

Автоматизация разработки параллельных программ для гетерогенных вычислительных кластеров

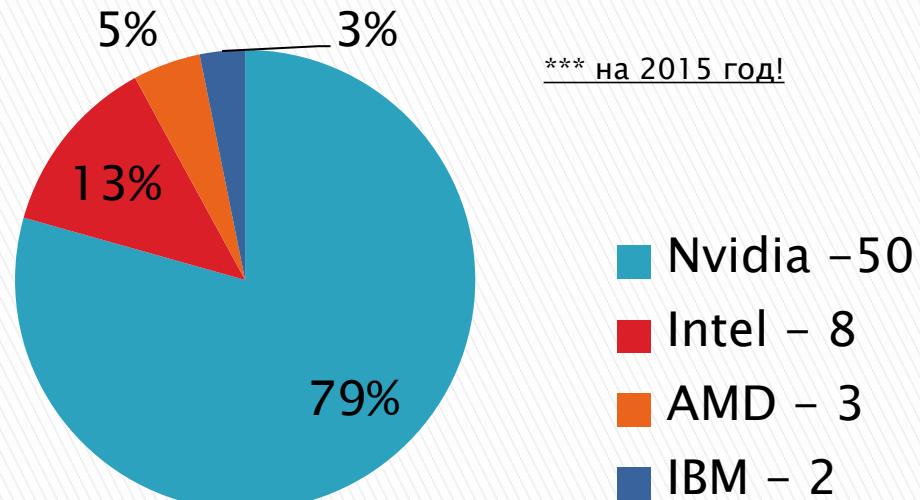
<http://dvm-system.org>

Институт прикладной математики им. М.В. Келдыша РАН
Колганов Александр alexander.k.s@mail.ru

Рейтинг TOP500 2022 год

(www.top500.org)

1. Frontier – HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X. **1102 PFlop/s**
2. Supercomputer Fugaku – Supercomputer Fugaku, A64FX 48C 2.2GHz, **442 PFlop/s**
3. LUMI – HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X. **151 PFlop/s**



Автоматизированно или Автоматически?

- ▶ Автоматическое распараллеливание – процесс отображения последовательной программы компиляторами (например, Intel, PGI) на параллельную архитектуру, состоящий в её преобразовании без участия пользователя для эффективного выполнения на параллельной вычислительной системе.
- ▶ Автоматизированное распараллеливание – процесс отображения последовательной программы на параллельную архитектуру, состоящий в ее преобразовании, в котором пользователь принимает активное участие. Автоматизированное распараллеливание может включать в себя процесс автоматического распараллеливания.

Проблемы распараллеливания последовательной программы на гетерогенный кластер

- ▶ Необходимость отображения данных на узлы кластера;

Проблемы распараллеливания последовательной программы на гетерогенный кластер

- ▶ Необходимость отображения данных на узлы кластера;
- ▶ Поиск оптимального (эффективного) способа распределения данных;

Проблемы распараллеливания последовательной программы на гетерогенный кластер

- ▶ Необходимость отображения данных на узлы кластера;
- ▶ Поиск оптимального (эффективного) способа распределения данных;
- ▶ Создание выходной программы с использованием стандартных языков параллельного программирования (*например, MPI, OpenMP, CUDA, OpenCL и т.д.*);

Проблемы распараллеливания последовательной программы на гетерогенный кластер

- ▶ Необходимость отображения данных на узлы кластера;
- ▶ Поиск оптимального (эффективного) способа распределения данных;
- ▶ Создание выходной программы с использованием стандартных языков параллельного программирования (*например, MPI, OpenMP, CUDA, OpenCL и т.д.*);
- ▶ Необходимость выполнения преобразований программы;

Проблемы распараллеливания последовательной программы на гетерогенный кластер

- ▶ Необходимость отображения данных на узлы кластера;
- ▶ Поиск оптимального (эффективного) способа распределения данных;
- ▶ Создание выходной программы с использованием стандартных языков параллельного программирования (*например, MPI, OpenMP, CUDA, OpenCL и т.д.*);
- ▶ Необходимость выполнения преобразований программы;
- ▶ Распределение работы в узле кластера между ускорителями и мультипроцессорами.

Программное обеспечение

- ▶ Низкоуровневые – MPI, SHMEM, pThread, TBB, CUDA, OpenCL;
- ▶ Высокоуровневые – OpenMP, OpenACC, DVMH, HPF, CoArray Fortran, UPC, Titanium, Chapel, X10, Fortress, XcalableACC, XcalableMP, HOPMA;
- ▶ Системы автоматизации – DVM, CAPTools, Parawise, FORGE Magic/DM, BERT77, ParalWare Trainer, Appolo, SAPFOR, Pluto, APC, ДВОР;

Высокоуровневые языки: OpenMP

- ▶ Директивные расширения языков C и Fortran;
- ▶ Стандарт, поддерживаемый большинством компиляторов (gnu, intel и др);
- ▶ В версии 4.0 поддержка ЦПУ и сопроцессоров Intel Xeon Phi (*offload* модель);
- ▶ Прозрачная масштабируемость на сотни потоков;
- ▶ Поддержка векторизации с помощью клаузы SIMD.

Высокоуровневые языки: OpenMP

сложение векторов

```
int A[L], B[L], C[L];  
  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}
```

Последовательный вариант

```
#pragma omp parallel for  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}  
  
#pragma omp parallel for simd  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}
```

Высокоуровневые языки: OpenMP

сложение векторов

```
int A[L], B[L], C[L];  
  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}
```

```
#pragma omp parallel for  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}
```

Параллельный вариант

```
#pragma omp parallel for simd  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}
```

Высокоуровневые языки: OpenMP

сложение векторов

```
int A[L], B[L], C[L];  
  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}  
  
#pragma omp parallel for  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}  
  
#pragma omp parallel for simd  
for (int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i]  
}
```

Параллельный вариант с использованием векторизации

Высокоуровневые языки: OpenACC

- ▶ Директивные расширения языков C и Fortran для отображения программы на GPU;
- ▶ Быстрая переносимость программы на GPU в рамках одного узла;
- ▶ Поддержка разных архитектур фирм AMD и NVidia;
- ▶ Большой набор директив для оптимизации программы.

Высокоуровневые языки: OpenACC

сложение векторов

```
int A[L], B[L], C[L];  
  
for(int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i];  
}
```

Последовательный вариант

```
#pragma acc update device(A,B)  
#pragma acc kernels  
{  
    for(int i = 0; i < L; ++i)  
    {  
        C[i] = A[i] + B[i];  
    }  
}  
#pragma acc update host(C)
```

Высокоуровневые языки: OpenACC

сложение векторов

```
int A[L], B[L], C[L];  
  
for(int i = 0; i < L; ++i)  
{  
    C[i] = A[i] + B[i];  
}
```

```
#pragma acc update device(A,B)  
#pragma acc kernels  
{  
    for(int i = 0; i < L; ++i)  
    {  
        C[i] = A[i] + B[i];  
    }  
}  
#pragma acc update host(C)
```

Параллельный вариант,
цикл вычисляется на GPU

Низкоуровневые языки: CUDA

- ▶ Архитектура и модель, реализованная компанией NVidia;
- ▶ Поддержка GPU только фирмы NVidia;
- ▶ CUDA API и CUDA Driver API для C и Fortran;
- ▶ Собственный компилятор nvcc (для C) и pgf (для фортрана);
- ▶ Мощные средства для оптимизации программы – программист может управлять памятью, регистрами, кэшем.

Низкоуровневые языки: CUDA

сложение векторов

```
__global__ void add(int *A, int *B, int *C, int N)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    if(i < N)
        C[i] = A[i] + B[i];
}

int main()
{
    int A[L], B[L], C[L];
    int *devA, *devB, *devC;

    cudaMalloc((void**)&devA, sizeof(int) * L);
    cudaMalloc((void**)&devB, sizeof(int) * L);
    cudaMalloc((void**)&devC, sizeof(int) * L);

    cudaMemcpy(devA, A, sizeof(int) * L, cudaMemcpyHostToDevice);
    cudaMemcpy(devB, B, sizeof(int) * L, cudaMemcpyHostToDevice);

    dim3 th = dim3(256);
    dim3 bl = dim3(L / th + 1);
    add <<<bl, th >>> (devA, devB, devC, L);
    cudaMemcpy(C, devC, sizeof(int) * L, cudaMemcpyDeviceToHost);
}
```

Низкоуровневые языки: OpenCL

- ▶ Открытый стандарт для параллельного программирования массивно-параллельных архитектур внутри одного узла;
- ▶ Поддерживается Nvidia, Intel, AMD и др.;
- ▶ Архитектурно независимая модель;
- ▶ Мощные средства для оптимизации программы – программист может управлять памятью, регистрами, кэшем.

Низкоуровневые языки: MPI

- ▶ Стандарт параллельного программирования, поддерживается многими компиляторами;
- ▶ Параллельные программы выполняются как внутри узла, так и на разных узлах;
- ▶ Можно использовать OpenMP для эффективной загрузки ядер ЦПУ внутри узла;
- ▶ Можно использовать CUDA, OpenACC, OpenCL для выполнения части программы на GPU внутри узла.

Система DVM

- ▶ Создана в 1993 г. в Институте прикладной математики им. М. В. Келдыша РАН;
- ▶ Объединяет достоинства модели параллелизма по данным и модели параллелизма по управлению;
- ▶ Аббревиатура DVM соответствует двум понятиям: Distributed Virtual Memory и Distributed Virtual Machine;
- ▶ Существует для двух языков: C-DVMH и Fortran-DVMH;
- ▶ Предназначена для использования на кластерах с аппаратурой различной.

Модель программирования в DVM

- ▶ На программиста возлагается ответственность за соблюдение правила собственных вычислений;
- ▶ Программист определяет общие (удаленные) данные, т.е. данные, вычисляемые на одних процессорах и используемые на других процессорах;
- ▶ Программист отмечает точки в последовательной программе, где происходит обновление значений общих данных;
- ▶ Программист распределяет по процессорам виртуальной параллельной машины не только данные, но и соответствующие вычисления.

