МЕЖДУНАРОДНАЯ НАУЧНАЯ КОНФЕРЕНЦИЯ
Суперкомпьютерные дни в России

# Реализация параллельного ввода-вывода в DVM-системе

27 сентября 2016 г. | Москва

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

# 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.

```
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

# Реализация параллельного ввода-вывода

- Каждый процесс читает/записывает в файл свою часть распределенного массива.
- Использование MPI-IO:
  - Все процессы работают с одним файлом, используя MPI_set_view или MPI_File_seek для позиционирования внутри файла.
  - Ввод-вывод на основе коллективных операций MPI.
- Выделение процессов ввода-вывода.

# Запись каждым процессом в отдельный файл

При таком подходе каждый процесс записывает в отдельный файл только свою часть распределенного массива.

Преимущества:

- Малый объем записываемых каждым процессом данных.
- Эффективное использование возможностей параллельных файловых систем.

Недостатки:

- С большим числом файлов работать сложнее.
- Получая вывод программы в виде множества отдельных файлов, пользователю требуется приложить усилия для объединения данных перед началом их анализа.

# MPI-IO . Индивидульные операции с общим файловым дескриптором

Все процессы пишут одновременно в один файл, используя MPI_set_view или MPI_File_seek для позиционирования внутри файла.

Недостатки: хуже чем первый метод поддерживается файловыми системами.

Преимущества: все данные находятся в одном файле.

# Коллективные операции MPI-IO

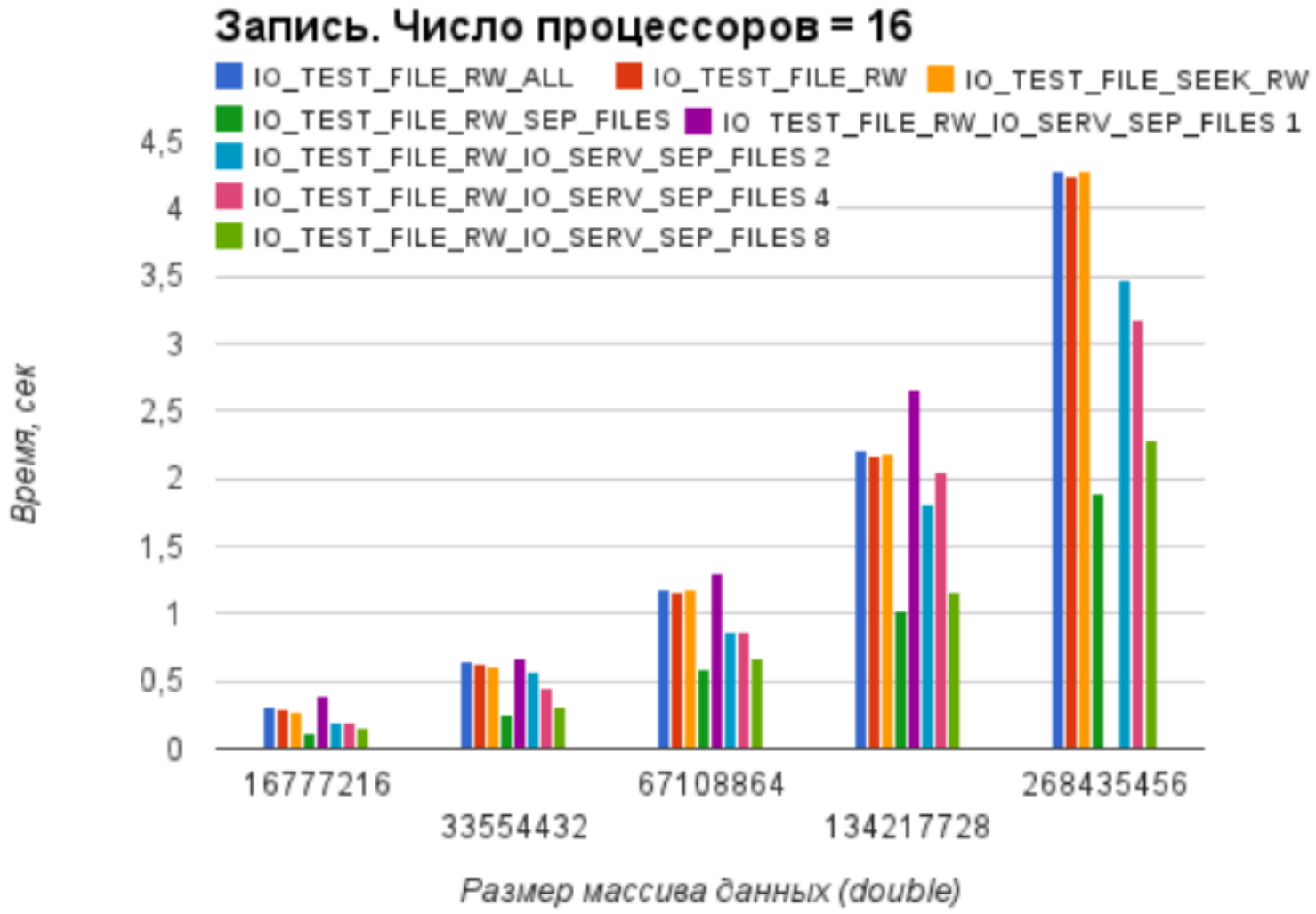
Все процессы вызывают фунцию для коллективной записи в один файл. Например, MPI_File_write_all.
Полностью полагаемся на реализацию MPI, которая может лучше учитывать архитектуру конкретной системы.

