Национальный Суперкомпьютерный Форум

Новые возможности системы автоматизации разработки параллельных программ (DVM-системы)

30 ноября 2016 г. | Переславль-Залесский

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

План доклада

- DVMH-модель параллельного программирования;
- Новые возможности отладки DVMH-программ;
- Использование средств DVMH в MPI-программах;
- Программы на регулярных и нерегулярных сетках;
- Новые возможности работы с нерегулярными сетками в компиляторе C-DVMH;
- Реализация параллельного ввода-вывода.

DVM-система

DVM-система предназначена для создания переносимых и эффективных вычислительных приложений на языках C-DVMH и Fortran-DVMH для многопроцессорных компьютеров с общей и распределенной памятью, включая и гибридные системы, в узлах которых вместе с универсальными многоядерными процессорами используются в качестве ускорителей и графические процессоры.

DVM-система состоит из:

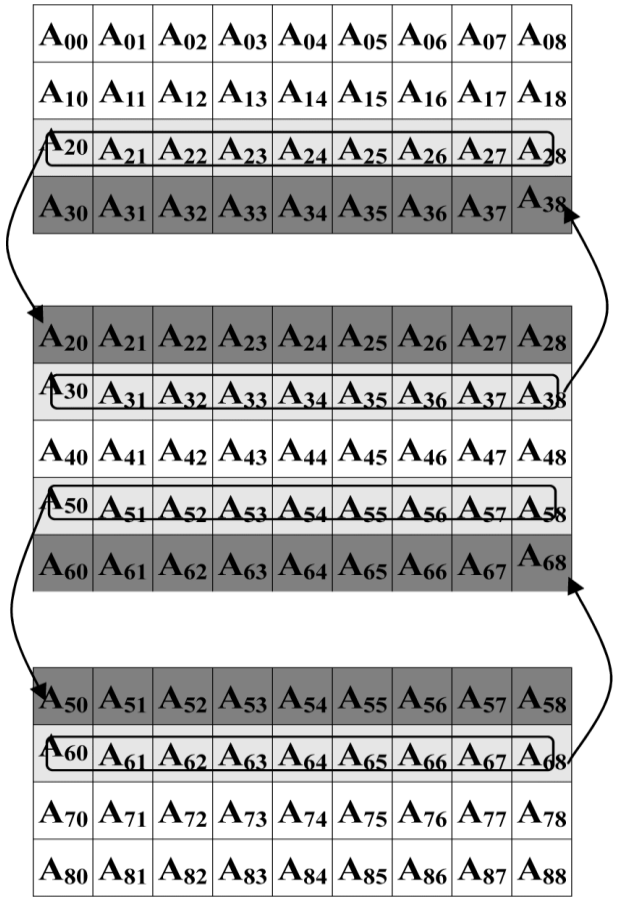
- Компилятор Fortran-DVMH
- Компилятор C-DVMH
- Библиотека поддержки Lib-DVMH
- Отладчик DVMH-программ
- Анализатор производительности DVMH-программ

```
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;
}
```