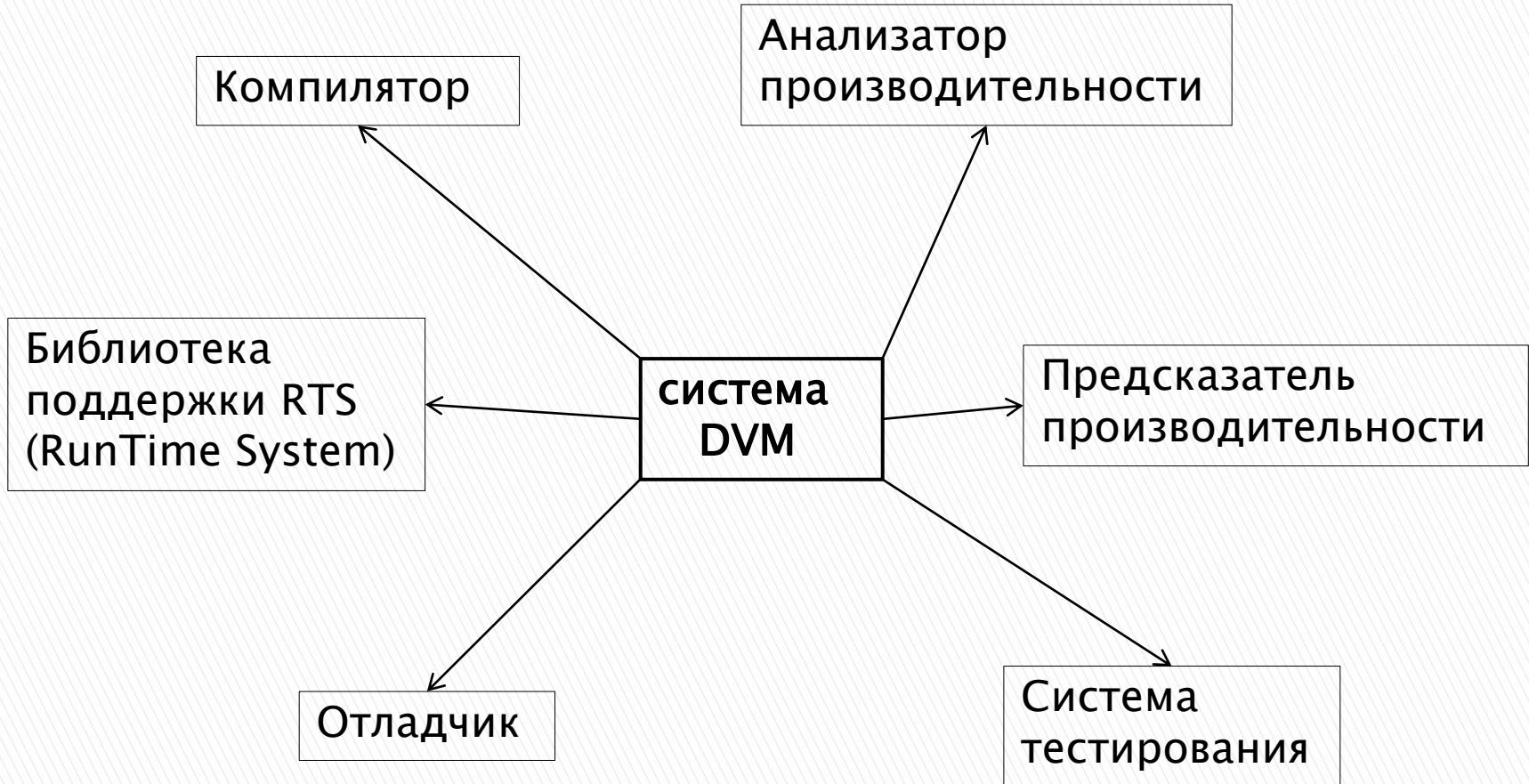
Средства программирования

C-DVMH = Язык Си 99 + специальные прагмы

Fortran-DVMH = Язык Фортран 95 + спец.комментарии

- ▶ Специальные комментарии 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

Компоненты DVM-системы



DVMH-модель

- ▶ Предложена в 2011 г. (*расширение DVM-модели*);
- ▶ Расширена в 2014 г. для эффективной поддержки Intel Xeon Phi;
- ▶ Позволяет использовать установленные на кластерах GPU Nvidia, а также ЦПУ и сопроцессоры Intel Xeon Phi;
- ▶ Расширение модели DVM – позволяет небольшими изменениями перевести DVM-программу в DVMH-программу.

Высокоуровневые языки: DVMH

сложение векторов

```
#pragma dvm array distribute[block]
int A[L];
#pragma dvm array align([i] with A[i])
int B[L];
#pragma dvm array align([i] with A[i])
int C[L];

for (int i = 0; i < L; ++i)
    C[i] = A[i] + B[i];
```

Последовательный вариант

```
#pragma dvm region
{
    #pragma dvm parallel([i] on A[i])
        for (int i = 0; i < L; ++i)
            C[i] = A[i] + B[i]
}
```

Высокоуровневые языки: DVMH

сложение векторов

```
#pragma dvm array distribute[block]
int A[L];
#pragma dvm array align([i] with A[i])
int B[L];
#pragma dvm array align([i] with A[i])
int C[L];

for (int i = 0; i < L; ++i)
    C[i] = A[i] + B[i];
```

Параллельный вариант,
цикл вычисляется на
нескольких узлах

```
#pragma dvm region
{
    #pragma dvm parallel([i] on A[i])
        for (int i = 0; i < L; ++i)
            C[i] = A[i] + B[i]
}
```

..... и на GPU и/или
мультипроцессоре

Программное обеспечение

- ▶ Низкоуровневые – MPI, SHMEM, pThread, TBB, CUDA, OpenCL;
- ▶ Высокоуровневые – OpenMP, OpenACC, DVMH, HPF, CoArray Fortran, UPC, Titanium, Chapel, X10, Fortress, XcalableACC, XcalableMP, HOPMA;
- ▶ Системы автоматизации – DVM, CAPTools, Parawise, FORGE Magic/DM, BERT77, ParalWare Trainer, Appolo, SAPFOR, Pluto, APC, ДВОР;

DVMH



MPI + OpenMP +
CUDA

Сравнение рассмотренных моделей

	OpenMP	OpenACC	OpenCL	CUDA	MPI	DVMH
Выполнение внутри узла	+	+	+	+	+	+
Выполнение на многих узлах					+	+
Сложность распараллеливания	1	2	5	4	3	2
Сложность отладки	3	3	5	3	5	2
Увеличение кода прогр.	1	1	5	4	3	1
Поддержка CPU	+	+	+	--	+	+
Поддержка GPU AMD	+*	+	+	--		--
Поддержка GPU NVidia	+*	+	+	+		+
Поддержка Xeon Phi	+	+	+	--	+	+

DVMH-модель распараллеливание на кластер

Распределение данных. DISTRIBUTE

```
float B[L][L];
float A[L][L];
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Распределение данных. DISTRIBUTE

#pragma dvm array

+

distribute

align

shadow

+

[]
block
wgtblock
genblock
multblock

Распределение данных. **DISTRIBUTE**

Как разбить массив
между узлами?

#pragma dvm array

+

distribute

align

shadow

+

[]
block
wgtblock
genblock
multblock

Блоchное
распределение:

- отложенное
- равными
- взвешенными
- неравными
- кратными т

Распределение данных. DISTRIBUT

```
#pragma dvm array distribute[block][block]
float B[L][L];
float A[L][L];
for (i = 1; i < L - 1; i++) {
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
}
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Распределение данных. DISTRIBUT

```
#pragma dvm array distribute[block][block]
float B[L][L];
float A[L][L];
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] -
            eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Отображение решетки
процессоров:

dvm run 2 3 JAC

Распределение данных. DISTRIBUT

```
#pragma dvm array distribute[ ][block]
float B[L][L];
float A[L][L];
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Распределение данных. DISTRIBUTE

```
#pragma dvm array distribute[ ][block]
float B[L][L];
float A[L][L];
for (i = 1; i < L - 1; i++) {
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] -
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
    for (i = 1; i < L - 1; i++)
        for (j = 1; j < L - 1; j++)
            B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Отображение решетки
процессоров:

dvm run 1 2 JAC

dvm run 2 1 JAC

dvm run 2 JAC

Локализация данных. ALIGN

#pragma dvm array

+

distribute

align

+

shadow

[]
with

Локализация данных. ALIGN

*Как связаны массивы
между собой?*

#pragma dvm array

+

distribute

align

shadow

+

[]
with

Правило (отображение)
выравнивания данных:

$B[i] \Rightarrow A[a^*i+b]$

Локализация данных. ALIGN

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Локализация данных. ALIGN

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];
for (i = 1; i <
     for (j = 1;
          float tm
          eps = Ma
          A[i][j] = B[i][j];
}
for (i = 1; i < L - 1; i++)
  for (j = 1; j < L - 1; j++)
    B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Правило выравнивания для массива A:
Элементы массива A[i][j] должны
находиться там же, где и элементы
массива B[i][j]

Локализация данных. ALIGN

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
```

После распределения данных необходимо распределить вычисления, иначе такая программа упадет из-за не соблюдения правила собственных вычислений!

Распределение данных. DISTRIBUTE

```
#pragma dvm array
float (*A)[L+1];
#pragma dvm array
float (*X);

/* create arrays */
A = malloc(L * (L + 1) * sizeof(float));
X = malloc((L / 2) * sizeof(float));
```

Распределение данных. DISTRIBUTE

```
#pragma dvm array  
float (*A)[L+1];  
  
#pragma dvm array  
float (*X);
```

Распределение динамически выделяемых массивов может быть только отложенным

```
/* create arrays */  
A = malloc(L * (L + 1) * sizeof(float));  
X = malloc((L / 2) * sizeof(float));
```

Распределение данных. **DISTRIBUTE**

```
#pragma dvm array  
float (*A)[L+1];  
  
#pragma dvm array  
float (*X);
```

Распределение динамически выделяемых массивов может быть только отложенным

```
/* create arrays */  
A = malloc(L * (L + 1) * sizeof(float));  
X = malloc((L / 2) * sizeof(float));
```

```
#pragma dvm redistribute (A[block][])  
#pragma dvm realign(X[i] with A[2*i][])

```

Правило выравнивания для массива X:
Элементы массива X[i] должны находиться там же, где и элементы массива A[2*i][]

Удаленные ссылки. SHADOW

#pragma dvm array

+

distribute

align

shadow

+

[low : high]

Удаленные ссылки. SHADOW

$$A[i] = B[i-d1] + B[i+d2]$$

*Есть ли пересечения
в обращении к
распределенным
данным?*

#pragma dvm array

+

distribute

align

shadow

В зависимости от
выравнивания
данных между A и B
ширина теневой
границы будет разной!

+

[low : high]

Удаленные ссылки. SHADOW

$$A[i] = B[i-d1] + B[i+d2]$$

*Есть ли пересечения
в обращении к
распределенным
данным?*

#pragma dvm array

+

distribute

align

shadow

В зависимости от
выравнивания
данных между A и B
ширина теневой
границы будет разной!

=> shadow[d1:d2]

+

[low : high]

Удаленные ссылки. SHADOW

$$A[i] = B[i-d1] + B[i+d2]$$

Есть ли пересечения
в обращении к
распределенным
данным?

#pragma dvm array +

distribute

align

shadow

В зависимости от
выравнивания
данных между A и B
ширина теневой
границы будет разной!

