  xmp shadow  A[1:1]
double A[L][L];
double B[L][L];
```

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 XACC

Распределение массива

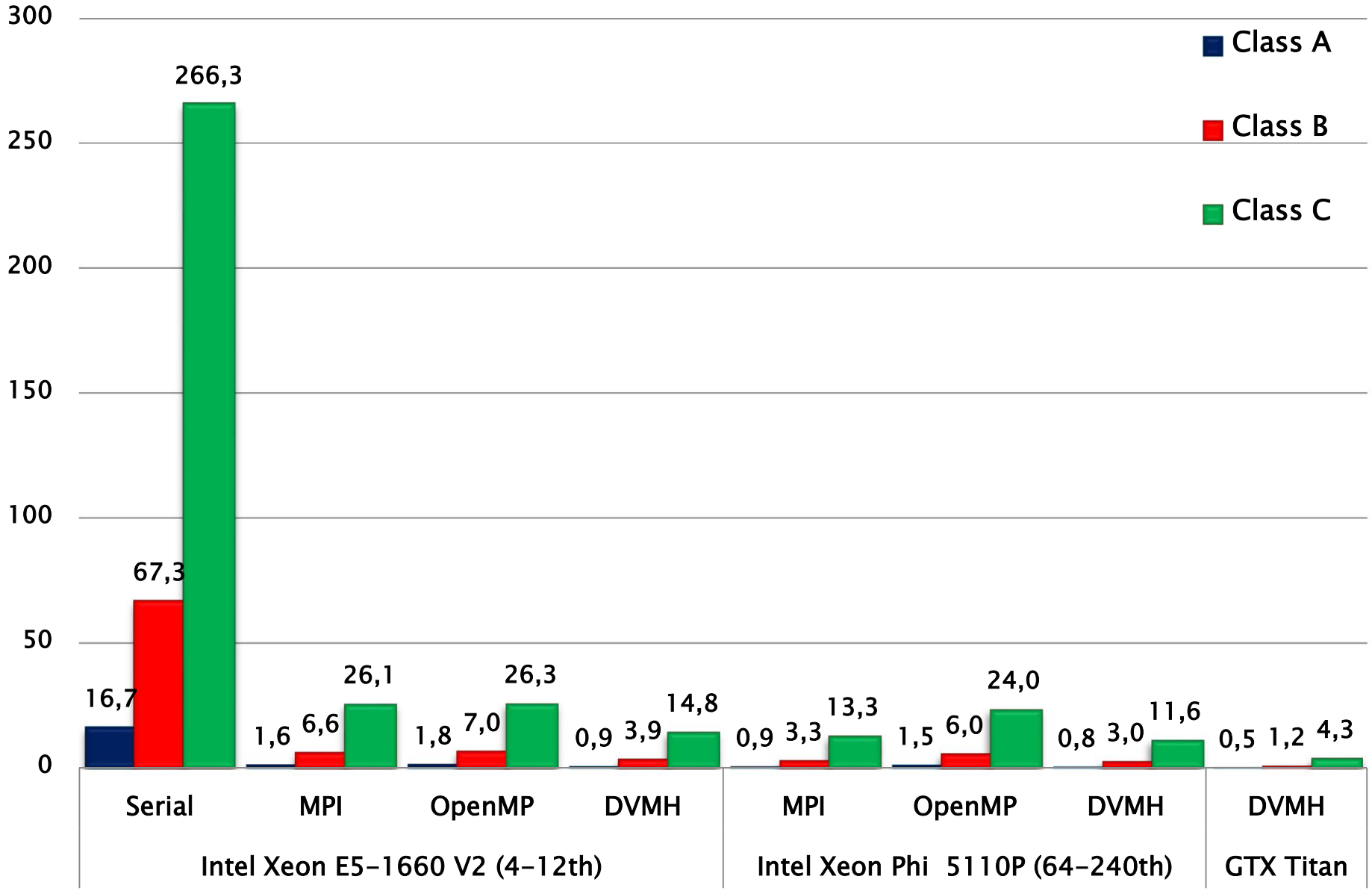


Распараллеливание тестов NAS

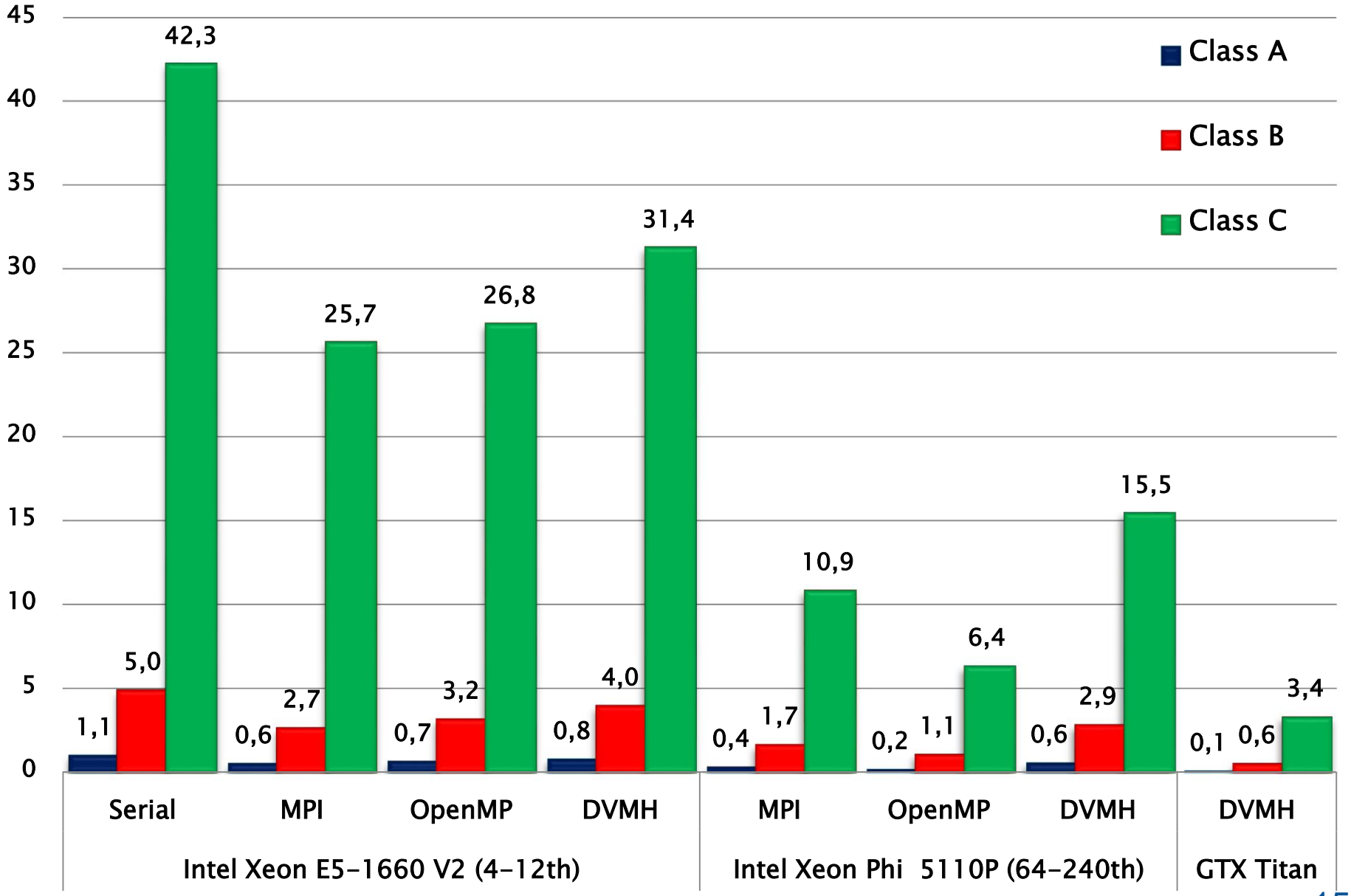
- **EP** - генерация пар случайных чисел Гаусса
- **MG** - приближенное решение трехмерного уравнения Пуассона. Метод MultiGrid
- **BT** - 3D Навье-Стокс, блочная трехдиагональная схема. Метод переменных направлений
- **LU** - 3D Навье-Стокс. Метод последовательной верхней релаксации
- **SP** - 3D Навье-Стокс.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

Подробные результаты исследования эффективности доступны на сайте http://www.dvm-system.org

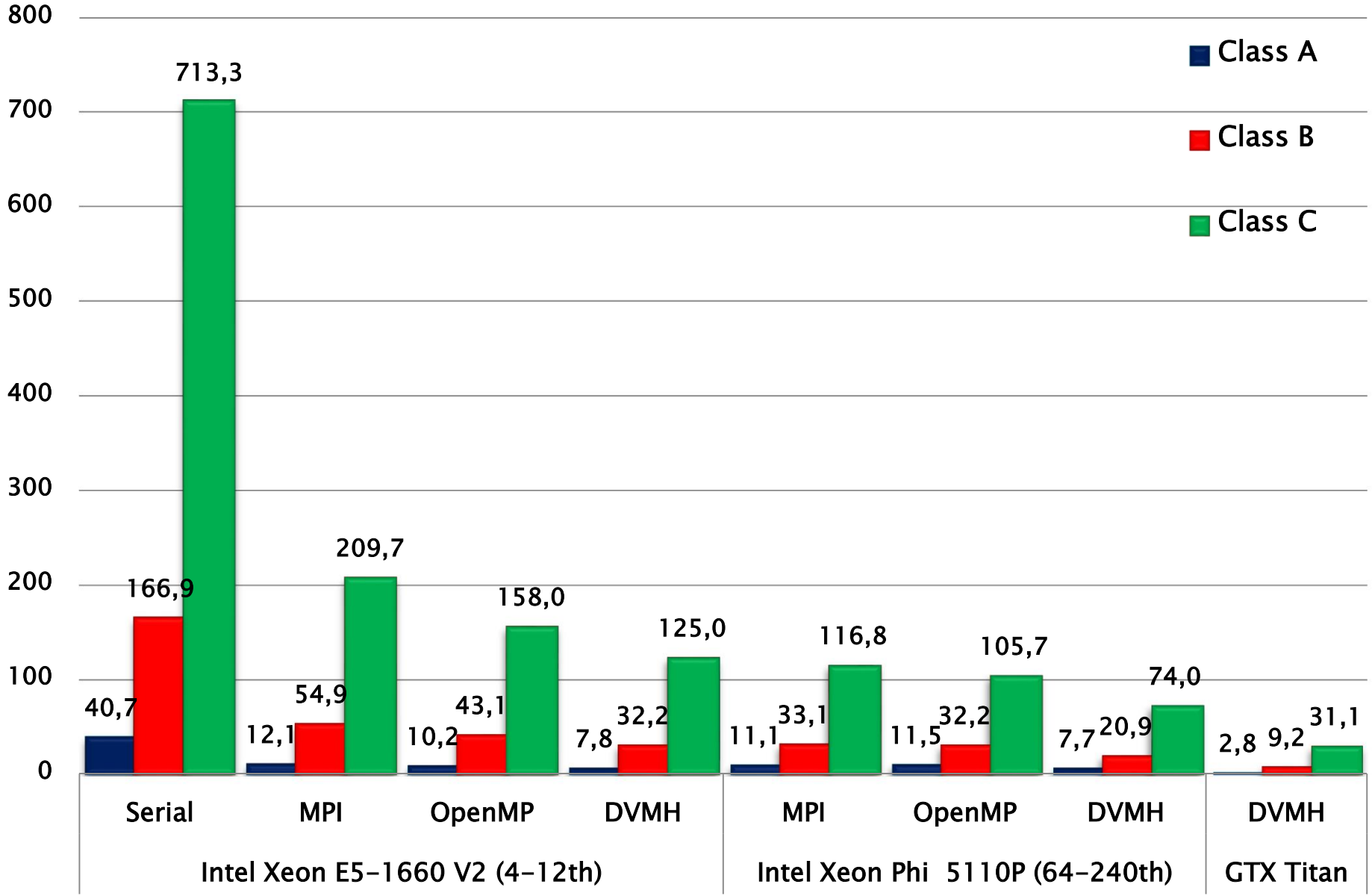
Времена выполнения теста EP, в секундах