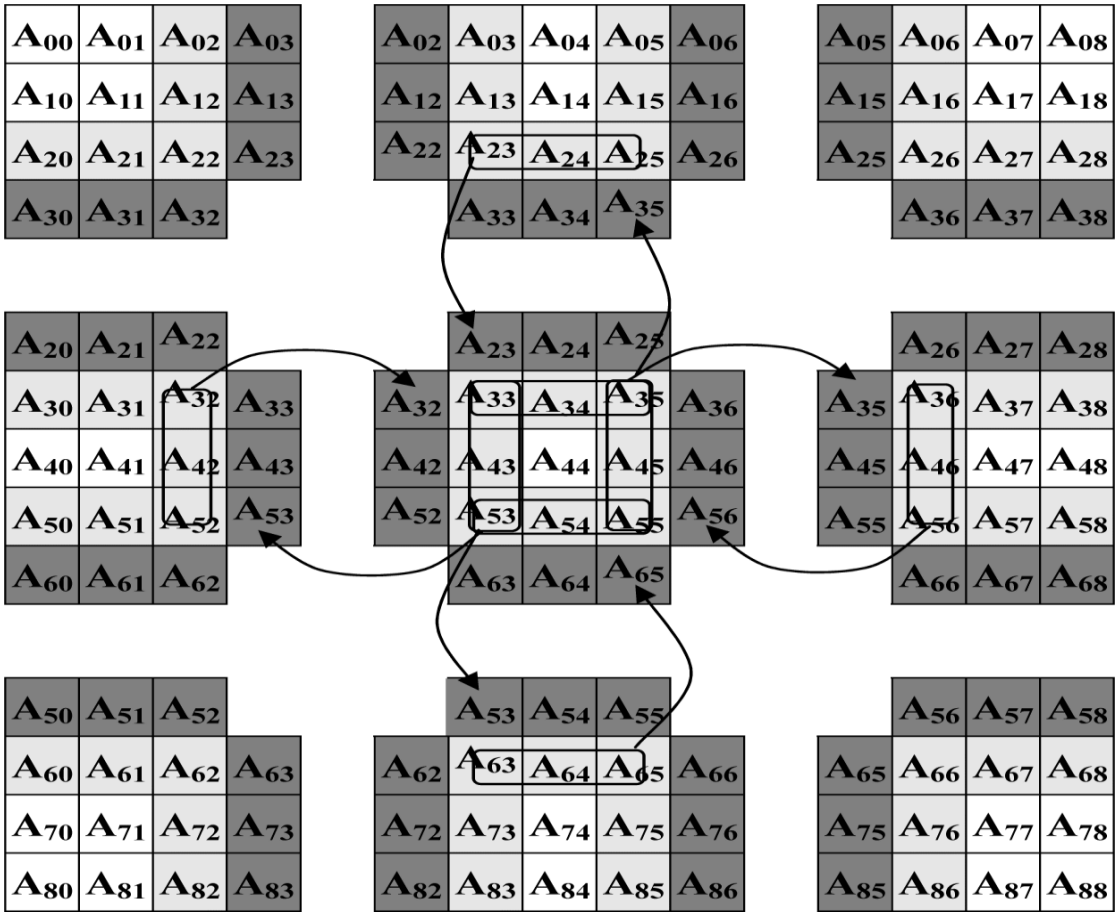
Алгоритм Якоби в модели DVMH

Динамическое распределение данных

./dvm run 3 1 jac

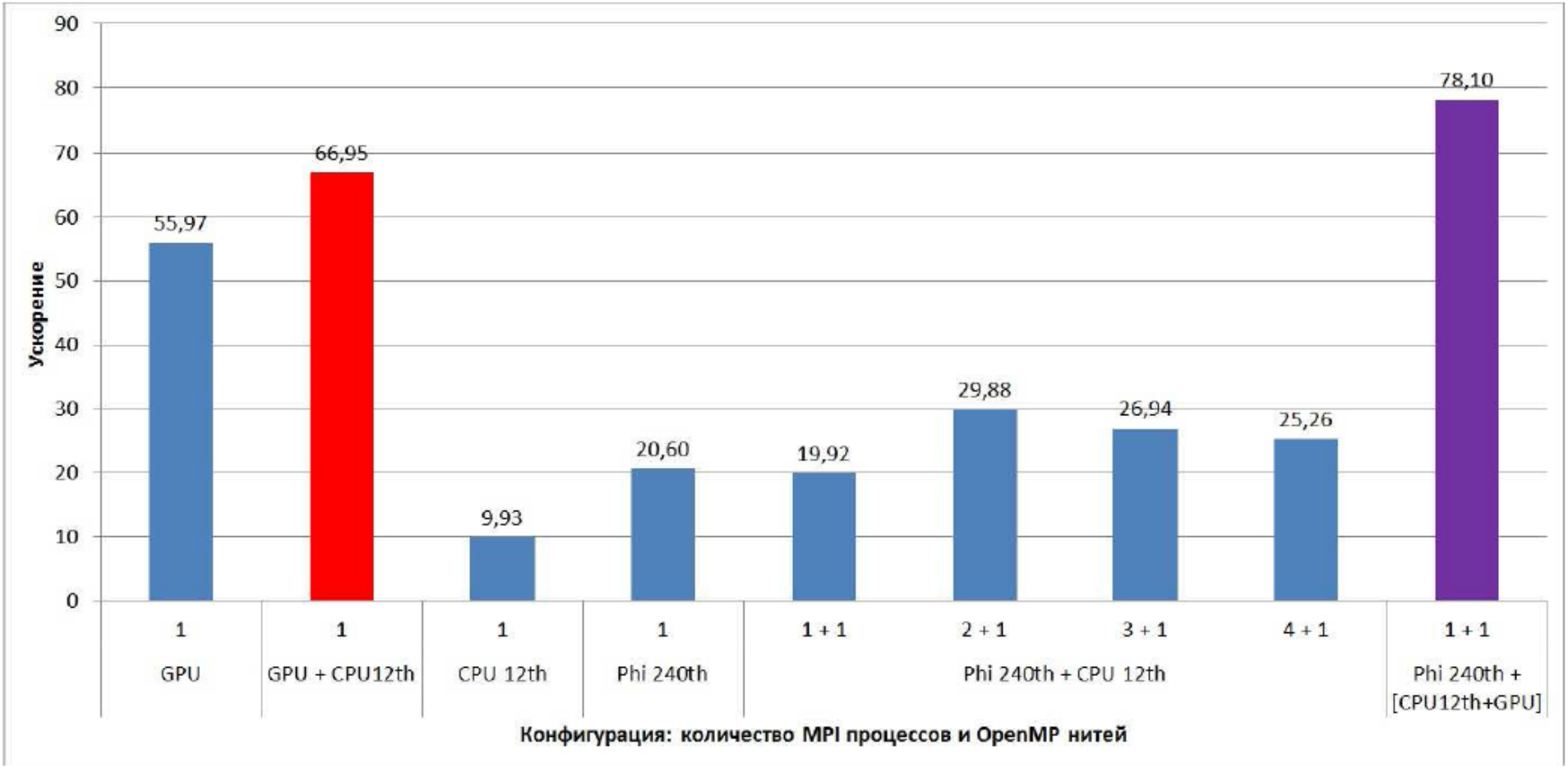


./dvm run 3 3 jac



Динамическое распределение вычислений

Выполнение теста EP из пакета NAS NPB при одновременном использовании ЦПУ, ГПУ и сопроцессора Xeon PHI



Новые возможности анализатора эффективности

```
--- The GPU characteristics ---

 ┌──────────────┐
 │ Proc: #1     │
 └──────────────┘
       │
   ┌──────────────────────┐
   │ GPU #1 (Tesla X2070) │
   ├──────────────────────┴──────────────────────────────────────────────────────────────────────────────────────┐
   │                                      #          Min          Max          Sum      Average    Productive         Lost │
   ├──────────────────────────────────────────────────────────────────────────────────────────────────────────────────┤
   │ [Region IN] Copy CPU to GPU          2           0B      976.624M       1.907G      976.624M       0.3252s            - │
   │ GET_ACTUAL                           1           0B      976.624M      976.624M      976.624M       0.1649s            - │
   │ Loop execution time                 41       0.0000        0.0907       2.5133        0.0613       2.5133s            - │
   │ Reduction                           20       0.0000        0.0084       0.0161        0.0008             -      0.0161s │
   ├──────────────────────────────────────────────────────────────────────────────────────────────────────────────────┤
   │ Productive time:     3.0033s                                                                                     │
   │ Lost time      :     0.0161s                                                                                     │
   └──────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

----------------------------------------------------------------------
Name (number) and performance time of processors
node6-155-07(1) 1.046391
min - node6-155-07(1) 1.046391; max - node6-155-07(1) 1.046391; mid - 1.046391
```

Новые возможности сравнительной отладки

- Специальный режим работы DVMH-программы, при котором все вычисления в регионах одновременно выполняются на ЦПУ и ГПУ
- Сравнение данных осуществляется при входе и выходе из региона
- Позволяет выявить и локализовать ошибки, проявляющиеся при работе на ускорителях
- Целочисленные данные сравниваются на совпадение, а вещественные числа сравниваются с заданной точностью по абсолютной и относительной погрешности
- В случае нахождения расхождений выдается информация о найденных расхождениях
- Далее в программе используется та версия данных, которая была получена при выполнении на ЦПУ

Использование средств DVMH в MPI-программах. Причины

- Уже накоплен фонд и опыт написания параллельных программ для кластера;
- Модель DVMH предполагает распараллеливание последовательных программ;
- Пользователь не желает отказываться от своей параллельной программы;
- Программа распараллелена вручную так, как не удастся (или потребует трюков) сделать средствами DVMH;
- Есть желание добавить параллелизм на нитях ЦПУ и/или на ГПУ, а также воспользоваться инструментарием DVMH (отладка и профилирование).

Использование средств DVMH в MPI-программах. Результаты

- Добавлен новый режим работы DVM-системы локально в каждом процессе;
- Добавлена конструкция нераспределенного параллельного цикла;
- Инкрементальное распараллеливание и быстрая оценка эффективности DVMH-модели на нитях ЦПУ и ГПУ перед проведением полного распараллеливания;
- Возможность использовать DVMH-распараллеливание внутри узла кластера в готовых MPI-программах.

Использование средств DVMH в MPI-программах

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

Алгоритм Якоби в модели DVMH

Использование средств DVMH в MPI-программах. Опыт

- Солвер явной схемы - часть большого развитого комплекса вычислительных программ:
  - C++, 39000 LOC, шаблоны, полиморфизм, и т.п.;
  - богатая платформа базовых понятий и структур данных;

- Модификации только локальные, в одном модуле в ~3000 строк, сводятся к добавлению нескольких директив (около 10);

- Получено ускорение
  - 2 CPU Intel Xeon X5670 12 ядер - 9.8 раз;
  - GPU NVidia GTX Titan (Kepler) - 18 раз.

Программы на регулярных сетках

Проблемы

- Единый шаг сетки по расчетной области - нет гибкости, неподъемные требования по памяти и вычислительной мощности при измельчении;
- Реализация численных методов зачастую привязана к форме сетки - двумерные, трехмерные, декартовы, цилиндрические, и т.д. Тем самым сложно заменять геометрию.

Положительные стороны

- Отношение соседства и пространственные координаты не хранятся явно - экономия памяти;
- Обращения к массивам с константными смещениями - свобода для компиляторных оптимизаций, ясность для распараллеливаниия (в том числе автоматического).

Программы на нерегулярных сетках

Положительные стороны

- Свобода вариации подробности сетки - поддержание нужной степени измельчения в отдельных частях области;
- Хорошие возможности по переиспользованию вычислительного кода, свобода в выборе формы расчетных областей.

Проблемы

- Отношение соседства и пространственные координаты приходится хранить явно;
- Косвенная индексация при обращениях к массивам - препятствие для компиляторных оптимизаций, сложность для распараллеливаниия (в особенности автоматического).

Обзор новых возможностей для работы с нерегулярными сетками

- Задание произвольных поэлементных распределений, в том числе получаемые пакетами Metis, Chaco, ...;
- Построение согласованных распределений на основе имеющихся (блочных или поэлементных);
- Задание произвольных по содержанию буферов удаленных элементов с эффективным однородным доступом к ним и обновлением;
- Сохранение быстрого доступа к распределенным массивам с помощью механизмов перехода на локальную индексацию;
- Возможность реорганизации данных - оптимизации шаблона доступа к памяти путем изменения порядка хранения локальных элементов.

Новые правила распределения

- **Косвенное** - задается массивом целых чисел, размер которого равен размеру косвенно распределяемого измерения, а значения задают номер домена.

  #pragma dvm array distribute A[indirect(B)]

- **Производное** - задается правилом, по форме похожим на правило выравнивания (ALIGN) модели DVMH: возможность согласованно распределять сеточные элементы.