# Выделение процессов ввода-вывода

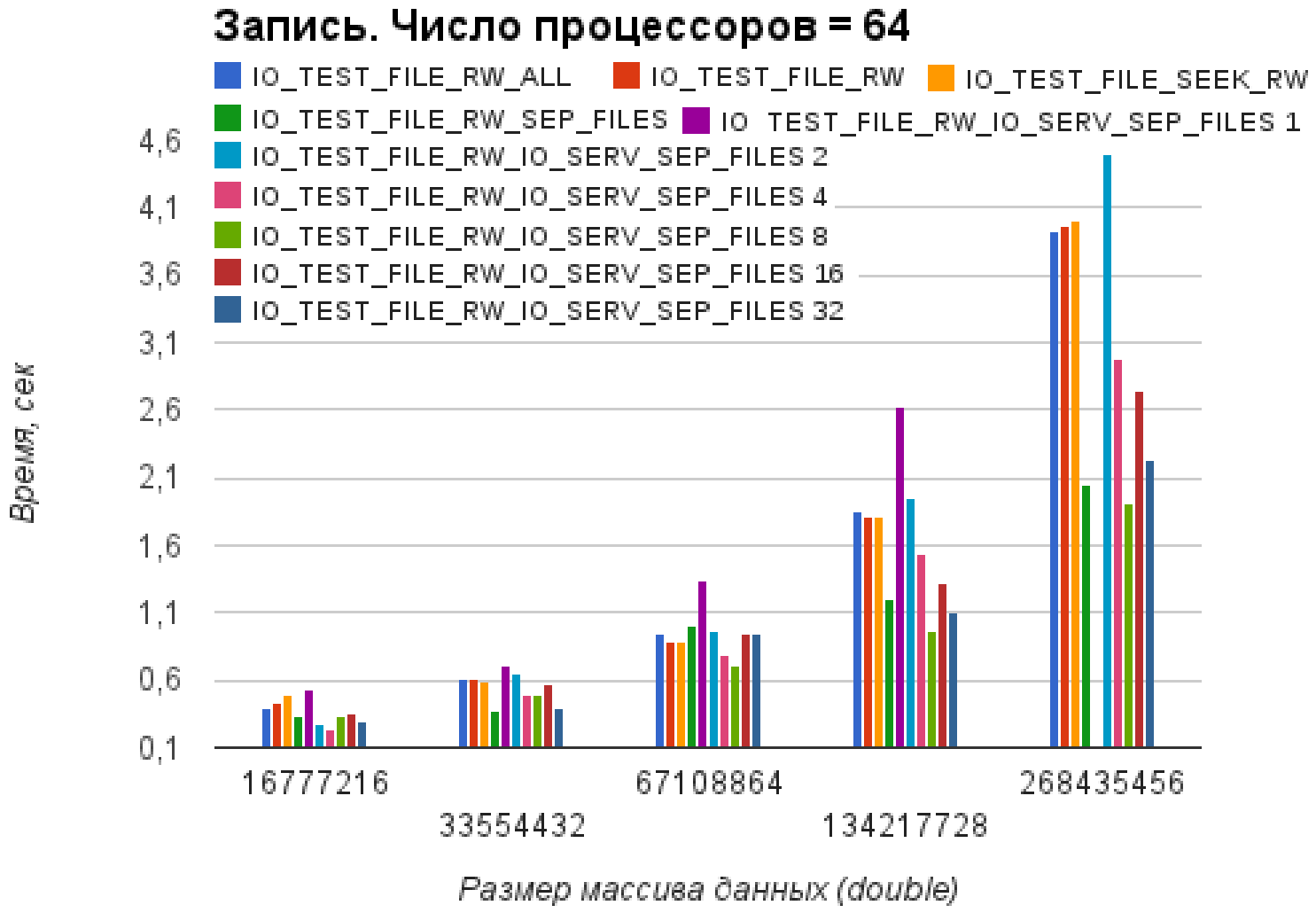
Все процессы делятся на группы.
В каждой группе выделяется один процесс ввода-вывода.
Каждый процесс, используя индивидуальные операции ввода-вывода, записывает в отдельный файл данные от всех процессов своей группы.

Преимущества: разделение на меньшее число файлов, больше возможностей для совмещения операций ввода-вывода с вычислениями на не участвующих в записи узлах.

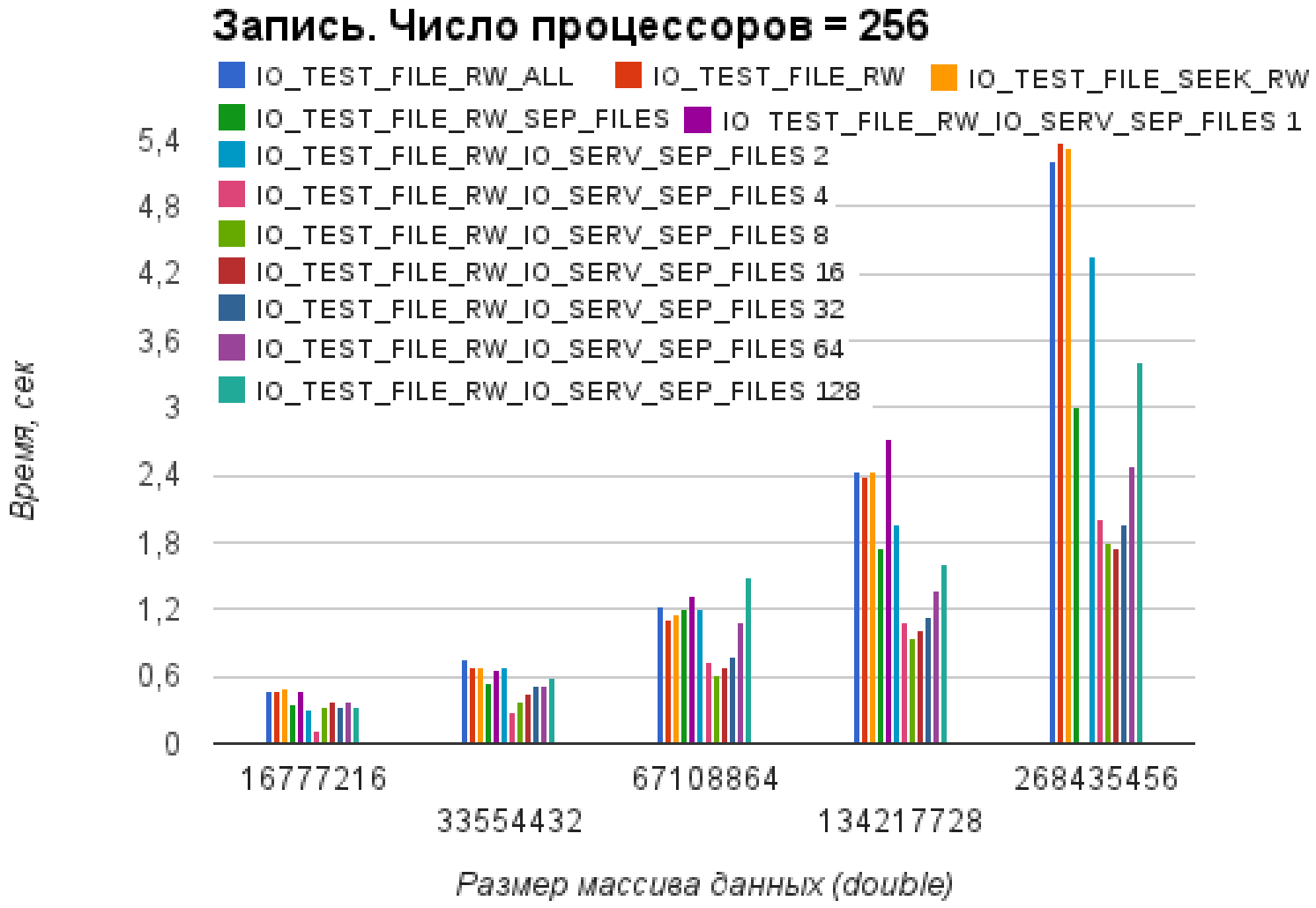
# Времена записи массива на МВС-100К при использовании различных режимов ввода-вывода



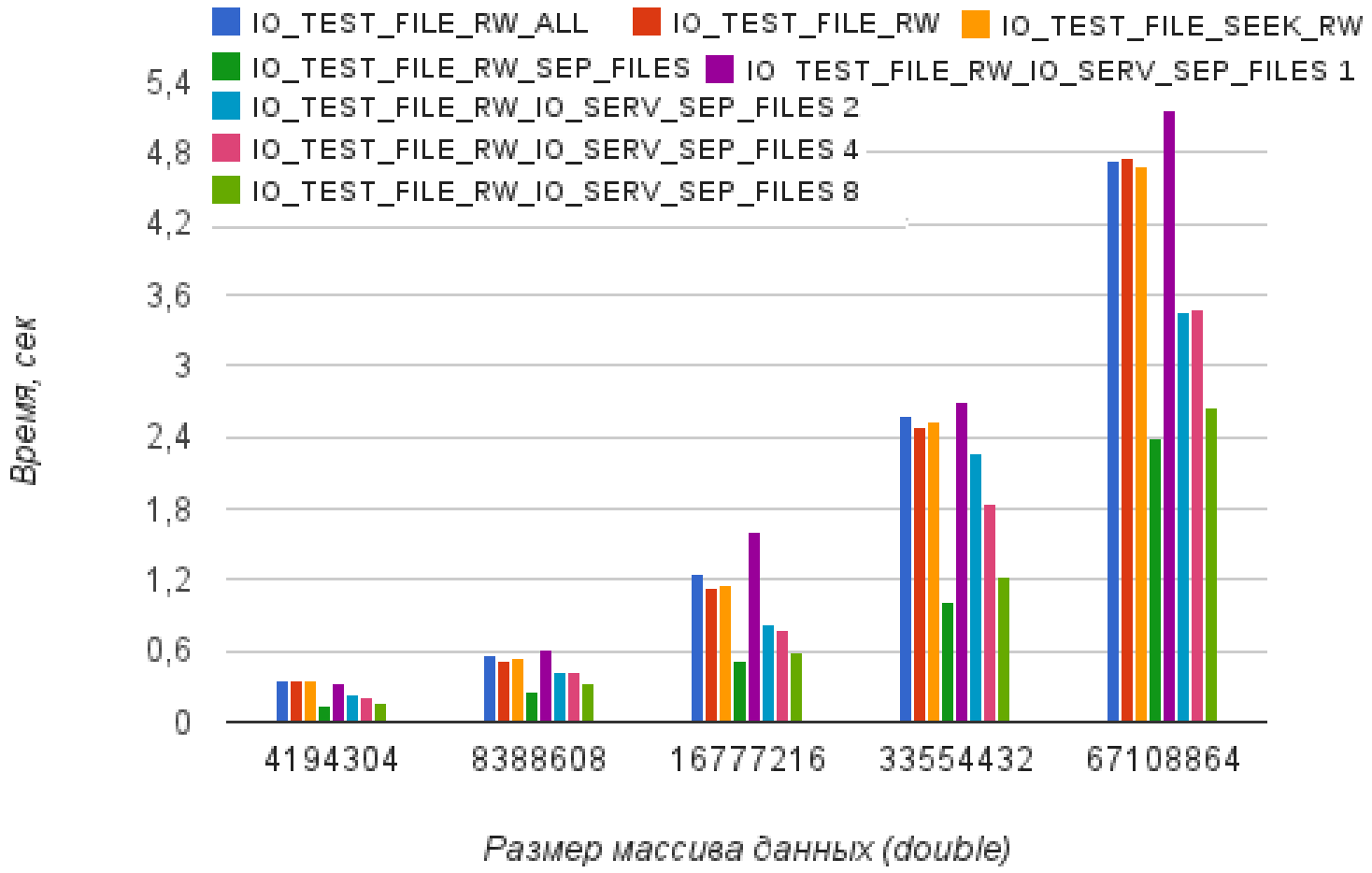
# Времена записи массива на МВС-100К при использовании различных режимов ввода-вывода



# Времена записи массива на МВС-100К при использовании различных режимов ввода-вывода



# Времена записи массива на IBM Regatta при использовании различных режимов ввода-вывода



# Проект GIMM_NANO

Разработка программного комплекса для численного решения больших задач современной электроники и лазерных нанотехнологий с использованием гибридных ЭВМ сверхвысокой производительности.

С использованием DVM-системы были разработаны:

- GIMM_APP_Quantum_Bit_States - программа трёхмерного нестационарного расчёта состояний кубитов квантового компьютера на основе решения нестационарного уравнения Шредингера для двух частиц с учетом спина;
- GIMM_APP_Quantum_Crystal_2d/GIMM_APP_Quantum_Crystal_3d - программы двухмерного/трёхмерного моделирования процессов объёмной кристаллизации при воздействии на образец лазерного или электронного пучка на основе многокомпонентной и многофазной гидродинамической модели;
- GIMM_APP_Quantum_Powder_2d/GIMM_APP_Quantum_Powder_3d - программы двухмерного/трёхмерного моделирования процессов плавления многокомпонентных порошков при селективном лазерном спекании на основе многокомпонентной и многофазной гидродинамической модели.

# Параллельный ввод-вывод в DVM-системе. Версия 1.0

В рамках проекта GIMM_NANO для DVM-программ разработана библиотека функций **ввода-вывода, хранения** и визуализации распределенных сеточных данных.

