Автоматизация разработки параллельных программ

Бахтин Владимир Александрович
Доцент кафедры системного программированния факультета ВМК, МГУ им. М. В. Ломоносова
к.ф.-м.н., вед. научный сотрудник Института прикладной математики им М.В.Келдыша РАН
bakhtin@keldysh.ru

Спецсеминар «Операционные системы и языки программирования распределенных вычислительных систем»

МГУ им. М.В. Ломоносова, Москва, 2019 г.

Содержание

- Современные направления развития параллельных вычислительных систем
- Технологии параллельного программирования
- Система Автоматизированной Параллелизации ФОРтран-программ (**САПФОР**)

Крюков Виктор Алексеевич
Профессор кафедры системного программирования факультета ВМК МГУ им. М.В. Ломоносова
д.ф.-м.н.,главный научный сотрудник Института прикладной математики им М.В.Келдыша РАН
krukov@keldysh.ru

Тенденции развития современных процессоров

В течение нескольких десятилетий развитие ЭВМ сопровождалось удвоением их быстродействия каждые 1.5-2 года. Это обеспечивалось и повышением тактовой частоты и совершенствованием архитектуры (параллельное и конвейерное выполнение команд).

Узким местом стала оперативная память. Знаменитый закон Мура, так хорошо работающий для процессоров, совершенно не применим для памяти, где скорости доступа удваиваются в лучшем случае каждые 5-6 лет.

Совершенствовались системы кэш-памяти, увеличивался объем, усложнялись алгоритмы ее использования.

Для процессора Intel Itanium:

Latency to L1: 1-2 cycles

Latency to L2: 5 - 7 cycles

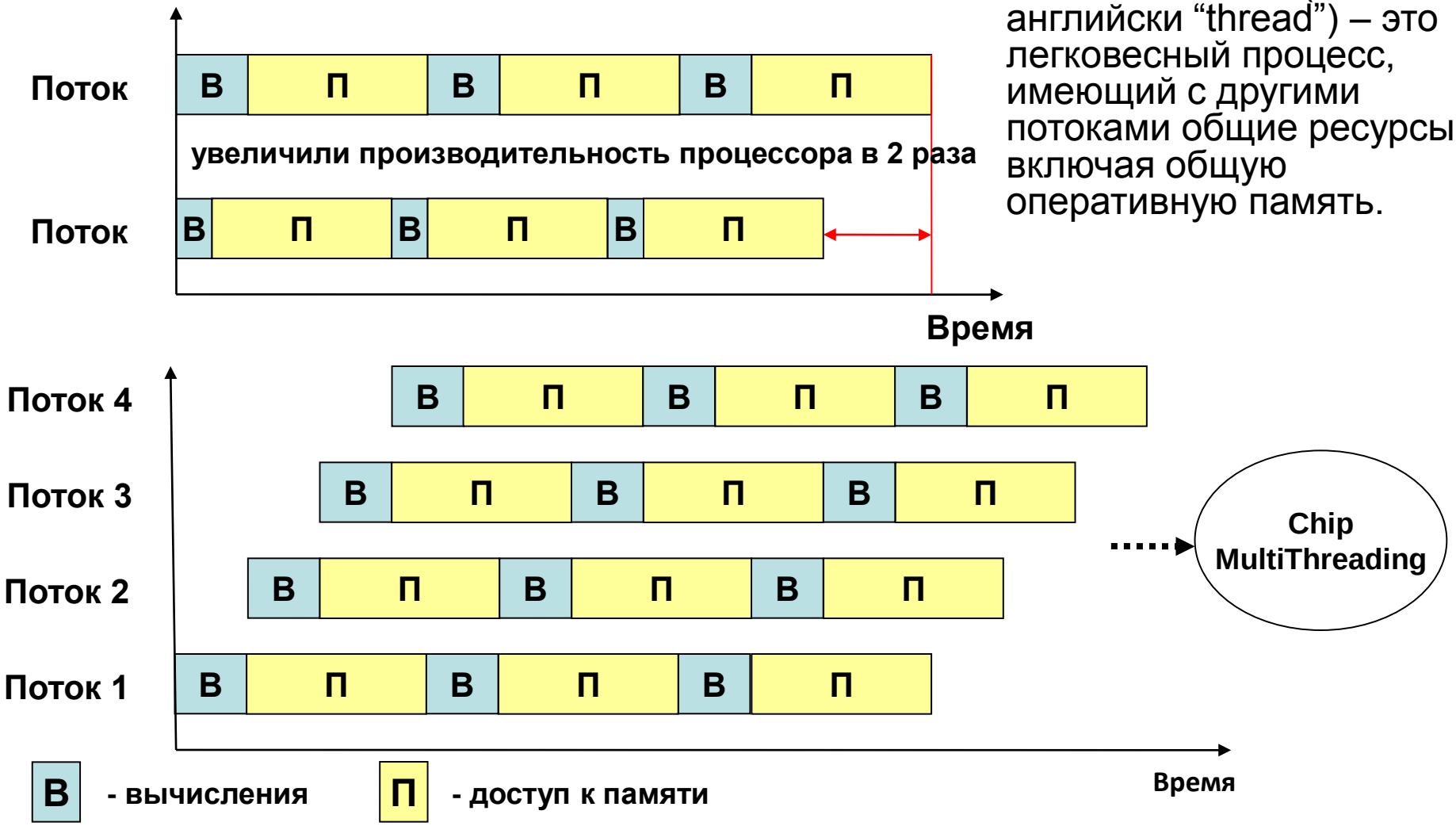
Latency to L3: 12 - 21 cycles

Latency to memory: 180 – 225 cycles

Важным параметром становится - **GUPS** (Giga Updates Per Second)

Тенденции развития современных процессоров

Поток или **нить** (по-английски “thread”) – это легковесный процесс, имеющий с другими потоками общие ресурсы, включая общую оперативную память.



TOP 500
The List.

PRESENTED BY

ICL INNOVATIVE COMPUTING LABORATORY

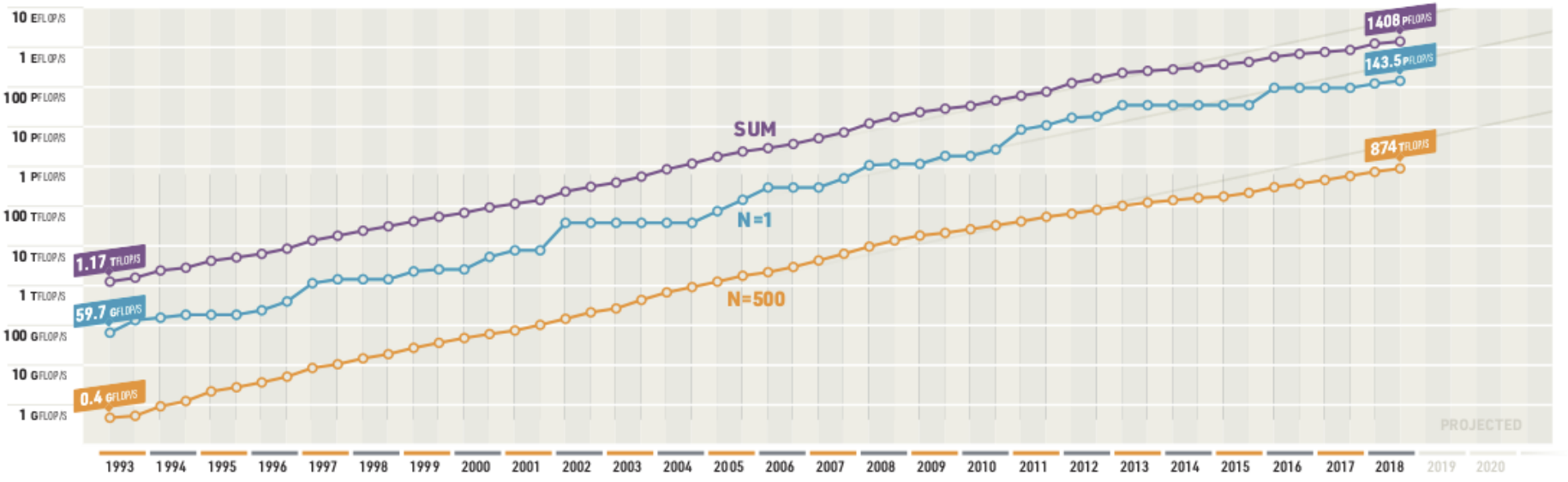
BERKELEY LAB Lawrence Berkeley National Laboratory

ISC GROUP Moving Forward.

FIND OUT MORE AT
top500.org

		SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	**Summit**	IBM POWER9 (22C, 3.07 GHz), NVIDIA Volta GV100 (80C), Dual-rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,282,544	143.5	11.1
2	**Sierra**	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-rail Mellanox EDR Infiniband	DOE/NNSA/LLNL	USA	1,572,480	94.6	7.44
3	**Sunway TaihuLight**	Shenwei SW26010 (260C 1.45 GHz) Custom interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
4	**Tianhe-2A (Milkyway-2A)**	Intel Ivy Bridge (12C 2.2 GHz) & TH Express-2, Matrix-2000	NSCC Guangzhou	China	4,981,760	61.4	18.5
5	**Piz Daint**	Cray XC50, Xeon E5-2690v3 (12C 2.6GHz), Aries interconnect , NVIDIA Tesla P100	CSCS	Switzerland	319,424	21.2	2.38

PERFORMANCE DEVELOPMENT



Современные суперкомпьютерные системы

№ 16 в Top 500

Суперкомпьютер K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect

- Пиковая производительность - 11280 TFlop/s
- Число ядер в системе — 705 024
- Производительность на Linpack - 10510 TFlop/s (93.17 % от пиковой)
- Энергопотребление комплекса - **12659.89 кВт**

Современные суперкомпьютерные системы

№ 3 в Top 500

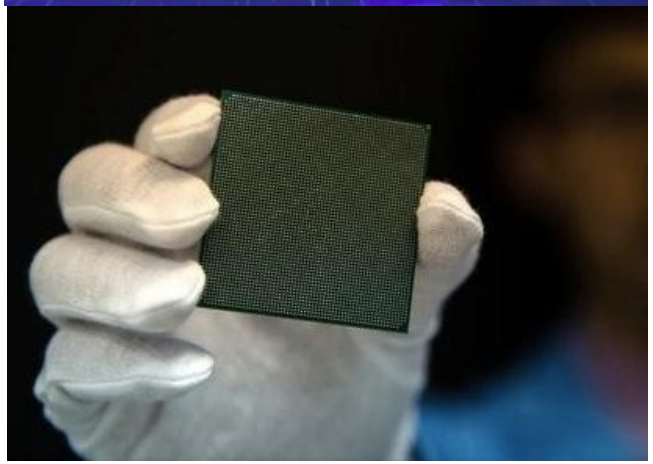
Суперкомпьютер Sunway TaihuLight, Sunway MPP, SW26010 260C 1.45GHz, Custom interconnect

- Пиковая производительность – 125435.9 TFlop/s
- Число ядер в системе — 10 649 600
- Производительность на Linpack – 93014.5 TFlop/s (74.15 % от пиковой)
- Энергопотребление комплекса **- 15371.0 кВт**

Важным параметром становится – **Power Efficency (Megaflops/watt)**

Как добиться максимальной производительности на Ватт => Chip MultiProcessing, многоядерность.