  Например, ячейки, ребра, вершины.
  #pragma dvm array distribute A ([cells[i][0]:cells[i][2]] with cells[@i])]

Новые теневые грани

Теневые грани - это набор элементов, не принадлежащих текущему процессу, для которых:

- Возможен доступ без специальных указаний из любой точки программы;
- Имеются специальные средства работы с ними: shadow_renew, shadow_compute, across

Любой набор удаленных элементов задается аналогично производному распределению:
shadow_add(nodes[neigh[i][0]:neigh[i][numneigh[i]-1] with nodes[@i]] = neighbours)

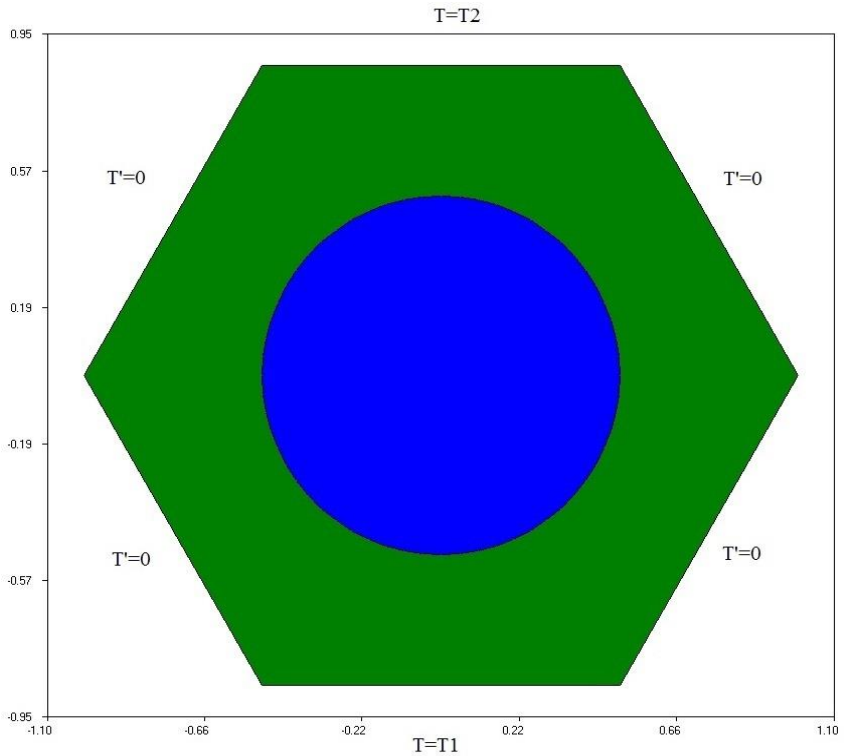
Переход к локальной индексации

- Процедура для перевода глобального (исходного) индекса в локальный (непосредственно для доступа к памяти) **слишком долгая**;
- Для блочных распределений они совпадают;
- Для поэлементных временно введена исполняемая директива локализации значений индексных массивов
  localize(neigh => nodes[:])
- В результате дальнейшие действия производятся над локальными индексами и нет необходимости менять компиляцию исполняемых конструкций.

Модельная задача

- Двумерная задача теплопроводности с постоянным, но разрывным коэффициентом в шестиграннике.

- Область состоит из двух материалов с различными коэффициентами температуропроводности.

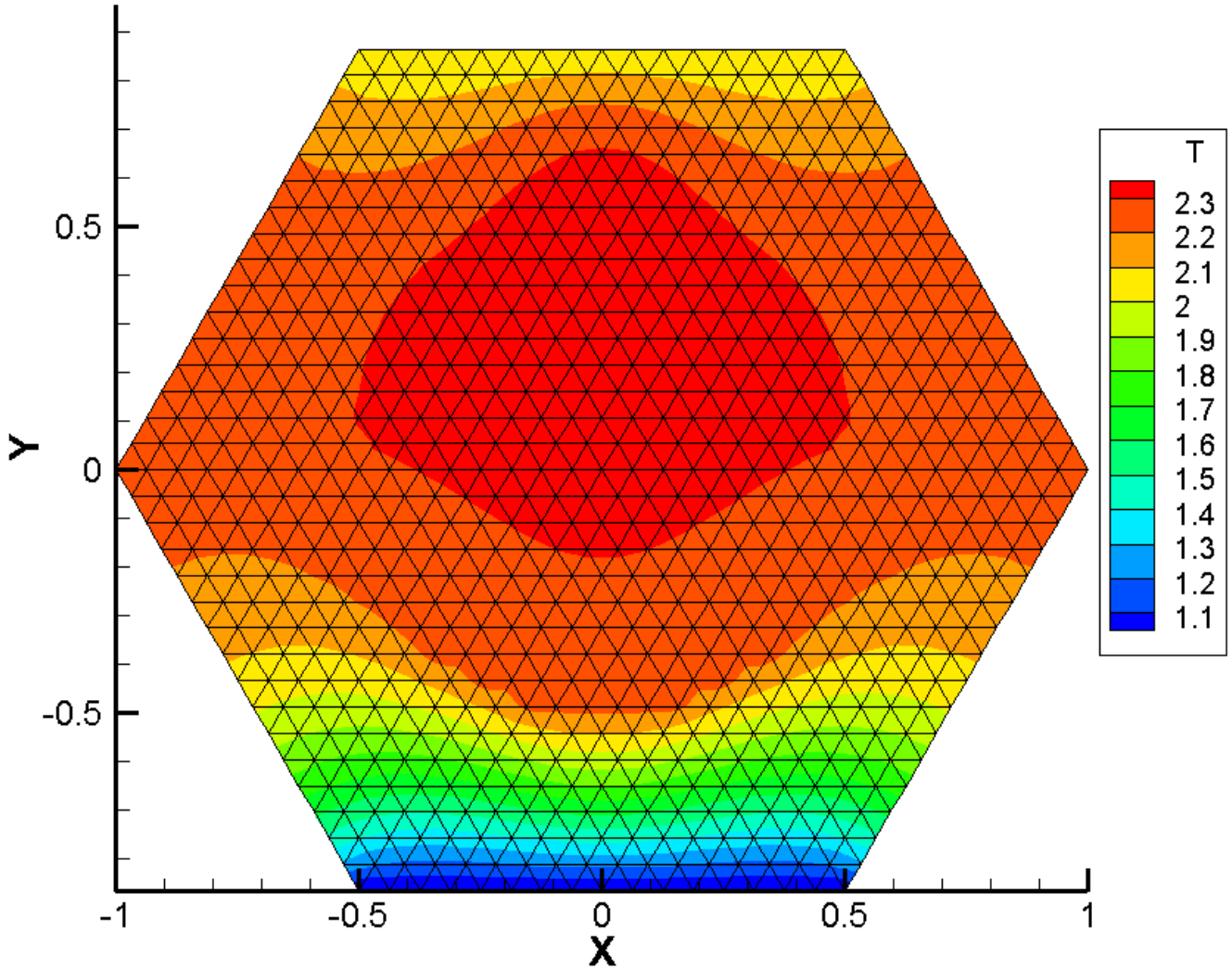


Модельная задача

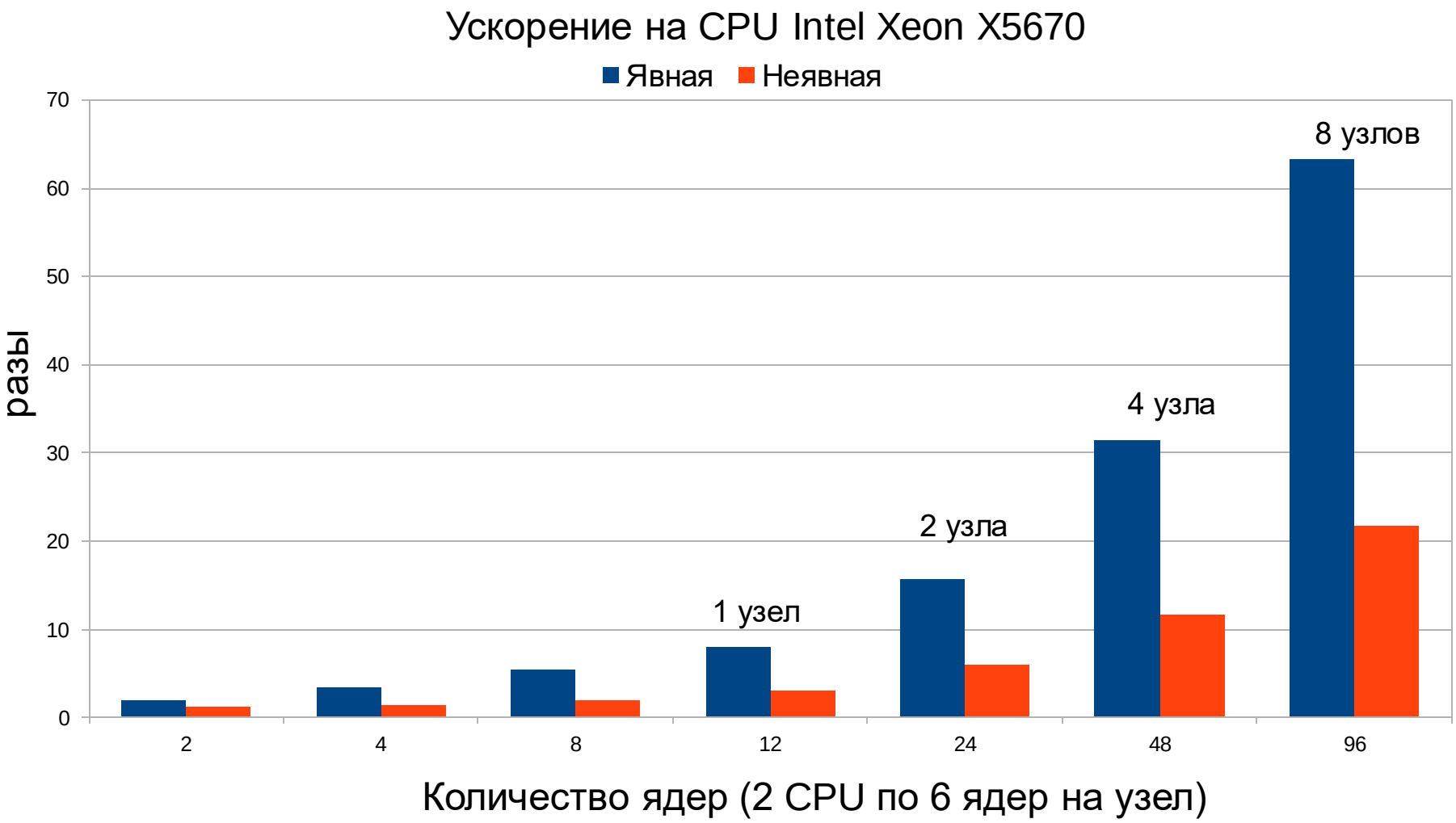
- Массивы величин одномерные - tt1,tt2
- Переменное число "соседей" - ii
- Связи задаются массивом - jj

```
do i = 1, np2
    nn = ii(i)
    nb = npa(i)
    if (nb.ge.0) then
      s1 = FS(xp2(i),yp2(i),tv)
      s2 = 0d0
      do j = 1, nn
        j1 = jj(j,i)
        s2 = s2 + aa(j,i) * tt1(j1)
      enddo
      s0 = s1 + s2
      tt2(i) = tt1(i) + tau * s0
    else if (nb.eq.-1) then
      tt2(i) = vtemp1
    else if (nb.eq.-2) then
      tt2(i) = vtemp2
    endif
    s0 = (tt2(i) - tt1(i)) / tau
    gt = DMAX1(gt,DABS(s0))
  enddo
  do i = 1, np2
    tt1(i) = tt2(i)
  enddo
```