```
INTEGER NS                                  ! Число серверов ввода-вывода
REAL*8,ALLOCATABLE:: P(:,:,:)               ! Распределенный массив
!DVM$ DISTRIBUTE (BLOCK,BLOCK,BLOCK) :: P
INTEGER IERR                                ! 0 в случае успеха или код ошибки
INTEGER ID                                  ! Уникальный номер записи
CHARACTER*256  STR                          !  Имя файла/архива
ALLOCATE(P(0:Nx,0:Ny,0:Nz))
IERR = GIMM_CPLOAD(ID, P,NS,STR(1:K)) ! Чтение распределенного массива
…
IERR = GIMM_CPSAVE(ID, P,NS,STR(1:K)) ! Запись распределенного массива
ID=ID+1
IERR = GIMM_CPSAVE(ID, P,NS,STR(1:K)) ! Запись распределенного массива
…
IERR = GIMM_CPLOAD(ID, P,NS,STR(1:K)) ! Чтение распределенного массива
```

Все процессы, которые выполняют DVM-программу, будут разбиты на NS-групп. Каждая из этих групп процессов будет передавать свои данные своему серверу ввода-вывода, который и будет записывать данные в файл. Всего будет записано NS файлов.

# Параллельный ввод-вывод в DVM-системе. Версия 1.0

В рамках проекта GIMM_NANO для DVM-программ разработана библиотека функций ввода-вывода, хранения и **визуализации** распределенных сеточных данных.

```
INTEGER NS                                  ! Число серверов ввода-вывода
REAL*8,ALLOCATABLE:: P(:,:,:)               ! Распределенный массив
!DVM$ DISTRIBUTE (BLOCK,BLOCK,BLOCK) :: P
!DVM$ ALIGN XW(i,j,k) WITH P(i,j,k)
!DVM$ ALIGN YW(i,j,k) WITH P(i,j,k)
!DVM$ ALIGN ZW(i,j,k) WITH P(i,j,k)
INTEGER IERR                                ! 0 в случае успеха или код ошибки
CHARACTER*256  STR                          !  Имя файла/архива
 IERR = GIMM_PLOT_3D(XW,YW,ZW,P,NX1,NY1,NZ1,
    &    NS,
    &    STR(1:K),
    &    'Distribution of P',
    &    'x','y','z','t',
    &    'e12.5','e12.5','e12.5','e12.5')

 IERR = GIMM_PLOT_2D(…)
 IERR = GIMM_PLOT_1D(…)
```

# Параллельный ввод-вывод в DVM-системе. Версия 2.0

- 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.