Тенденции развития современных процессоров



ShenWei SW26010

64-разрядный RISC-процессор с поддержкой SIMD-инструкций и внеочередным исполнением команд

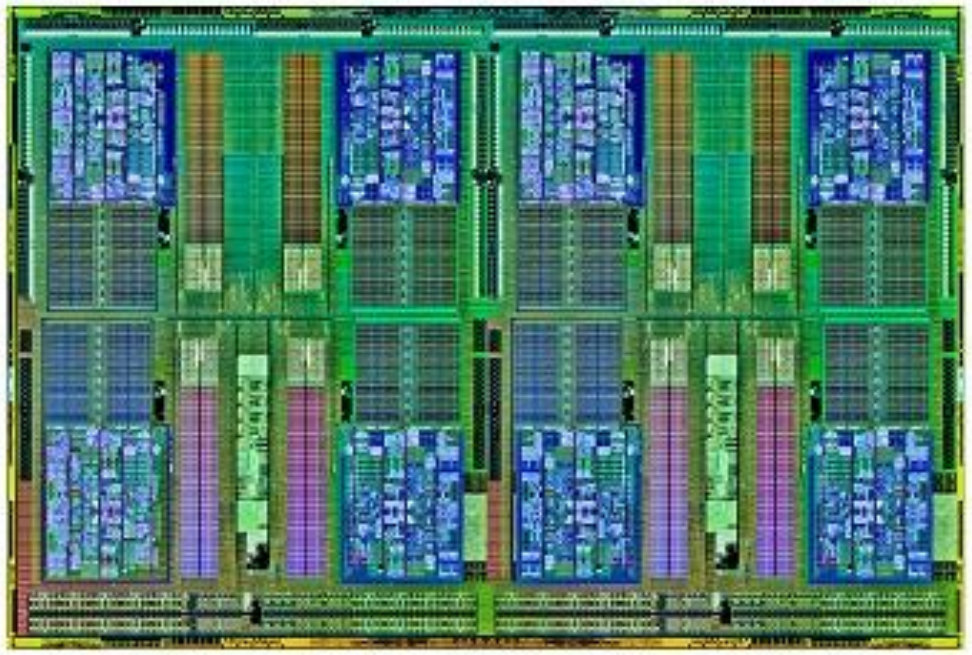
Изготовлен по схеме, предусматривающей использование четырех кластеров с 64 вычислительными ядрами (CPE) и одним управляющим ядром (MPE) в каждом.

В каждом кластере также имеется собственный контроллер памяти, суммарная пропускная способность на один процессорный разъем достигает 136,5 ГБ/с.

На каждое ядро выделено 12 КБ кэш-памяти инструкций и 64 КБ кэш-памяти данных.

Рабочая частота процессора - 1,45 ГГц.

Тенденции развития современных процессоров



AMD Opteron серии 6300

6380 SE 16 ядер @ 2,5 ГГц, 16 МБ L3 Cache

6348 12 ядер @ 2,8 ГГц, 16 МБ L3 Cache

6328 8 ядер @ 3,2 ГГц, 16 МБ L3 Cache

6308 4 ядра @ 3,5 ГГц, 16 МБ L3 Cache

технология AMD Turbo CORE

встроенный контроллер памяти (4 канала памяти DDR3)

4 канала «точка-точка» с использованием HyperTransort 3.0

Тенденции развития современных процессоров

Intel Xeon Processor серии E5

E5-2699 v4 (55M Cache, 2.20 GHz) 22 ядра, 44 нити

E5-2698 v4 (50M Cache, 2.20 GHz) 20 ядер, 40 нитей

E5-2697 v4 (45M Cache, 2.30 GHz) 18 ядер, 36 нитей

E5-2697A v4 (40M Cache, 2.60 GHz) 16 ядер, 32 нити

E5-2667 v4 (25M Cache, 3.20 GHz) 8 ядер, 16 нитей

Intel® Turbo Boost

Intel® Hyper-Threading

Intel® Intelligent Power

Intel® QuickPath

Тенденции развития современных процессоров

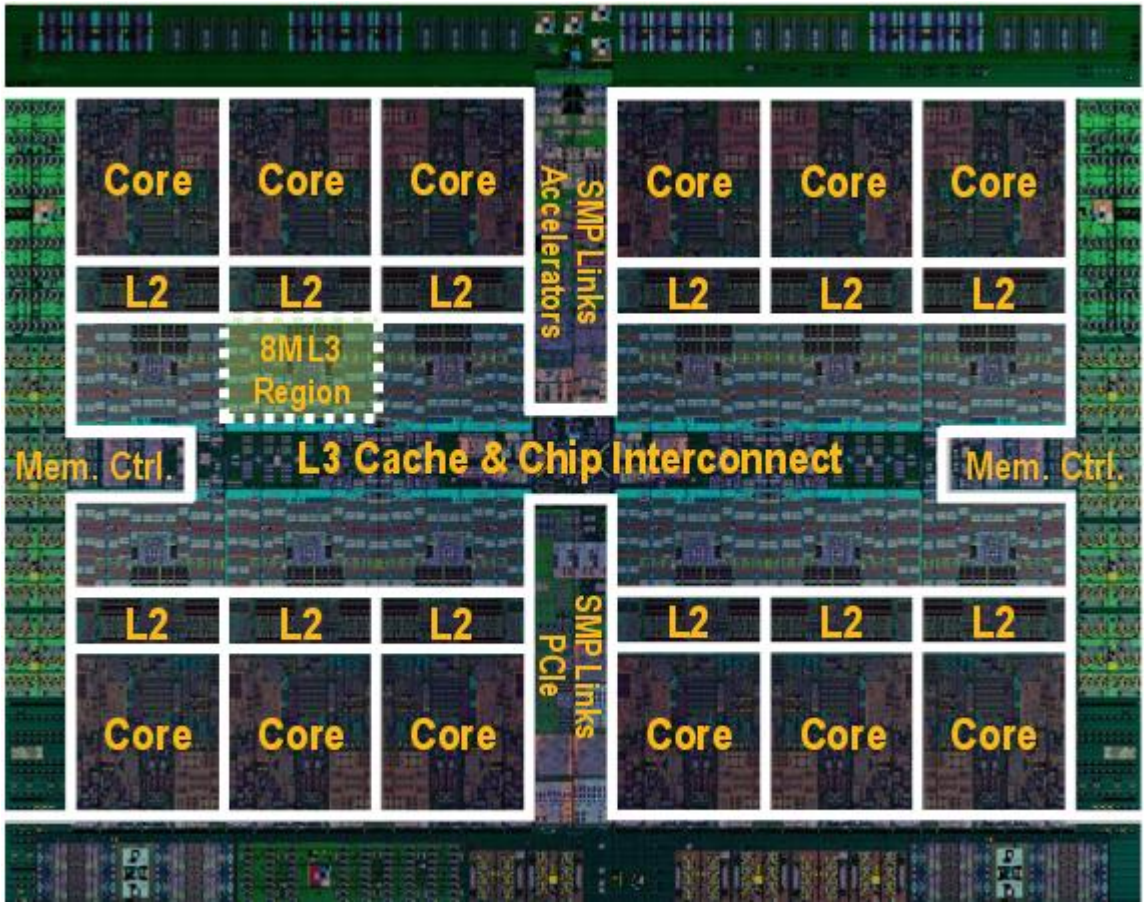
Intel Core i7 High End Desktop Processors

i7-6950X Extreme Edition (25M Cache, up to 3.50 GHz),
10 ядер, 20 нитей

i7-6900K Processor (20M Cache, up to 3.70 GHz),
8 ядер, 16 нитей

i7-6850K Processor (15M Cache, up to 3.80 GHz),
6 ядер, 12 нитей

Тенденции развития современных процессоров



IBM Power8

- 2,75 – 4,2 ГГц
- 12 ядер x 8 нитей Simultaneuos MultiThreading
- 64 КБ Data Cache + 32КБ instruction Cache
- L2 512 КБ
- L3 96 МБ

www.idh.ch/IBM_TU_2013/Power8.pdf

Тенденции развития современных процессоров

IBM Power9

- 2,75 – 4,2 ГГц
- 12 ядер x 8 нитей

  24 ядра x 4 нити
- L2 512 КБ
- L3 120 МБ (10 МБ на 2 ядра)

Тенденции развития современных процессоров

Количество ядер	8
Кэш-память 2го уровня	8 * 512 КБ
Кэш-память 3го уровня	16 МБ
Рабочая частота	1.3 ГГц
Производительность	~250 ГФлопс
Тип контроллеров памяти	DDR3-1600
Кол-во контроллеров памяти	4
Поддержка многопроцессорных систем	До 4 процессоров
Каналы межпроцессорного обмена (пропускная способность)	3 (16 ГБ/с)
Технологический процесс	28 нм
Площадь кристалла	350 кв. мм
Рассеиваемая мощность на уровне	60 – 90 Вт

Тенденции развития современных процессоров

- Темпы уменьшения латентности памяти гораздо ниже темпов ускорения процессоров + прогресс в технологии изготовления кристаллов => CMT (Chip MultiThreading)
- Опережающий рост потребления энергии при росте тактовой частоты + прогресс в технологии изготовления кристаллов => CMP (Chip MultiProcessing, многоядерность)
- И то и другое требует более глубокого распараллеливания для эффективного использования аппаратуры

Содержание

- Современные направления развития параллельных вычислительных систем
- Технологии параллельного программирования
- Система Автоматизированной Параллелизации ФОРтран-программ (**САПФОР**)

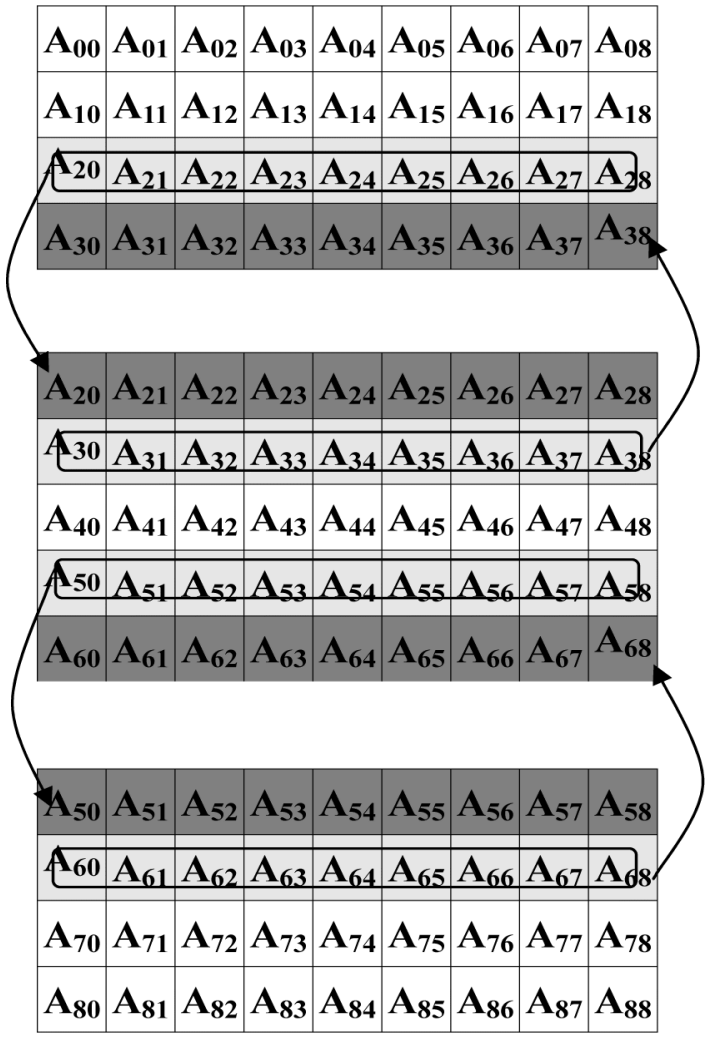
Алгоритм Якоби. Последовательная версия

```
/* Jacobi program */
#include <stdio.h>
#define L 1000
#define ITMAX 100
int i,j,it;
double A[L][L];
double B[L][L];
int main(int an, char **as)
{
        printf("JAC STARTED\n");
        for(i=0;i<=L-1;i++)
                for(j=0;j<=L-1;j++)
                {
                        A[i][j]=0.;
                        B[i][j]=1.+i+j;
                }
```