=> shadow[d1+d2:0]

+ [low : high]

Удаленные ссылки. SHADOW

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. SHADOW

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```



Распределение витков цикла. PARALLEL

- ▶ Моделью выполнения DVMH-программы является модель SPMD;
- ▶ Спецификация распределения вычислений разрешается только для циклов:
 - цикл является тесно-гнездовым циклом с прямоугольным индексным пространством;
 - распределенные измерения массивов индексируются только регулярными выражениями типа $a^*J + b$, где J – индекс цикла;
 - левая часть оператора присваивания является ссылкой на распределенный массив, редукционную переменную или приватную переменную;
 - выполняется правило собственных вычислений;
 -

Распределение витков цикла. PARALLEL

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm parallel ([i][j] on A[i][j])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Распределение витков цикла. PARALLEL

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm parallel ([i][j] on A[i][j])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. SHADOW

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm parallel ([i][j] on A[i][j])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. SHADOW

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm parallel ([i][j] on A[i][j])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```



Удаленные ссылки. SHADOW

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm parallel ([i][j] on A[i][j])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(
            A[i][j] = shadow_renew(A) - сокращенная
            запись от shadow_renew(A[1:1][1:1])
        }
#pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. SHADOW

A ₀₀	A ₀₁	A ₀₂	A ₀₃	A ₀₄	A ₀₅	A ₀₆	A ₀₇	A ₀₈
A ₁₀	A ₁₁	A ₁₂	A ₁₃	A ₁₄	A ₁₅	A ₁₆	A ₁₇	A ₁₈
A ₂₀	A ₂₁	A ₂₂	A ₂₃	A ₂₄	A ₂₅	A ₂₆	A ₂₇	A ₂₈
A ₃₀	A ₃₁	A ₃₂	A ₃₃	A ₃₄	A ₃₅	A ₃₆	A ₃₇	A ₃₈
A ₂₀	A ₂₁	A ₂₂	A ₂₃	A ₂₄	A ₂₅	A ₂₆	A ₂₇	A ₂₈
A ₃₀	A ₃₁	A ₃₂	A ₃₃	A ₃₄	A ₃₅	A ₃₆	A ₃₇	A ₃₈
A ₄₀	A ₄₁	A ₄₂	A ₄₃	A ₄₄	A ₄₅	A ₄₆	A ₄₇	A ₄₈
A ₅₀	A ₅₁	A ₅₂	A ₅₃	A ₅₄	A ₅₅	A ₅₆	A ₅₇	A ₅₈
A ₆₀	A ₆₁	A ₆₂	A ₆₃	A ₆₄	A ₆₅	A ₆₆	A ₆₇	A ₆₈
A ₅₀	A ₅₁	A ₅₂	A ₅₃	A ₅₄	A ₅₅	A ₅₆	A ₅₇	A ₅₈
A ₆₀	A ₆₁	A ₆₂	A ₆₃	A ₆₄	A ₆₅	A ₆₆	A ₆₇	A ₆₈
A ₇₀	A ₇₁	A ₇₂	A ₇₃	A ₇₄	A ₇₅	A ₇₆	A ₇₇	A ₇₈
A ₈₀	A ₈₁	A ₈₂	A ₈₃	A ₈₄	A ₈₅	A ₈₆	A ₈₇	A ₈₈

Локальная часть процессора P1

Локальная часть процессора P2

Локальная часть процессора P3

Удаленные ссылки. SHADOW

A ₀₀	A ₀₁	A ₀₂	A ₀₃	A ₀₄	A ₀₅	A ₀₆	A ₀₇	A ₀₈
A ₁₀	A ₁₁	A ₁₂	A ₁₃	A ₁₄	A ₁₅	A ₁₆	A ₁₇	A ₁₈
A ₂₀	A ₂₁	A ₂₂	A ₂₃	A ₂₄	A ₂₅	A ₂₆	A ₂₇	A ₂₈
A ₃₀	A ₃₁	A ₃₂	A ₃₃	A ₃₄	A ₃₅	A ₃₆	A ₃₇	A ₃₈

Локальная часть процессора P1

A ₂₀	A ₂₁	A ₂₂	A ₂₃	A ₂₄	A ₂₅	A ₂₆	A ₂₇	A ₂₈
A ₃₀	A ₃₁	A ₃₂	A ₃₃	A ₃₄	A ₃₅	A ₃₆	A ₃₇	A ₃₈
A ₄₀	A ₄₁	A ₄₂	A ₄₃	A ₄₄	A ₄₅	A ₄₆	A ₄₇	A ₄₈
A ₅₀	A ₅₁	A ₅₂	A ₅₃	A ₅₄	A ₅₅	A ₅₆	A ₅₇	A ₅₈

Локальная часть процессора P2

A ₅₀	A ₅₁	A ₅₂	A ₅₃	A ₅₄	A ₅₅	A ₅₆	A ₅₇	A ₅₈
A ₆₀	A ₆₁	A ₆₂	A ₆₃	A ₆₄	A ₆₅	A ₆₆	A ₆₇	A ₆₈
A ₇₀	A ₇₁	A ₇₂	A ₇₃	A ₇₄	A ₇₅	A ₇₆	A ₇₇	A ₇₈
A ₈₀	A ₈₁	A ₈₂	A ₈₃	A ₈₄	A ₈₅	A ₈₆	A ₈₇	A ₈₈

Локальная часть процессора P3

Удаленные ссылки. REDUCTION

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm parallel ([i][j] on A[i][j]) reduction(max(eps))
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;

printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
```

```
float B[L][L];
```

```
#pragma dvm array align([i][j] with B[i][j])
```

```
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on B[i][j]) remote_access(A[][6])
```

```
for (i = 1; i < L - 1; i++)
```

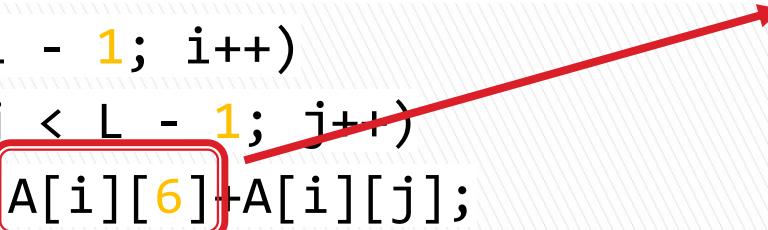
```
    for (j = 1; j < L - 1; j++)
```

```
        B[i][j] = A[i][6]+A[i][j];
```

Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on B[i][j]) remote_access(A[][6])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = A[i][6] + A[i][j];
```



Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on B[i][j]) remote_access(A[][6])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = A[i][6] + A[i][j];
```

!! Не реализовано в C-DVMH (*работает в F-DVMH*)

Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];
```

```
float buf[L]; ← Не распределенный массив!
#pragma dvm remote_access(A[][6])
{
    for (i = 1; i < L - 1; i++)
        buf[i] = A[i][6];
}
```

```
#pragma dvm parallel ([i][j] on B[i][j]) remote_access(A[][6])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = A[i][6] + A[i][j];
```

Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];
```

```
float buf[L];
#pragma dvm remote_access(A[][6])
{
    for (i = 1; i < L - 1; i++)
        buf[i] = A[i][6];
}
```

Ручное копирование в буфер

```
#pragma dvm parallel ([i][j] on B[i][j]) remote_access(A[][6])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = A[i][6] + A[i][j];
```

Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];

float buf[L];
#pragma dvm remote_access(A[][6])
{
    for (i = 1; i < L - 1; i++)
        buf[i] = A[i][6];
}
```

Использование буфера в цикле

```
#pragma dvm parallel ([i][j] on B[i][j])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = buf[i]+A[i][j];
```

Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];

#pragma dvm parallel ([i][j] on B[i][j]) remote_access(A[][6])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = A[i][6]+A[i][j];

X = A[6][61];
```

Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];

#pragma dvm parallel ([i][j] on B[i][j]) remote_access(A[][6])
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = A[i][6]+A[i][j];

#pragma dvm remote_access(A[6][61])
{
    X = A[6][61];
}
```

Удаленные ссылки. REMOTE

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j])
float A[L][L];
```



```
#pragma dvm parallel
for (i = 1; i < L -
    for (j = 1; j < L;
        B[i][j] = A[i][j] + A[i][j],
```

правило выравнивания для массива А
+
правило собственных вычислений

```
#pragma dvm remote_access(A[6][61])
{
    X = A[6][61];
}
```



Удаленные ссылки. ACROSS

```
#pragma dvm array distribute[block][block]
```

```
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on A[i][j])
```

```
for (i = 1; i < L - 1; i++)
```

```
    for (j = 1; j < L - 1; j++)
```

```
        A[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
```

```
printf(" IT = %4i    EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. ACROSS

```
#pragma dvm array distribute[block][block]
```

```
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on A[i][j])
```

```
for (i = 1; i < L - 1; i++)
```

```
    for (j = 1; j < L - 1; j++)
```

```
        A[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
```

```
printf(" IT = %i EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. ACROSS

```
#pragma dvm array distribute[block][block]
```

```
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on A[i][j]) across(A[1:1][1:1])
```

```
for (i = 1; i < L - 1; i++)
```

```
    for (j = 1; j < L - 1; j++)
```

```
        A[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
```

```
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. ACROSS

```
#pragma dvm array distribute[block][block]
```

```
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on A[i][j]) across(A[1:1][ ])
```

```
for (i = 1; i < L - 1; i++)
```

```
    for (j = 1; j < L - 1; j++)
```

```
        A[i][j] = (A[i-1][j]+A[i][j]+A[i+1][j])/3;
```

```
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. ACROSS

```
#pragma dvm array distribute[block][ ]
```

```
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on A[i][j]) across(A[1:1][1:1])
```

```
for (i = 1; i < L - 1; i++)
```

```
    for (j = 1; j < L - 1; j++)
```

```
        A[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
```

```
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Удаленные ссылки. ACROSS + STAGE

```
#pragma dvm array distribute[block][block]
```

```
float A[L][L];
```

```
#pragma dvm parallel ([i][j] on A[i][j]) across(A[1:1][1:1]),
```

```
stage(2)
```

```
for (i = 1; i < L - 1; i++)
```

```
    for (j = 1; j < L - 1; j++)
```

```
        A[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
```

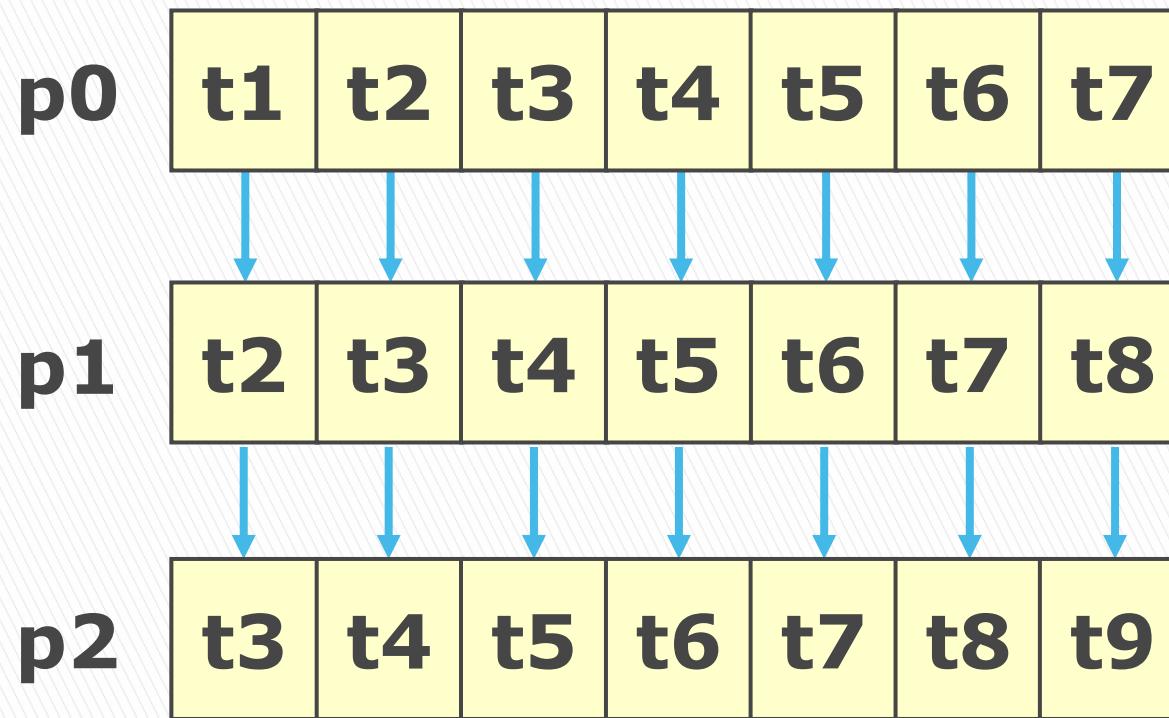
```
printf(" IT = %4i    EPS = %14.7E\n", it, eps);
```

Пользователь может указать количество шагов конвейера с помощью stage (expr).