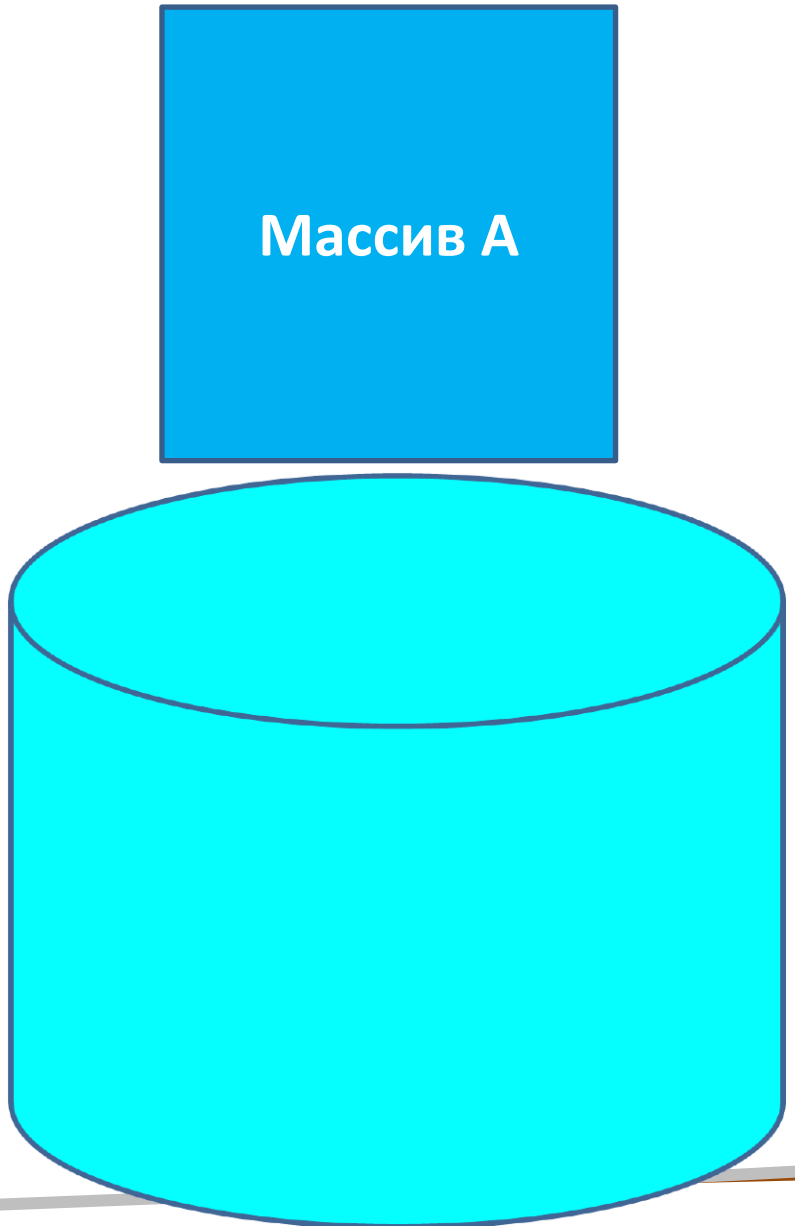
# Параллельный синхронный ввод-вывод

- **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

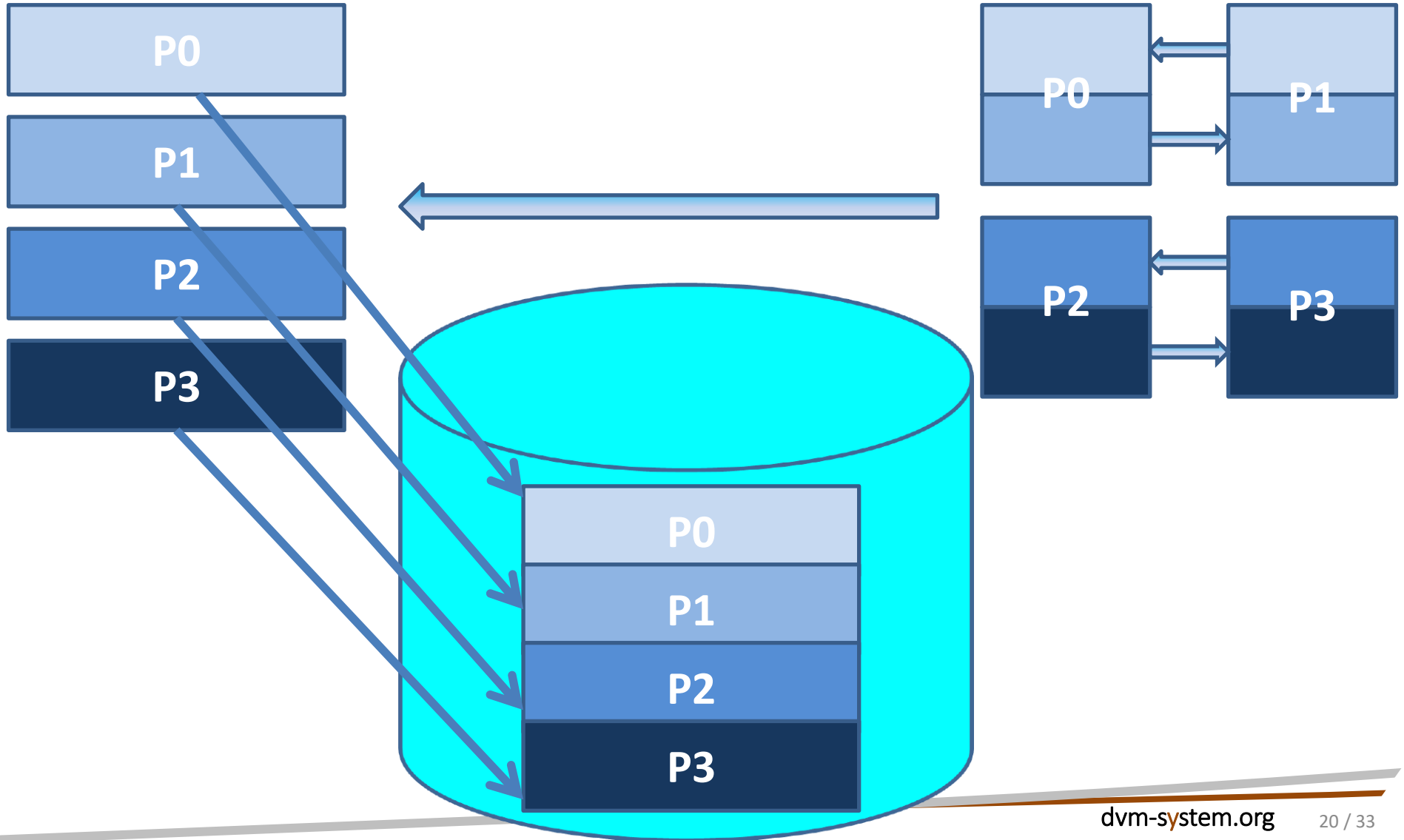

1) Передать позицию в файле от процесса ВВ/ВЫВ всем остальным процессам текущей многопроцессорной системы.
2) Каждый процесс устанавливает позицию в файле в ту точку, с которой он будет читать (или записывать).
3) Каждый процесс выполняет свою часть работы.
   - Если запрошена операция чтения в размноженную 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).