Алгоритм Якоби. Последовательная версия

```
        /******  iteration loop  ************************/
        for(it=1; it<ITMAX;it++)
        {
              for(i=1;i<=L-2;i++)
                    for(j=1;j<=L-2;j++)
                          A[i][j] = B[i][j];
              for(i=1;i<=L-2;i++)
                    for(j=1;j<=L-2;j++)
                          B[i][j] = (A[i-1][j]+A[i+1][j]+A[i][j-1]+A[i][j+1])/4.;
        }
        return 0;
}
```

Алгоритм Якоби. MPI-версия



Shadow edges

Imported elememts

Алгоритм Якоби. MPI-версия

```
/* Jacobi-1d program */
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"
#define m_printf if (myrank==0)printf
#define L 1000
#define ITMAX 100

int i,j,it,k;
int ll,shift;
double (* A)[L];
double (* B)[L];
```

Алгоритм Якоби. MPI-версия

```
int main(int argc, char **argv)
{
        MPI_Request req[4];
        int myrank, ranksize;
        int startrow,lastrow,nrow;
        MPI_Status status[4];
        double t1, t2, time;
        MPI_Init (&argc, &argv); /* initialize MPI system */
        MPI_Comm_rank(MPI_COMM_WORLD, &myrank);/*my place in MPI system*/
        MPI_Comm_size (MPI_COMM_WORLD, &ranksize); /* size of MPI system */
        MPI_Barrier(MPI_COMM_WORLD);
        /* rows of matrix I have to process */
        startrow = (myrank *L) / ranksize;
        lastrow = (((myrank + 1) * L) / ranksize)-1;
        nrow = lastrow - startrow + 1;
        m_printf("JAC1 STARTED\n");
```

Алгоритм Якоби. MPI-версия

```
/* dynamically allocate data structures */
A = malloc ((nrow+2) * L * sizeof(double));
B = malloc ((nrow) * L * sizeof(double));
for(i=1; i<=nrow; i++)
        for(j=0; j<=L-1; j++)
        {
                A[i][j]=0.;
                B[i-1][j]=1.+startrow+i-1+j;
        }
```

Алгоритм Якоби. MPI-версия

```
/******  iteration loop  ************************/
t1=MPI_Wtime();
for(it=1; it<=ITMAX; it++)
{
        for(i=1; i<=nrow; i++)
        {
                if (((i==1)&&(myrank==0))||((i==nrow)&&(myrank==ranksize-1)))
                    continue;
                for(j=1; j<=L-2; j++)
                {
                        A[i][j] = B[i-1][j];
                }
        }
```

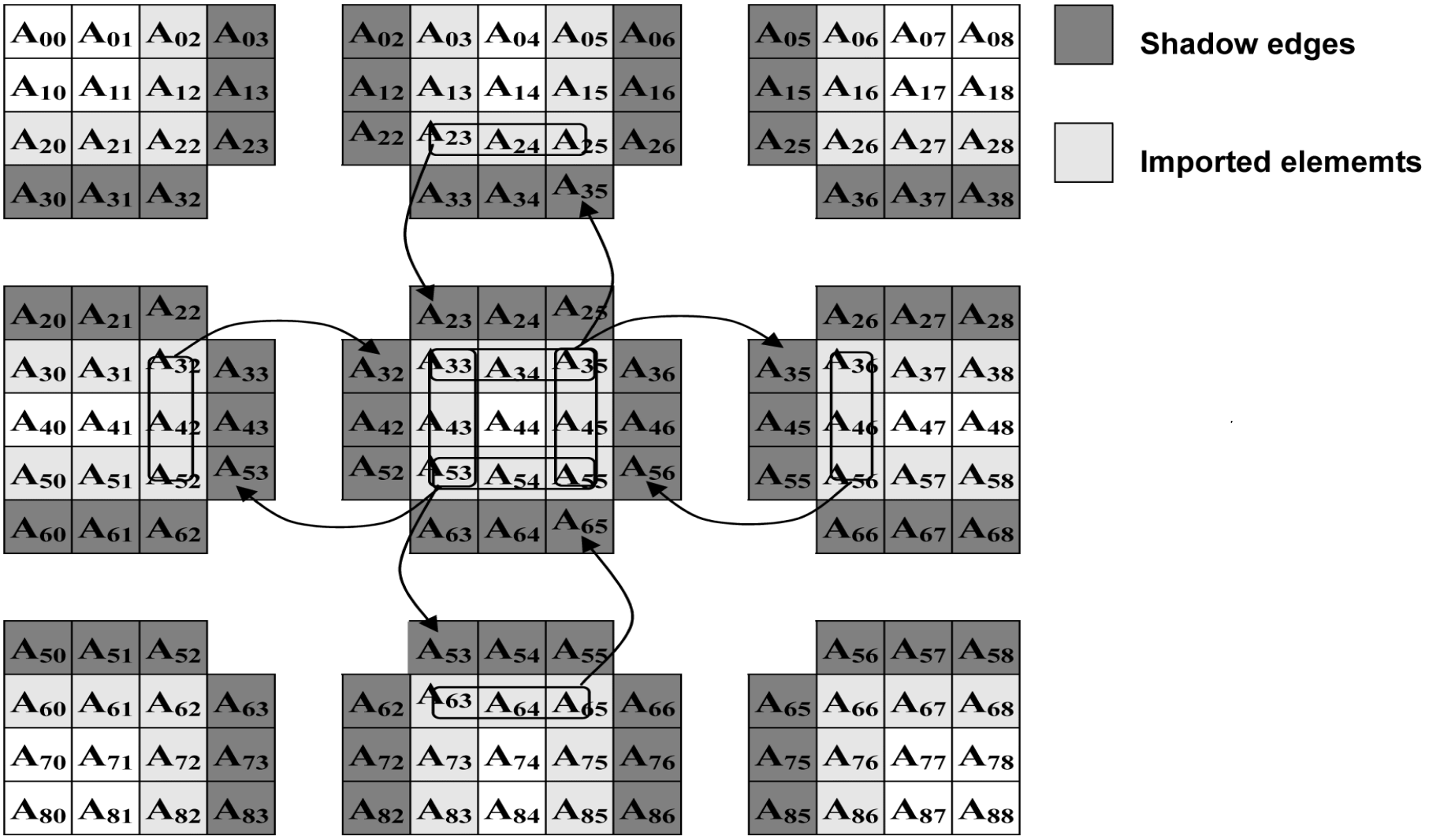
Алгоритм Якоби. MPI-версия

```
if(myrank!=0)
        MPI_Irecv(&A[0][0],L,MPI_DOUBLE, myrank-1, 1215,
          MPI_COMM_WORLD, &req[0]);
if(myrank!=ranksize-1)
        MPI_Isend(&A[nrow][0],L,MPI_DOUBLE, myrank+1, 1215,
          MPI_COMM_WORLD,&req[2]);
if(myrank!=ranksize-1)
        MPI_Irecv(&A[nrow+1][0],L,MPI_DOUBLE, myrank+1, 1216,
          MPI_COMM_WORLD, &req[3]);
if(myrank!=0)
        MPI_Isend(&A[1][0],L,MPI_DOUBLE, myrank-1, 1216,
          MPI_COMM_WORLD,&req[1]);
ll=4; shift=0;
if (myrank==0) {ll=2;shift=2;}
if (myrank==ranksize-1) {ll=2;}
MPI_Waitall(ll,&req[shift],&status[0]);
```

Алгоритм Якоби. MPI-версия

```
            for(i=1; i<=nrow; i++)
            {
                  if (((i==1)&&(myrank==0))||((i==nrow)&&(myrank==ranksize-1))) continue;
                  for(j=1; j<=L-2; j++)
                        B[i-1][j] = (A[i-1][j]+A[i+1][j]+
                              A[i][j-1]+A[i][j+1])/4.;
            }
      }/*DO it*/
printf("%d: Time of task=%lf\n",myrank,MPI_Wtime()-t1);
MPI_Finalize ();
return 0;
}
```

Алгоритм Якоби. MPI-версия



Алгоритм Якоби. MPI-версия

```
/*Jacobi-2d program */
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"
#define m_printf if (myrank==0)printf
#define L 1000
#define LC 2
#define ITMAX 100

int i,j,it,k;

double (* A)[L/LC+2];
double (* B)[L/LC];
```

Алгоритм Якоби. MPI-версия

```
int main(int argc, char **argv)
{
MPI_Request req[8];
int myrank, ranksize;
int srow,lrow,nrow,scol,lcol,ncol;
MPI_Status status[8];
double t1;
int isper[] = {0,0};
int dim[2];
int coords[2];
MPI_Comm newcomm;
MPI_Datatype vectype;
int pleft,pright, pdown,pup;
MPI_Init (&argc, &argv);       /* initialize MPI system */
MPI_Comm_size (MPI_COMM_WORLD, &ranksize);  /* size of MPI system */
MPI_Comm_rank (MPI_COMM_WORLD, &myrank);    /* my place in MPI system */
```

Алгоритм Якоби. MPI-версия

```
dim[0]=ranksize/LC;
dim[1]=LC;
if ((L%dim[0])||(L%dim[1]))
{
      m_printf("ERROR: array[%d*%d] is not distributed on %d*%d
processors\n",L,L,dim[0],dim[1]);
      MPI_Finalize();
      exit(1);
}
MPI_Cart_create(MPI_COMM_WORLD,2,dim,isper,1,&newcomm);
MPI_Cart_shift(newcomm,0,1,&pup,&pdown);
MPI_Cart_shift(newcomm,1,1,&pleft,&pright);
MPI_Comm_rank (newcomm, &myrank);    /* my place in MPI system */
MPI_Cart_coords(newcomm,myrank,2,coords);
```

Алгоритм Якоби. MPI-версия

```
/* rows of matrix I have to process */
srow = (coords[0] * L) / dim[0];
lrow = (((coords[0] + 1) * L) / dim[0])-1;
nrow = lrow - srow + 1;
/* columns of matrix I have to process */
scol = (coords[1] * L) / dim[1];
lcol = (((coords[1] + 1) * L) / dim[1])-1;
ncol = lcol - scol + 1;
MPI_Type_vector(nrow,1,ncol+2,MPI_DOUBLE,&vectype);
MPI_Type_commit(&vectype);
m_printf("JAC2 STARTED on %d*%d processors with %d*%d array,
it=%d\n",dim[0],dim[1],L,L,ITMAX);
/* dynamically allocate data structures */
A = malloc ((nrow+2) * (ncol+2) * sizeof(double));
B = malloc (nrow * ncol * sizeof(double));
```

Алгоритм Якоби. MPI-версия

```
for(i=0; i<=nrow-1; i++)
{
      for(j=0; j<=ncol-1; j++)
      {
            A[i+1][j+1]=0.;
            B[i][j]=1.+srow+i+scol+j;
      }
}
/******  iteration loop  ************************/
MPI_Barrier(newcomm);
t1=MPI_Wtime();
for(it=1; it<=ITMAX; it++)
{
      for(i=0; i<=nrow-1; i++)
      {
            if (((i==0)&&(pup==MPI_PROC_NULL))||((i==nrow-1)&&(pdown==MPI_PROC_NULL))) continue;
            for(j=0; j<=ncol-1; j++)
            {
                  if (((j==0)&&(pleft==MPI_PROC_NULL))||((j==ncol-1)&&(pright==MPI_PROC_NULL)))
                        continue;
                  A[i+1][j+1] = B[i][j];
            }
      }
```