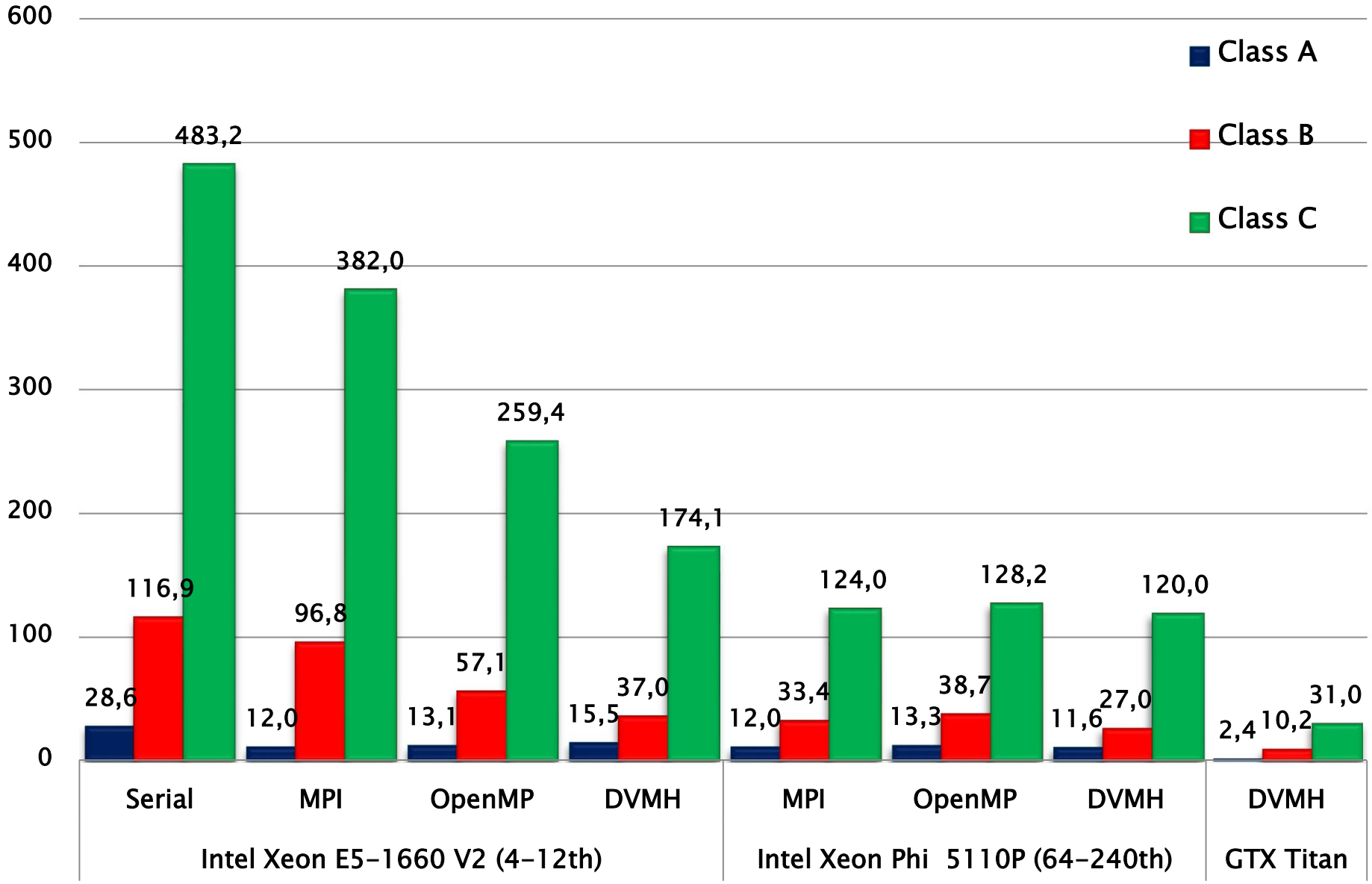
Времена выполнения теста MG, в секундах



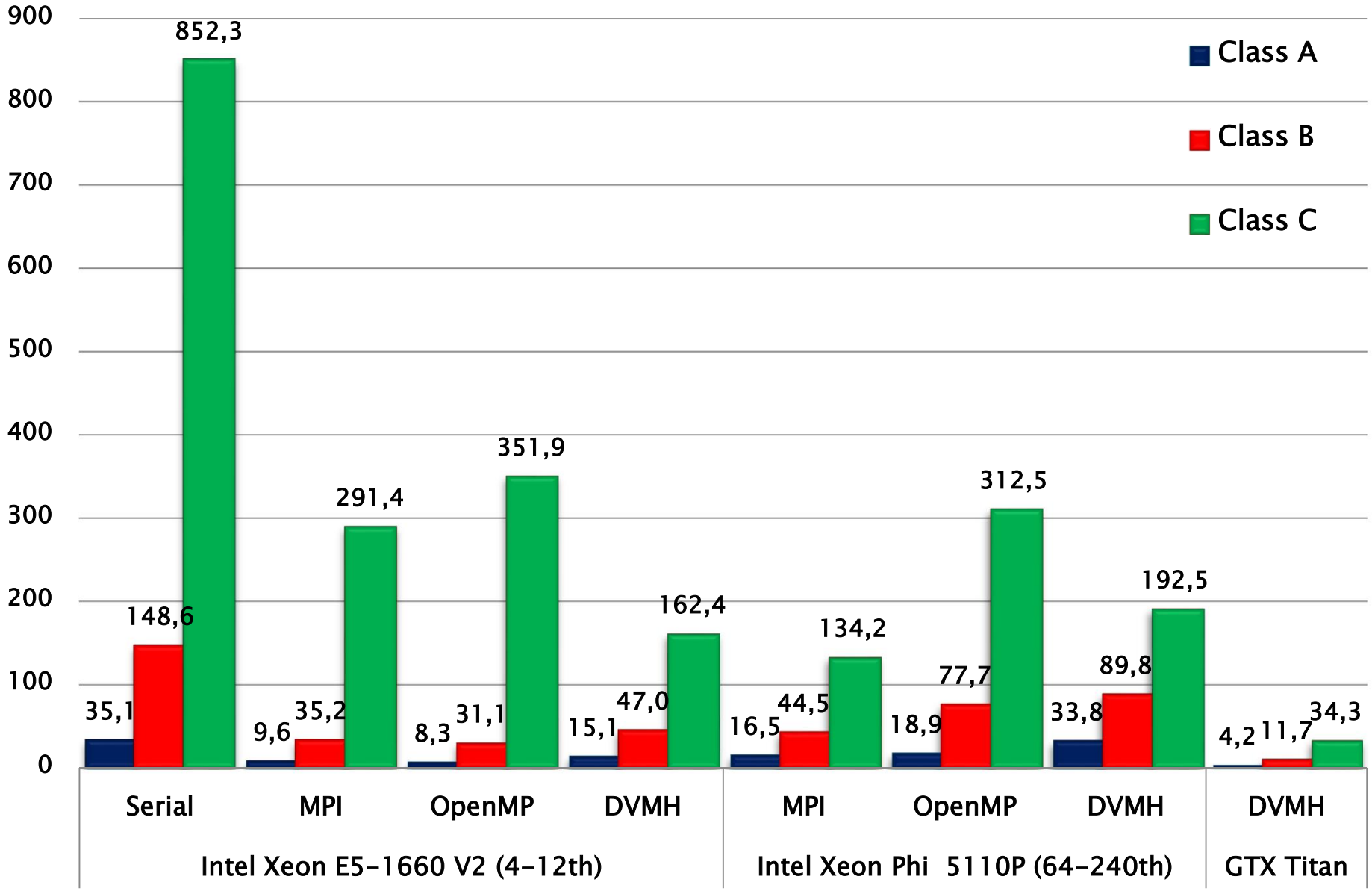
Времена выполнения теста BT, в секундах



Времена выполнения теста SP, в секундах



Времена выполнения теста LU, в секундах



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

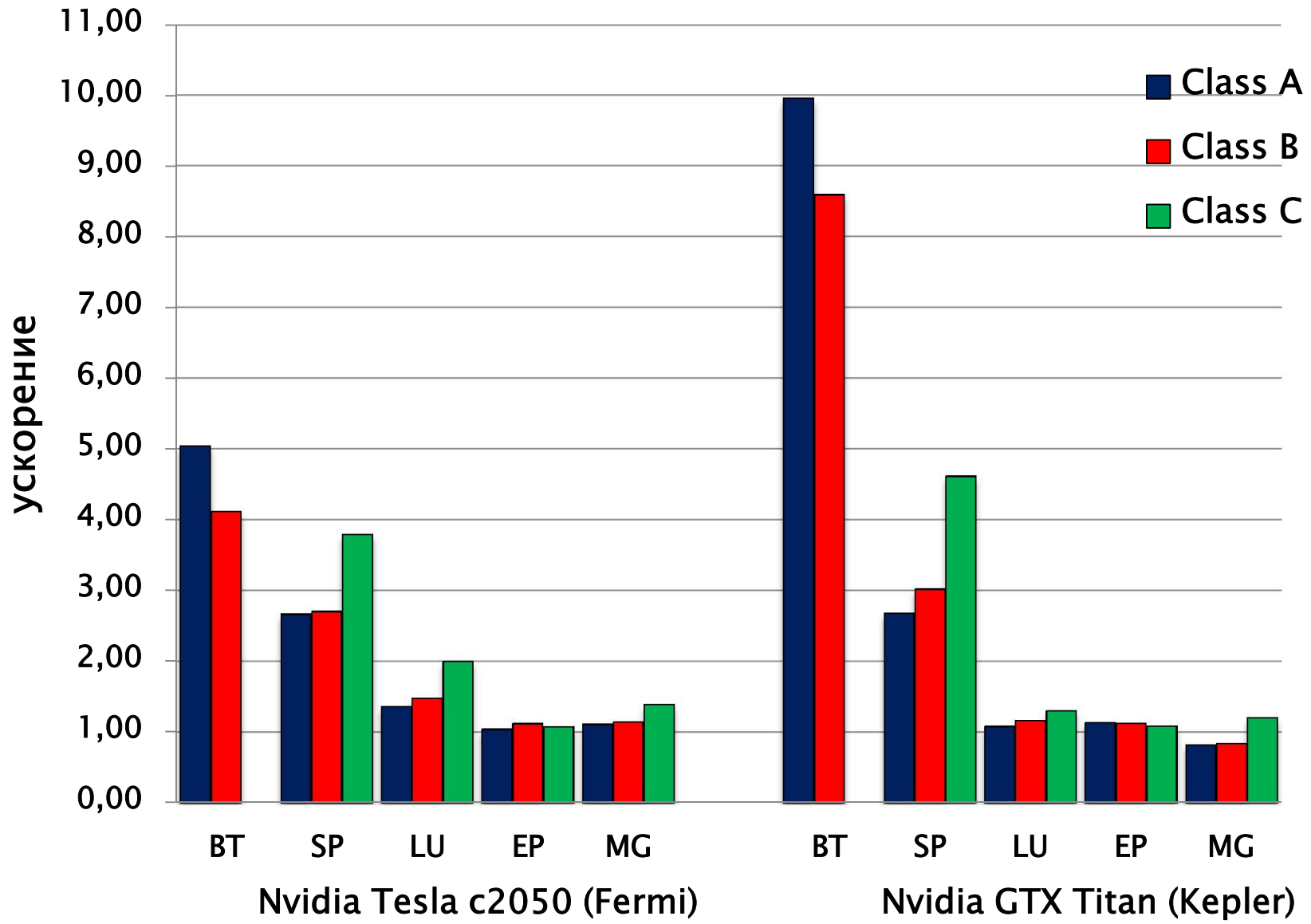
  of Rio Grande do Sul (EP, CG, FT)
- OpenACC-версии

  HPC Tools group, Department of Computer

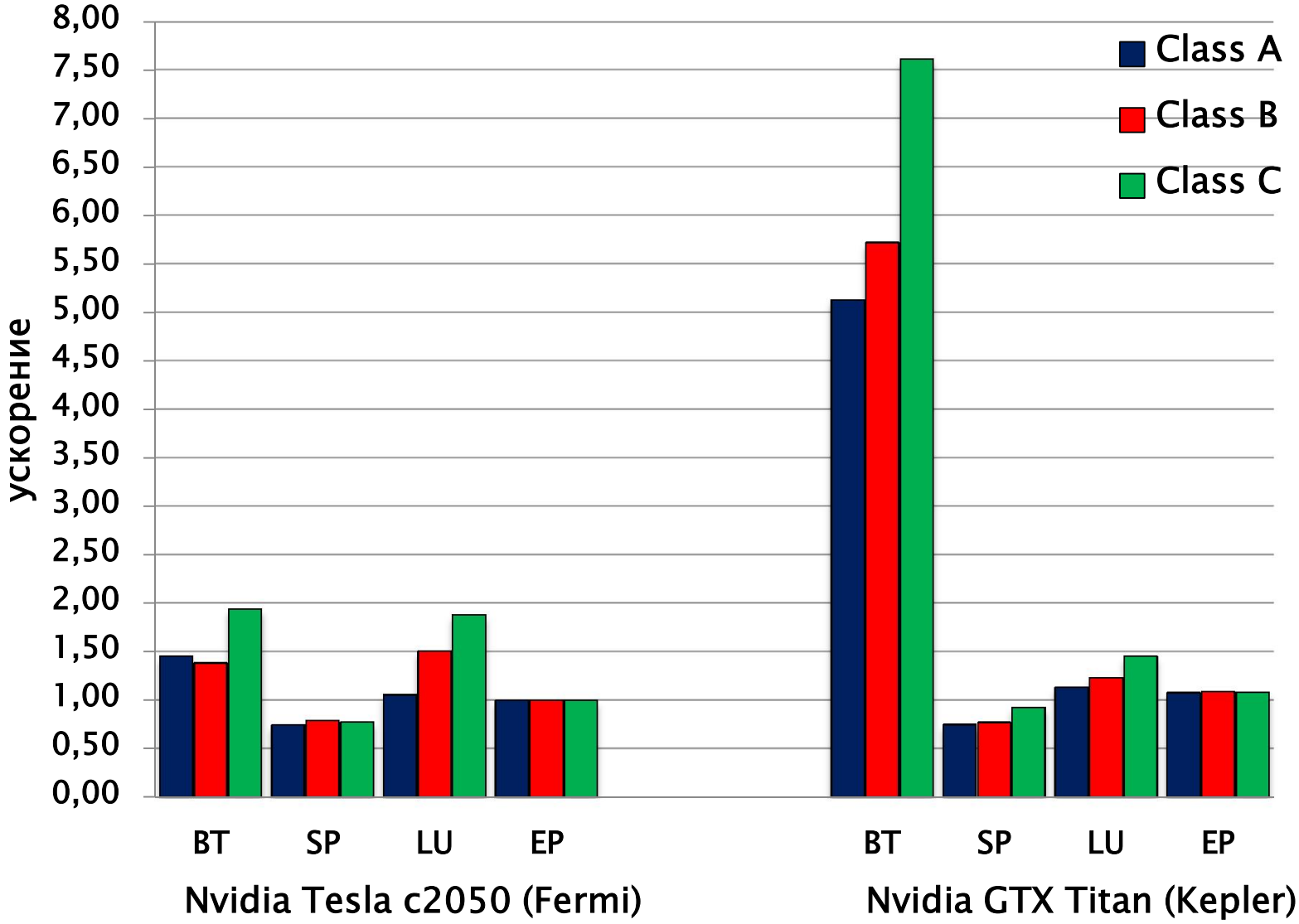
  Science University of Houston

  Rengan Xu, Barbara Chapman et al.

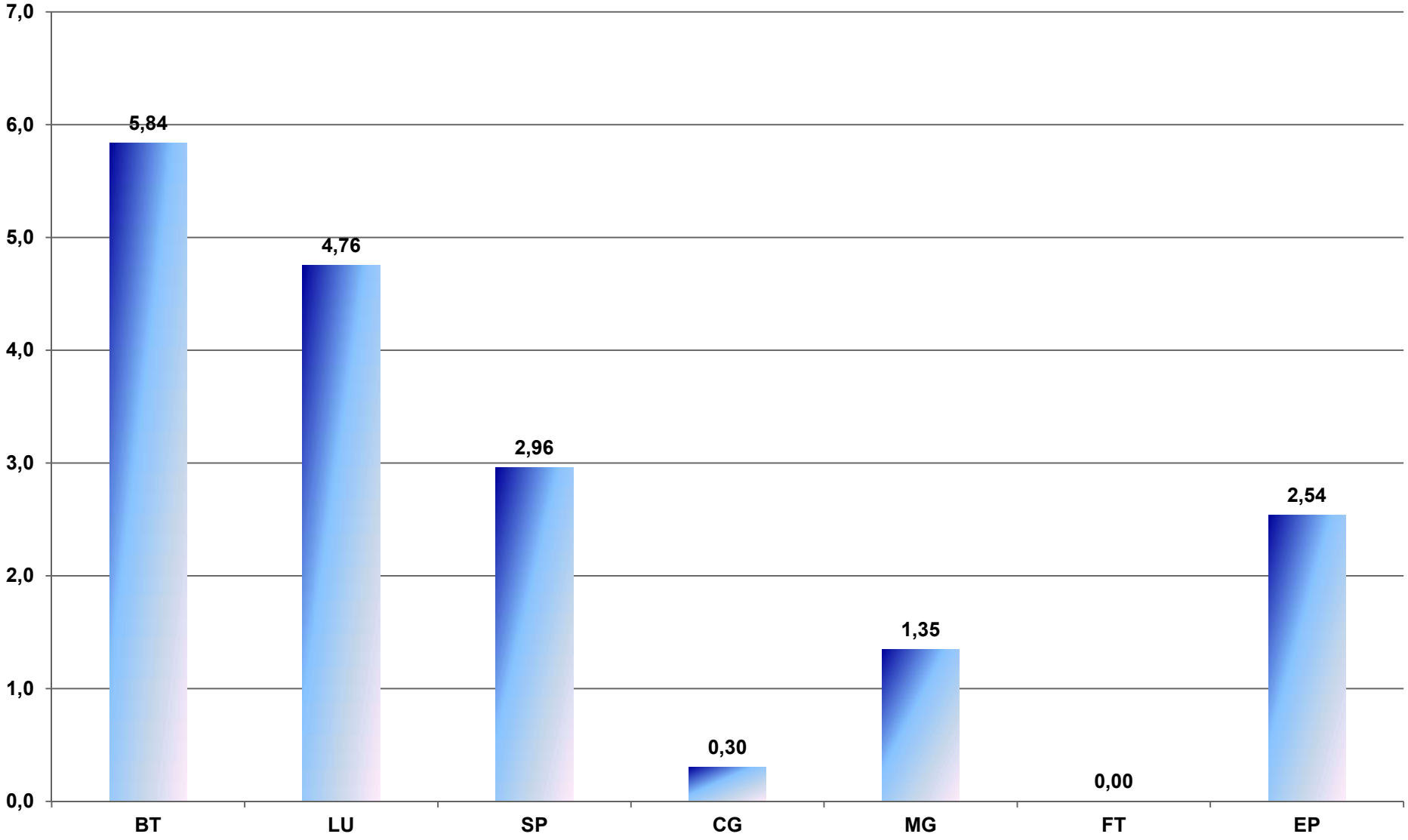
Ускорение выполнения DVMH-версий тестов NAS NPB по сравнению с OpenCL-версиями



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



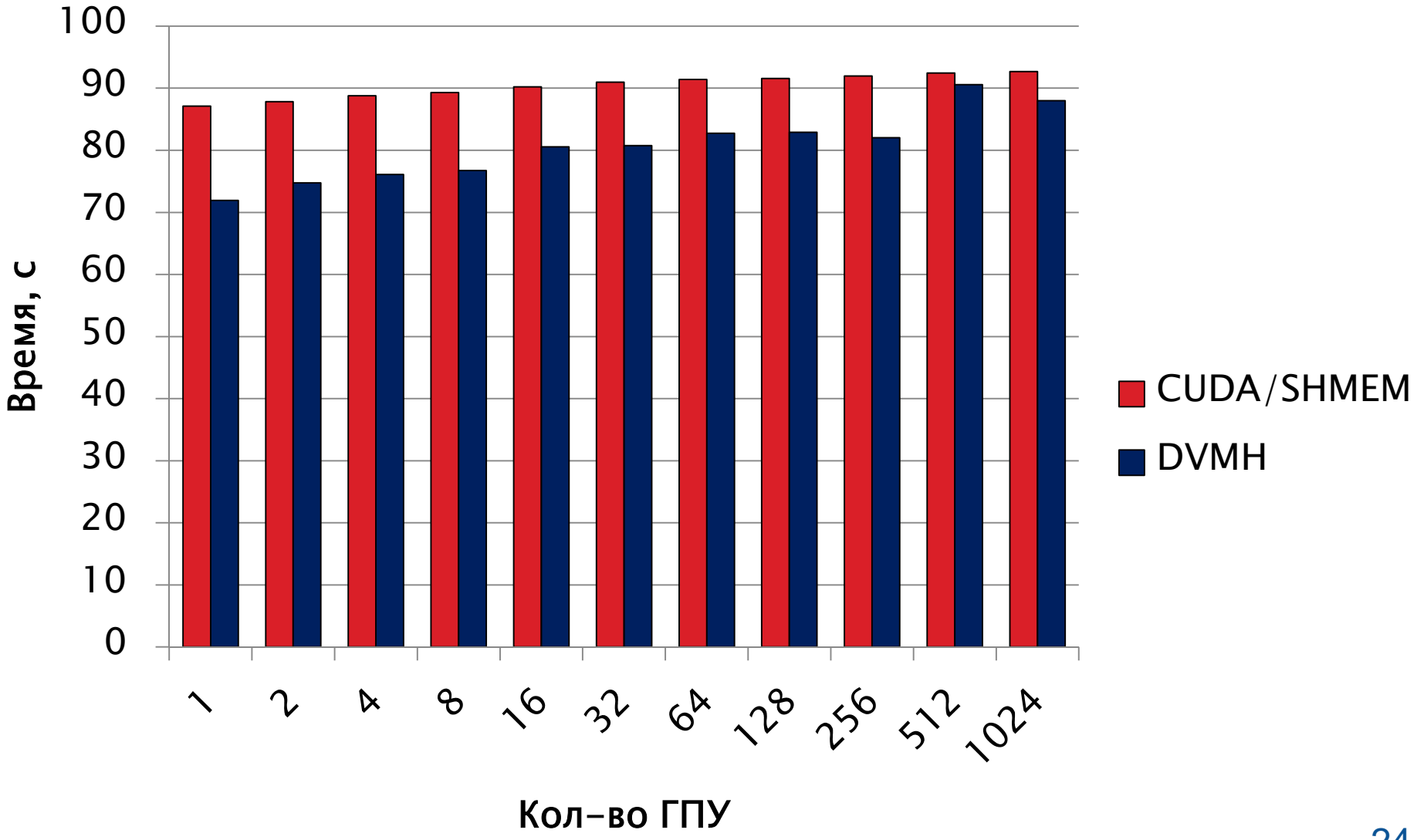