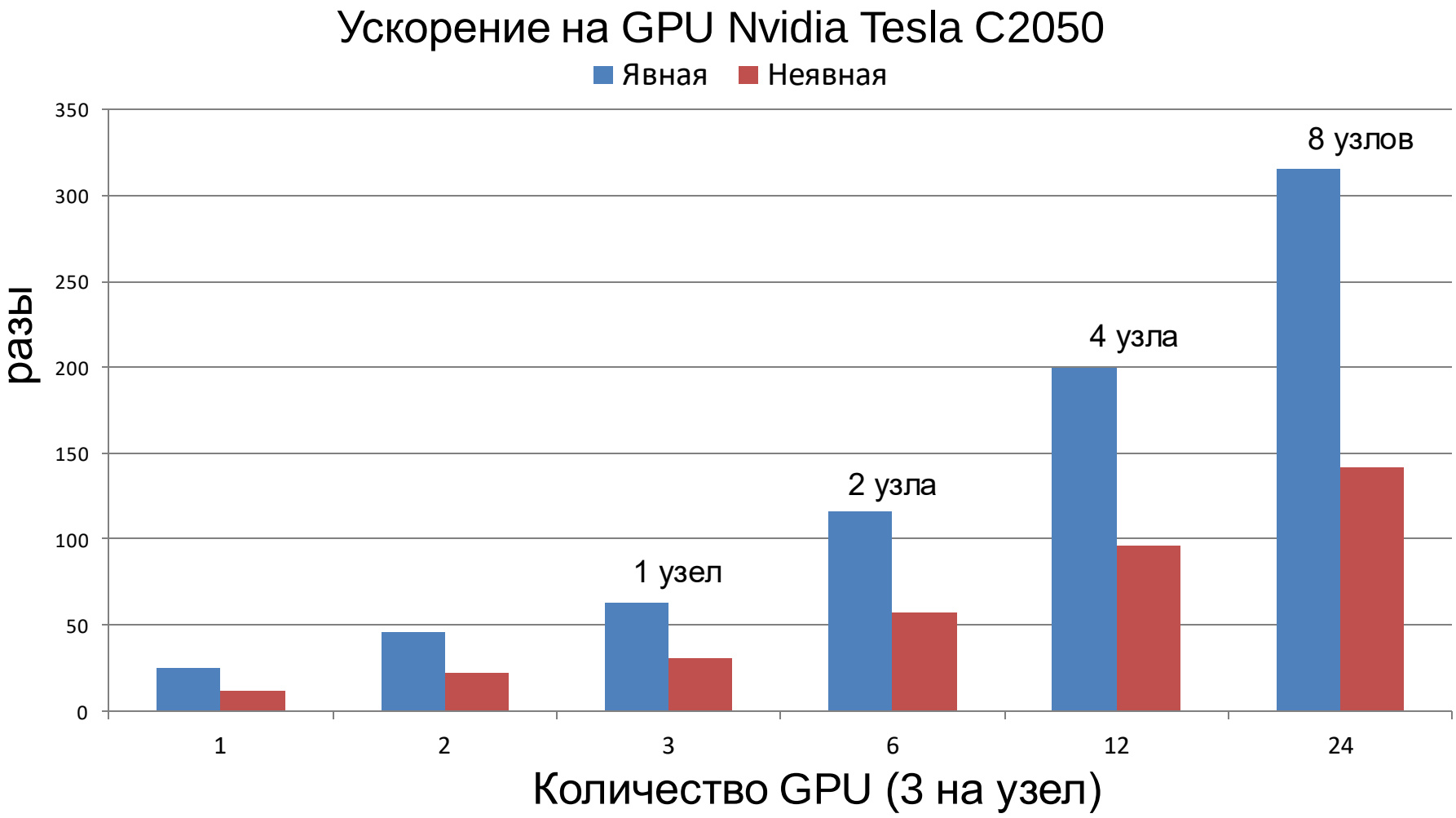
Расчетная сетка



Полученные ускорения, 8 млн узлов



Полученные ускорения, 8 млн узлов



Параллельный ввод-вывод в DVM-системе

- Последовательный синхронный ввод-вывод в компиляторе C-DVMH
- Параллельный синхронный ввод-вывод:
  - В локальный файл
  - В параллельный файл
- Асинхронный параллельный ввод-вывод

Последовательный синхронный ввод-вывод в компиляторе C-DVMH

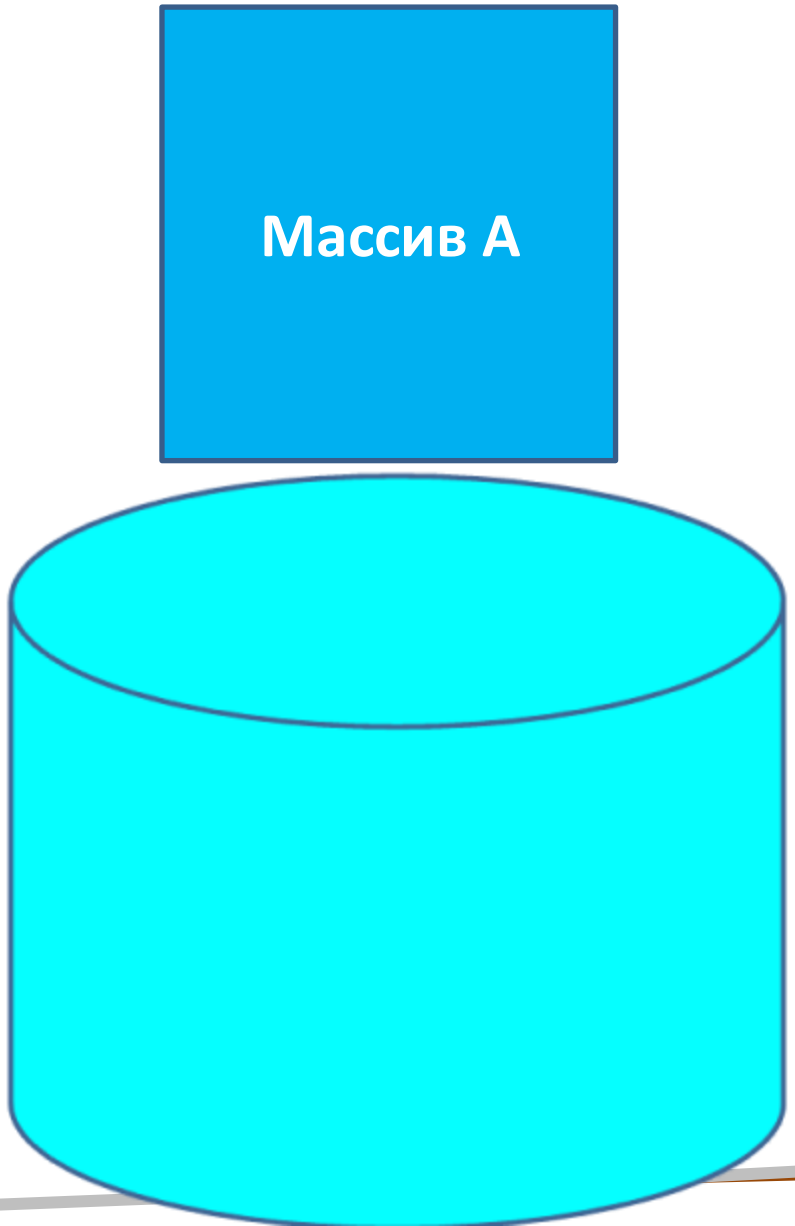
- Новая версия языка C-DVMH => новая версия компилятора на базе CLANG+LLVM