Алгоритм Якоби. MPI-версия

```
MPI_Irecv(&A[0][1],ncol,MPI_DOUBLE,
    pup, 1215, MPI_COMM_WORLD, &req[0]);
MPI_Isend(&A[nrow][1],ncol,MPI_DOUBLE,
     pdown, 1215, MPI_COMM_WORLD,&req[1]);
MPI_Irecv(&A[nrow+1][1],ncol,MPI_DOUBLE,
     pdown, 1216, MPI_COMM_WORLD, &req[2]);
MPI_Isend(&A[1][1],ncol,MPI_DOUBLE,
     pup, 1216, MPI_COMM_WORLD,&req[3]);
MPI_Irecv(&A[1][0],1,vectype,
     pleft, 1217, MPI_COMM_WORLD, &req[4]);
MPI_Isend(&A[1][ncol],1,vectype,
     pright, 1217, MPI_COMM_WORLD,&req[5]);
MPI_Irecv(&A[1][ncol+1],1,vectype,
     pright, 1218, MPI_COMM_WORLD, &req[6]);
MPI_Isend(&A[1][1],1,vectype,
     pleft, 1218, MPI_COMM_WORLD,&req[7]);
MPI_Waitall(8,req,status);
```

Алгоритм Якоби. MPI-версия

```
        for(i=1; i<=nrow; i++)
        {
              if (((i==1)&&(pup==MPI_PROC_NULL))||
                    ((i==nrow)&&(pdown==MPI_PROC_NULL))) continue;
              for(j=1; j<=ncol; j++)
              {
                    if (((j==1)&&(pleft==MPI_PROC_NULL))||
                          ((j==ncol)&&(pright==MPI_PROC_NULL))) continue;
                    B[i-1][j-1] = (A[i-1][j]+A[i+1][j]+A[i][j-1]+A[i][j+1])/4.;
              }
        }
  }
  printf("%d: Time of task=%lf\n",myrank,MPI_Wtime()-t1);
  MPI_Finalize ();
  return 0;
}
```

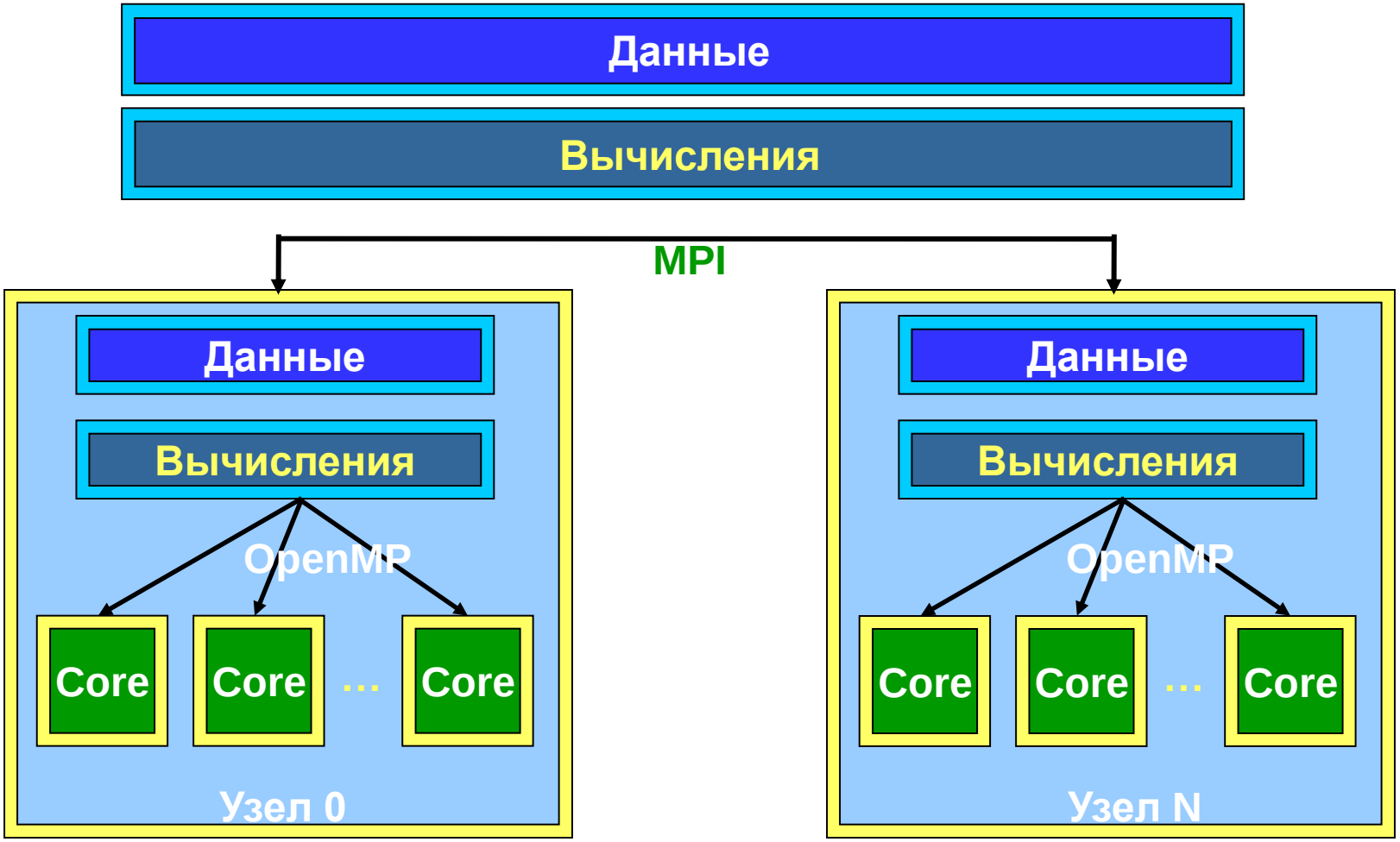
Алгоритм Якоби. OpenMP-версия

```
/* Jacobi program */
#include <stdio.h>
#define L 1000
#define ITMAX 100
int i,j,it;
double A[L][L];
double B[L][L];
int main(int an, char **as)
{
      printf("JAC STARTED\n");
     #pragma omp parallel for private(i,j)
      for(i=0;i<=L-1;i++)
            for(j=0;j<=L-1;j++)
            {
                  A[i][j]=0.;
                  B[i][j]=1.+i+j;
            }
```

Алгоритм Якоби. OpenMP-версия

```
        /******  iteration loop  ************************/
        for(it=1; it<ITMAX;it++)
        {
              #pragma omp parallel for private(i,j)
              for(i=1;i<=L-2;i++)
                    for(j=1;j<=L-2;j++)
                          A[i][j] = B[i][j];
             #pragma omp parallel for private(i,j)
              for(i=1;i<=L-2;i++)
                    for(j=1;j<=L-2;j++)
                          B[i][j] = (A[i-1][j]+A[i+1][j]+A[i][j-1]+A[i][j+1])/4.;
        }
        return 0;
}
```

Гибридная модель MPI/OpenMP



Алгоритм Якоби. MPI/OpenMP-версия

```
/* Jacobi-1d program */
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"
#define m_printf if (myrank==0)printf
#define L 1000
#define ITMAX 100

int i,j,it,k;
int ll,shift;
double (* A)[L];
double (* B)[L];
```

Алгоритм Якоби. MPI/OpenMP-версия

```
int main(int argc, char **argv)
{
        MPI_Request req[4];
        int myrank, ranksize;
        int startrow,lastrow,nrow;
        MPI_Status status[4];
        double t1, t2, time;
        MPI_Init (&argc, &argv); /* initialize MPI system */
        MPI_Comm_rank(MPI_COMM_WORLD,&myrank); /*my place in MPI system */
        MPI_Comm_size (MPI_COMM_WORLD, &ranksize); /* size of MPI system */
        MPI_Barrier(MPI_COMM_WORLD);
        /* rows of matrix I have to process */
        startrow = (myrank * N) / ranksize;
        lastrow = (((myrank + 1) * N) / ranksize)-1;
        nrow = lastrow - startrow + 1;
        m_printf("JAC1 STARTED\n");
```

Алгоритм Якоби. MPI/OpenMP-версия

```
/* dynamically allocate data structures */
A = malloc ((nrow+2) * N * sizeof(double));
B = malloc ((nrow) * N * sizeof(double));
for(i=1; i<=nrow; i++)
        #pragma omp parallel for
        for(j=0; j<=L-1; j++)
        {
                A[i][j]=0.;
                B[i-1][j]=1.+startrow+i-1+j;
        }
```

Алгоритм Якоби. MPI/OpenMP-версия

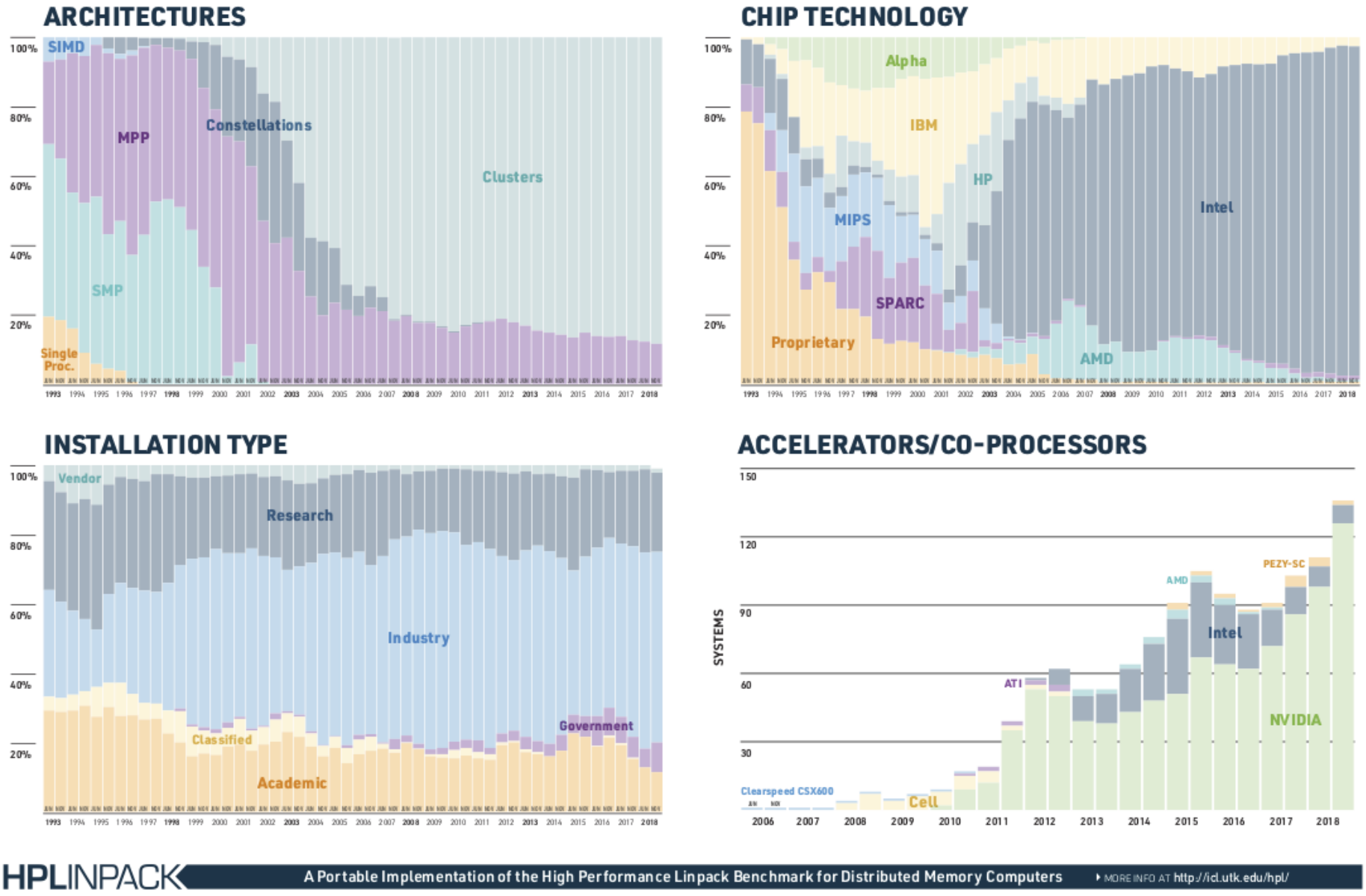
```
/******  iteration loop  ************************/
t1=MPI_Wtime();
for(it=1; it<=ITMAX; it++)
{
        for(i=1; i<=nrow; i++)
        {
                if (((i==1)&&(myrank==0))||((i==nrow)&&(myrank==ranksize-1)))
                    continue;
                #pragma omp parallel for
                for(j=1; j<=L-2; j++)
                {
                        A[i][j] = B[i-1][j];
                }
        }
```