Ускорение выполнения DVMH-версий тестов NAS NPB по сравнению с OpenACC версиями (класс C)



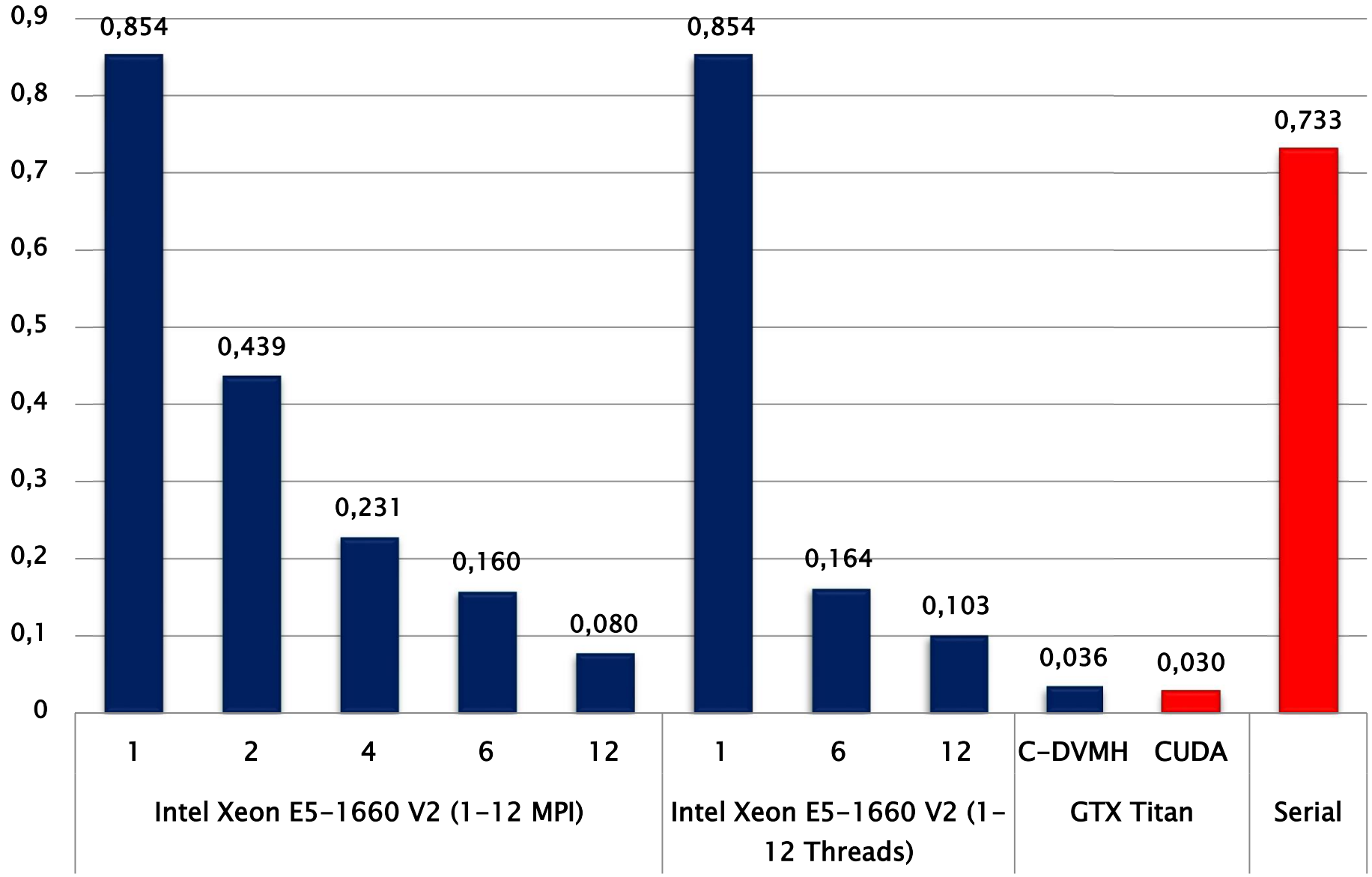
Применение DVMH для приложений

- **Контейнер** - моделирование течения вязкой тяжелой жидкости под действием силы тяжести в прямоугольном контейнере с открытой верхней стенкой и отверстием в одной из боковых стенок в трехмерной постановке (Fortran-DVMH)
- **POLINOM** - расчет коэффициентов полиномиальной реконструкции (C-DVMH)
- **FLUXES -** расчет конвективных потоков через внутренние грани расчетных ячеек (C-DVMH)
- **Композит –** композиционная модель многокомпонентной фильтрации при разработке месторождений нефти и газа (Fortran-DVMH)

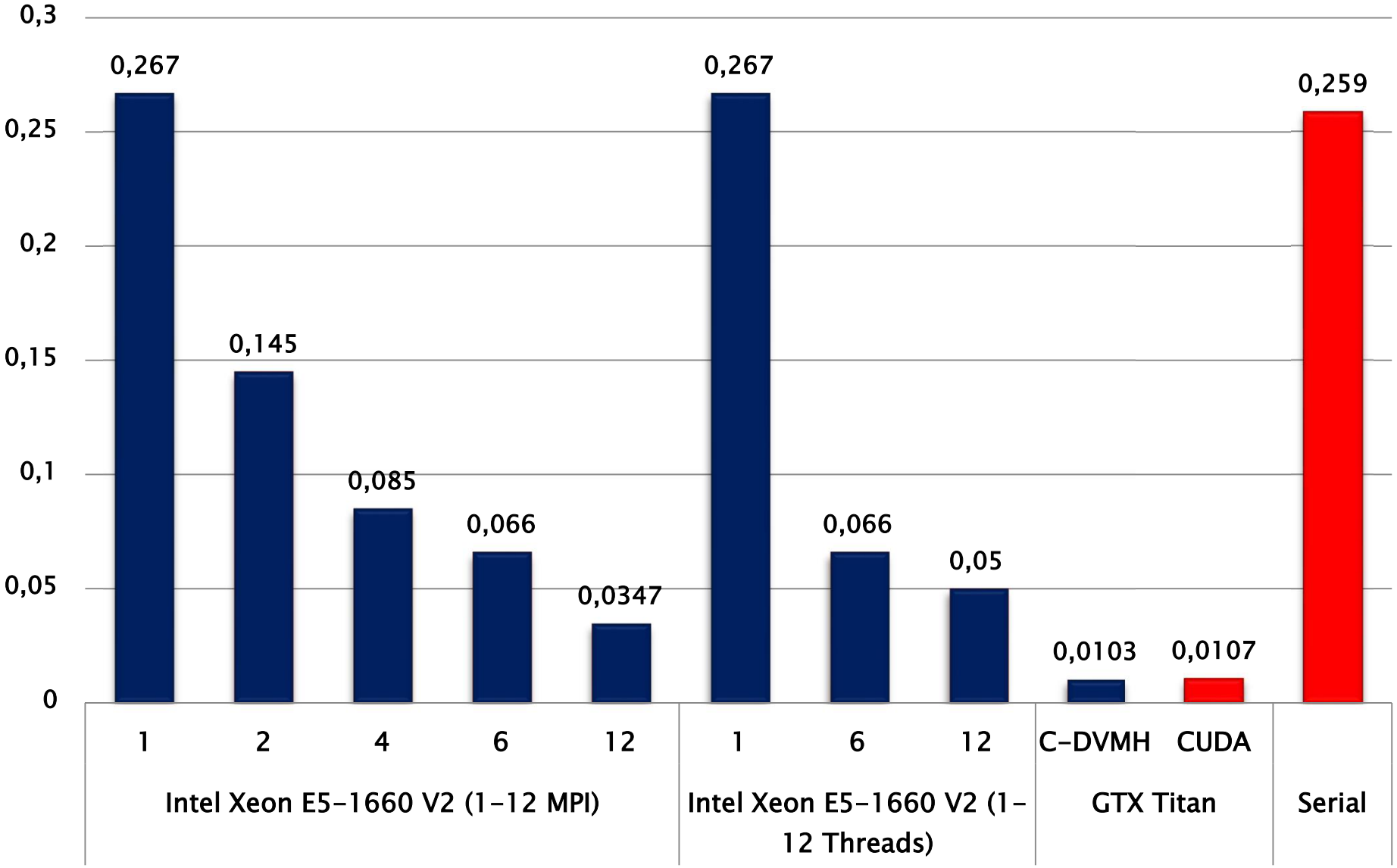
Программа Контейнер.
Сравнение DVMH vs C+SHMEM+CUDA