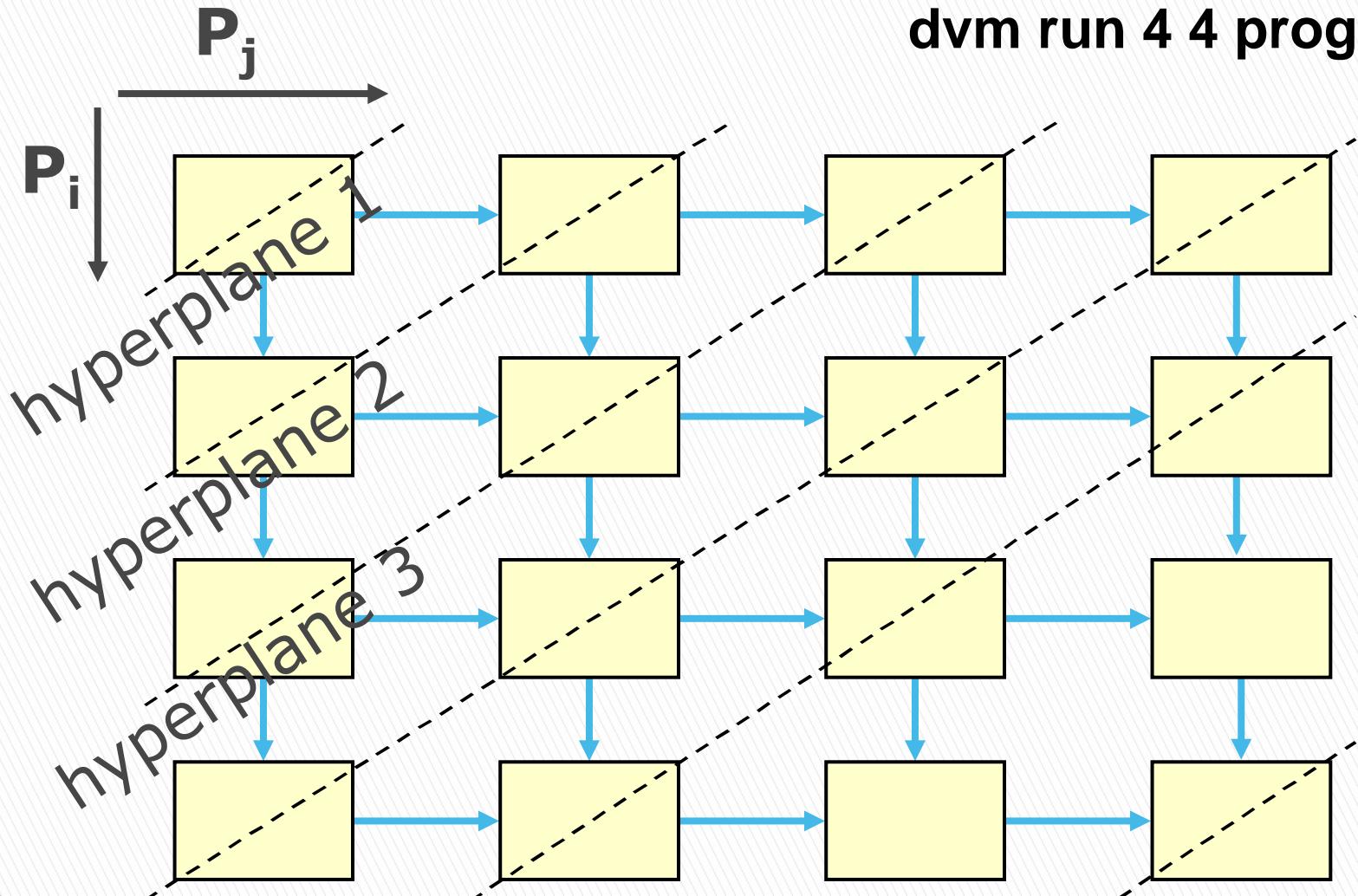
stage(1) – последовательное выполнение.

Конвейеризация + stage. 1D

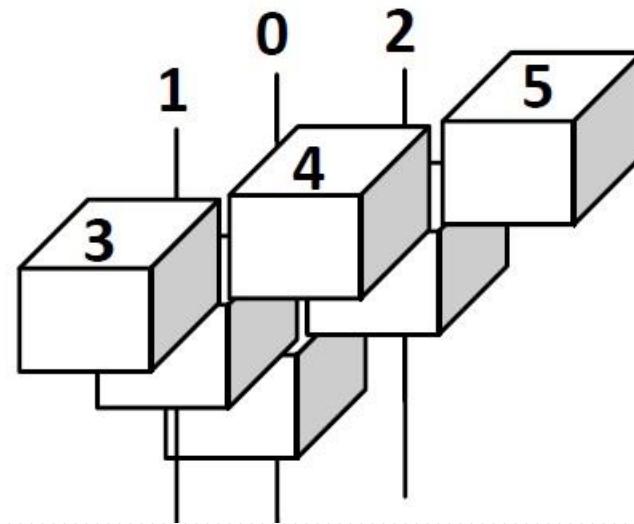
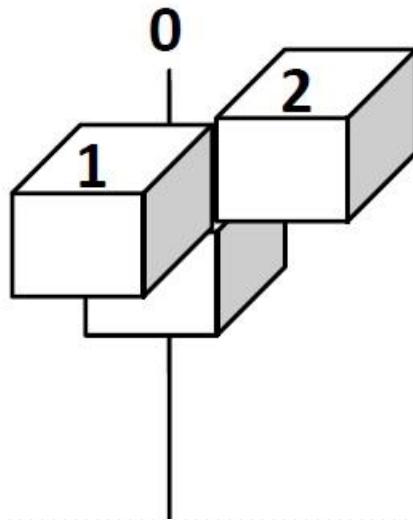
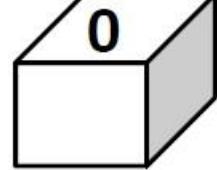
dvm run 3 1 prog



Метод гиперплоскостей. 2D



Метод гиперплоскостей. 3D



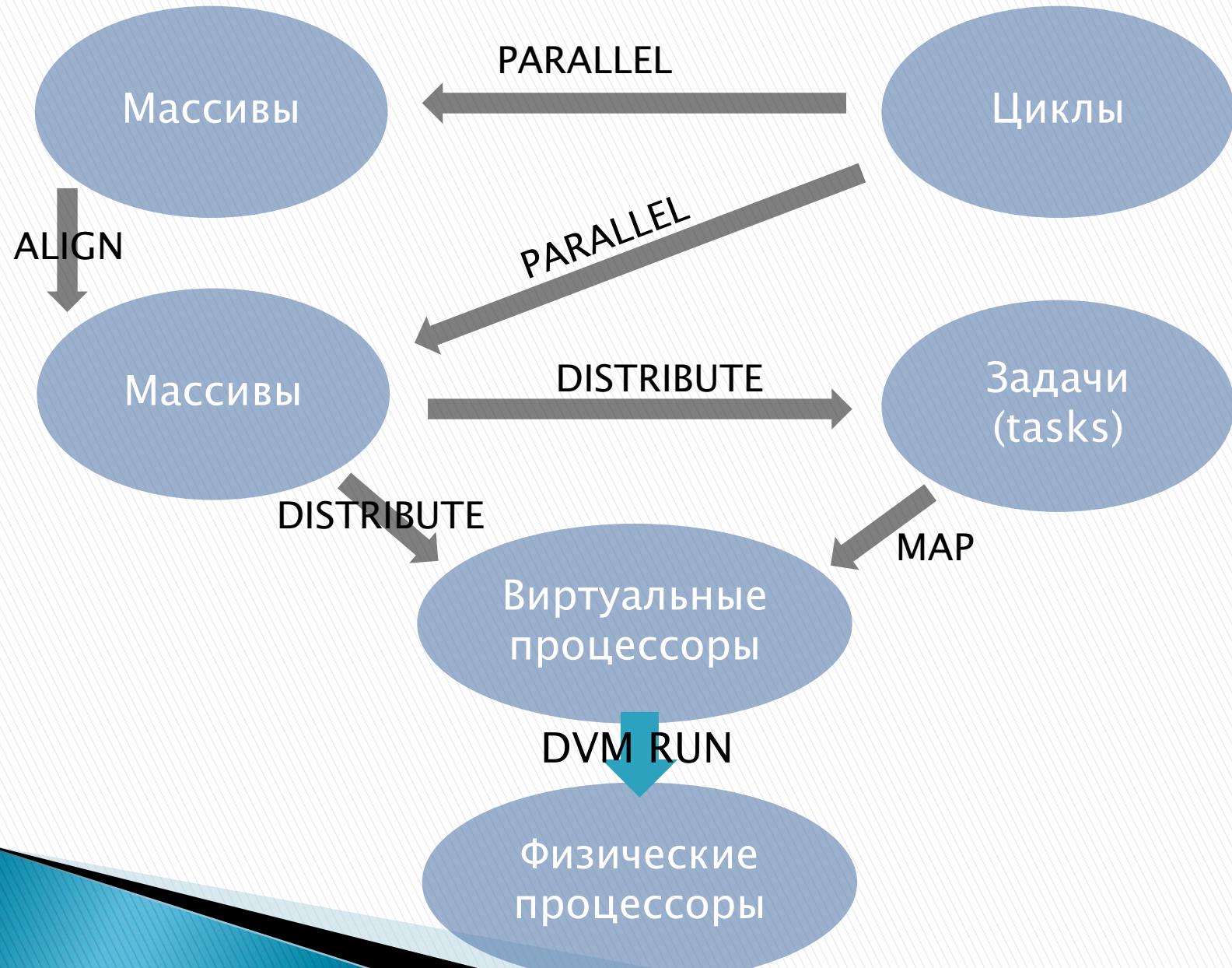
1я
плоскость

2я
плоскость

3я
плоскость

И т.д.

Отображение последовательной программы



DVMH-модель распараллеливание на узел

Распределение вычислений. REGION

- ▶ Регион выделяет часть программы (с одним входом и одним выходом) для возможного выполнения на одном или нескольких вычислительных устройствах (GPU, CPU, Xeon Phi);
- ▶ Регион может быть выполнен на любых устройствах (*спецификация targets*);
- ▶ Вложенные регионы не допускаются.