Алгоритм Якоби. MPI/OpenMP-версия

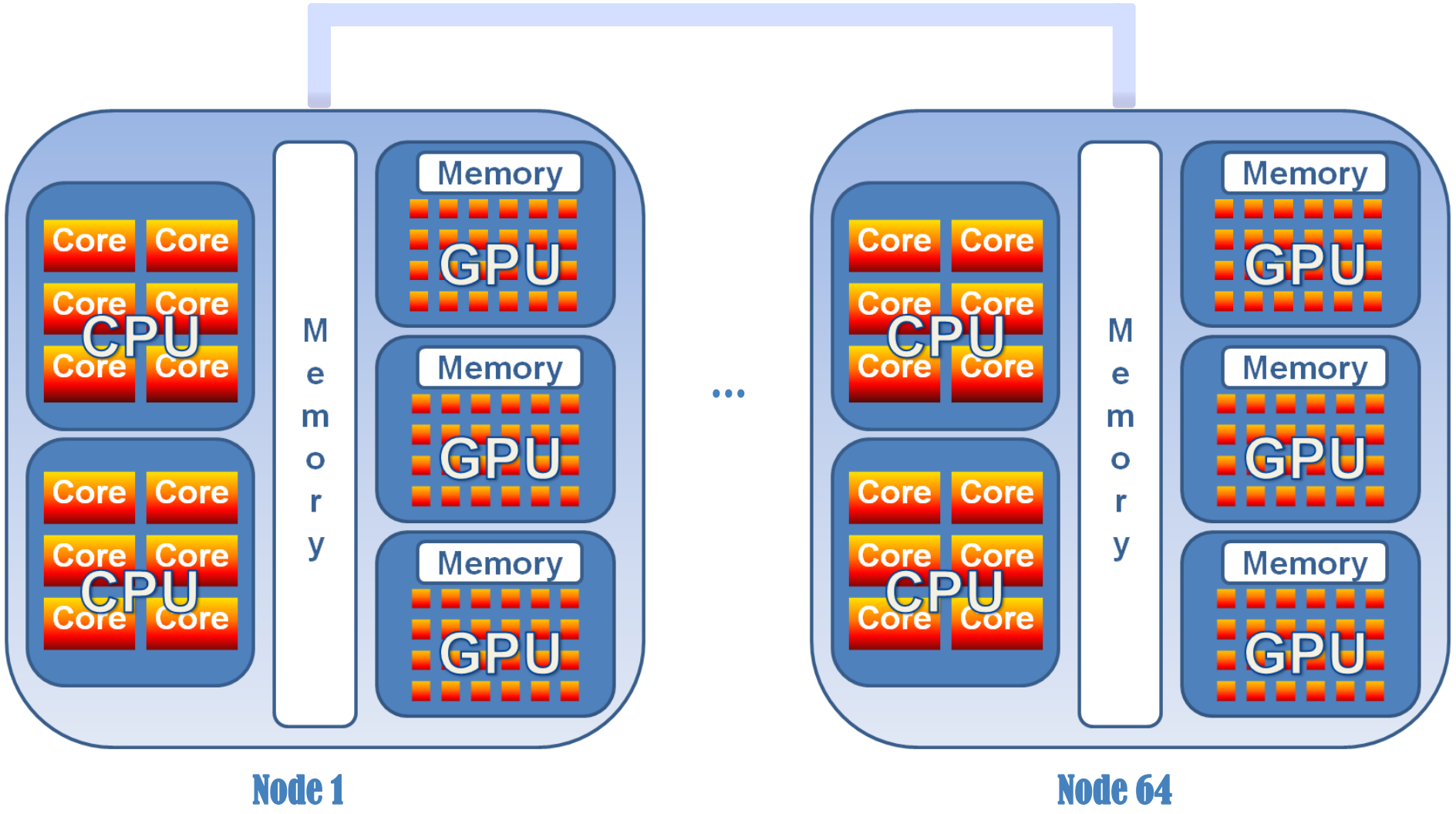
```
if(myrank!=0)
        MPI_Irecv(&A[0][0],L,MPI_DOUBLE, myrank-1, 1215,
            MPI_COMM_WORLD, &req[0]);
if(myrank!=ranksize-1)
        MPI_Isend(&A[nrow][0],L,MPI_DOUBLE, myrank+1, 1215,
            MPI_COMM_WORLD,&req[2]);
if(myrank!=ranksize-1)
        MPI_Irecv(&A[nrow+1][0],L,MPI_DOUBLE, myrank+1, 1216,
            MPI_COMM_WORLD, &req[3]);
if(myrank!=0)
        MPI_Isend(&A[1][0],L,MPI_DOUBLE, myrank-1, 1216,
            MPI_COMM_WORLD,&req[1]);
ll=4; shift=0;
if (myrank==0) {ll=2;shift=2;}
if (myrank==ranksize-1) {ll=2;}
MPI_Waitall(ll,&req[shift],&status[0]);
```

Алгоритм Якоби. MPI/OpenMP-версия

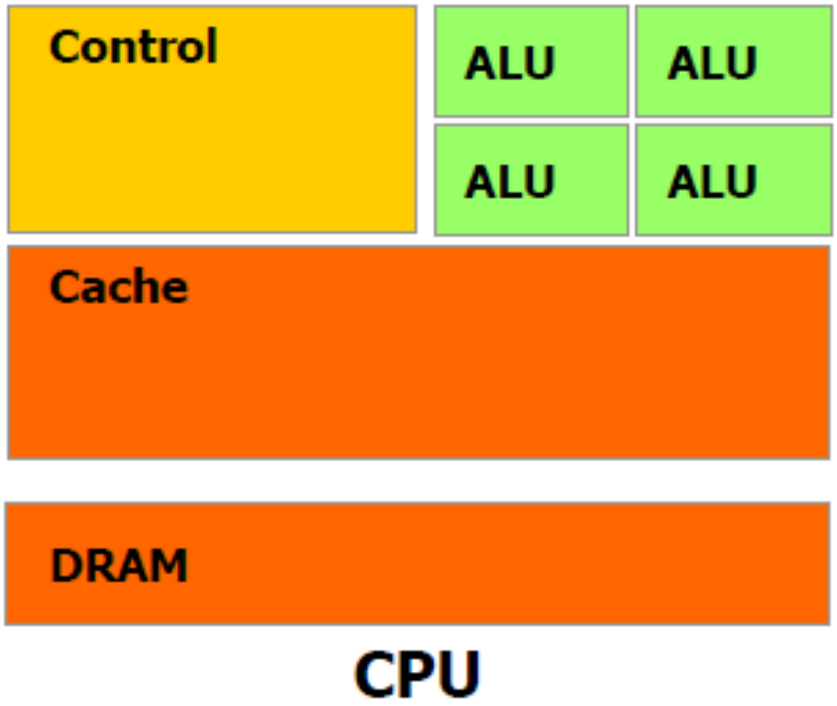
```
            for(i=1; i<=nrow; i++)
            {
                  if (((i==1)&&(myrank==0))||((i==nrow)&&(myrank==ranksize-1))) continue;
                  #pragma omp parallel for
                  for(j=1; j<=L-2; j++)
                        B[i-1][j] = (A[i-1][j]+A[i+1][j]+
                              A[i][j-1]+A[i][j+1])/4.;
            }
      }/*DO it*/
printf("%d: Time of task=%lf\n",myrank,MPI_Wtime()-t1);
MPI_Finalize ();
return 0;
}
```

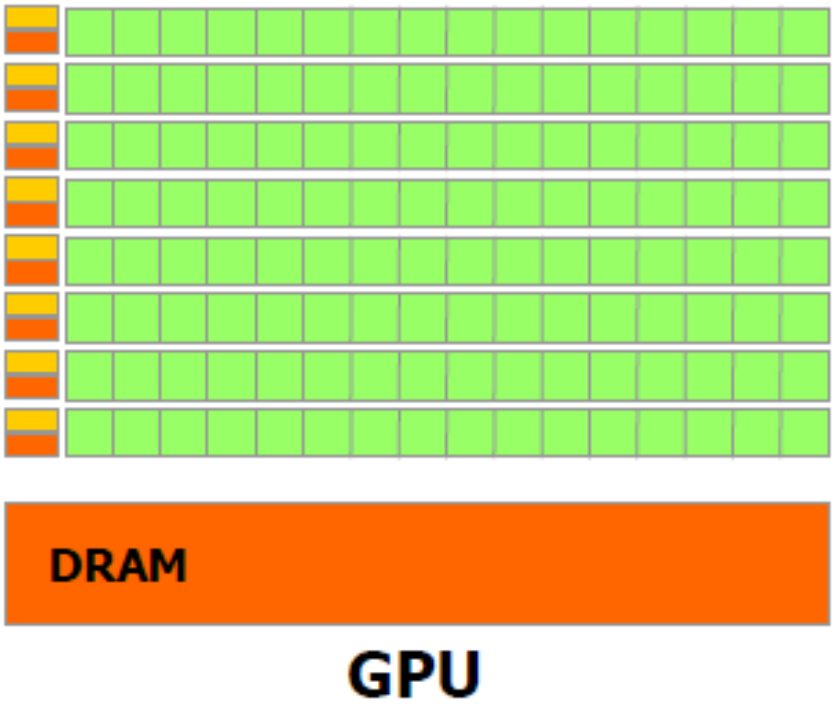

ARCHITECTURES
100%
80%
60%
40%
20%
SIMD
MPP
Constellations
Clusters
SMP
Single Proc.
1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
CHIP TECHNOLOGY
100%
80%
60%
40%
20%
Alpha
IBM
HP
MIPS
Intel
SPARC
Proprietary
AMD
1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
INSTALLATION TYPE
100%
80%
60%
40%
20%
Vendor
Research
Industry
Government
Classified
Academic
1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
ACCELERATORS/CO-PROCESSORS
SYSTEMS
150
120
90
60
30
PEZY-SC
AMD
Intel
ATI
NVIDIA
Clearspeed CSX600
Cell
2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
HPLINPACK
A Portable Implementation of the High Performance Linpack Benchmark for Distributed Memory Computers
MORE INFO AT http://icl.utk.edu/hpl/

Гибридная вычислительная система K-100



CPU или GPU





Алгоритм Якоби на языке Fortran

```
      PROGRAM   JACOB_SEQ
     PARAMETER   (L=4096,  ITMAX=100)
     REAL    A(L,L), B(L,L)
      PRINT *,  '**********  TEST_JACOBI  **********'
     DO IT  =  1,  ITMAX
          DO  J  =  2, L-1
              DO  I  =  2, L-1
                  A(I, J)  =  B(I, J)
              ENDDO
          ENDDO
            DO  J = 2,  L-1
                DO  I = 2,  L-1
                    B(I, J) =  (A(I-1, J) + A(I, J-1) +  A(I+1, J) +
     *                    A(I, J+1)) / 4
                ENDDO
            ENDDO
      ENDDO
      PRINT *,B
      END
```