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

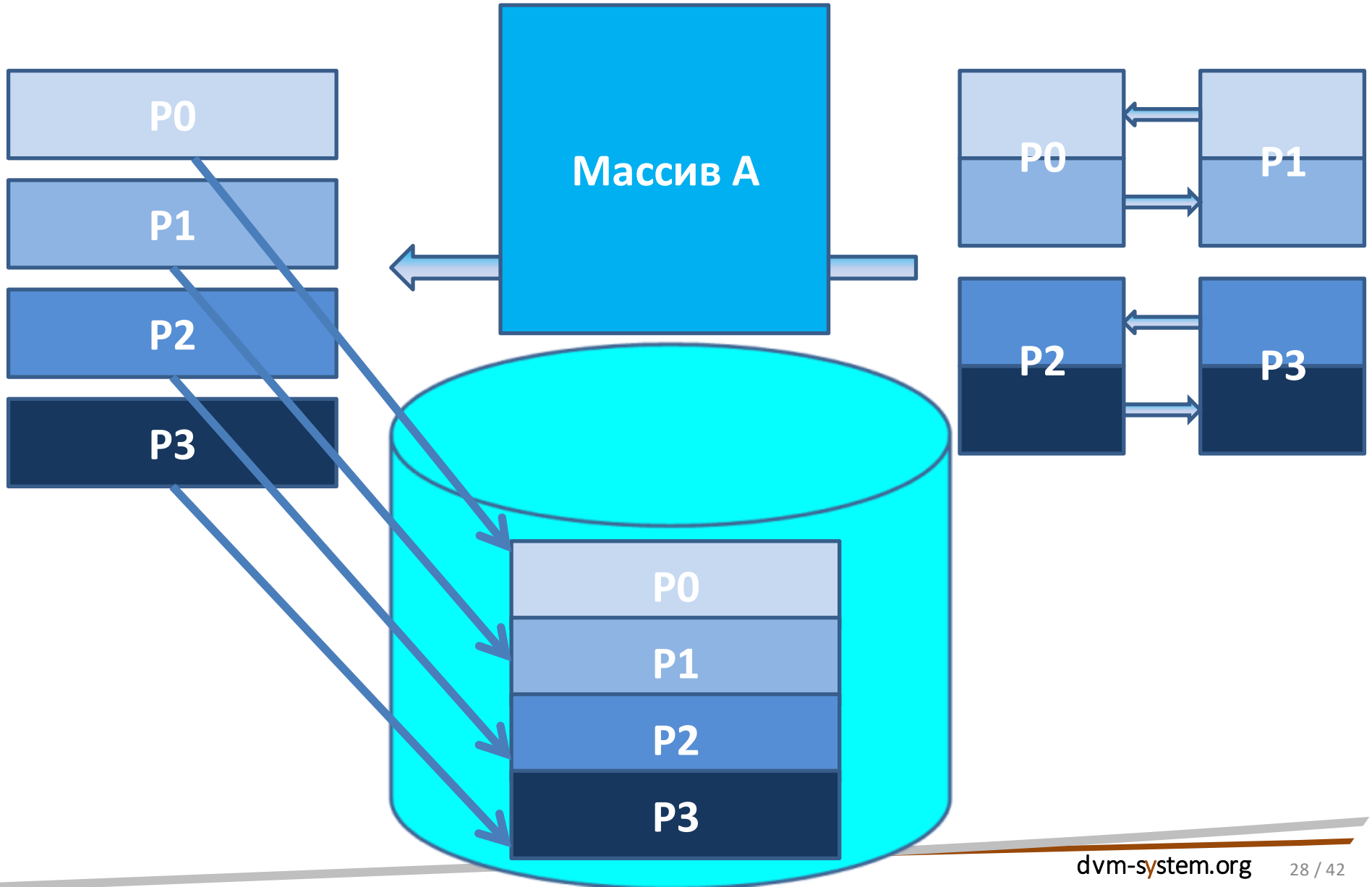
- **Локальный файл**
  Открывается каждым процессом независимо. Все операции над локальными файлами обрабатываются каждым процессом независимо и никаких коммуникаций не вызывают. При чтении (или записи) распределенного массива из локального файла, читается (или записывается) только локальная часть распределенного 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