Распределение вычислений. REGION

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm region
{
#pragma dvm parallel ([i][j] on A[i][j]) reduction(max(eps))
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
}
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Распределение вычислений. REGION

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm region
{
    #pragma dvm parallel ([i][j] on A[i][j]) reduction(max(eps))
    for (i = 1; i < L - 1; i++)
        for (j = 1; j < L - 1; j++) {
            float tmp = fabs(B[i][j] - A[i][j]);
            eps = Max(tmp, eps);
            A[i][j] = B[i][j];
        }
    #pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
    for (i = 1; i < L - 1; i++)
        for (j = 1; j < L - 1; j++)
            B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
}
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Два параллельных цикла в регионе

Распределение вычислений. REGION

Спецификации для региона:

- ▶ IN – входные данные: в регионе должны быть самые последние значения этих данных;
- ▶ OUT – выходные данные: значения указанных переменных в регионе изменяются и могут быть использованы далее;
- ▶ LOCAL – локальные данные: значения указанных переменных в регионе изменяются, но эти изменения не будут использованы далее;
- ▶ INOUT, INLOCAL – сокращенная запись одновременно двух спецификаций IN и OUT/ LOCAL;
- ▶ TARGETS – указание списка типов вычислителей;

Распределение вычислений. REGION

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm region inout(A, B)
{
#pragma dvm parallel ([i][j] on A[i][j]) reduction(max(eps))
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
}
printf(" IT = %4i      EPS = %14.7E\n", it, eps);
```

Актуализация данных. ACTUAL/GET_ACTUAL

- ▶ Директива **GET_ACTUAL** **делает** все необходимые **обновления** для того, чтобы в памяти ЦПУ были актуальные (т.е. самые новые) значения данных;
- ▶ Директива **ACTUAL** **объявляет тот факт**, что указанные в списке массивы и скаляры имеют самые новые значения в памяти ЦПУ. Значения указанных переменных и элементов массивов, находящиеся в памяти ускорителей, считаются устаревшими и перед использованием **будут при необходимости обновлены**.

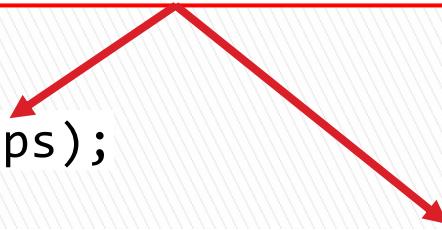
Актуализация данных. ACTUAL/GET_ACTUAL

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm region inout(A, B)
{
#pragma dvm parallel ([i][j] on A[i][j]) reduction(max(eps))
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++)
        B[i][j] = (A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j])/4;
}
#pragma dvm get_actual(eps, A)
printf(" IT = %4i    EPS = %14.7E\n", it, eps);
for (i=1; i < L-1; i++)
    for (j=1; j < L-1; j++) printf("A[%d,%d]=%f\n", i, j, A[i,j]);
```