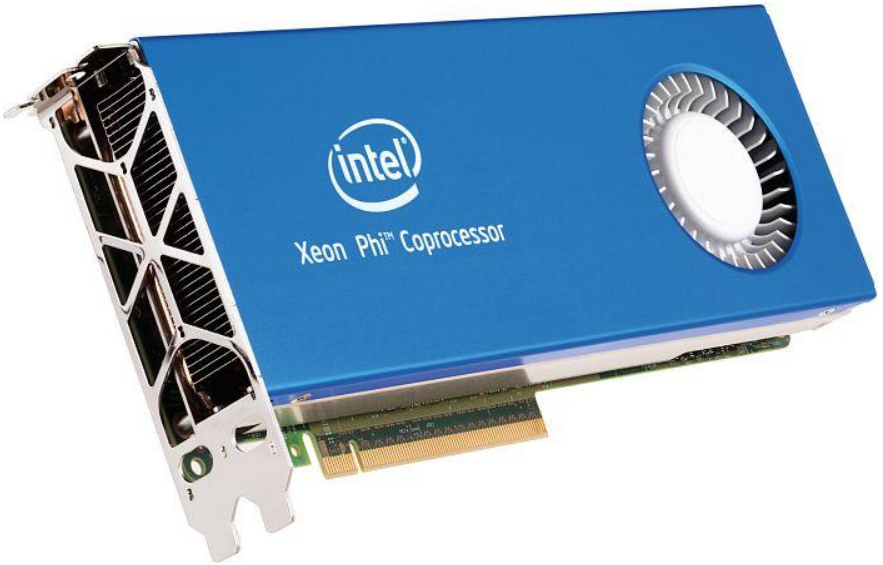
Алгоритм Якоби на языке Fortran CUDA

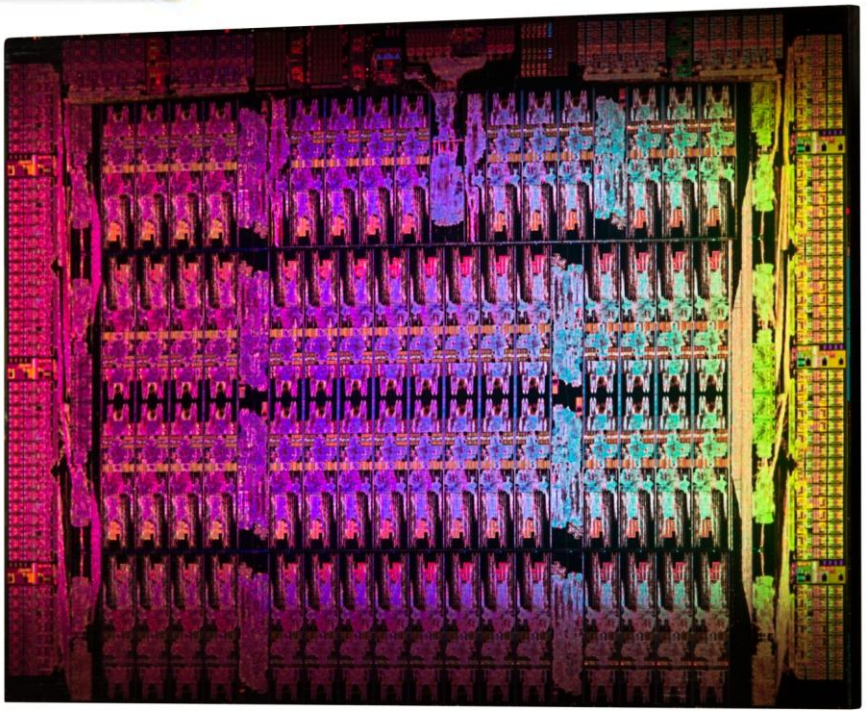
```
program JACOB_CUDA
 use cudafor
 use jac_cuda
 parameter (k=4096, itmax = 100, block_dim = 16)
 real, device, dimension(k, k) :: a, b
 integer it
 type(dim3) :: grid, block
 print *, '**********  test_jacobi   **********‘
 grid = dim3(k / block_dim, k / block_dim, 1)
 block = dim3(block_dim, block_dim, 1)
 do it = 1, itmax
      call arr_copy<<<grid, block>>>(a, b, k)
      call arr_renew<<<grid, block>>>(a, b, k)
 end do
 end program JACOB_CUDA
```

Алгоритм Якоби на языке Fortran CUDA

```
module jac_cuda
    contains
     attributes(global) subroutine arr_copy(a, b, k)
     real, device, dimension(k, k) :: a, b
     integer, value :: k
     integer i, j
     i = (blockIdx%x - 1) * blockDim%x + threadIdx%x
     j = (blockIdx%y - 1) * blockDim%y + threadIdx%y
     if (i.ne.1 .and. i.ne.k .and. j.ne.1 .and. j.ne.k) a(i, j) = b(i, j)
end subroutine arr_copy
attributes(global) subroutine arr_renew(a, b, k)
     real, device, dimension(k, k) :: a, b
     integer, value :: k
     integer i, j
     i = (blockIdx%x - 1) * blockDim%x + threadIdx%x
     j = (blockIdx%y - 1) * blockDim%y + threadIdx%y
     if (i.ne.1.and.i.ne.k.and.j.ne.1.and.j.ne.k) b(i,j)=(a(i-1,j)+a(i+1,j)+a(i,j-1)+a(I,j+1))/4
end subroutine arr_renew
end module jac_cuda
```

Intel Xeon Phi





DVM-система

DVM-система состоит из следующих компонент:

- Компилятор Fortran-DVMH
- Компилятор C-DVMH
- Библиотека поддержки LIB-DVMH
- DVM-отладчик
- Предсказатель выполнения DVM-программ
- Анализатор производительности DVM-программ

Аббревиатура DVM (**D**istributed **V**irtual **M**emory, **D**istributed **V**irtual **M**achine) отражает поддержку виртуальной общей памяти на распределенных системах

Средства программирования

C-DVMH = Язык Си + специальные прагмы

Fortran-DVMH = Язык Фортран 95 + специальные комментарии

- Специальные комментарии и прагмы являются высокоуровневыми спецификациями параллелизма в терминах последовательной программы
- Отсутствуют низкоуровневые передачи данных и синхронизации
- Последовательный стиль программирования
- Спецификации параллелизма «невидимы» для стандартных компиляторов
- Существует только один экземпляр программы для последовательного и параллельного счета

Алгоритм Якоби. DVM-версия

```
      PROGRAM    JAC_DVM
      PARAMETER    (L=4096,  ITMAX=100)
      REAL     A(L,L), B(L,L)
!DVM$   DISTRIBUTE    ( BLOCK,   BLOCK)   ::   A
!DVM$   ALIGN  B(I,J)  WITH  A(I,J)
      PRINT *,  '**********  TEST_JACOBI   **********'
      DO IT  =  1,  ITMAX
!DVM$     PARALLEL (J,I) ON  A(I,  J)
          DO  J  =  2, L-1
              DO  I  =  2, L-1
                  A(I, J)  =  B(I, J)
              ENDDO
          ENDDO
```

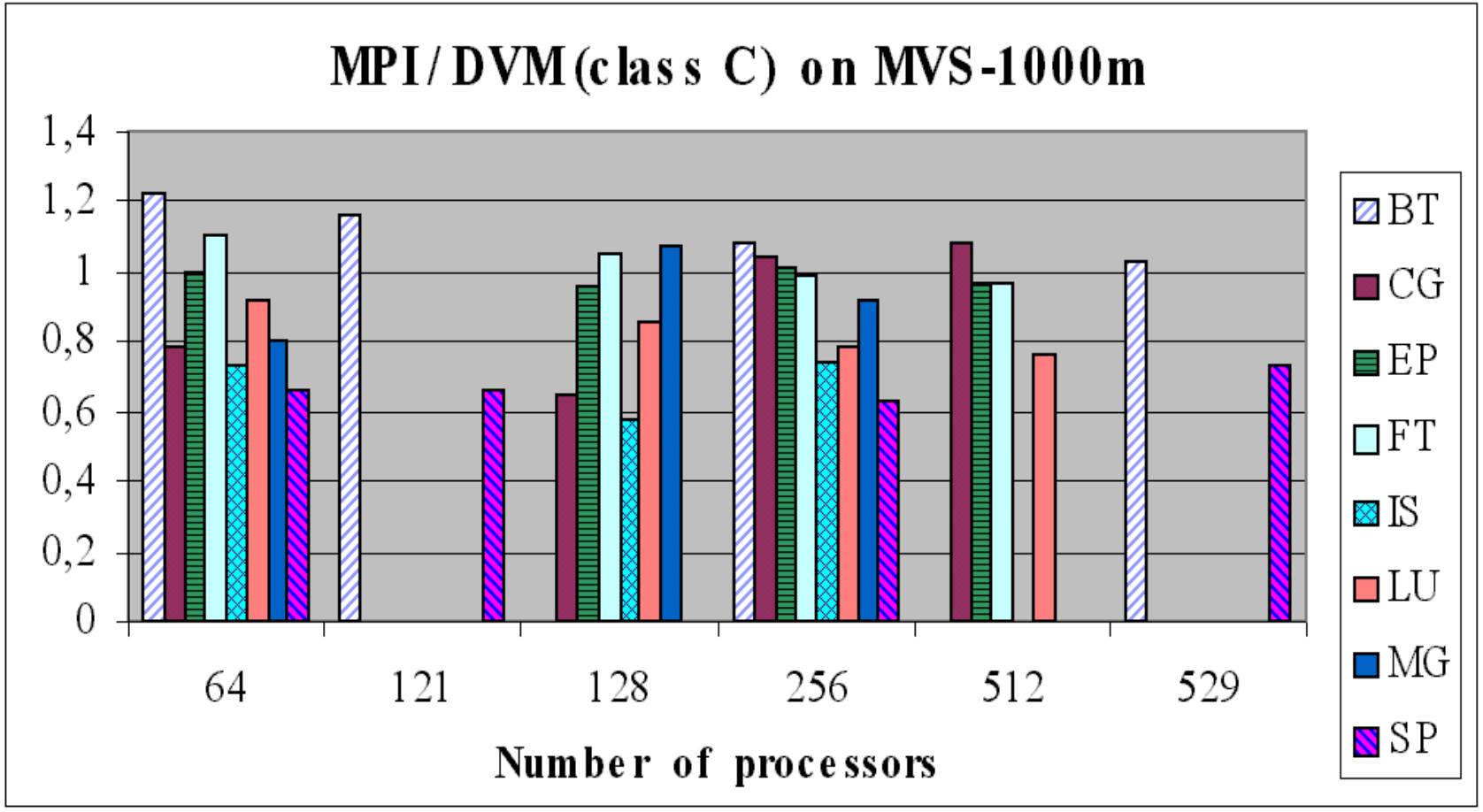
Алгоритм Якоби. DVM-версия

```
!DVM$      PARALLEL (J,I) ON B(I,  J), SHADOW_RENEW (A)
           DO  J = 2,  L-1
                 DO  I = 2,  L-1
                     B(I, J) =  (A(I-1, J) + A(I, J-1) +  A(I+1, J) +  A(I, J+1)) / 4
                 ENDDO
            ENDDO
      ENDDO
      PRINT *,B
      END
```

Тесты NAS

Тест	Характеристика теста	SEQ	MPI	DVM	MPI/ SEQ	DVM/S EQ
BT	3D Навье-Стокс, метод переменных направлений	3929	5744	3991	1.46	1.02
CG	Оценка наибольшего собственного значения симметричной разреженной матрицы	1108	1793	1118	1.62	1.01
EP	Генерация пар случайных чисел Гаусса	641	670	649	1.04	1.01
FT	Быстрое преобразование Фурье, 3D спектральный метод	1500	2352	1605	1.57	1.07
IS	Параллельная сортировка	925	1218	1067	1.32	1.17
LU	3D Навье-Стокс, метод верхней релаксации	4189	5497	4269	1.31	1.02
MG	3D уравнение Пуассона, метод Multigrid	1898	2857	2131	1.50	1.12
SP	3D Навье-Стокс, Beam-Warning approximate factorization	3361	5020	3630	1.49	1.08
Σ		17551	25151	18460	1.43	1.05