# Реализация операций fread, fwrite, fputs

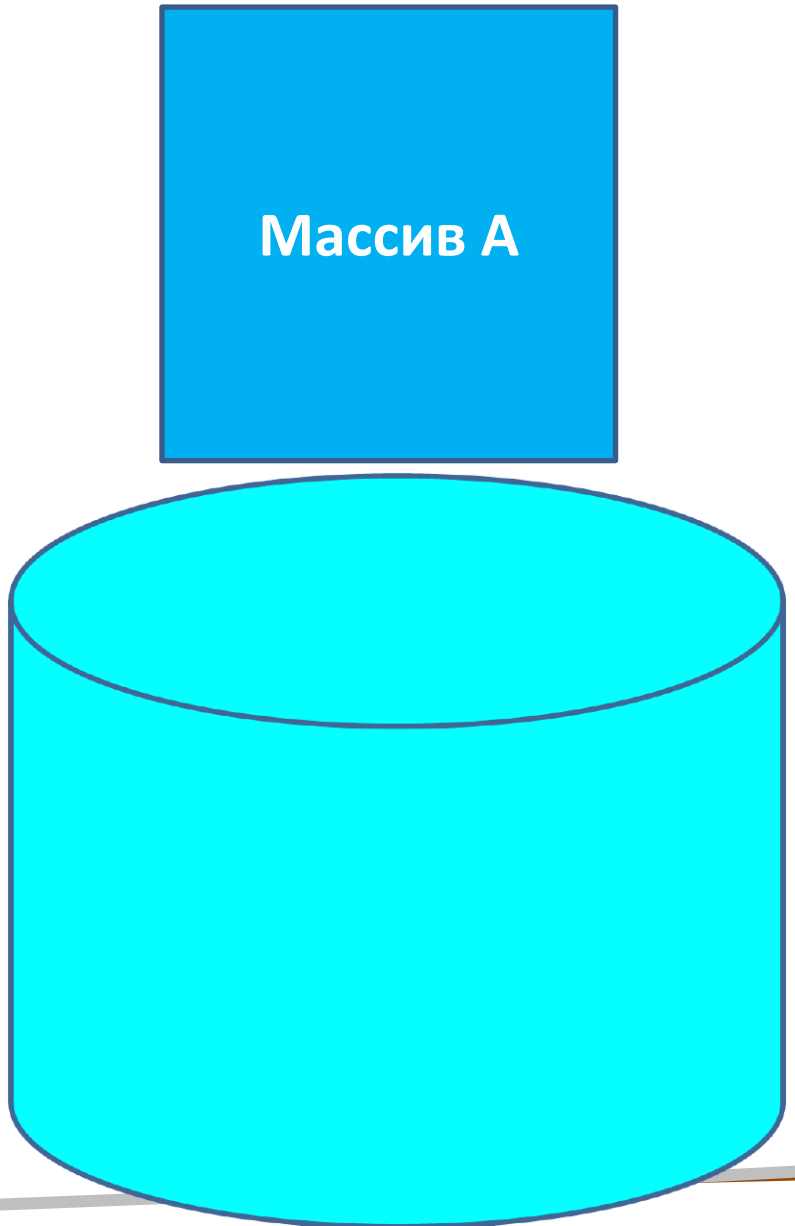


# Реализация операций fread, fwrite, fputs

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

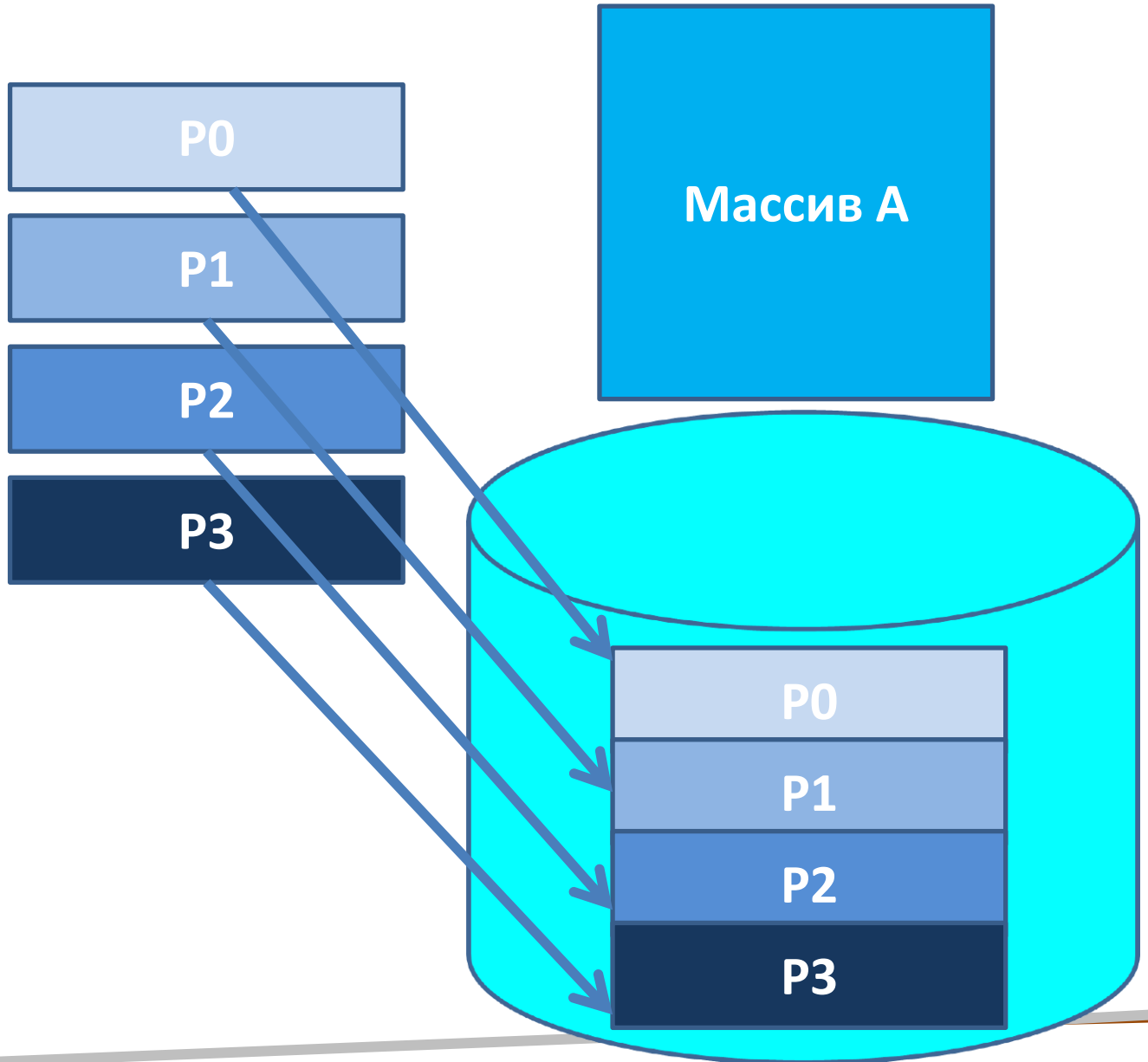


# Реализация операций fread, fwrite, fputs



# Реализация операций fread, fwrite, fputs

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



# 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

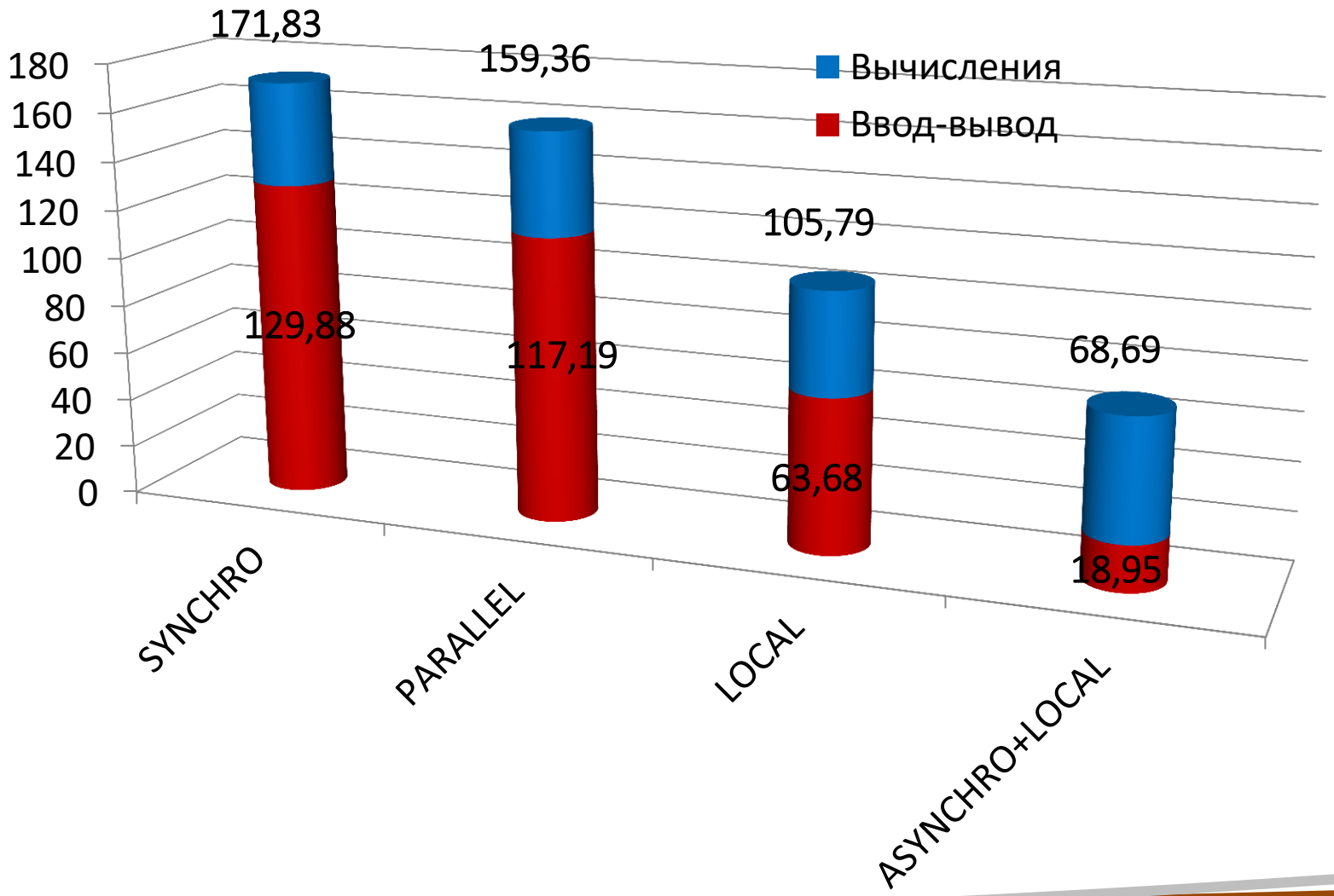
```
!DVM$ IO_MODE ([PARALLEL]
               [[,]LOCAL]
               [[,]ASYNC])

      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;
}
```

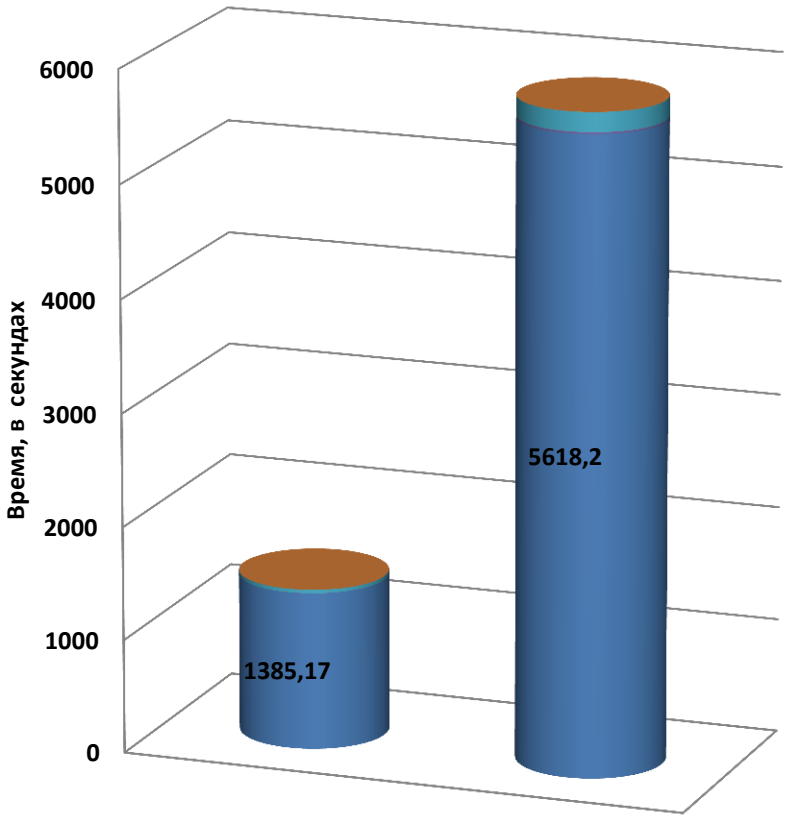
# Алгоритм Якоби в модели DVMH

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



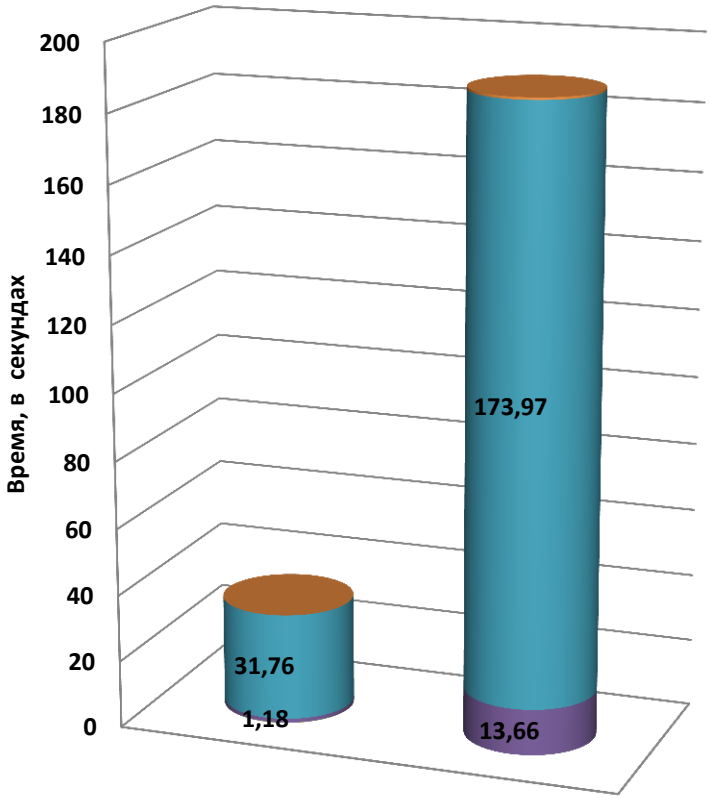
# МВС-10П. Intel Xeon E5-2690 vs Intel Xeon Phi 7110X
# Программа Якоби, 10800x10800, 10000 итераций

## Полное время работы



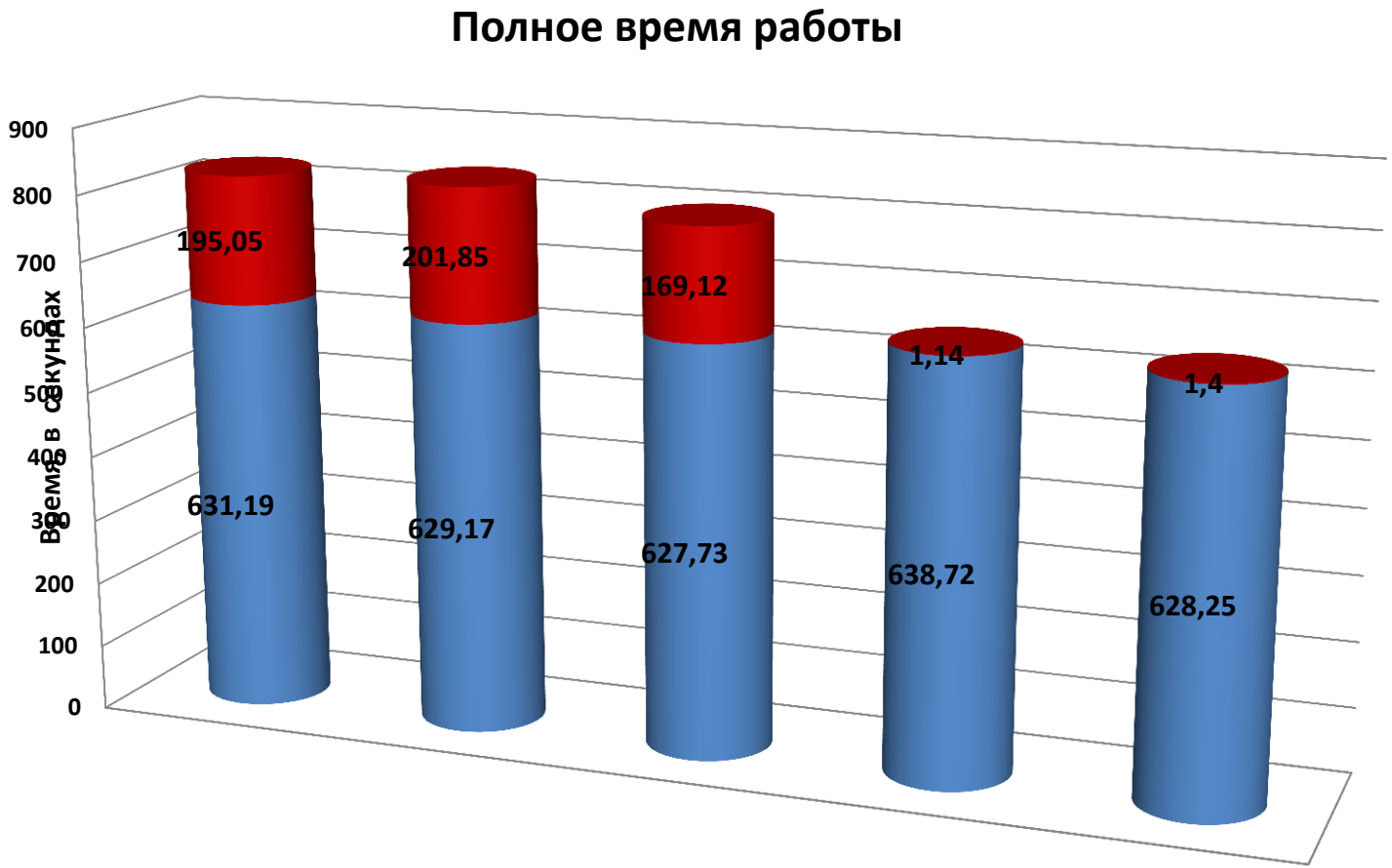