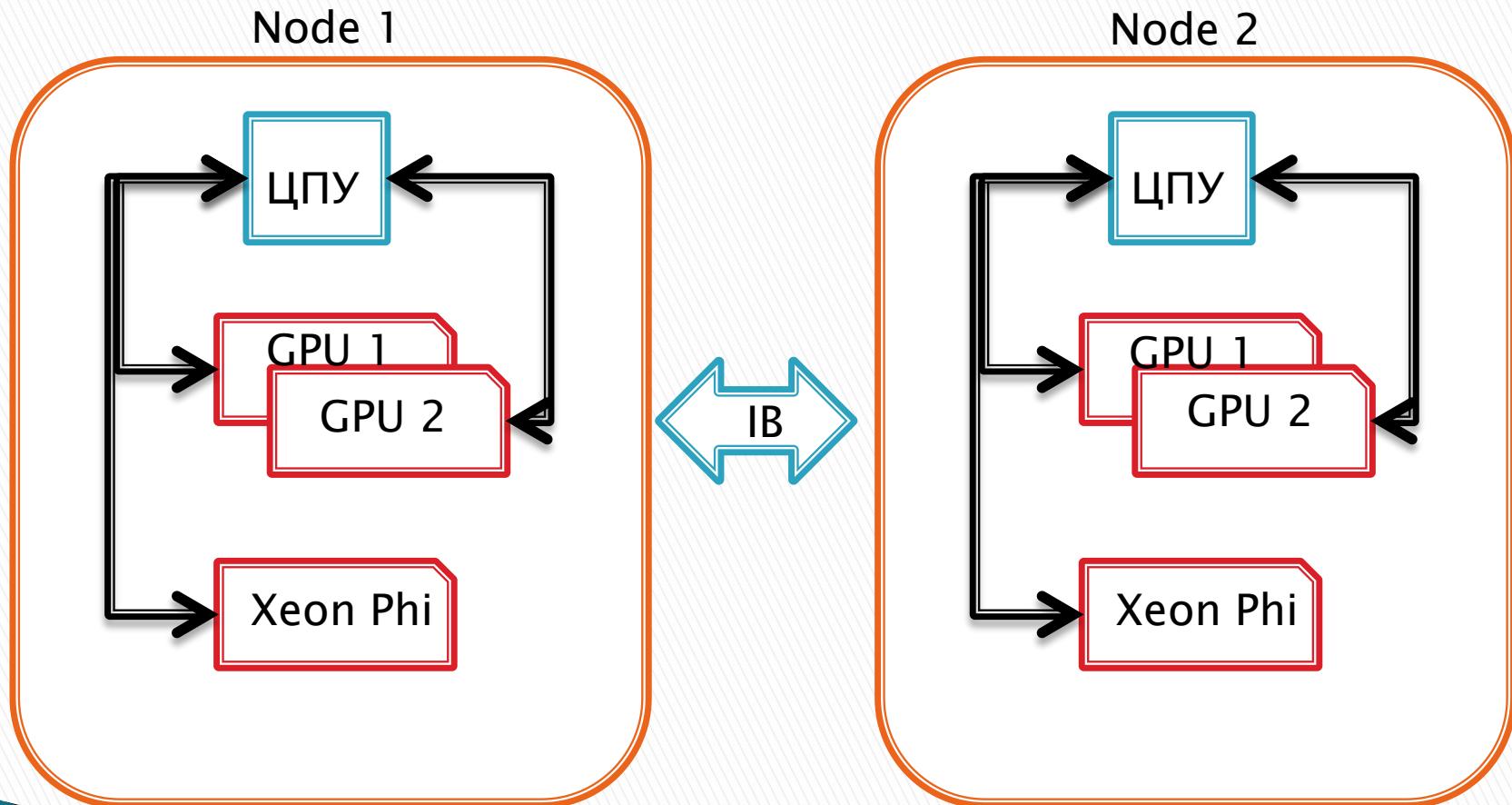
Актуализация данных. ACTUAL/GET_ACTUAL

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];
#pragma dvm region inout(A, B)
{
#pragma dvm parallel ([i][j] on A[i][j]) reduction(max(eps))
for (i = 1; i < L - 1; i++)
    for (j = 1; j < L - 1; j++) {
        float tmp = fabs(B[i][j] - A[i][j]);
        eps = Max(tmp, eps);
        A[i][j] = B[i][j];
    }
#pragma dvm parallel ([i][j] on B[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
    for (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;
#pragma dvm get_actual(eps, A)
printf(" IT = %4i EPS = %14.7E\n", it, eps);
for (i=1; i < L-1; i++)
    for (j=1; j < L-1; j++) printf("A[%d,%d]=%f\n", i, j, A[i,j]);
```

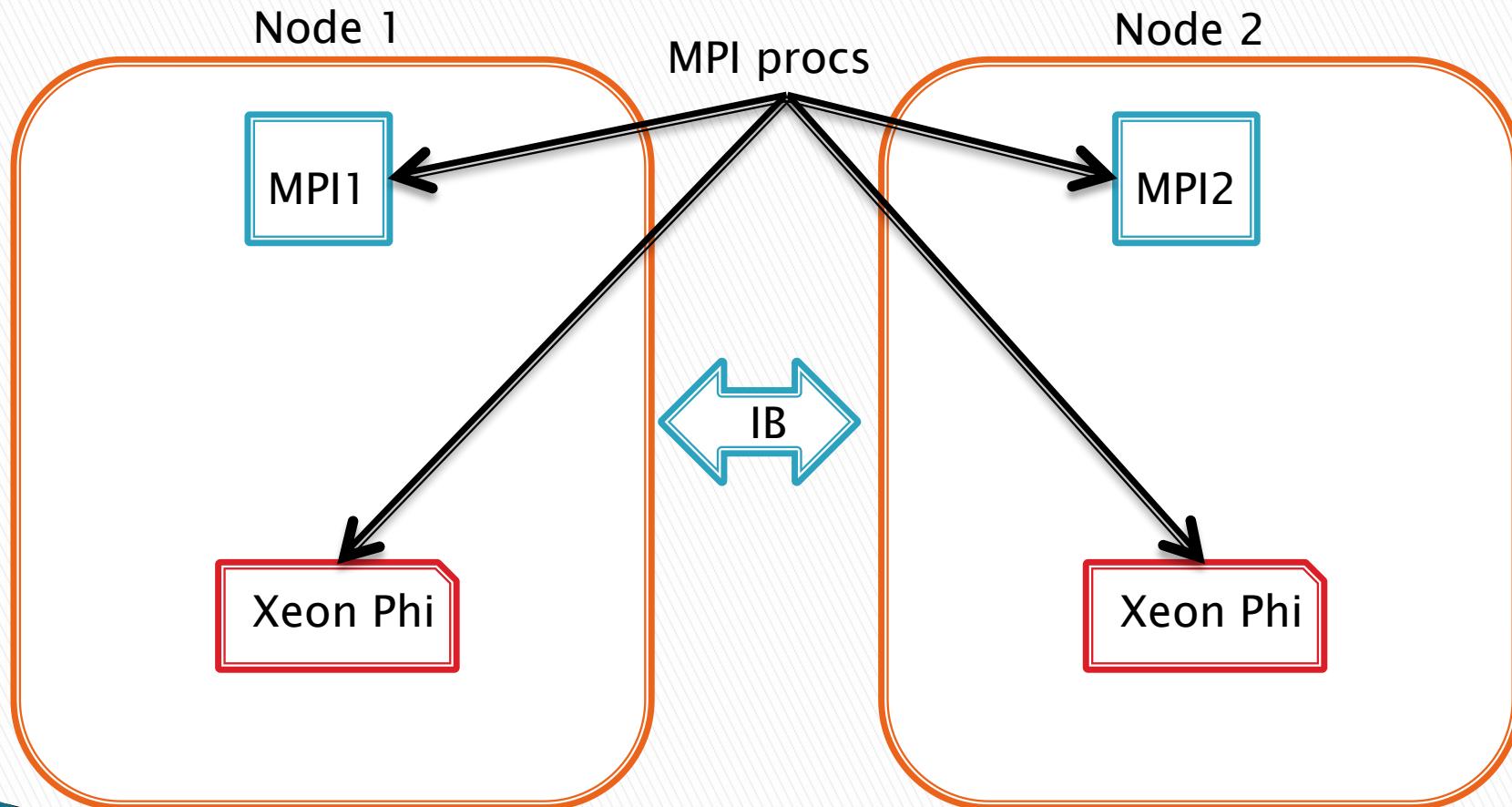
Чтение данных на хосте,
измененных в регионе



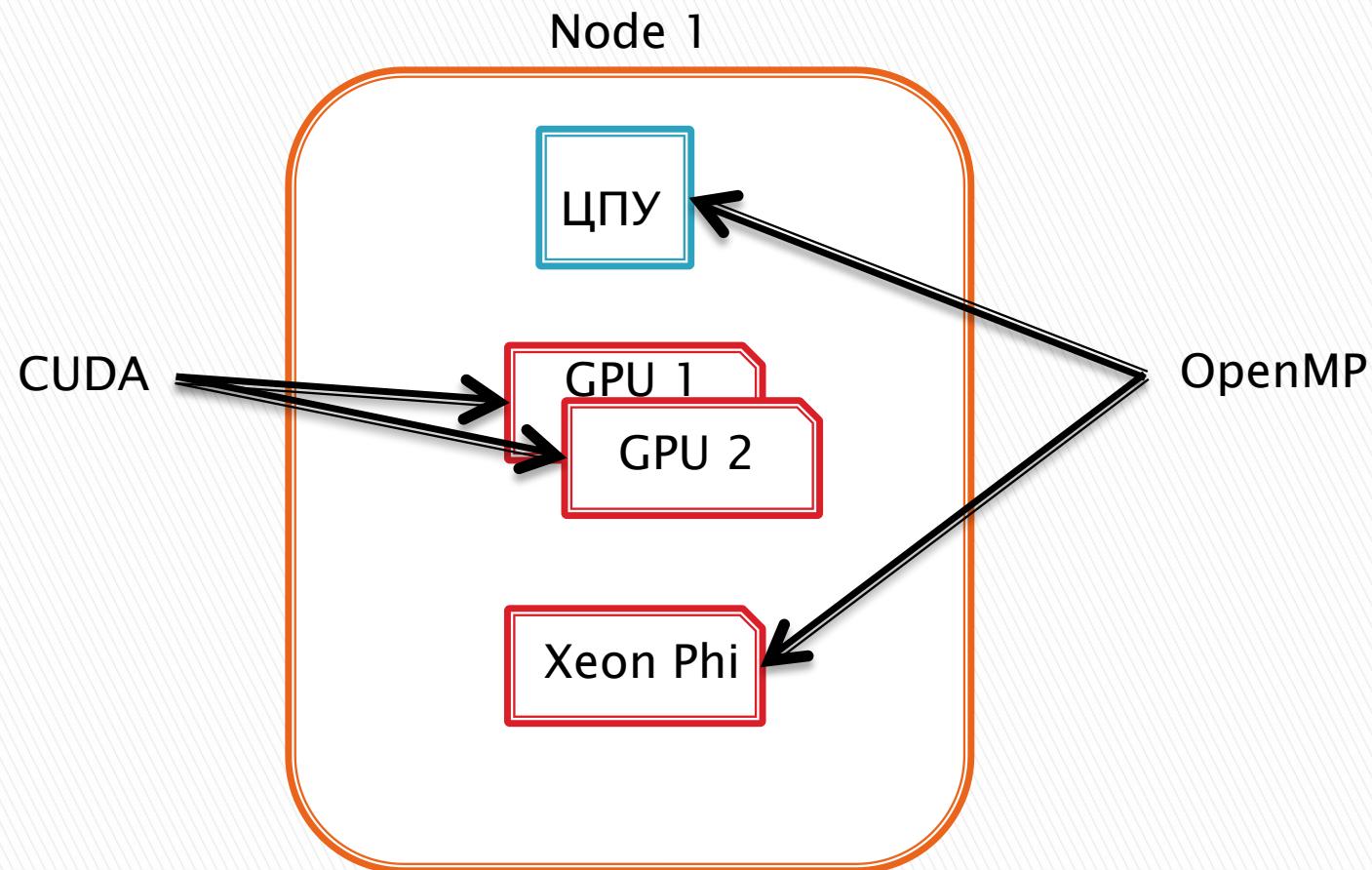
Двухуровневая балансировка в DVMH-модели.



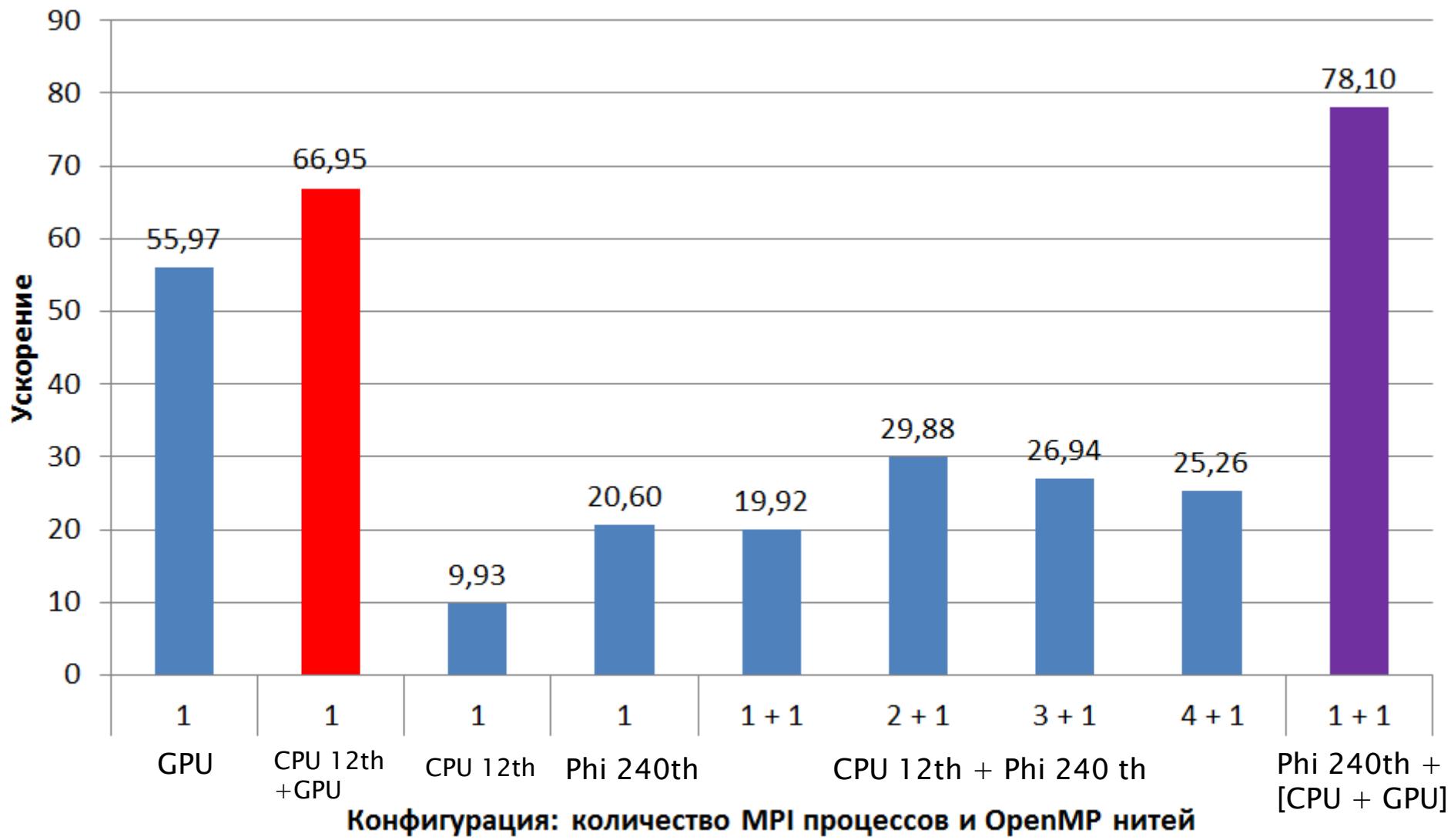
Уровень 1: отображение MPI-процессов.



Уровень 2: загрузка устройств узла



Балансировка нагрузки на примере теста EP

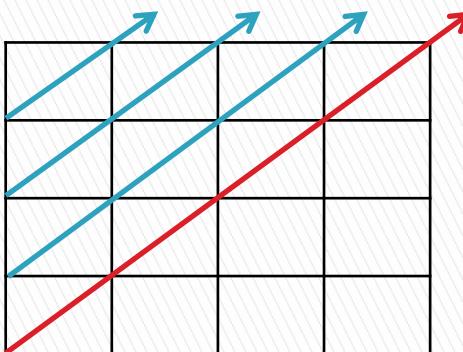


Реализация спецификации ACROSS в DVMH

Особенности реализации для GPU

- ▶ Зависимость по 1 измерению – его сериализация;
- ▶ Зависимость по 2, 3 и 4м измерениям – выполнение гиперплоскостями;
- ▶ Изменение порядка выполнения витков – изменение шаблона доступа в память;
- ▶ Переупорядочивание элементов массивов;

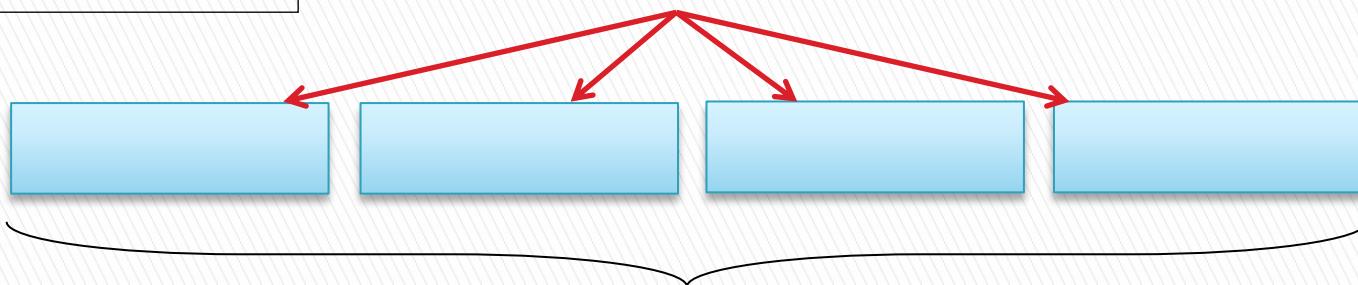
Особенности реализации для GPU



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

4 диагональ

warp load/store operations



Array A(4, 4)

Особенности реализации для GPU: динамическая транформация данных

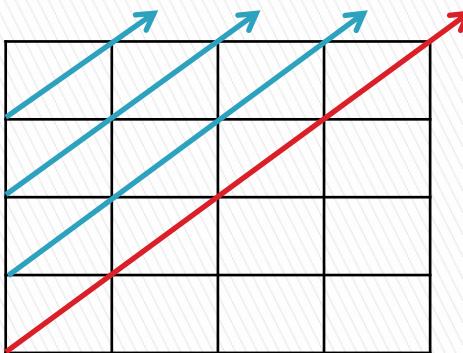
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



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

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

Особенности реализации для GPU



```
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



Array A(4, 4)