Времена выполнения программы POLINOM



Времена выполнения программы FLUXES



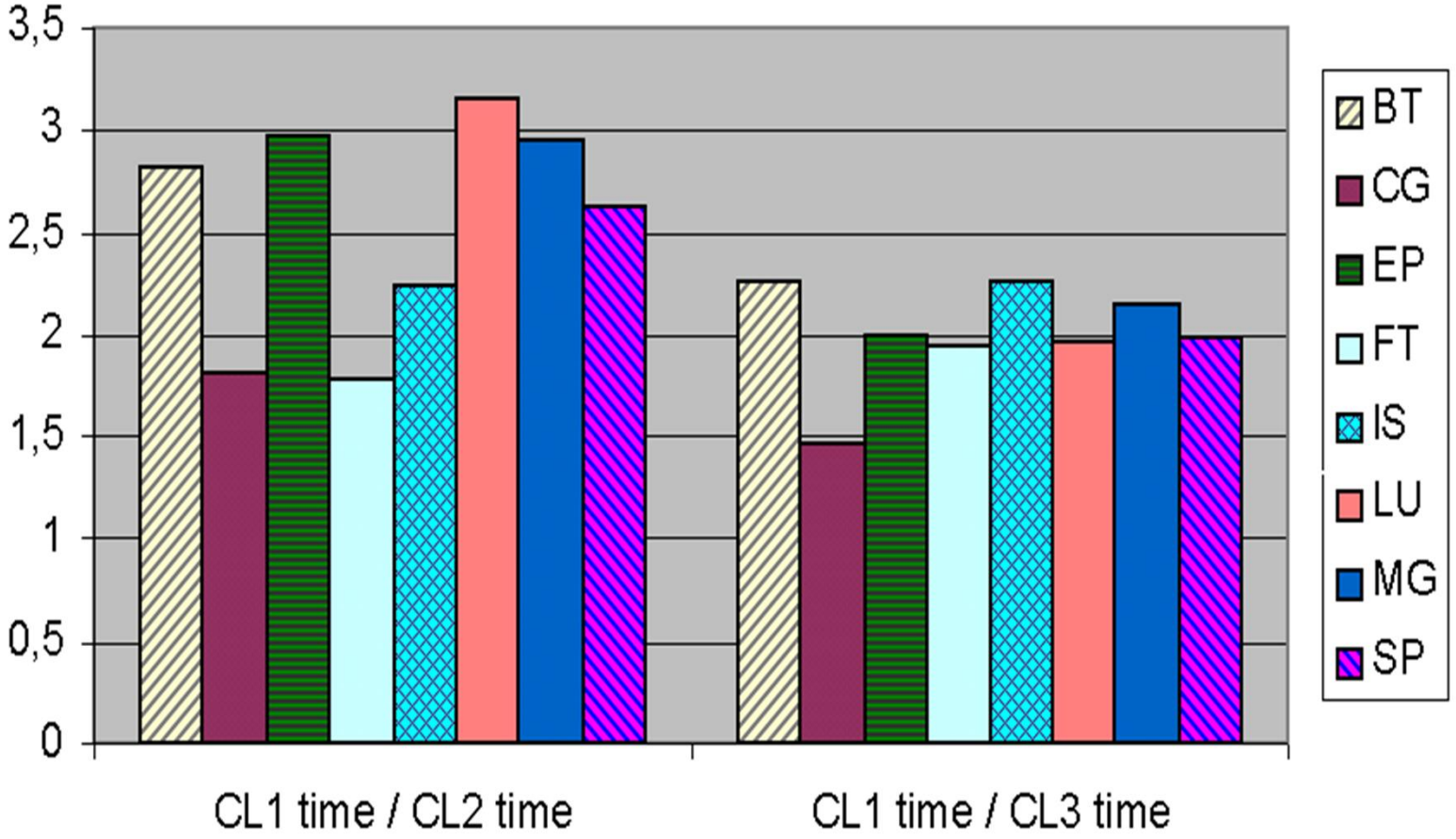
Методы динамической настройки DVMH-программ

- 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

Методы динамической настройки DVMH-программ

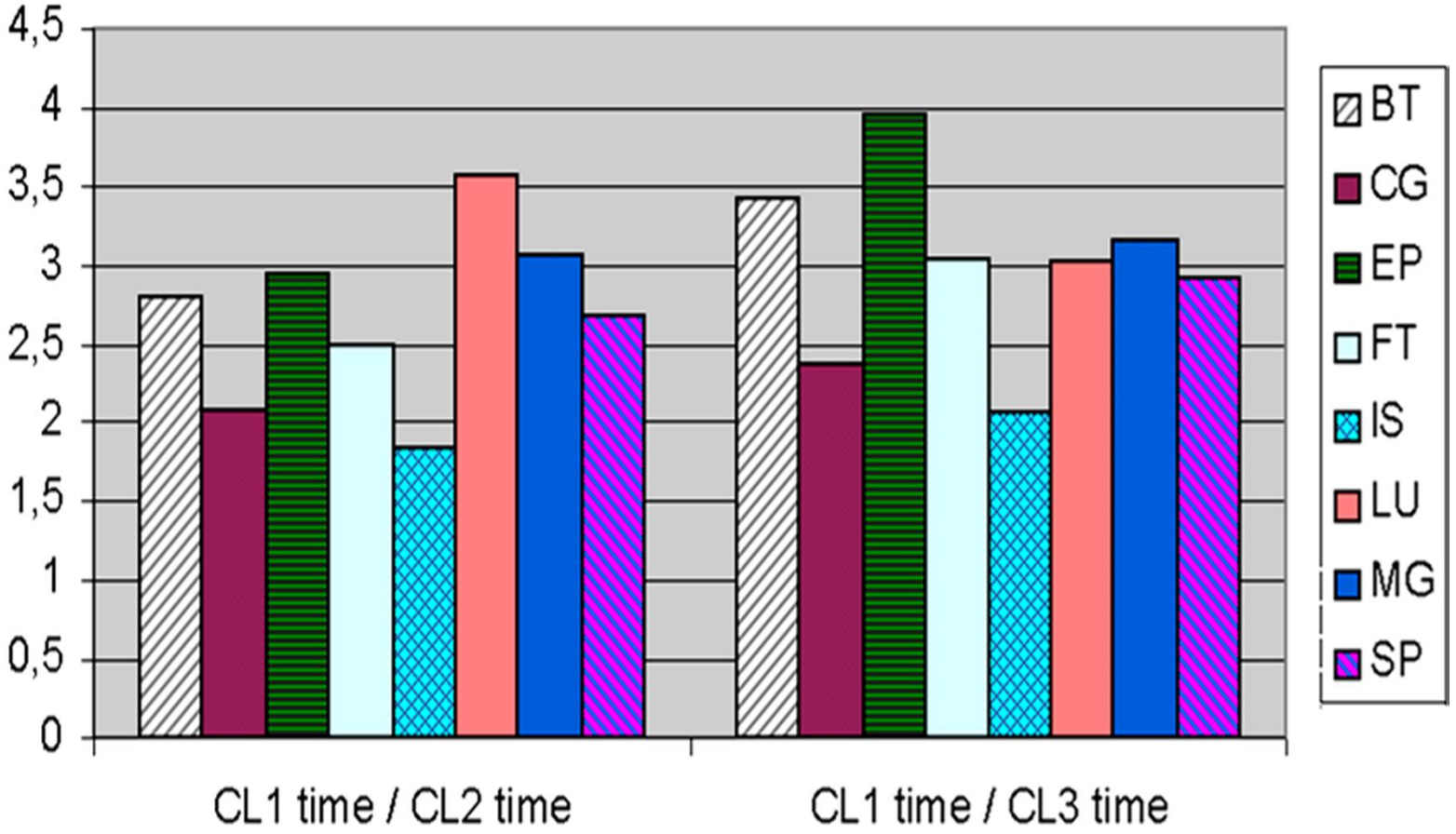
- Отображение массивов и циклов на узлы кластера с учетом их производительности
- Отображение массивов и циклов на устройства узла с учетом их производительности
- Трансформация массивов
- Использование динамической компиляции

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

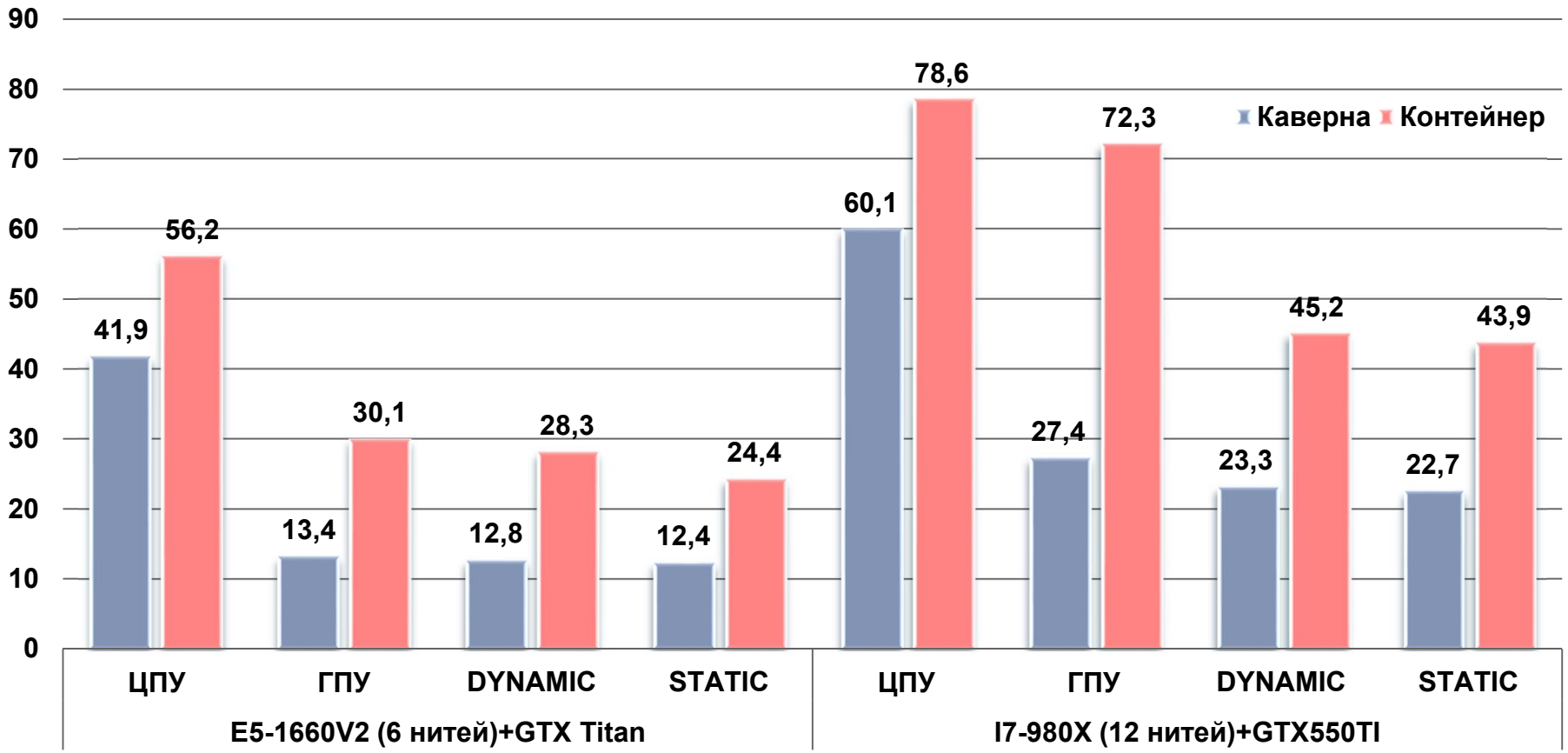
Динамический режим

- 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

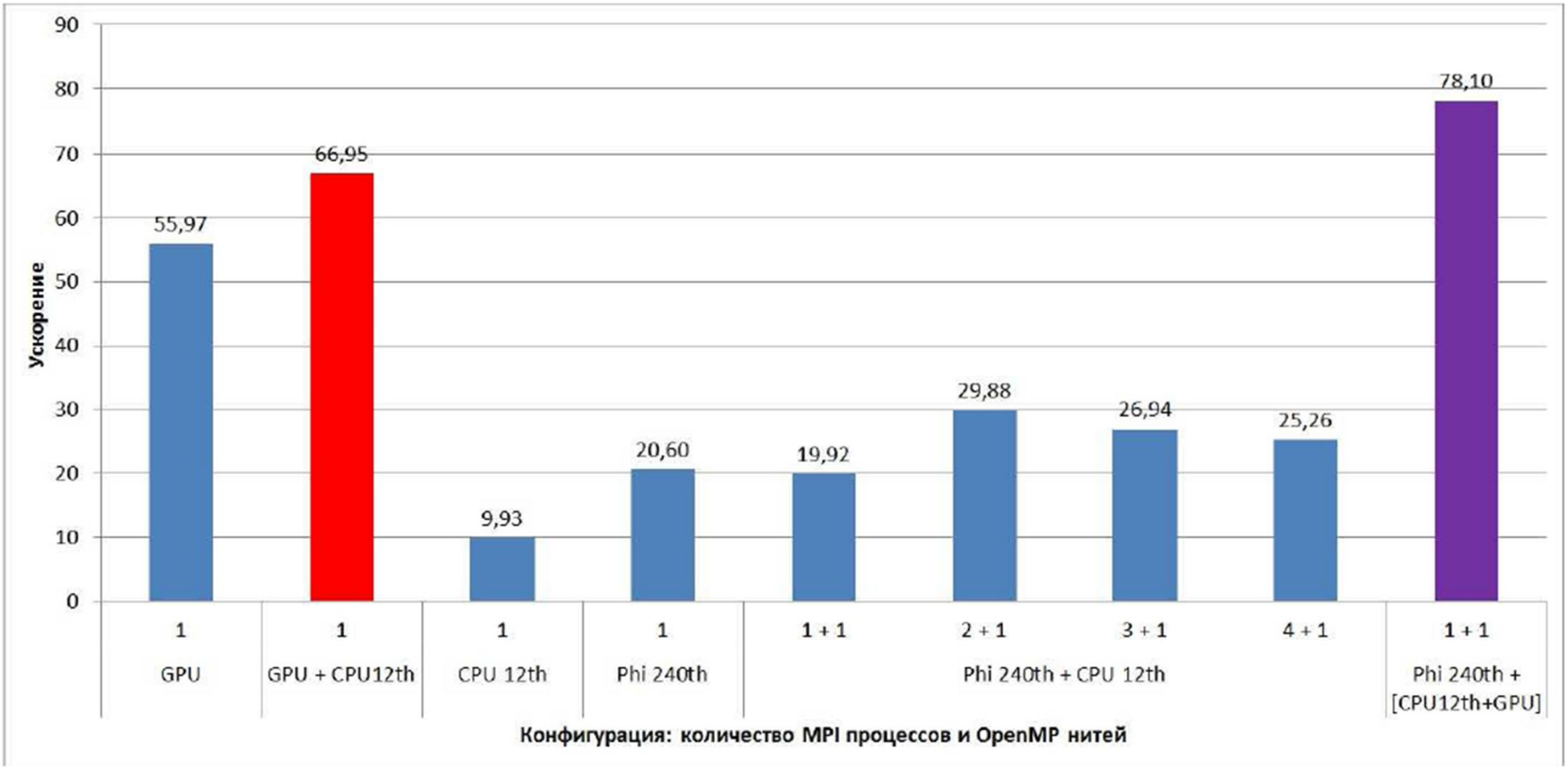
Зависимость производительности от объема вычислений