| | XEON | XEON PHI |
|---|---|---|
| REMOVE | 0 | 0,69 |
| FCLOSE | 31,76 | 173,97 |
| FWRITE | 1,18 | 13,66 |
| FOPEN | 0,09 | 0,01 |
| EXE | 1385,17 | 5618,2 |

## Время ввода-вывода



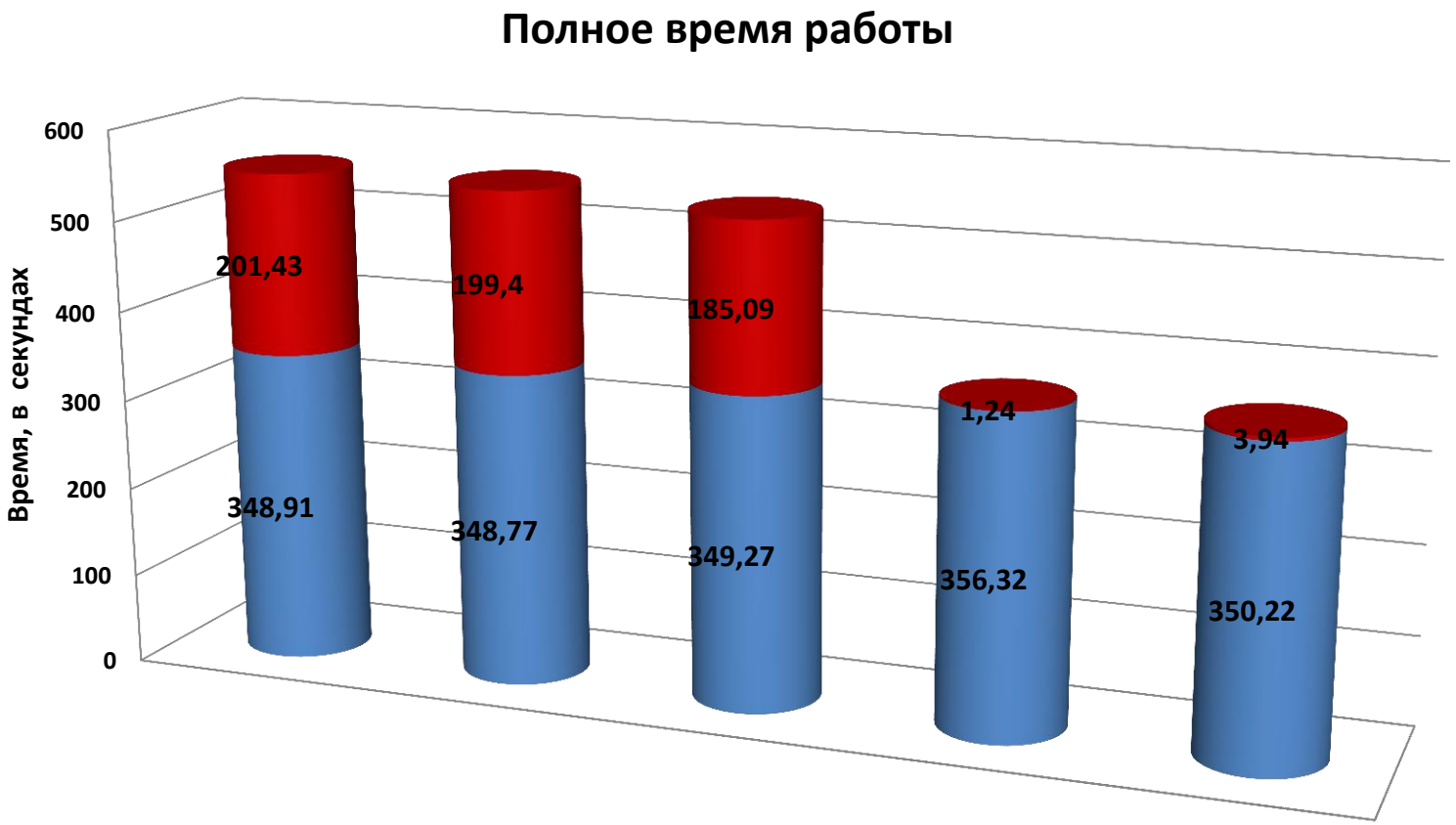

| | XEON | XEON PHI |
|---|---|---|
| REMOVE | 0 | 0,69 |
| FCLOSE | 31,76 | 173,97 |
| FWRITE | 1,18 | 13,66 |
| FOPEN | 0,09 | 0,01 |

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



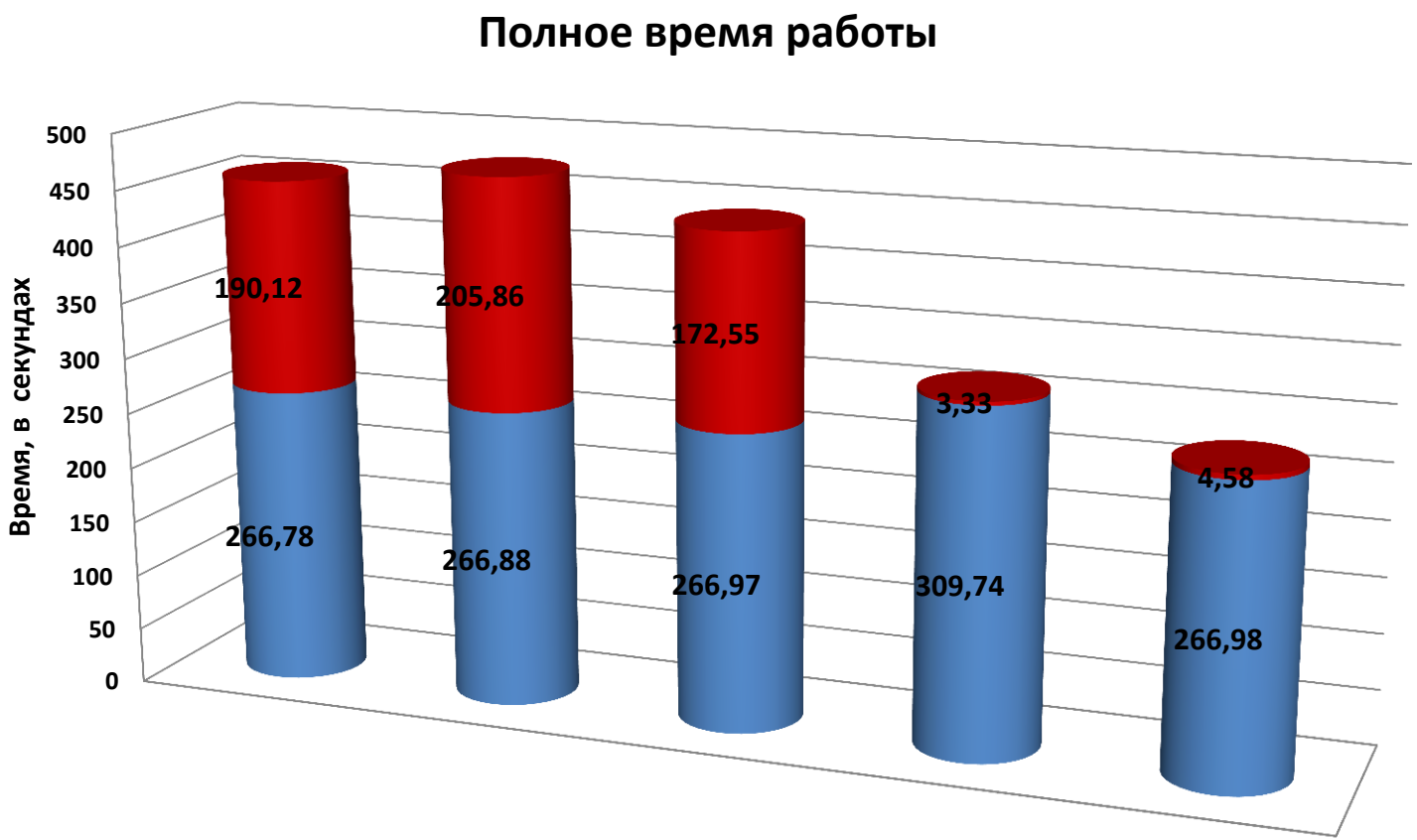

| | WB | WBP | WBL | WBSP | WBSL |
|---|---|---|---|---|---|
| Ввод-вывод | 195,05 | 201,85 | 169,12 | 1,14 | 1,4 |
| Вычисления | 631,19 | 629,17 | 627,73 | 638,72 | 628,25 |

# 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 |

# Параллельный ввод-вывод в DVM-системе. Версия 3.0. Планы

**Запись контрольной точки**

**#pragma dvm cp_save** *cp-save-clause*-list

*cp-save-clause* ::= **filename (** *string-expr*-list **)** | **data (** *var-ref*-list **)** | **async** [ **(** *id-var* **)** ] | **flexible**