Включается опцией при
компиляции `-autoTfm`

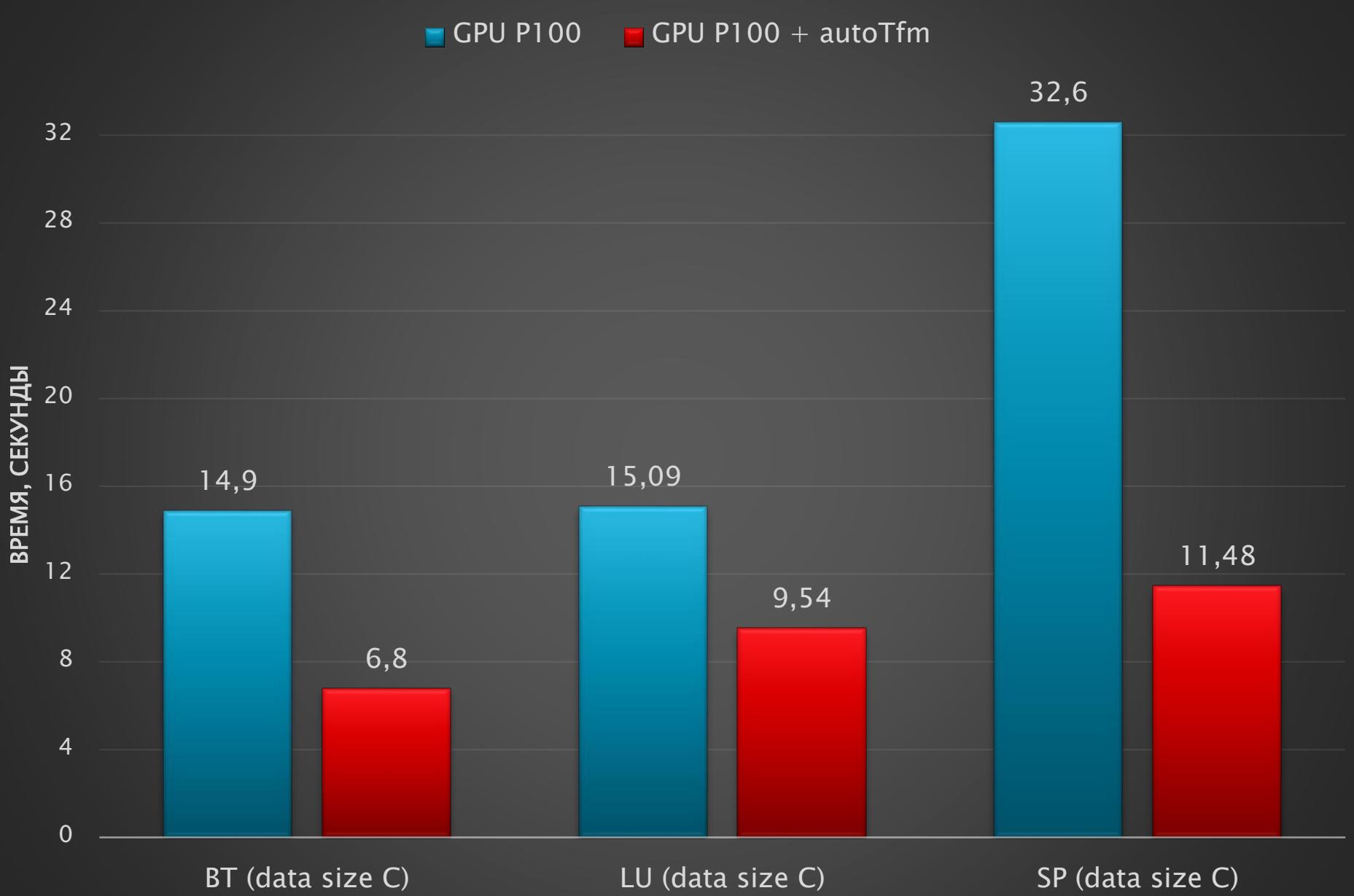
Динамическая транформация данных

- ▶ Никаких дополнительных указаний в DVMN-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.

Распараллеливание прикладных задач

- ▶ **NAS Parallel Benchmarks** — набор тестов производительности нацеленных на проверку возможностей высоко параллельных суперкомпьютеров. Последнее обновление в июле 2020 года. Всего 11 тестов.
- ▶ BT, SP и LU – решают синтетическую систему нелинейных диф. уравнений в частных производных (3-мерная система уравнений Навье — Стокса для сжимаемой жидкости или газа), используя три алгоритма:
 - блочная трёхдиагональная схема с методом переменных направлений (BT),
 - скалярная пятидиагональная схема (SP),
 - метод симметричной последовательной верхней релаксации (алгоритм SSOR, задача LU).

Распараллеливание прикладных задач



Компиляция и запуск DVMN-программ

Настройка DVM-системы

- ▶ Компиляция и запуск выполняются с помощью скрипта **dvm** (*в каталоге <path>/user/dvm*);
- ▶ Необходимо скопировать к себе скрипт **dvm** для компиляции и **dvm** и **usr.par** для запуска;
- ▶ Все дальнейшие действия производить с помощью скрипта **dvm**;
- ▶ Для настройки запуска (компиляции) необходимо изменить настройки **dvm**-скрипта.

Компиляция DVMH-программ

- ▶ Компиляция и запуск выполняются с помощью скрипта **dvm** (*в каталоге <path>/user/dvm*);

Конвертация, компиляция и линковка:

dvm c <опции> <список файлов>

Конвертация:

dvm cdv <опции> <список файлов>

Компиляция:

dvm cc <опции> <список файлов>

Линковка:

dvm clink <опции> <список файлов>

Компиляция DVMH-программ

- ▶ Компиляция выполняется с помощью скрипта **dvm** (*в каталоге <path>/user/dvm*);

dvm с <опции> <список файлов>

- **-s** – режим игнорирования всех директив;
- **-noH** – режим игнорирования директив, обеспечивающих выполнение программы на GPU;
- **-autoTfm** – режим динамического переупорядочивания массива (режим перестановки измерений массива для оптимизации доступа к глобальной памяти);
- **-noCuda** – управление процессом компиляции – компилятор не готовит выполнение регионов на CUDA-устройствах.

Компиляция DVMH-программ

```
dvmdir='/polusfs/home/kolganov.a/DVM/dvm_r8114/dvm_sys'  
export dvmbuild='Version 5.0, revision 8114, platform POLUS, build 37'  
  
----- One can customize compiler options:  
# export PCC='mpicc -g -O3 -fopenmp' # C compiler  
# export PCXX='mpic++ -g -O3 -fopenmp' # C++ compiler  
# export PFORT='mpif77 -g -O3 -fopenmp' # Fortran compiler  
# export NVCC='/usr/local/cuda/bin/nvcc -arch=sm_60 -DHAVE_EXPLICIT_CAST  
           -DPGI_COMPILE_BITCODE -DHAVE_EXPLICIT_CAST' # NVIDIA CUDA C++ compiler  
  
----- One can add libraries (additional linker flags):  
# export USER_LIBS=''
```

Опции компиляции и линковки, установленные DVM-системой по умолчанию. Их можно менять.

Запуск DVMH-программ

- ▶ Запуск выполняется с помощью скрипта **dvm** (*в каталоге <path>/user/dvm*);

dvm run [N1 [N2 [N3 [N4]]]] <exec DVMH file>

- **N1, N2, N3, N4** – размеры матрицы виртуальных процессоров (по умолчанию – 1 1 1 1);
- При запуске DVMH-программы размеры матрицы виртуальных процессоров определяют число процессов, которые будут выполняться параллельно: $\text{TOTAL_PROC} = N1 * N2 * N3 * N4$.

Запуск DVMH-программ

#----- One can set launch options:

```
export DVMH_PPN='20'          # Number of processes per node
# export DVMH_STACKSIZE=''    # Stack size to set for the task
export DVMH_NUM_THREADS='1'   # Number of CPU threads per process
export DVMH_NUM_CUDAS='0'     # Number of GPUs per process
# export DVMH_CPU_PERF=''     # Performance of all cores of CPU per process
# export DVMH_CUDAS_PERF=''   # Performance of each GPU per device
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU
# export DVMH_EXCLUSIVE=0     # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;

Запуск DVMH-программ

#----- One can set launch options:

```
export DVMH_PPN='20'          # Number of processes per node
# export DVMH_STACKSIZE=' '   # Stack size to set for the task
export DVMH_NUM_THREADS='1'    # Number of CPU threads per process
export DVMH_NUM_CUDAS='0'      # Number of GPUs per process
# export DVMH_CPU_PERF=' '     # Performance of all cores of CPU per process
# export DVMH_CUDAS_PERF=' '   # Performance of each GPU per device
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU
# export DVMH_EXCLUSIVE=0      # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ DVMH_PPN – сколько процессов запускать на узел. На Polus доступно максимум 20 процессов на узел!

Запуск DVMH-программ

#----- One can set launch options:

```
export DVMH_PPN='20'          # Number of processes per node
# export DVMH_STACKSIZE=' '   # Stack size to set for the task
export DVMH_NUM_THREADS='1'    # Number of CPU threads per process
export DVMH_NUM_CUDAS='0'      # Number of GPUs per process
# export DVMH_CPU_PERF=' '     # Performance of all cores of CPU per process
# export DVMH_CUDAS_PERF=' '   # Performance of each GPU per device
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU
# export DVMH_EXCLUSIVE=0     # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ DVMH_PPN – сколько процессов запускать на узел. На Polus доступно максимум 20 процессов на узел!
dvm run 5 1 2 + PPN=4 -> будет использовано 3 узла

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'          # Number of processes per node  
# export DVMH_STACKSIZE=''    # Stack size to set for the task  
export DVMH_NUM_THREADS='1'   # Number of CPU threads per process  
export DVMH_NUM_CUDAS='0'     # Number of GPUs per process  
# export DVMH_CPU_PERF=''    # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF=''  # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0     # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ Количество нитей и GPU в каждом процессе. По умолчанию каждый процесс НЕ использует GPU и использует 1 нить.

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'      # Number of processes per node  
# export DVMH_STACKSIZE='' # Stack size to set for the task  
export DVMH_NUM_THREADS='1' # Number of CPU threads per process  
export DVMH_NUM_CUDAS='0'   # Number of GPUs per process  
# export DVMH_CPU_PERF=''  # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF='' # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0    # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ Количество нитей и GPU в каждом процессе. По умолчанию каждый процесс НЕ использует GPU и использует 1 нить;
- ▶ Конфигурация нитей и GPU может сильно влиять на производительность.

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'          # Number of processes per node  
# export DVMH_STACKSIZE=''    # Stack size to set for the task  
export DVMH_NUM_THREADS='8'   # Number of CPU threads per process  
export DVMH_NUM_CUDAS='0'     # Number of GPUs per process  
# export DVMH_CPU_PERF=''    # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF=''  # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0     # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ Использовать 8 нитей на процесс и 0 GPU

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'          # Number of processes per node  
# export DVMH_STACKSIZE=''    # Stack size to set for the task  
export DVMH_NUM_THREADS='0'    # Number of CPU threads per process  
export DVMH_NUM_CUDAS='1'      # Number of GPUs per process  
# export DVMH_CPU_PERF=''     # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF=''   # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0     # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ Использовать 0 нитей на процесс и 1 GPU – считать только на GPU.

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'          # Number of processes per node  
# export DVMH_STACKSIZE=''    # Stack size to set for the task  
export DVMH_NUM_THREADS='0'    # Number of CPU threads per process  
export DVMH_NUM_CUDAS='2'      # Number of GPUs per process  
# export DVMH_CPU_PERF=''     # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF=''   # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0     # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ Использовать 0 нитей на процесс и 2 GPU – считать только на 2x GPU **в каждом процессе.**

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'      # Number of processes per node  
# export DVMH_STACKSIZE='' # Stack size to set for the task  
export DVMH_NUM_THREADS='8' # Number of CPU threads per process  
export DVMH_NUM_CUDAS='1'   # Number of GPUs per process  
# export DVMH_CPU_PERF=''  # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF='' # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0    # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ *_PERF – задавать соотношение производительности GPU и нитей CPU. По умолчанию 50% на 50%.

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'      # Number of processes per node  
# export DVMH_STACKSIZE='' # Stack size to set for the task  
export DVMH_NUM_THREADS='8' # Number of CPU threads per process  
export DVMH_NUM_CUDAS='1'   # Number of GPUs per process  
# export DVMH_CPU_PERF=''  # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF='' # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0    # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ *_PERF – задавать соотношение производительности GPU и нитей CPU. По умолчанию 50% на 50%;
- ▶ Производительность задается как доля CUDAS_{perf} к доле NUM_THREADS_{perf}

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'      # Number of processes per node  
# export DVMH_STACKSIZE=' ' # Stack size to set for the task  
export DVMH_NUM_THREADS='8' # Number of CPU threads per process  
export DVMH_NUM_CUDAS='1'   # Number of GPUs per process  
export DVMH_CPU_PERF='0.2'  # Performance of all cores of CPU per process  
export DVMH_CUDAS_PERF='0.8' # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0    # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ В каждом процессе 8 нитей и 1 GPU, с соотношением производительности 20% (8 нитей) и 80% GPU;
- ▶ С таким соотношением будет выполняться распределение работы в циклах между GPU и CPU.

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'      # Number of processes per node  
# export DVMH_STACKSIZE='' # Stack size to set for the task  
export DVMH_NUM_THREADS='8' # Number of CPU threads per process  
export DVMH_NUM_CUDAS='1'   # Number of GPUs per process  
# export DVMH_CPU_PERF=''  # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF='' # Performance of each GPU per device  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0    # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ Ручной вариант подбора *_PERF – запустить только на 8 нитях, запустить только на 1 GPU, использовать полученные времена для вычисления средних *_PERF.

Запуск DVMH-программ