```
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

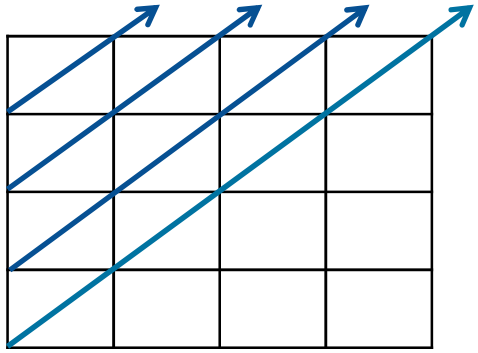
- Отображение массивов и циклов на узлы кластера с учетом их производительности
- Отображение массивов и циклов на устройства узла с учетом их производительности
- Трансформация массивов
- Использование динамической компиляции

```
#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;
}
```

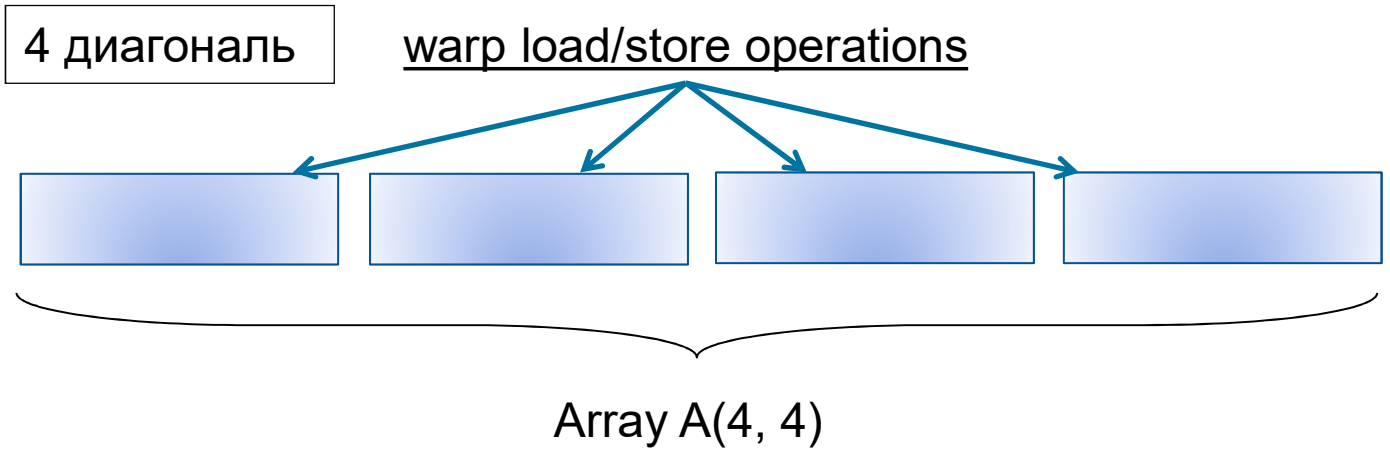
Алгоритм SOR в модели DVMH

Выполнение гиперплоскостями

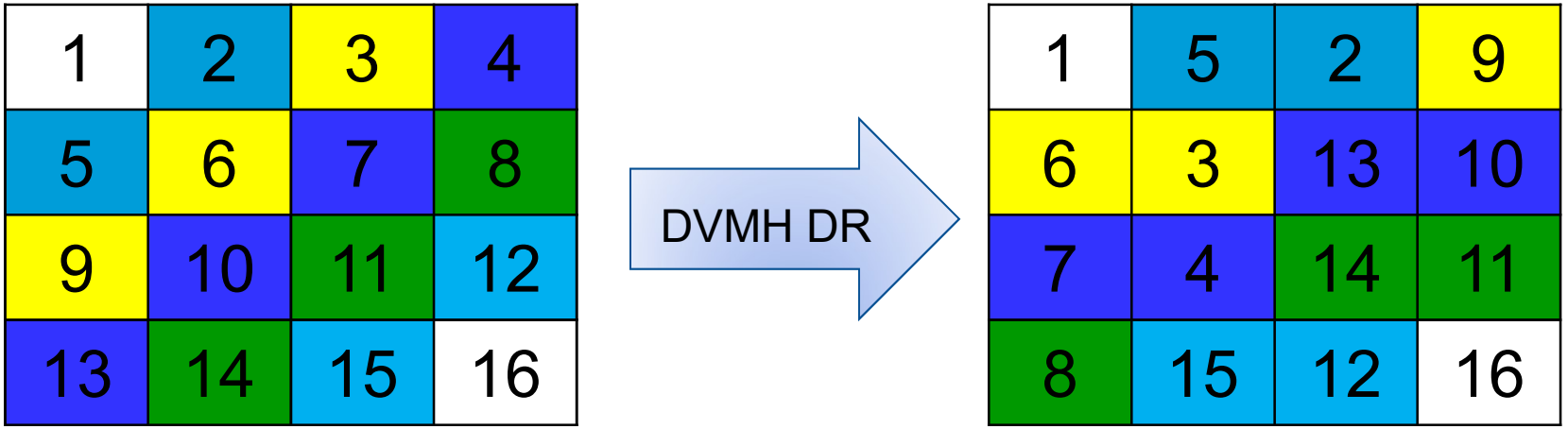
```
double  A[N][N];
```



```
#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] …;
```



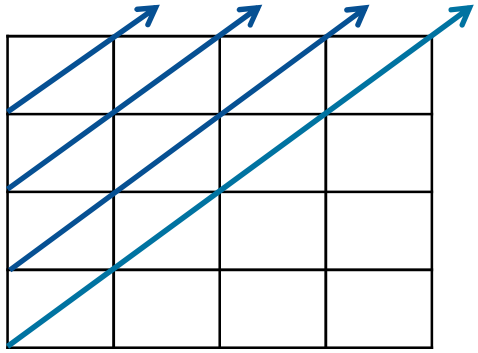
Динамическая транформация данных



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

Выполнение гиперплоскостями

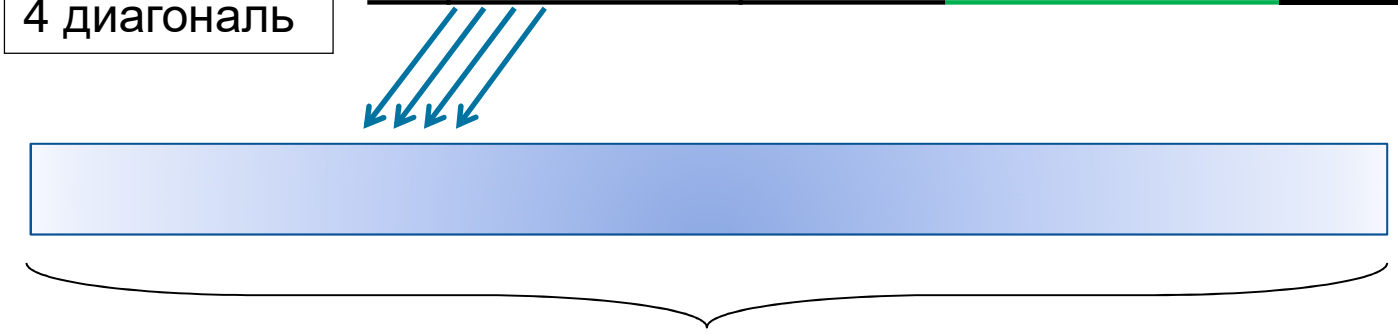
double A[N][N];