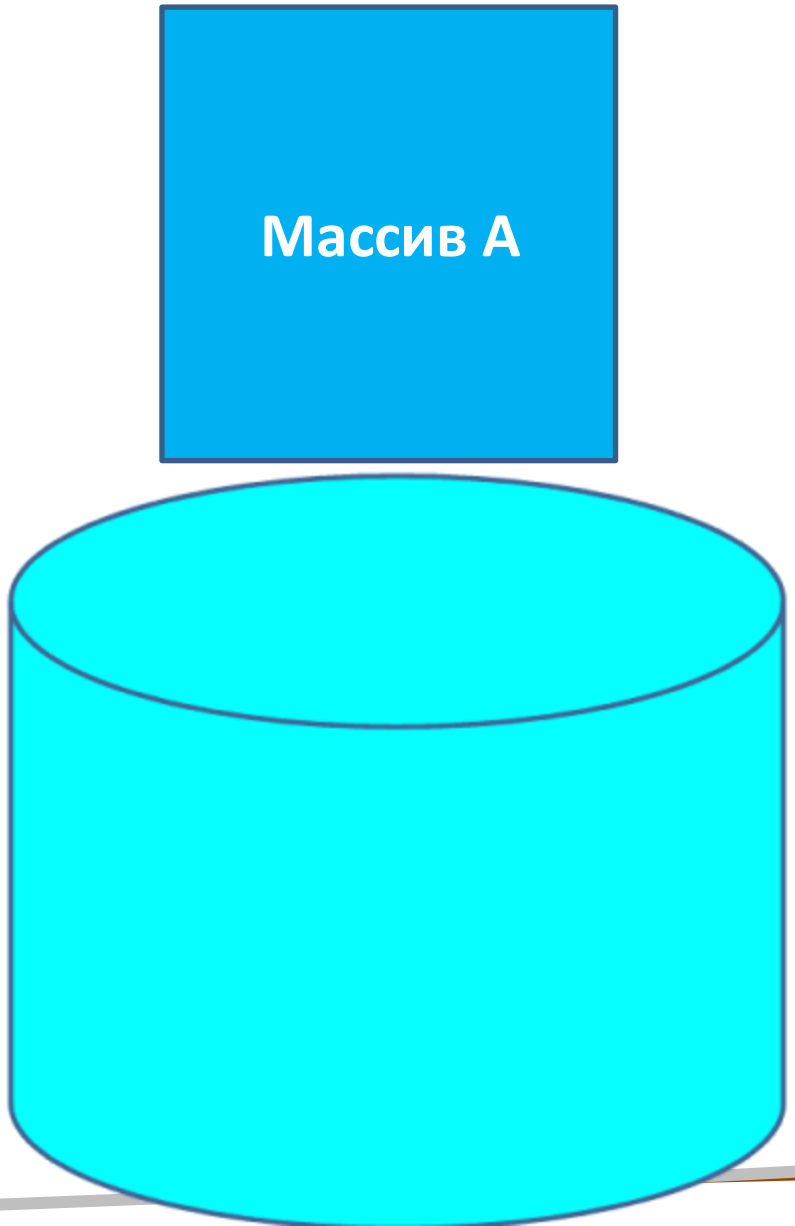


Реализация операций fread, fwrite, fputs

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

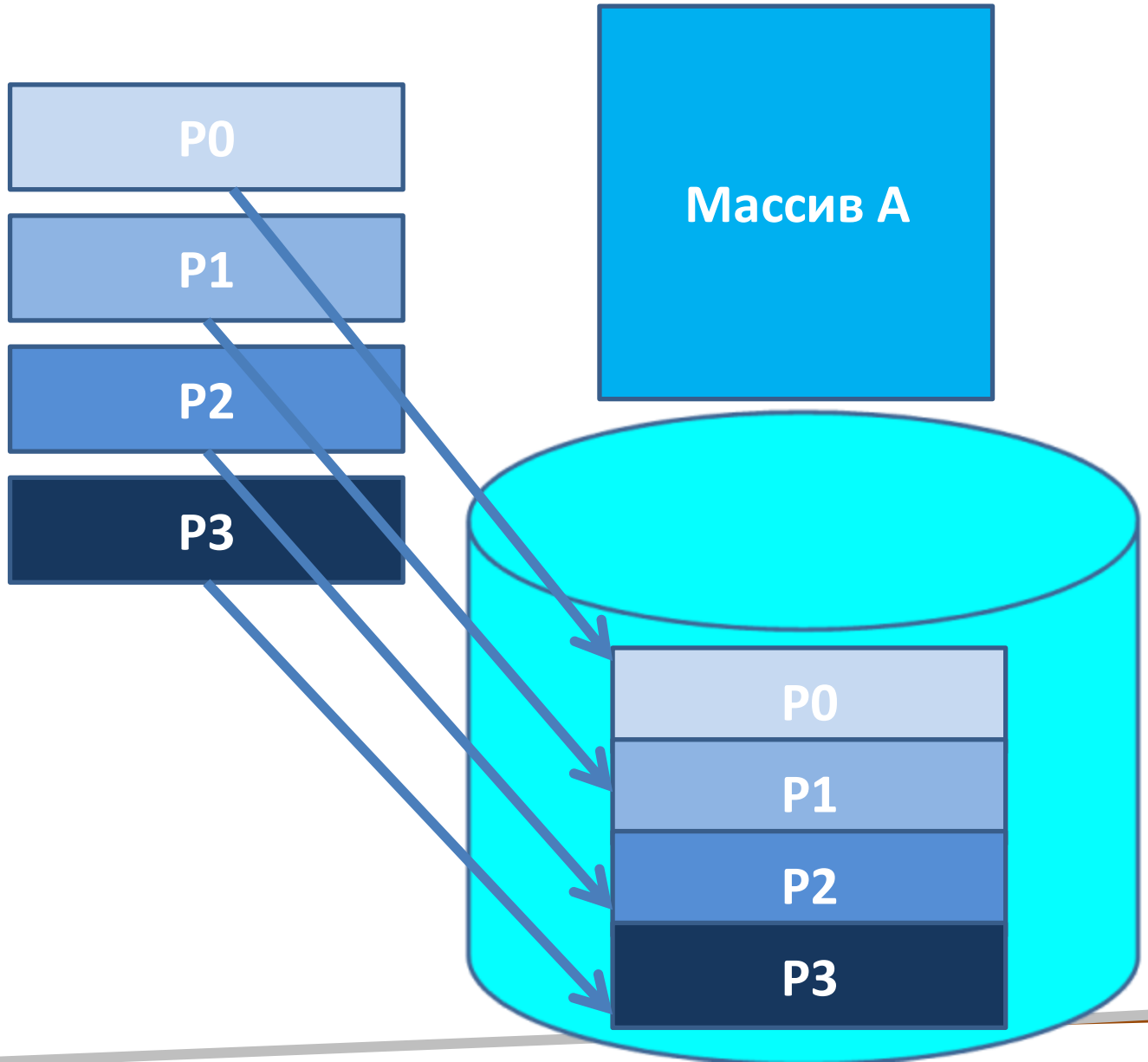


Реализация операций fread, fwrite, fputs



Реализация операций fread, fwrite, fputs

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



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).

Асинхронный параллельный ввод-вывод

- **Асинхронный файл**

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 (требуется режим MPI_THREAD_MULTIPLE).
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

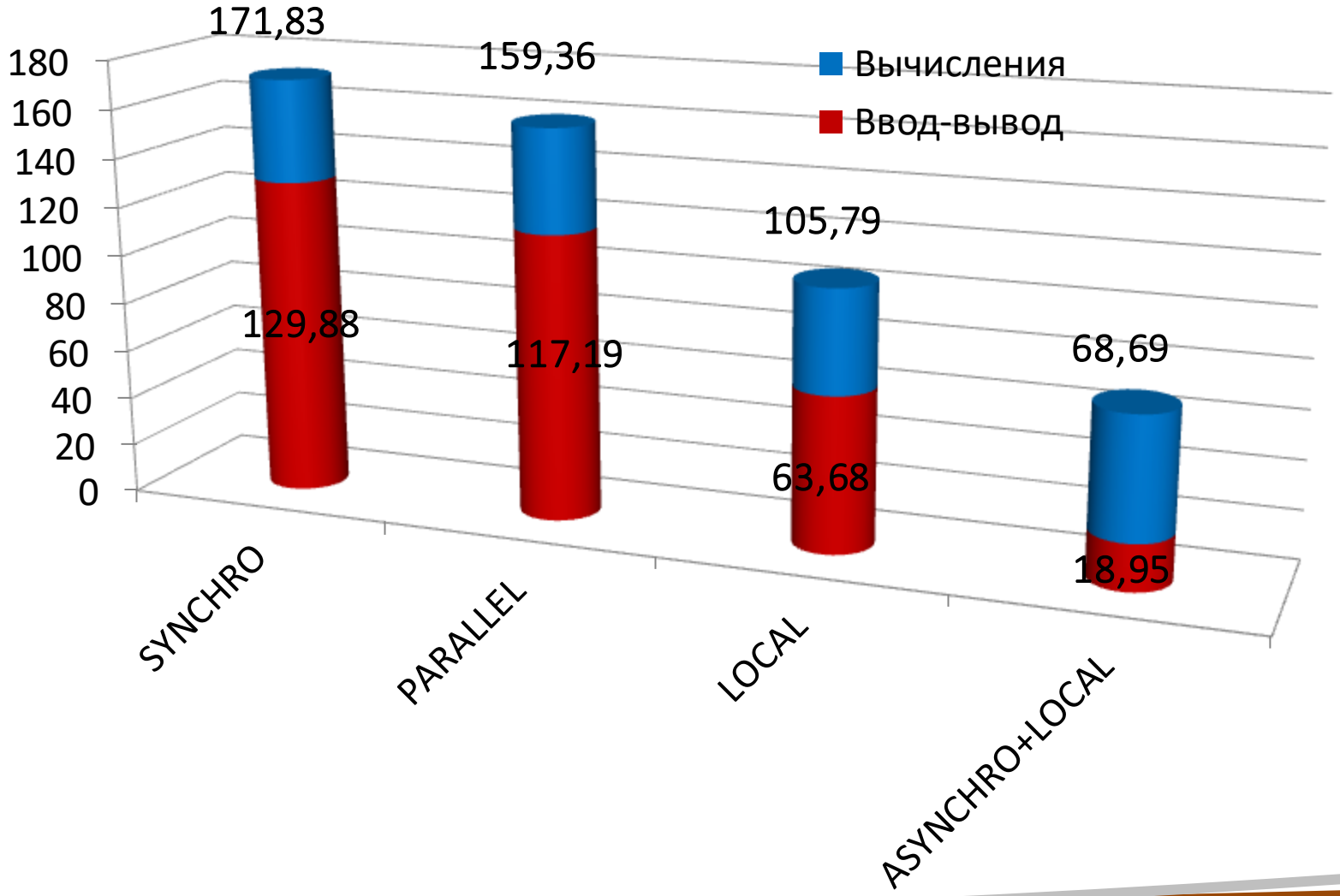
```
!DVM$ IO_MODE ([PARALLEL]
               [[,]LOCAL]
               [[,]ASYNC])

      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
```