```
#----- One can set launch options:  
export DVMH_PPN='20'      # Number of processes per node  
# export DVMH_STACKSIZE='' # Stack size to set for the task  
export DVMH_NUM_THREADS='8' # Number of CPU threads per process  
export DVMH_NUM_CUDAS='1'   # Number of GPUs per process  
# export DVMH_CPU_PERF=''  # Performance of all cores of CPU per process  
# export DVMH_CUDAS_PERF='' # Performance of each GPU per device  
  
# export DVMH_NO_DIRECT_COPY=0 # Use standard cudaMemcpy functions instead of direct copying with GPU  
# export DVMH_EXCLUSIVE=0     # Use exclusive running mode on LSF
```

- ▶ Данные переменные окружения позволяют выполнить отображение запущенной программы на физические ресурсы кластера;
- ▶ Дополнительные опции. Например, использовать стандартные функции копирования или CUDA-ядра;
- ▶ Использовать эксклюзивный запуск на узле или в режиме разделения ресурсов!

Другие опции доступны по выдаче **dvm help**.

Отладка DVMH-программ

- ▶ Создание лога системы поддержки при выполнении DVMH-программы;

```
#----- Debugging options:  
# export DVMH_LOGLEVEL=3 # Levels of debugging: 1-errors only, 2-warning, 3-info, 4-debug, 5-trace  
# export DVMH_LOGFILE='dvmh %d.log' # Log file name for each process  
  
# export DVMH_COMPARE_DEBUG=0 # An alternative way to turn comparative debugging mode on  
# export dvmsave=0 # Save convertation results  
# export dvmshow=0 # Show commands executed by the DVM driver  
  
export dvmcopy=1  
  
exec "$dvmdir/bin/dvm_drv" "$@"
```

Отладка DVMH-программ

- ▶ Создание лога системы поддержки при выполнении DVMH-программы;
- ▶ Выполнение программы в режиме сравнения CPU и GPU в конце каждого региона. Аналог dvm cmph <prog>;

```
#----- Debugging options:  
# export DVMH_LOGLEVEL=3 # Levels of debugging: 1-errors only, 2-warning, 3-info, 4-debug, 5-trace  
# export DVMH_LOGFILE='dvmh %d.log' # Log file name for each process  
# export DVMH_COMPARE_DEBUG=0 # An alternative way to turn comparative debugging mode on  
# export dvmsave=0 # Save convertation results  
# export dvmshow=0 # Show commands executed by the DVM driver  
export dvmcopy=1  
exec "$dvmdir/bin/dvm_drv" "$@"
```

Отладка DVMH-программ

- ▶ Создание лога системы поддержки при выполнении DVMH-программы;
- ▶ Выполнение программы в режиме сравнения CPU и GPU в конце каждого региона. Аналог dvm cmph <prog>;
- ▶ Сохранение промежуточных файлов конвертации, а также отображение хода компиляции и линковки.

```
#----- Debugging options:  
# export DVMH_LOGLEVEL=3 # Levels of debugging: 1-errors only, 2-warning, 3-info, 4-debug, 5-trace  
# export DVMH_LOGFILE='dvmh_%d.log' # Log file name for each process  
# export DVMH_COMPARE DEBUG=0 # An alternative way to turn comparative debugging mode on  
# export dvmsave=0 # Save convertation results  
# export dvmshow=0 # Show commands executed by the DVM driver  
  
export dvmcopy=1  
exec "$dvmdir/bin/dvm_drv" "$@"
```

Отладка производительности DVMН-программ

- ▶ После запуска программы через «`dvm run prog`» по завершении программы появляется файл со статистикой выполнения «`prog.sts.gz+`»;
- ▶ С помощью команды «`dvm pa prog.sts.gz+ out.txt`» можно получить информацию о работе программы;
- ▶ Есть возможность расстановки системы интервалов в коде программы с помощью директив

`#pragma dvm interval (expr)`

`#pragma dvm endinterval`

Отладка производительности DVMH-программ

```
#pragma dvm array distribute[block][block]
float B[L][L];
#pragma dvm array align([i][j] with B[i][j]), shadow[1:1][1:1]
float A[L][L];

#pragma dvm interval 2
#pragma dvm region inout(A, B)
{
    #pragma dvm parallel ([i][j] on A[i][j])
    for (i = 1; i < L - 1; i++)
        for (j = 1; j < L - 1; j++) {
            float tmp = fabs(B[i][j] - A[i][j]);
            eps = Max(tmp, eps);
            A[i][j] = B[i][j];
        }
}
#pragma dvm endinterval
```

Отладка производительности DVMH-программ

```
#pragma dvm array distribute[block][block]
```

```
float B[L][L];
```

```
#pragma dvm array
```

```
float A[L][L];
```

Интервал для сбора статистики. Может состоять из последовательного и параллельного кода.

```
#pragma dvm interval 2
```

```
#pragma dvm region inout(A, B)
```

```
{
```

```
#pragma dvm parallel ([i][j] on A[i][j])
```

```
for (i = 1; i < L - 1; i++)
```

```
    for (j = 1; j < L - 1; j++) {
```

```
        float tmp = fabs(B[i][j] - A[i][j]);
```

```
        eps = Max(tmp, eps);
```

```
        A[i][j] = B[i][j];
```

```
}
```

```
}
```

```
#pragma dvm endinterval
```

Отладка производительности DVMH-программ

► Общая статистика

```
Processor system=1*1*1*1
Statistics has been accumulated on DVM-system version 5.0, platform POLUS
Analyzer is executing on DVM-system version 5.0, platform POLUS
-----
INTERVAL ( NLINE=17 SOURCE=bt.fdv ) LEVEL=0  EXE_COUNT=1
--- The main characteristics ---
Parallelization efficiency      1.0000
Execution time                  15.4326
Processors                      1
Threads amount                  1
Total time                      15.4326
Productive time                 15.4325 ( CPU= 15.4172 Sys= 0.0149 I/O= 0.0005 )
Lost time                        0.0000
  Communication                  0.0000 ( Real_sync= 0.0000 Starts= 0.0000 )
Productive time GPU              14.9914
Lost time GPU                    0.1263
          |   |   |   | Nop   Communic
I/O                           42     0.0000
Reduction                      2     0.0000
--- The comparative characteristics ---
          |   |   |   |   |   | Tmin  N proc    Tmax  N proc    Tmid
Lost time                      0.0000   1     0.0000   1     0.0000
Communication                   0.0000   1     0.0000   1     0.0000
Execution time                  15.4326   1     15.4326   1     15.4326
User CPU time                   15.4172   1     15.4172   1     15.4172
Sys. CPU time                  0.0149   1     0.0149   1     0.0149
I/O time                       0.0005   1     0.0005   1     0.0005
Productive time GPU             14.9914   1     14.9914   1     14.9914
Lost time GPU                   0.1263   1     0.1263   1     0.1263
Processors                      1     1     1     1     1
          |   |   |   |   |   | Communic
Reduction           Tmin   0.0000   1
Reduction           Tmax   0.0000   1
Reduction           Tmid   0.0000
--- The execution characteristics ---
          |   |   |   |   |   |   |
Lost time                      0.0000
Communication                   0.0000
Execution time                  15.4326
User CPU time                   15.4172
Sys. CPU time                  0.0149
I/O time                       0.0005
Productive time GPU             14.9914
Lost time GPU                   0.1263
Processors                      1
          |   |   |   |   |   |   | Communication
Reduction           0.0000
```

Отладка производительности DVMH-программ

▶ Общая статистика для GPU

--- The GPU characteristics ---

Proc: #1							
GPU #1 (Tesla P100-SXM2-16GB)	#	Min	Max	Sum	Average	Productive	Lost
[Region IN] Copy CPU to GPU	4	12B	165.205M	330.410M	82.603M	0.0204s	-
Loop execution	1023	0.0009	0.0381	14.9709	0.0146	14.9709s	-
Reduction	10	0.0001	0.0001	0.0007	0.0001	-	0.0007s
Page lock host memory	30	0.0001	0.0614	0.1256	0.0042	-	0.1256s
Productive time:	14.9914s						
Lost time :	0.1263s						

Отладка производительности DVMH-программ

▶ Статистика для интервала

```
INTERVAL ( NLINE=107 SOURCE=bt.fdv ) LEVEL=1 USER EXE_COUNT=200 EXPR=1
--- The main characteristics ---
Parallelization efficiency      1.0000
Execution time                  1.7632
Processors                      1
Threads amount                  1
Total time                      1.7632
Productive time                 1.7632 ( CPU= 1.7603 Sys= 0.0029 I/O= 0.0000 )
Productive time GPU              1.6869
```

```
--- The comparative characteristics ---
| | | | | Tmin N proc   Tmax N proc   Tmid
Execution time                  1.7632   1   1.7632   1   1.7632
User CPU time                   1.7603   1   1.7603   1   1.7603
Sys. CPU time                  0.0029   1   0.0029   1   0.0029
Productive time GPU             1.6869   1   1.6869   1   1.6869
Processors                      1   1   1   1   1
```

```
--- The execution characteristics ---
| | | | | 1
Execution time                  1.7632
User CPU time                   1.7603
Sys. CPU time                  0.0029
Productive time GPU             1.6869
Processors                      1
```

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

Proc: #1	#	Min	Max	Sum	Average	Productive	Lost
GPU #1 (Tesla P100-SXM2-16GB)	400	0.0038	0.0047	1.6869	0.0042	1.6869s	-
Loop execution	400	0.0038	0.0047	1.6869	0.0042	1.6869s	-
Productive time:	1.6869s						
Lost time :	0.0000s						

Отладка производительности DVMN-программ

▶ Статистика для интервала

```
INTERVAL ( NLINE=107 SOURCE=bt.fdv ) LEVEL=1 USER EXE_COUNT=200 EXPR=1
--- The main characteristics ---
Parallelization efficiency      1.0000
Execution time                  1.7632
Processors                      1
Threads amount                  1

```

Уровень в
иерархии
вложенности

```
Processors                      1 1
--- The execution characteristics ---
| | |
| | 1
Execution time                  1.7632
User CPU time                   1.7603
Sys. CPU time                  0.0029
Productive time GPU            1.6869
Processors                      1
```

```
.7603 Sys= 0.0029 I/O= 0.0000 )
Tmax N proc          Tmid
7632   1           1.7632
7603   1           1.7603
0029   1           0.0029
6869   1           1.6869

```

Выражение при
директиве
interval

Количество
входов

--- The GPU characteristics ---

Proc: #1							
GPU #1 (Tesla P100-SXM2-16GB)							
#	Min	Max	Sum	Average	Productive	Lost	
Loop execution	400	0.0038	0.0047	1.6869	0.0042	1.6869s	-
Productive time:	1.6869s						
Lost time :	0.0000s						

Вопросы?

сайт системы – www.dvm-system.org

Описание CDVMH и FDVMH

Примеры программ на CDVMH и FDVMH

<http://dvm-system.org/ru/#dvm-docs>

На Polus система доступна:

module load OpenMPI

/polusfs/home/kolganov.a/DVM/dvm_r8114/dvm_sys/