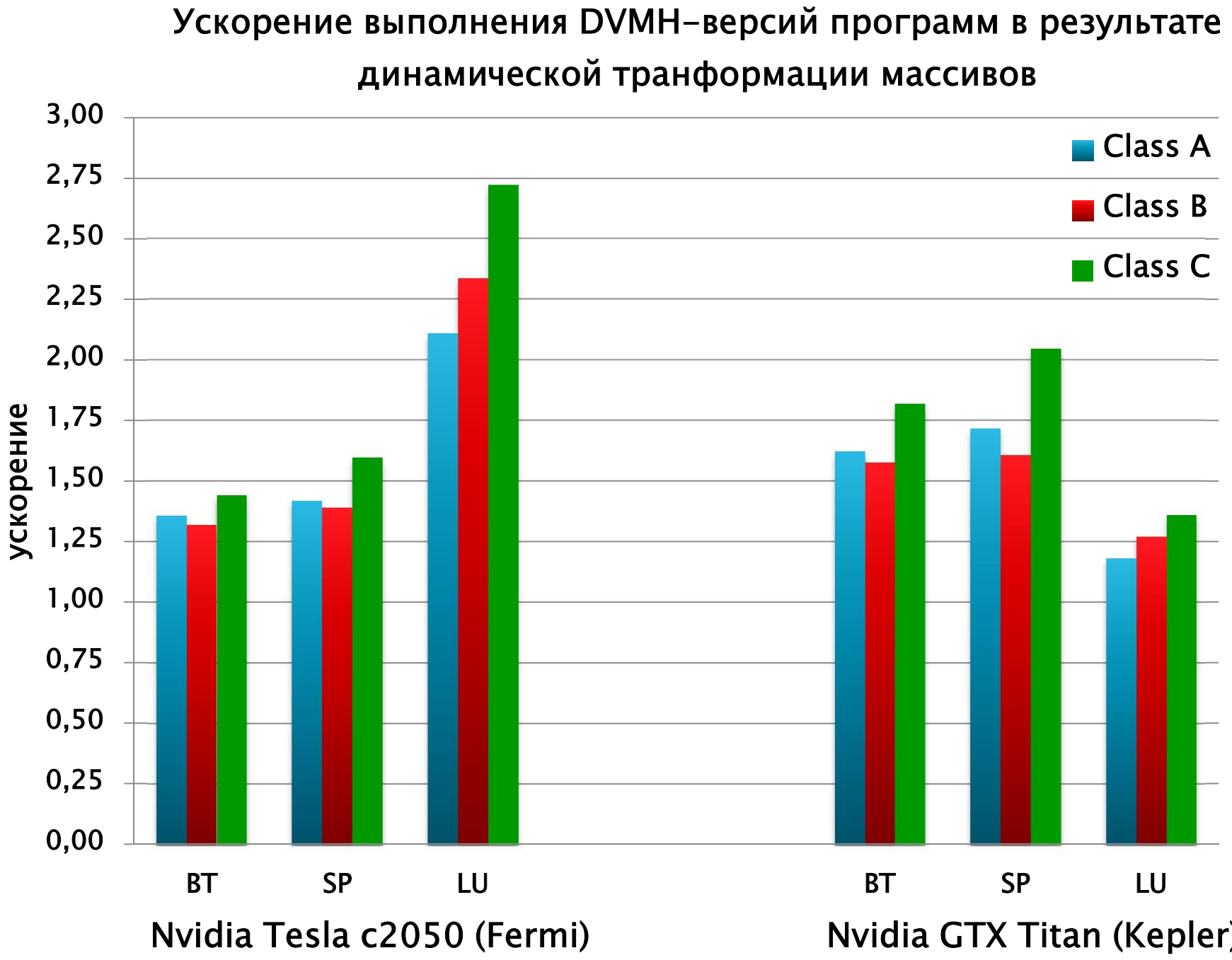
```
#pragma dvm parallel([i][j] on A[i][j]), across(A[1:1][1:1])
 for (i = 1; i < N - 1; i++)
      for (j = 1; j < N - 1; j++)
          A[i][j] = A[i - 1][j] + A[i + 1][j] + A[i][j - 1] …;
```

4 диагональ

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



Array A(4, 4)


Ускорение выполнения DVMH-версий программ в результате динамической транформации массивов
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
ускорение
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	35%	74	66.5	11%
x_solve	153	112	27%	64.5	42.9	34%
y_solve	132	107	19%	64.4	42.6	34%
z_xolve	128	107	17%	53	48	10%

- Общая производительность всей программы увеличилась на 28%

Режимы ввода-вывода в DVM-системе

- **Последовательный синхронный ввод-вывод**
- **Параллельный синхронный ввод-вывод:**
  - **В локальный файл**
  - **В параллельный файл**
- **Асинхронный параллельный ввод-вывод**

```
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", "wbsl");
   #pragma dvm get_actual(B)
   fwrite(B, sizeof(double), L * L, f);
   fclose(f);
   return 0;
}
```

Алгоритм Якоби в модели DVMH

```
      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$ IO_MODE(LOCAL, ASYNCHRONOUS)
      OPEN(3, FILE='JAC.DAT', ACCESS='STREAM')
!DVM$ GET_ACTUAL(B)
      WRITE(3, *) B
      CLOSE(3)
      END
```

Алгоритм Якоби в модели DVMH

```
FILE *f;
double A[L][L];
double B[L][L];
int main(int argc, char *argv[]) {
   for(int it = 0; it < ITMAX; it++) {
      #pragma dvm region
      {
         #pragma dvm parallel(2)
         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(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.;
      }
   }
  f = fopen("jacobi.dat", "wbsl");
  #pragma dvm get_actual(B)
  fwrite(B, sizeof(double), L * L, f);
  fclose(f);
  return 0;
}
```

Алгоритм Якоби в модели DVMH

Последовательный синхронный ввод-вывод

1) Файл открывается только одним процессом из текущей многопроцессорной системы (ВВ/ВЫВ).
2) Все операции над файлом производит только процесс ВВ/ВЫВ.
3) При запросе ВВ/ВЫВ распределенного массива т.к. данные распределены и их объем таков, что не позволяет сконцентрировать их в одном процессе, чтение (или запись) делается порциями около 100МБ, после (перед) которого (которой) делается рассылка (сбор) данных на (с) процессы (процессов) ими владеющие (владеющих).
4) Если операция предполагает запись пользовательских переменных или возврат значения, то они рассылаются процессом ВВ/ВЫВ.
5) Если в процессе выполнения операции возникла ошибка и стандарт предписывает установку errno, то его значение также рассылается процессом ВВ/ВЫВ.
6) Операции не над файловыми дескрипторами (remove, rename, tmpnam) выполняются процессом ВВ/ВЫВ текущей многопроцессорной системы с рассылкой результатов выполнения на остальные процессы.

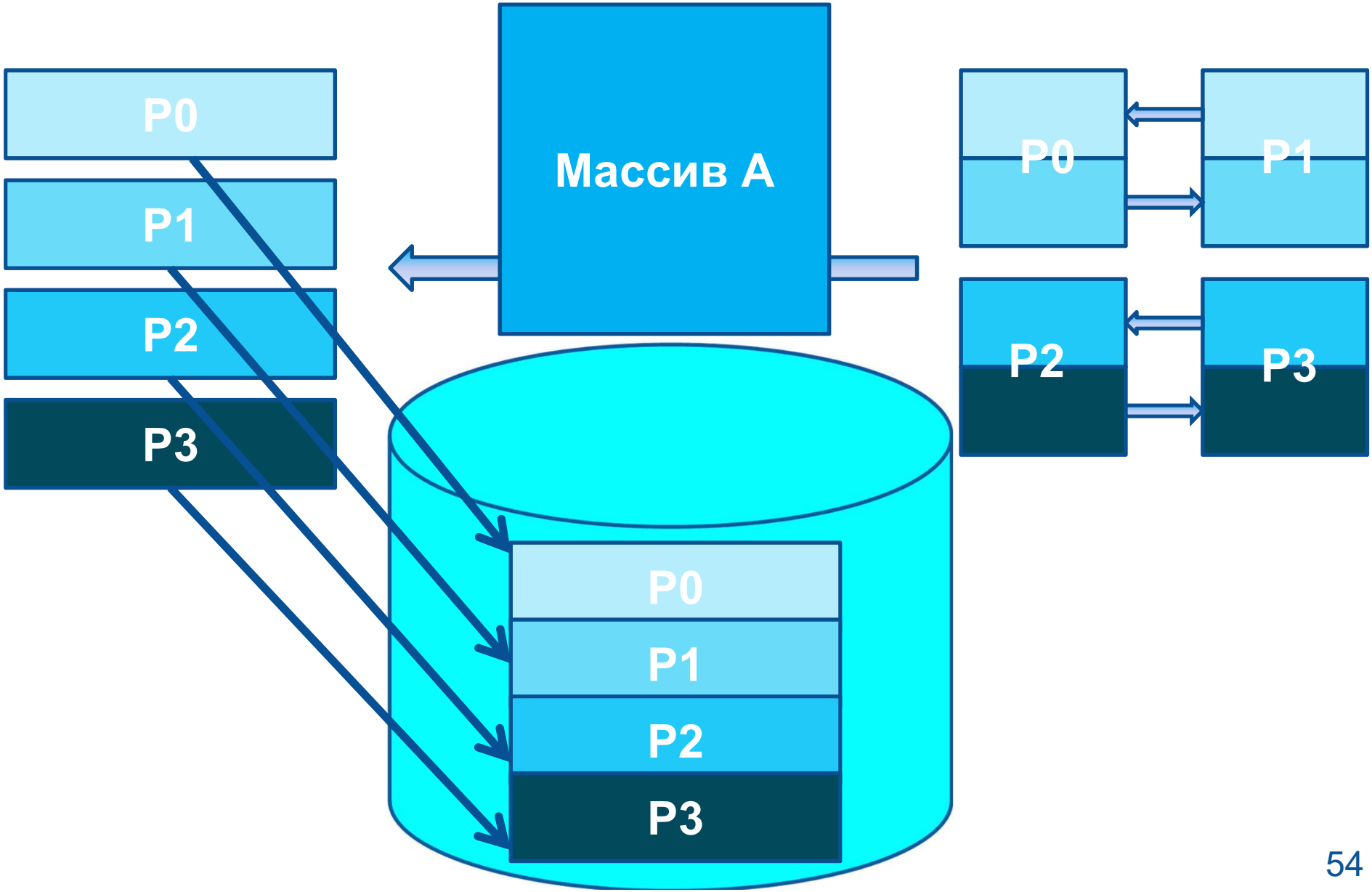
Параллельный синхронный ввод-вывод

- **Локальный файл**
  Открывается каждым процессом независимо. Все операции над локальными файлами обрабатываются каждым процессом независимо и никаких коммуникаций не вызывают. При чтении (или записи) распределенного массива из локального файла, читается (или записывается) только локальная часть распределенного массива. Требует совпадения распределений при записи файлов и при их последующем чтении.
- **Параллельный файл**
  Содержимое файла сохраняется таким же, как если бы он записывался последовательной программой, что позволяет читать файлы, записанные программой, работавшей на любом количестве процессов.
  Параллельный файл открывается каждым процессом.
  Все операции над параллельными файлами выполняются так же, как и над обычными файлами, за исключением операций fread, fwrite, fputs.