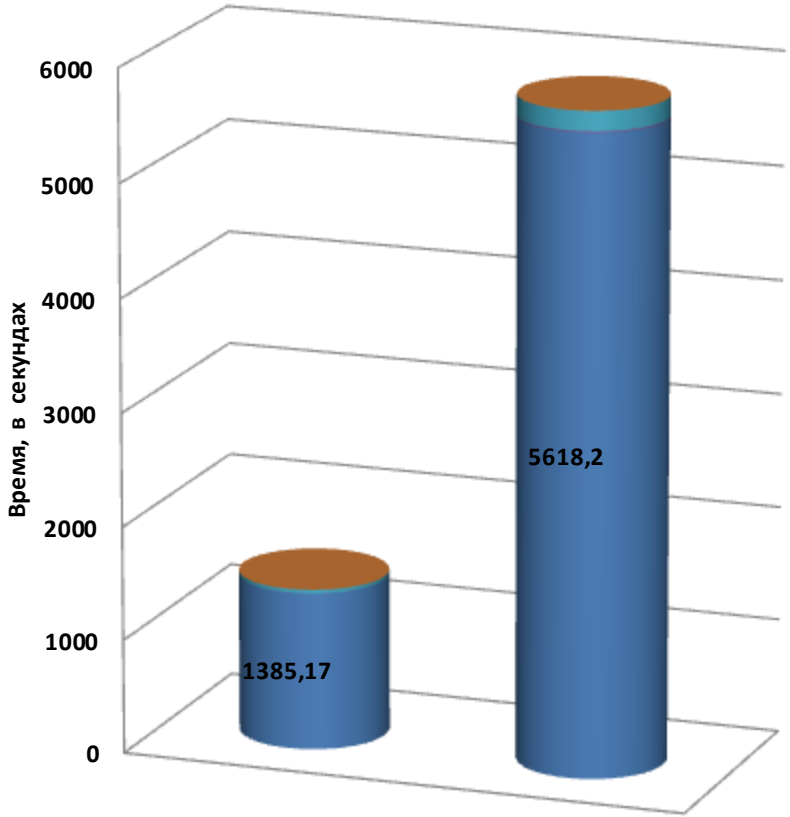
```
FILE *f;
#pragma dvm array distribute[block][block] shadow[1:1][1:1]
double A[L][L];
#pragma dvm array align ([i][j] with A[i][j])
double B[L][L];
int main(int argc, char *argv[]) {
   for(int it = 0; it < ITMAX; it++) {
      #pragma dvm region inout (A,B)
      {
         #pragma dvm parallel([i][j] 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

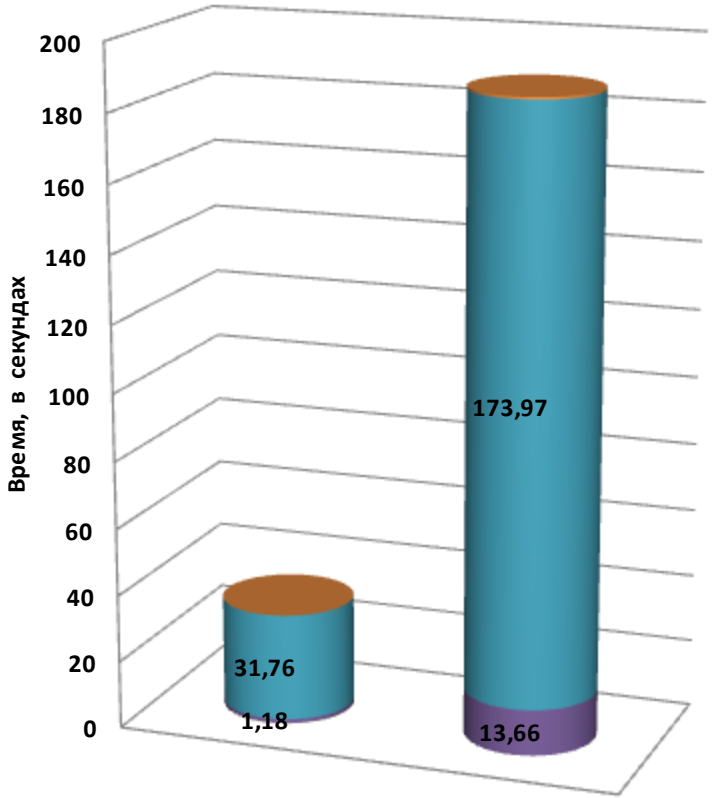
Времена выполнения 100 итераций программы Якоби 32000x32000 при сохранении 10 контрольных точек на «Ломоносове»



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

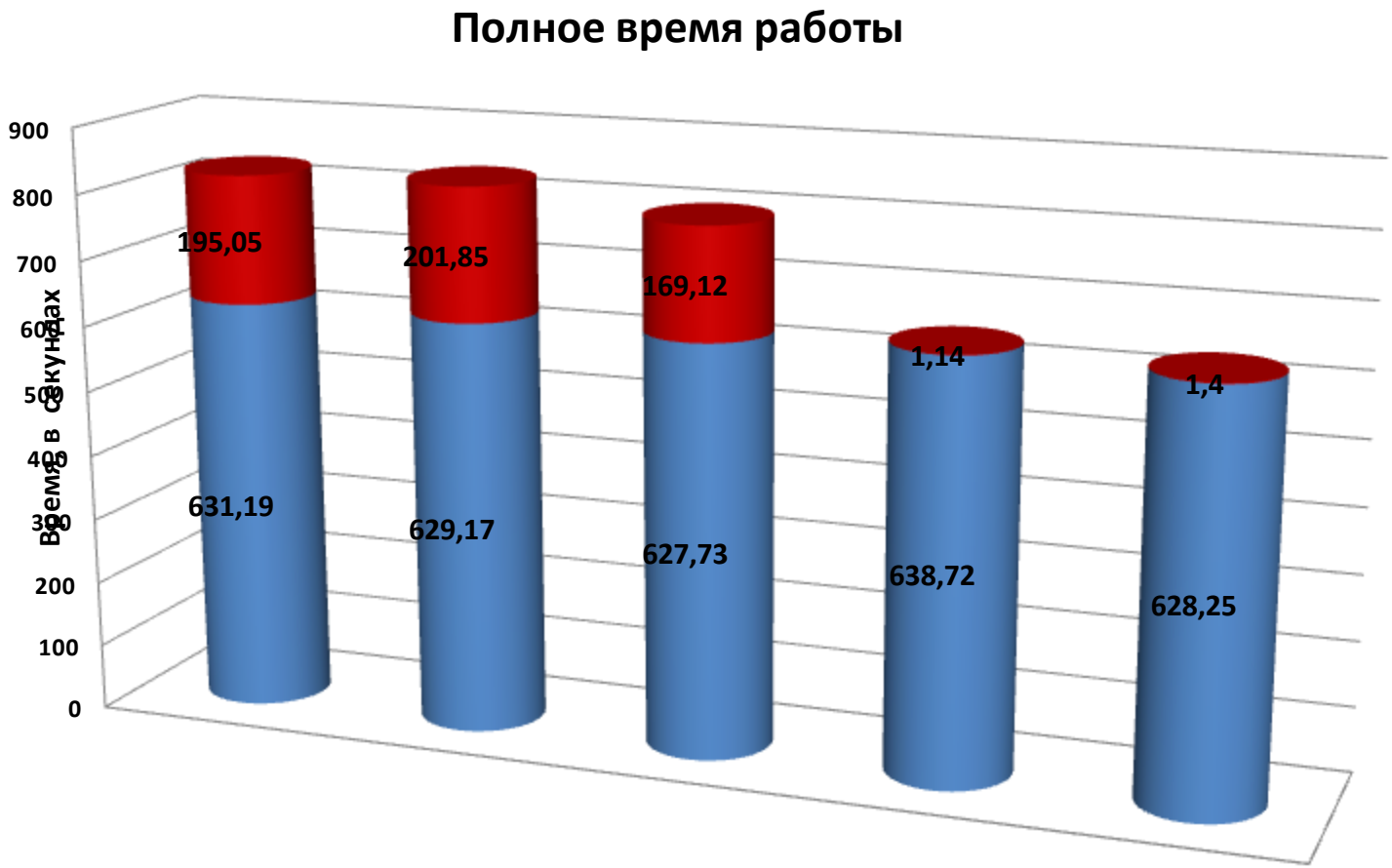


	XEON	XEON PHI
REMOVE	0	0,69
FCLOSE	31,76	173,97
FWRITE	1,18	13,66
FOPEN	0,09	0,01
EXE	1385,17	5618,2



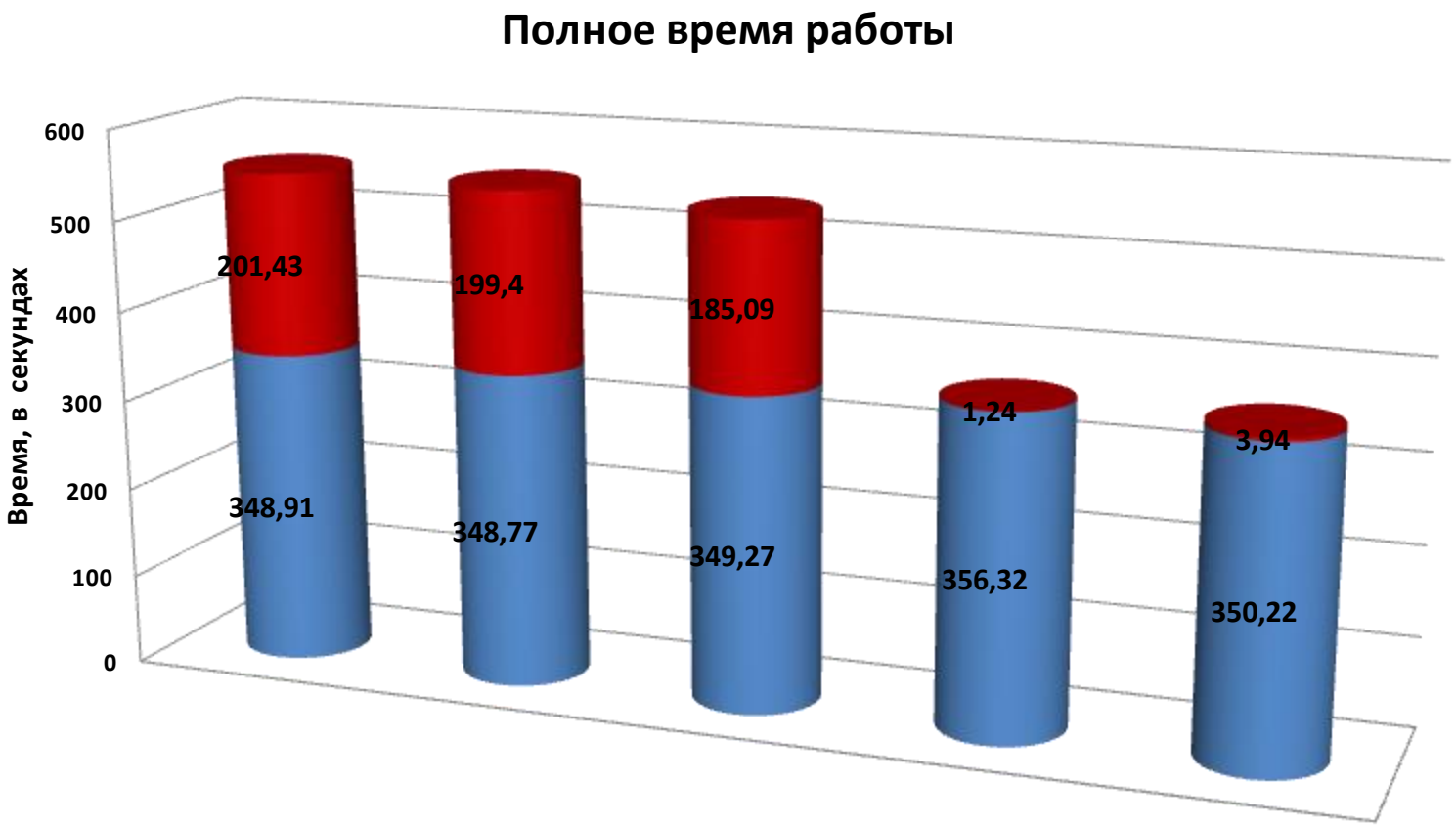
	XEON	XEON PHI
REMOVE	0	0,69
FCLOSE	31,76	173,97
FWRITE	1,18	13,66
FOPEN	0,09	0,01

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



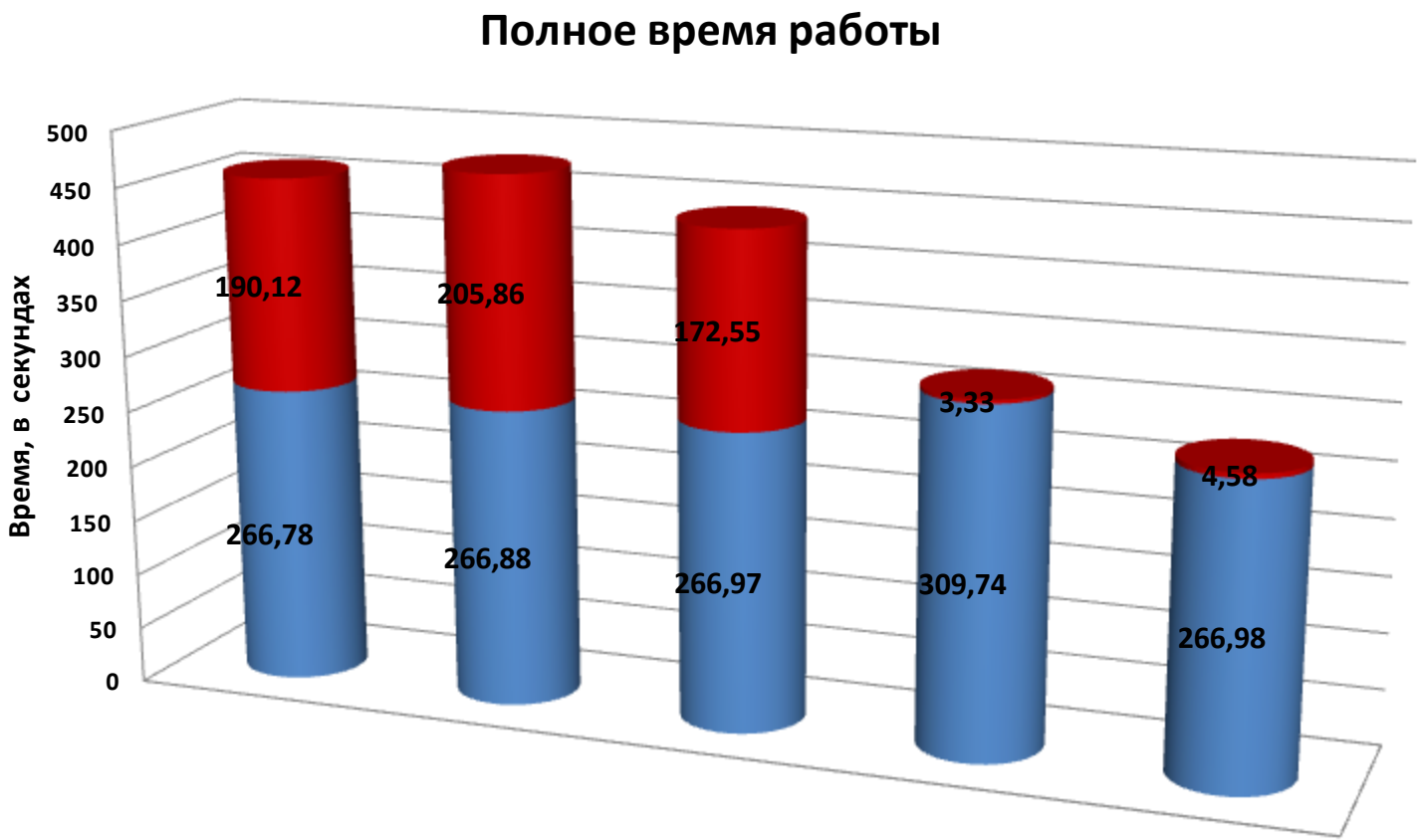
	WB	WBP	WBL	WBSP	WBSL
Ввод-вывод	195,05	201,85	169,12	1,14	1,4
Вычисления	631,19	629,17	627,73	638,72	628,25

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

Выводы

- DVM-система позволяет разрабатывать эффективные параллельные программы для гибридных вычислительных кластеров, использующих многоядерные универсальные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi.
- Наличие развитых средств для отладки и анализа производительности DVMH-программ существенно упрощает процесс разработки параллельных программ.
- Сделаны шаги по расширению возможностей DVM-системы для решения задач, использующих нерегулярные сетки и позволяющие распараллелить их на кластер с многоядерными и графическими процессорами:
  - возможность перехода на ручное распределение данных (например, средствами MPI);
  - новые экспериментальные конструкции языка C-DVMH и их реализация в компиляторе и системе поддержки.
- Разработаны средства параллельного ввода-вывода, обеспечивающие эффективное использование возможностей файловых систем современных кластеров.

Спасибо за внимание

http://dvm-system.org