*var-ref* ::= *subarray* | *datalist-name*

*id-var* ::= *int-var-name*

**Чтение контрольной точки**

**#pragma dvm cp_load** *cp-load-clause*-list

*cp-load-clause* ::= **filename (** *string-expr*-list **)** | **data (** *var-ref*-list **)** | **async (** *id-var* **)**

**Задание списка переменных**

**#pragma dvm datalist**

**void *lst;** /* Объявляет переменную хранилищем списка переменных */

**Добавление/удаление переменных из списка**

**#pragma dvm datalist_add (** *datalist-var-name* **) data (** *var-ref*-list **)**

*var-ref* ::= *subarray* | *datalist-name*

**#pragma dvm datalist_clear (** *datalist-var-name* **)**

# Выводы

- В системе автоматизации разработки параллельных программ (DVM-системе) разработаны средства параллельного ввода-вывода, обеспечивающие эффективное использование возможностей файловых систем современных кластеров.
- Одно из преимуществ разработанных средств параллельного ввода-вывода – это простота использования. В своей программе прикладной программист использует привычные ему операторы ввода-вывода последовательного языка программирования (Си или Фортран), переключение между режимами осуществляется изменением всего одного параметра функций fopen и freopen (для языка C-DVMH) или при помощи директивы IO_MODE (для языка Fortran-DVMH).
- Использование разработанных средств параллельного ввода-вывода в тестовых программах позволило существенно сократить время, требуемое для выполнения операций ввода-вывода.
- Разработанные средства асинхронного ввода-вывода являются универсальными – могут использоваться и для сохранения контрольных точек, и для сохранения данных, полученных на очередном временном шаге алгоритма (например, для визуализации), и для вычислений с массивами на внешней памяти.

# Спасибо за внимание


http://dvm-system.org