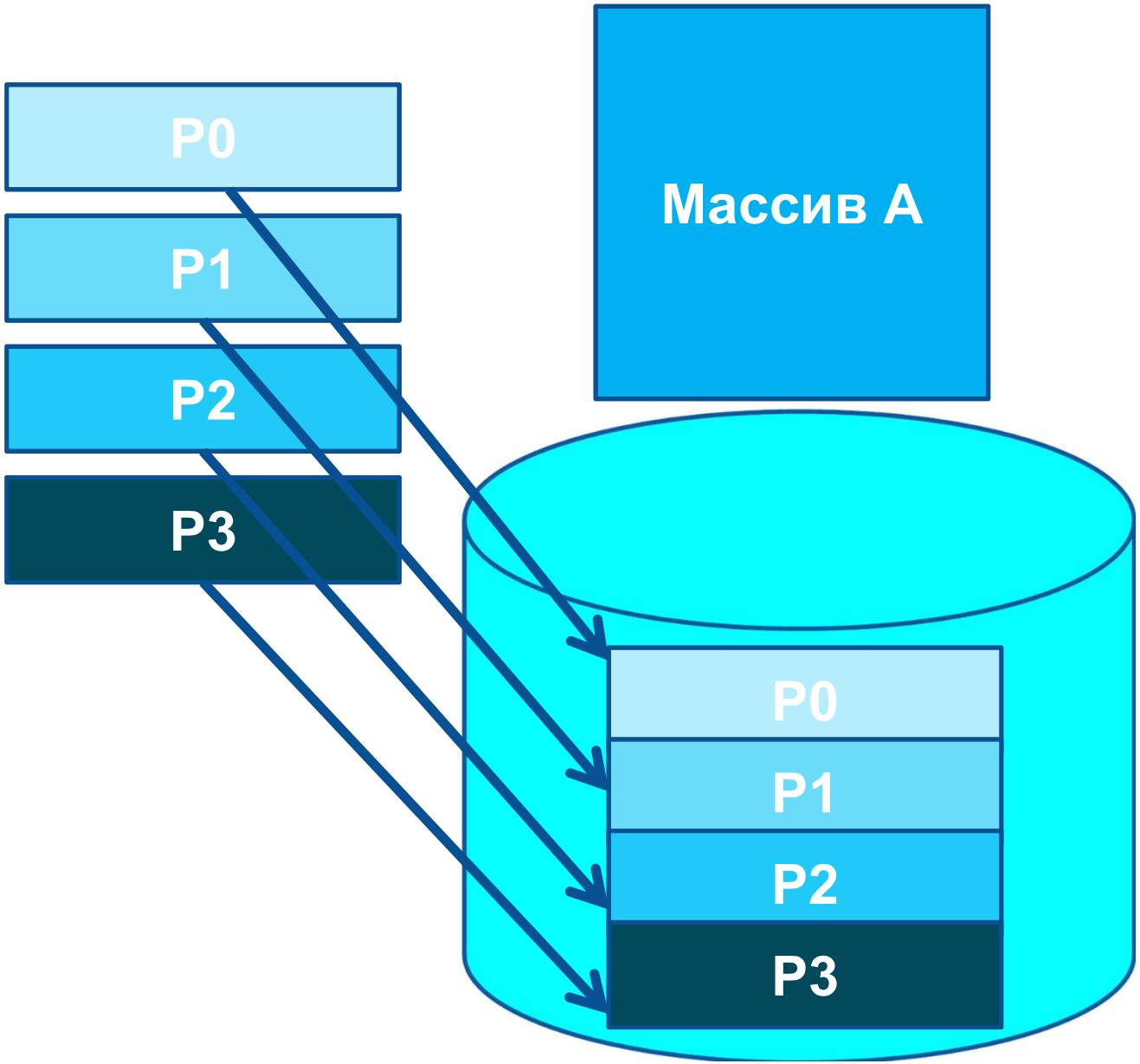
Реализация операций fread, fwrite, fputs

1) Передать позицию в файле от процесса ВВ/ВЫВ всем остальным процессам текущей многопроцессорной системы.
2) Каждый процесс устанавливает позицию в файле в ту точку, с которой он будет читать (или записывать).
3) Каждый процесс выполняет свою часть работы.
   - Если запрошена операция чтения в размноженную переменную, то после чтения части производится объединение прочитанных участков (операция типа Allgather).
   - Если запрошена операция с распределенным массивом, то каждый процесс выполняет сбор нужных ему данных со всех других процессов перед записью либо выполняет рассылку после чтения своей части файла (операция типа Alltoall).
4) Все процессы синхронизируют свои изменения на файловую систему. Для синхронизации содержимого файла в памяти с содержимым на диске используется системный вызов fdatasync в Linux или _commit в Windows.
5) Процесс ВВ/ВЫВ устанавливает позицию за последним участком чтения (или записи).

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



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



Асинхронный параллельный ввод-вывод

Асинхронными могут быть: обычные, локальные и параллельные файлы.

Для обработки асинхронных операций каждым процессом заводится задаваемое параметром количество нитей, формирующих пул.

Все операции над одним файлом выполняются последовательно.

Операции над разными файлами могут выполняться разными нитями ввода-вывода.

Любая недопускающая асинхронного выполнения операция приводит к ожиданию всех асинхронных операций над соответствующим файлом перед началом выполнения синхронной операции.

Операция будет синхронной, если ее возвращаемое значение используется в программе или она должна устанавливать errno в случае ошибки.

Нити осуществляют все необходимые для работы с файлом коммуникации с использованием MPI.

Для сериализации асинхронных операций над каждым файлом используется механизм зависимых задач и параллельного выполнения графа задач.

Параллельный ввод-вывод в языке C-DVMH

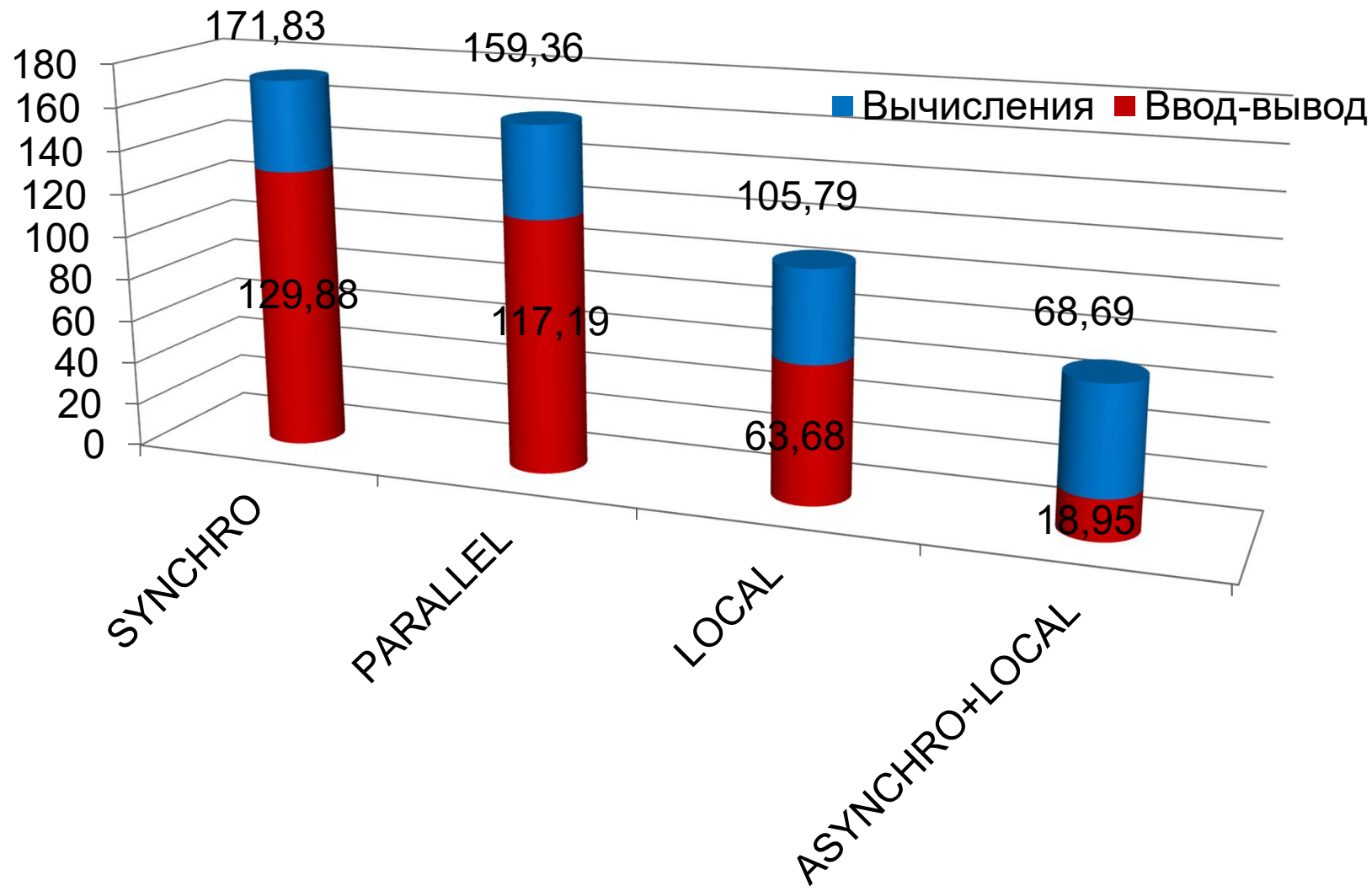
```
const char *mode = "wb";
FILE *cp = fopen("jac_%02d.dat", mode);
```

Параметр mode	Режим ввода-вывода
“wb”	Синхронный последовательный ввод-вывод
“wbl”	Синхронный параллельный ввод-вывод в локальный файл
“wbp”	Синхронный параллельный ввод-вывод в параллельный файл
“wbs”	Асинхронный последовательный ввод-вывод
“wbsl”	Асинхронный параллельный ввод-вывод в локальный файл
“wbsp”	Асинхронный параллельный ввод-вывод в параллельный файл

Параллельный ввод-вывод в языке Fortran-DVMH

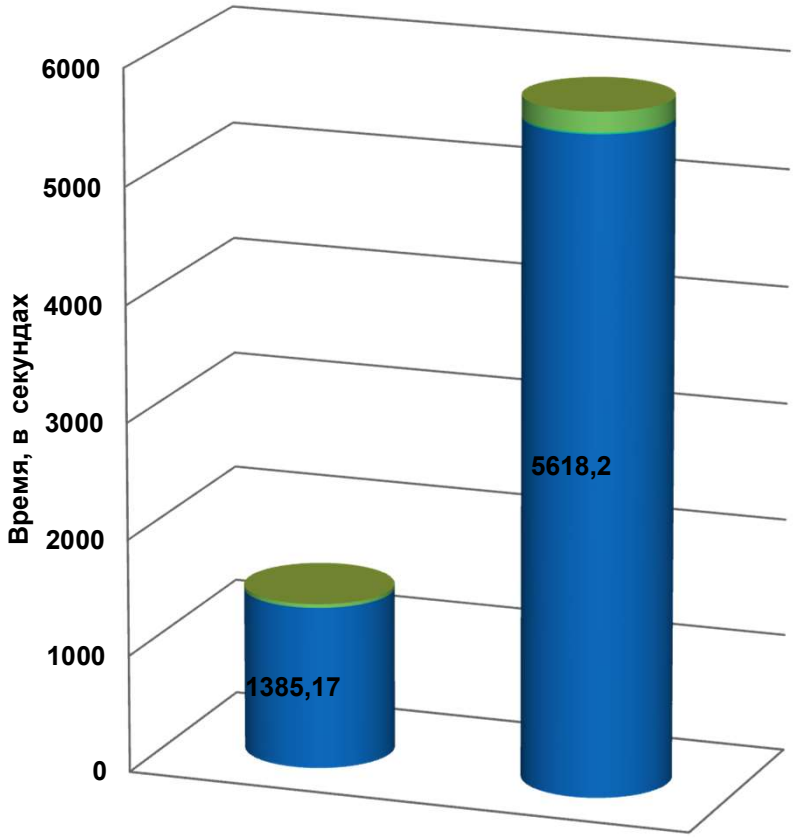
```
!DVM$ IO_MODE ([PARALLEL]
                [[,]LOCAL]
                [[,]ASYNC])

      PARAMETER (L=16000)
      DOUBLE PRECISION A(L,L), B(L,L)
!DVM$ DISTRIBUTE ( BLOCK, BLOCK) :: A
!DVM$ ALIGN B(I,J) WITH A(I,J)
...
!DVM$  IO_MODE (LOCAL, ASYNC)
      OPEN(4, ACCESS='STREAM', FILE = ' DATA_%02d.DAT', ERR=44)
        ...
      WRITE(4) A(2:L-1,2:L-1), B
        ...
      CLOSE(4)
        ...
44:   PRINT *, 'ERROR HAPPENED! PROGRAM TERMINATES'
      STOP
```

Времена выполнения 100 итераций программы Якоби 32000x32000 при сохранении 10 контрольных точек на СК «Ломоносов»



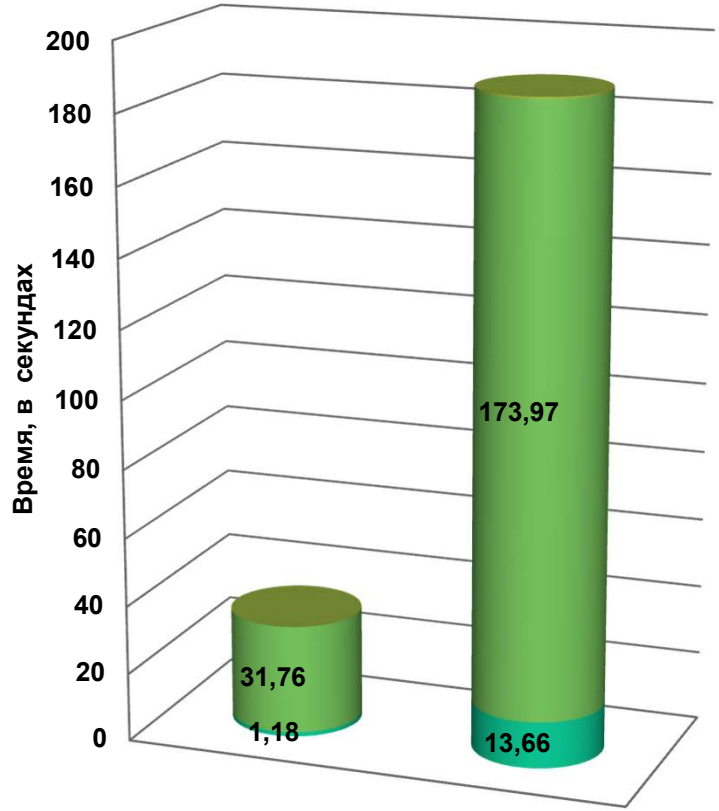
МВС-10П. Intel Xeon E5-2690 vs Intel Xeon Phi 7110X
Программа Якоби, 10800x10800, 10000 итераций