Тесты NAS


MPI/DVM(class C) on MVS-1000m
1,4
1,2
1
0,8
0,6
0,4
0,2
0
64
121
128
256
512
529
Number of processors
BT
CG
EP
FT
IS
LU
MG
SP

Алгоритм Якоби. DVM/OpenMP-версия

```
      PROGRAM    JAC_OpenMP_DVM
      PARAMETER    (L=4096,  ITMAX=100)
      REAL     A(L,L), B(L,L)
!DVM$   DISTRIBUTE    ( BLOCK,   BLOCK)   ::   A
!DVM$   ALIGN  B(I,J)  WITH  A(I,J)
      PRINT *,  '**********  TEST_JACOBI   **********'
      DO IT  =  1,  ITMAX
!DVM$     PARALLEL (J,I) ON  A(I,  J)
!$OMP     PARALLEL DO COLLAPSE (2)
        DO  J  =  2, L-1
          DO  I  =  2, L-1
            A(I, J)  =  B(I, J)
          ENDDO
        ENDDO
```

Алгоритм Якоби. DVM/OpenMP-версия

```
!DVM$      PARALLEL (J,I) ON B(I,  J), SHADOW_RENEW (A)
 !$OMP     PARALLEL DO COLLAPSE (2)
           DO  J = 2,  L-1
                 DO  I = 2,  L-1
                     B(I, J) =  (A(I-1, J) + A(I, J-1) +  A(I+1, J) +  A(I, J+1)) / 4
                 ENDDO
            ENDDO
       ENDDO
       PRINT *, B
       END
```

Алгоритм Якоби. DVMH-версия

```
          PROGRAM    JAC_GPU_DVM
          PARAMETER    (L=4096,  ITMAX=100)
          REAL     A(L,L), B(L,L)
!DVM$     DISTRIBUTE     ( BLOCK,   BLOCK)   ::   A
!DVM$     ALIGN  B(I,J)  WITH  A(I,J)
          PRINT *,  '**********  TEST_JACOBI   **********'
          DO IT  =  1,  ITMAX
!DVM$       REGION INOUT(B), IN (A)
!DVM$       PARALLEL (J,I) ON  A(I,  J)
            DO  J  =  2, L-1
                  DO  I  =  2, L-1
                      A(I, J)  =  B(I, J)
                  ENDDO
             ENDDO
```

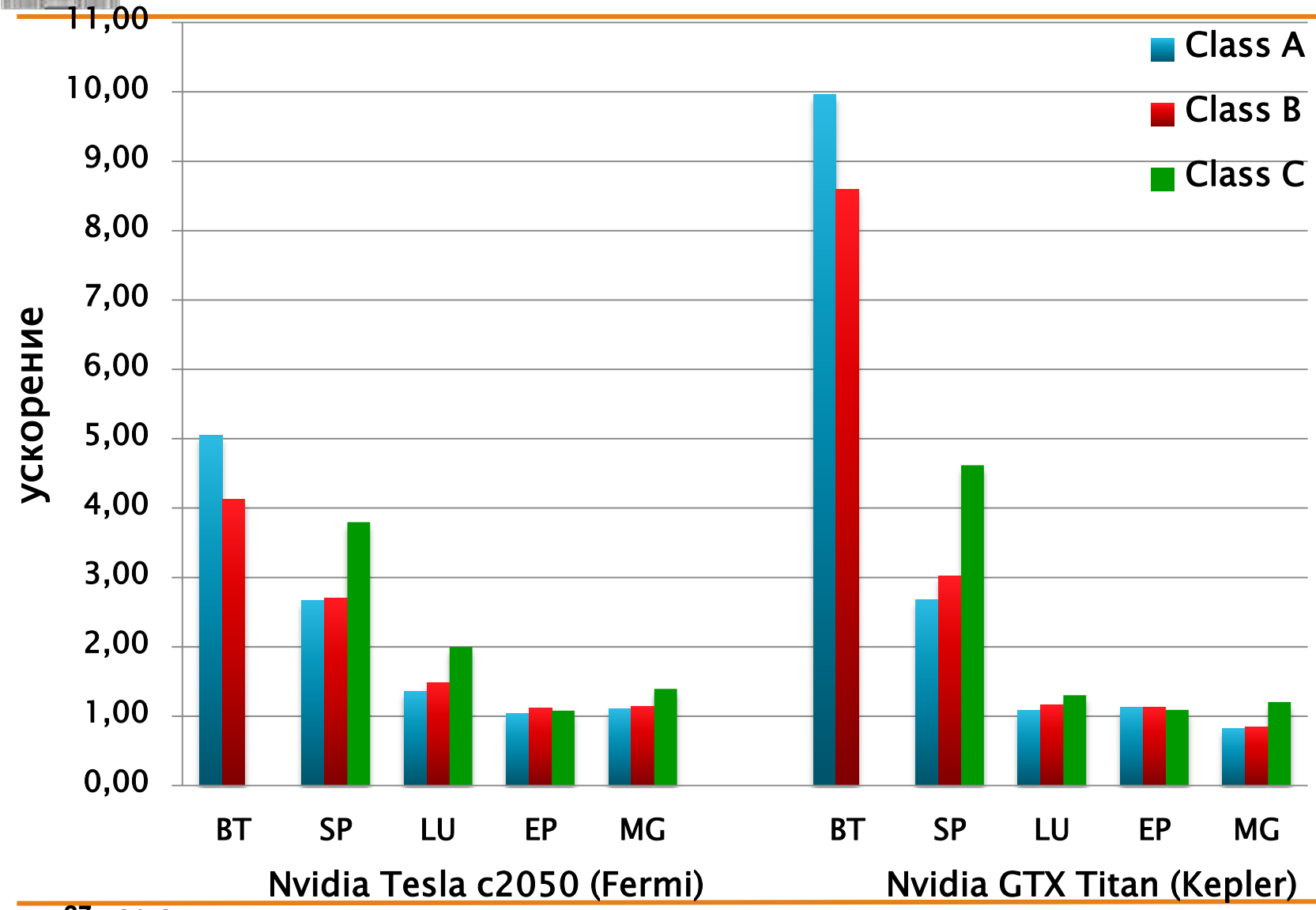
Алгоритм Якоби. DVMH-версия

```
!DVM$       PARALLEL (J,I) ON B(I,  J), SHADOW_RENEW (A)
            DO  J = 2,  L-1
                  DO  I = 2,  L-1
                      B(I, J) =  (A(I-1, J) + A(I, J-1) +  A(I+1, J) +  A(I, J+1)) / 4
                  ENDDO
             ENDDO
!DVM$       END REGION
          ENDDO
!DVM$     GET_ACTUAL(B)
          PRINT *, B
          END
```

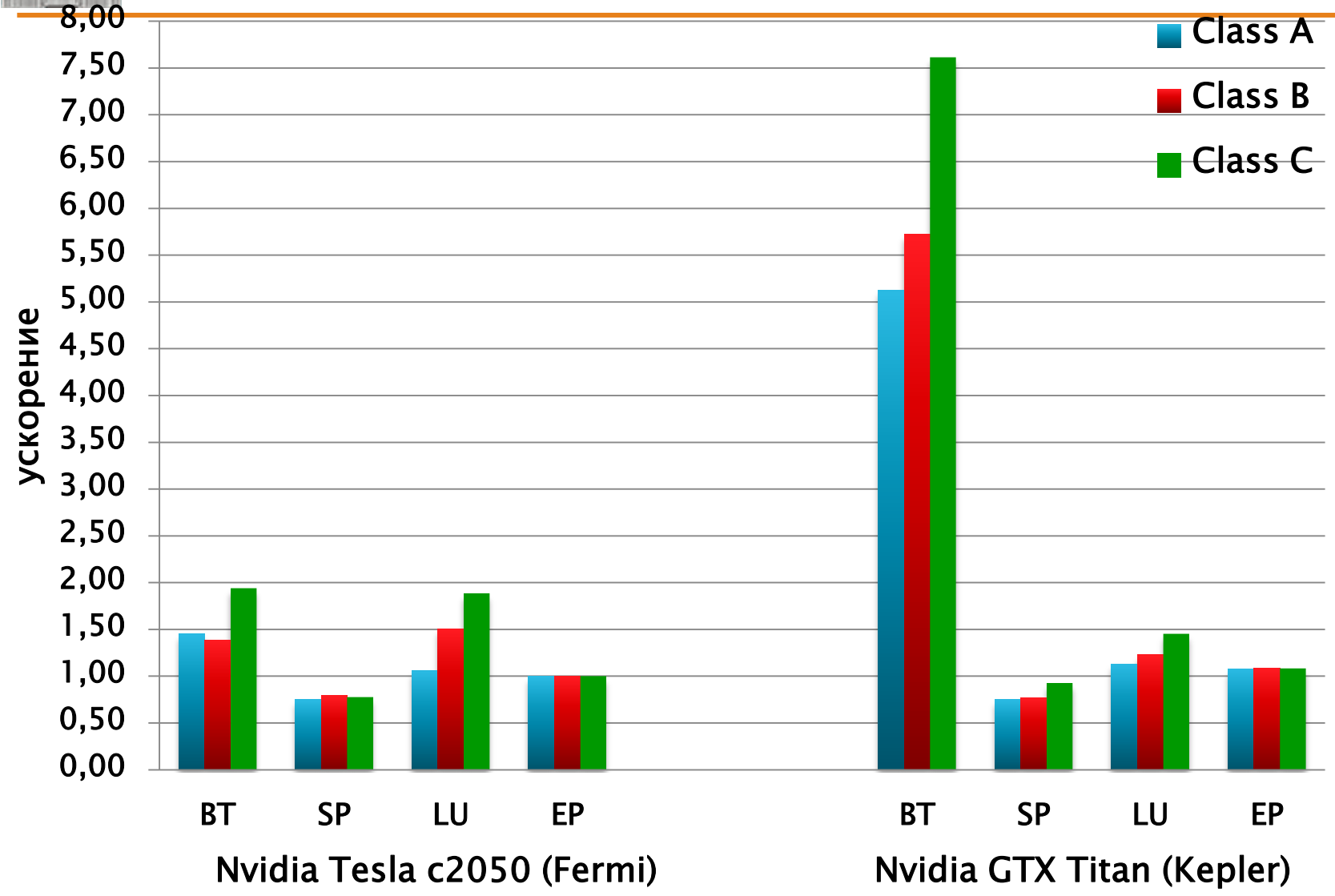
Алгоритм Якоби. DVMH-версия

```
#define L 4096
#define ITMAX 100
#pragma dvm array A(distribute[block][block], shadow[1:1][1:1])
double A[L][L];
#pragma dvm array B(align([i][j] with A[i][j]))
double B[L][L];
int main(int argc, char *argv[]) {
   for(int it = 0; it < ITMAX; it++) {
       #pragma dvm region
      {
         #pragma dvm parallel([i][j] on A[i][j])
         for (int i = 1; I < L - 1; i++)
            for (int j = 1; j < L-1; j++)  A[i][j] = B[i][j];
         #pragma dvm parallel([i][j] on B[i][j]), shadow_renew(A)
         for (int i = 1; I < L - 1; i++)
            for (int j = 1; j < L - 1; j++)
               B[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i][j - 1] + A[i][j + 1]) / 4.;
      }
   }
   return 0;
}
```