Полное время работы



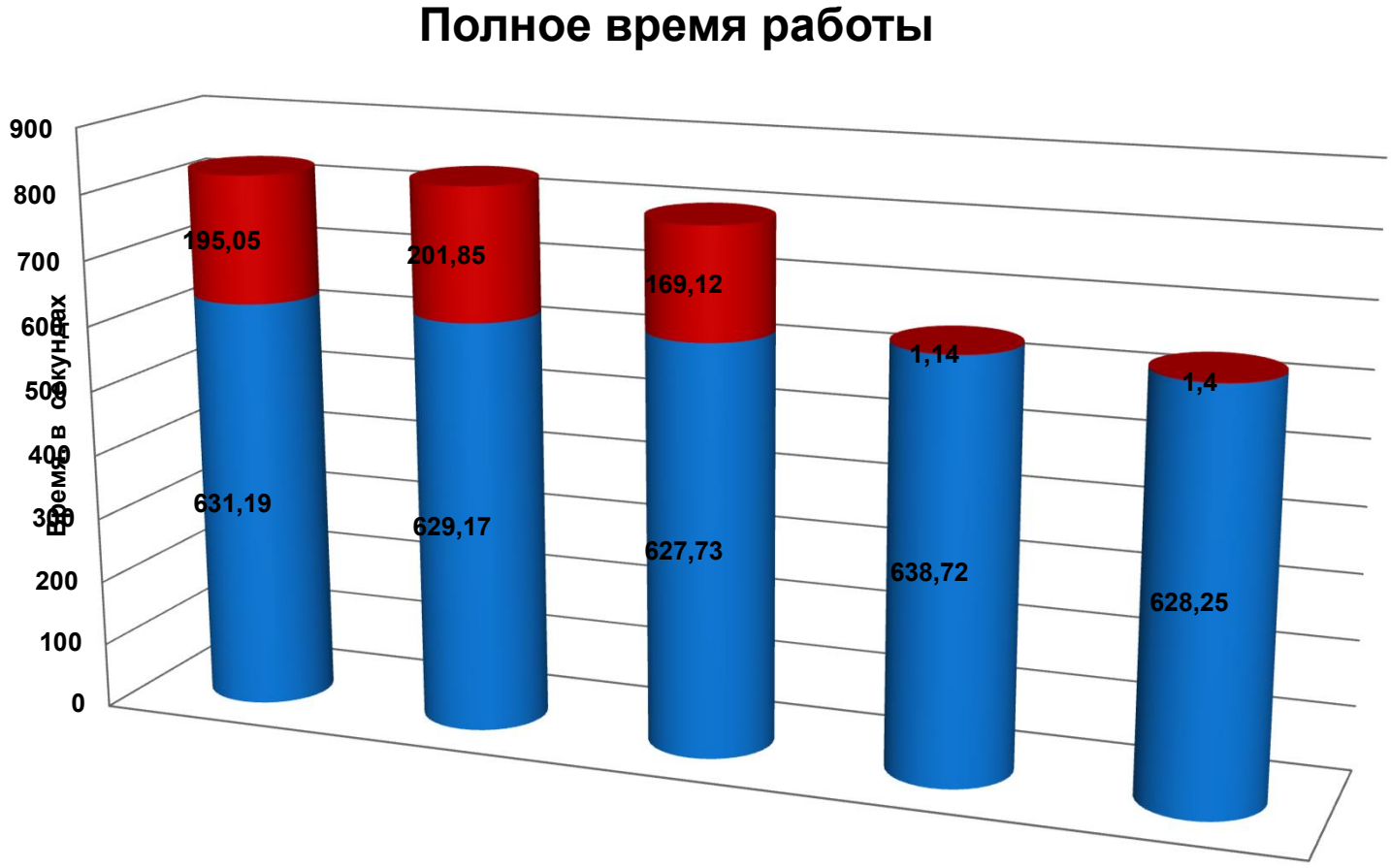
	XEON	XEON PHI
REMOVE	0	0,69
FCLOSE	31,76	173,97
FWRITE	1,18	13,66
FOPEN	0,09	0,01
EXE	1385,17	5618,2

Время ввода-вывода



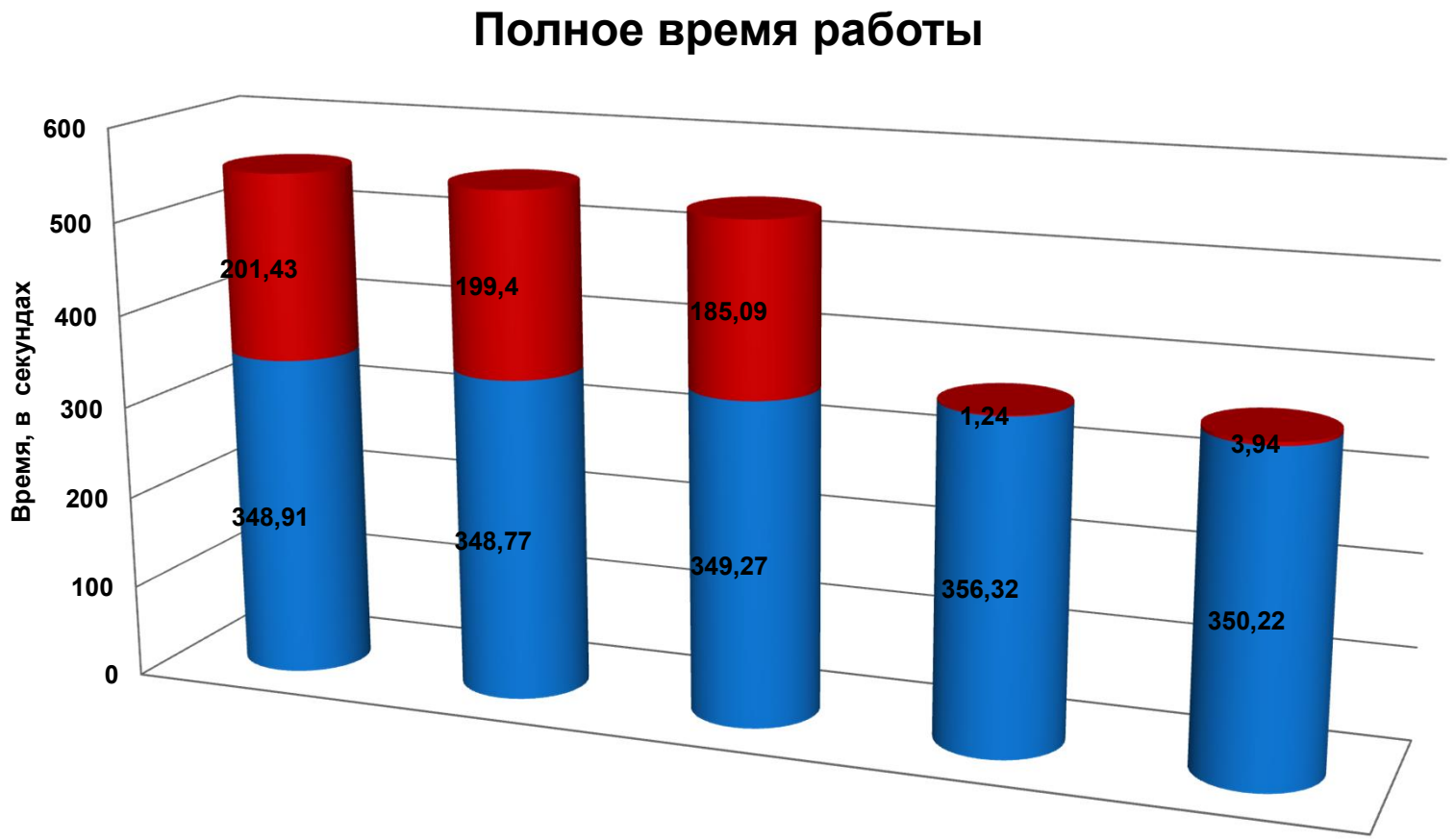
	XEON	XEON PHI
REMOVE	0	0,69
FCLOSE	31,76	173,97
FWRITE	1,18	13,66
FOPEN	0,09	0,01

МВС-10П. Ввод-вывод с Intel Xeon Phi 7110X. 10 MPI-процессов



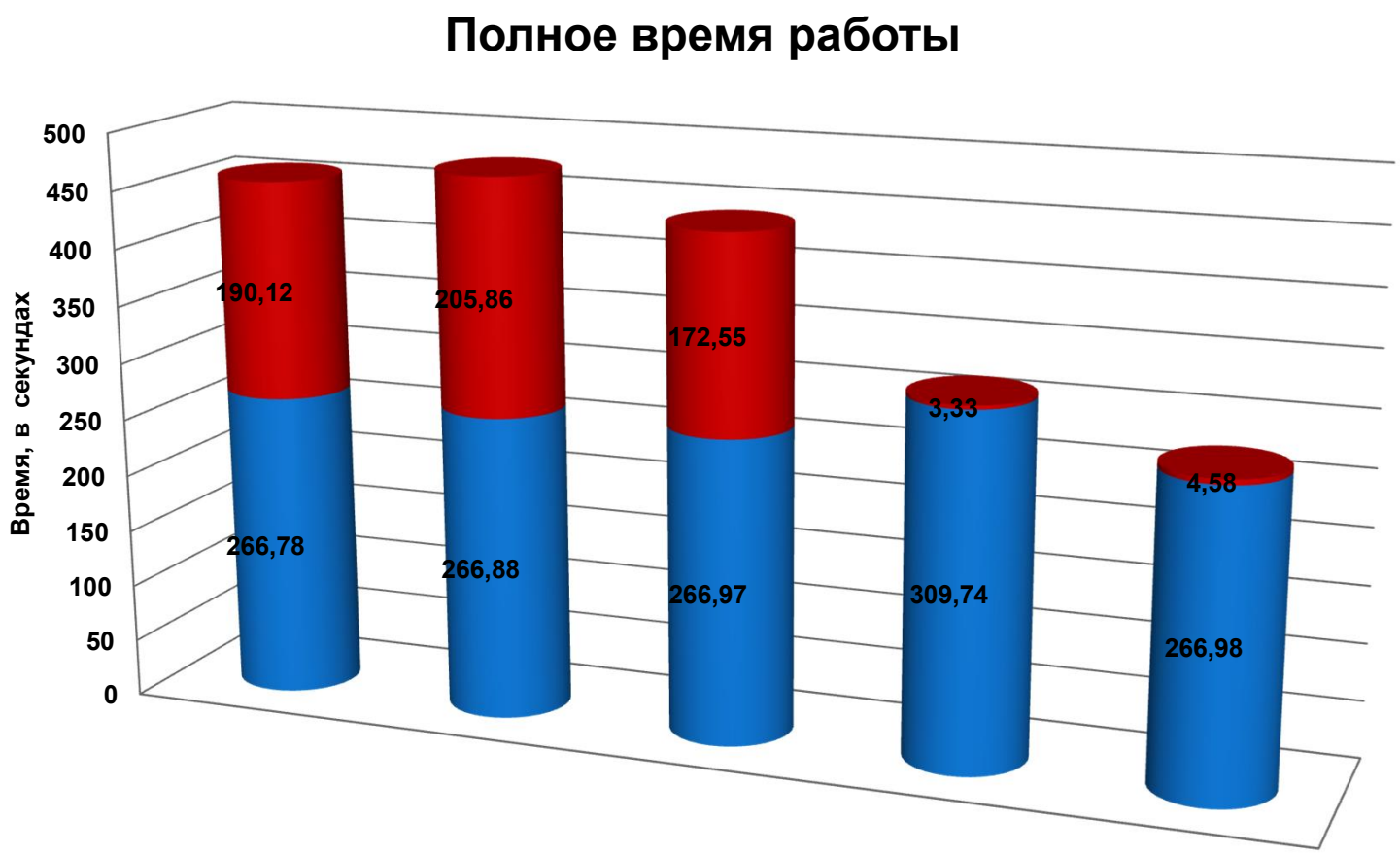
	WB	WBP	WBL	WBSP	WBSL
Ввод-вывод	195,05	201,85	169,12	1,14	1,4
Вычисления	631,19	629,17	627,73	638,72	628,25

МВС-10П. Ввод-вывод с Intel Xeon Phi 7110X. 20 MPI-процессов



	WB	WBP	WBL	WBSP	WBSL
Ввод-вывод	201,43	199,4	185,09	1,24	3,94
Вычисления	348,91	348,77	349,27	356,32	350,22

МВС-10П. Ввод-вывод с Intel Xeon Phi 7110X. 30 MPI-процессов



	WB	WBP	WBL	WBSP	WBSL
Ввод-вывод	190,12	205,86	172,55	3,33	4,58
Вычисления	266,78	266,88	266,97	309,74	266,98