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



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



Содержание

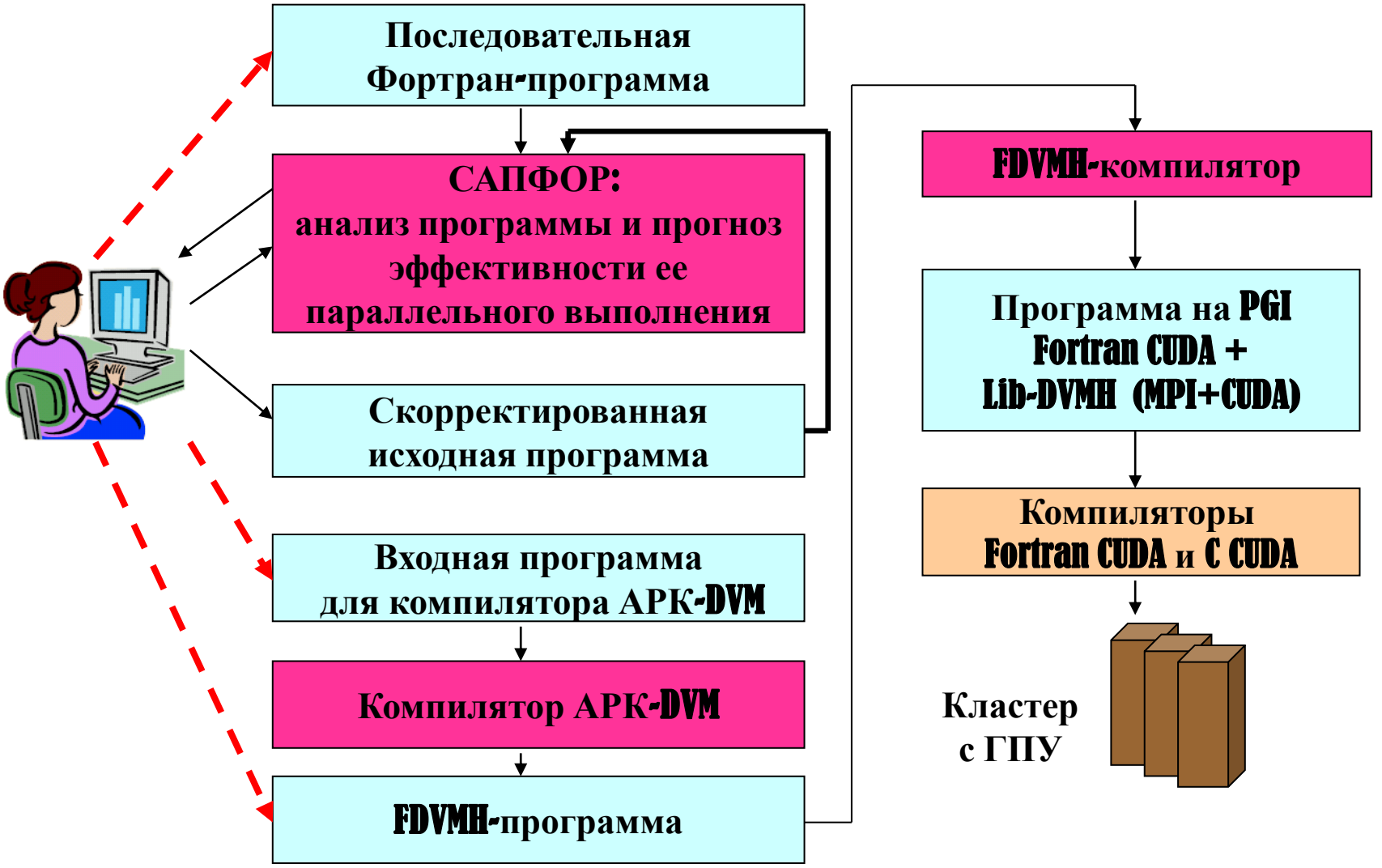
- Современные направления развития параллельных вычислительных систем
- Технологии параллельного программирования
- Система Автоматизированной Параллелизации ФОРтран-программ (**САПФОР**)

Система САПФОР

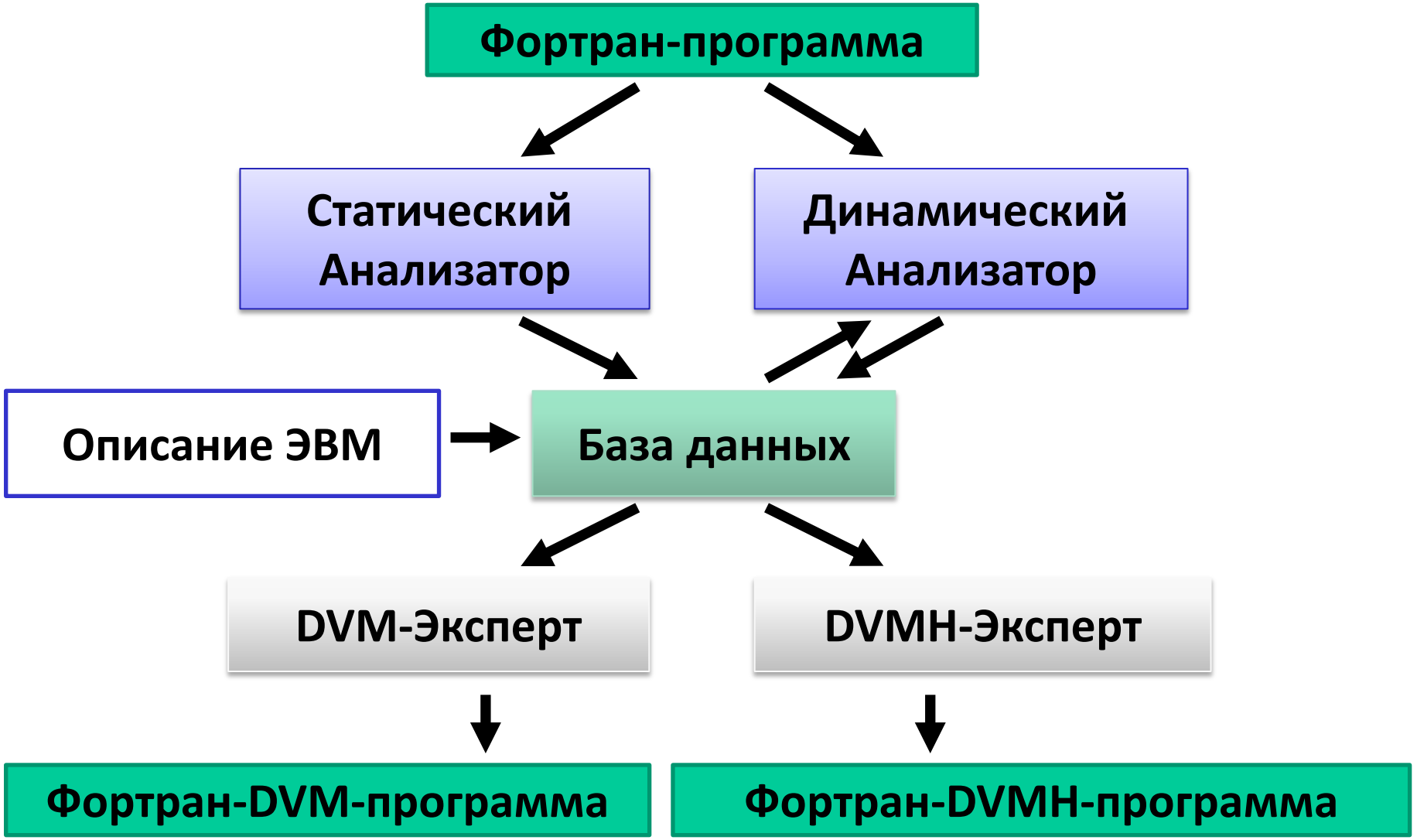
Помогает программисту эффективно отображать его программы на многоядерные кластеры с ускорителями.

Взаимодействует с программистом в терминах последовательной программы.

Система САПФОР



Автоматически распараллеливающий компилятор



Автоматически распараллеливающий компилятор

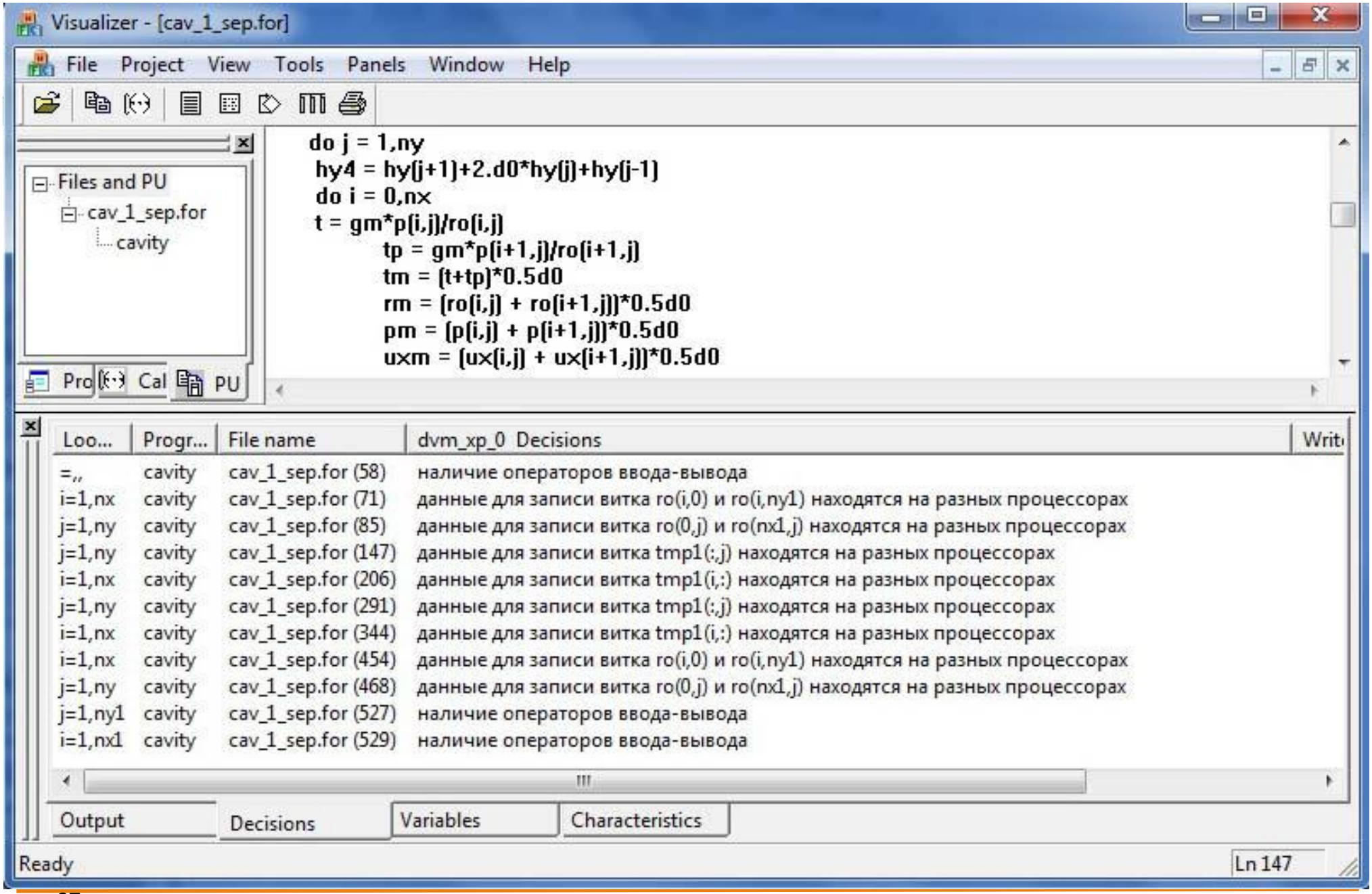
Далее приводятся результаты работы распараллеливающего компилятора на:

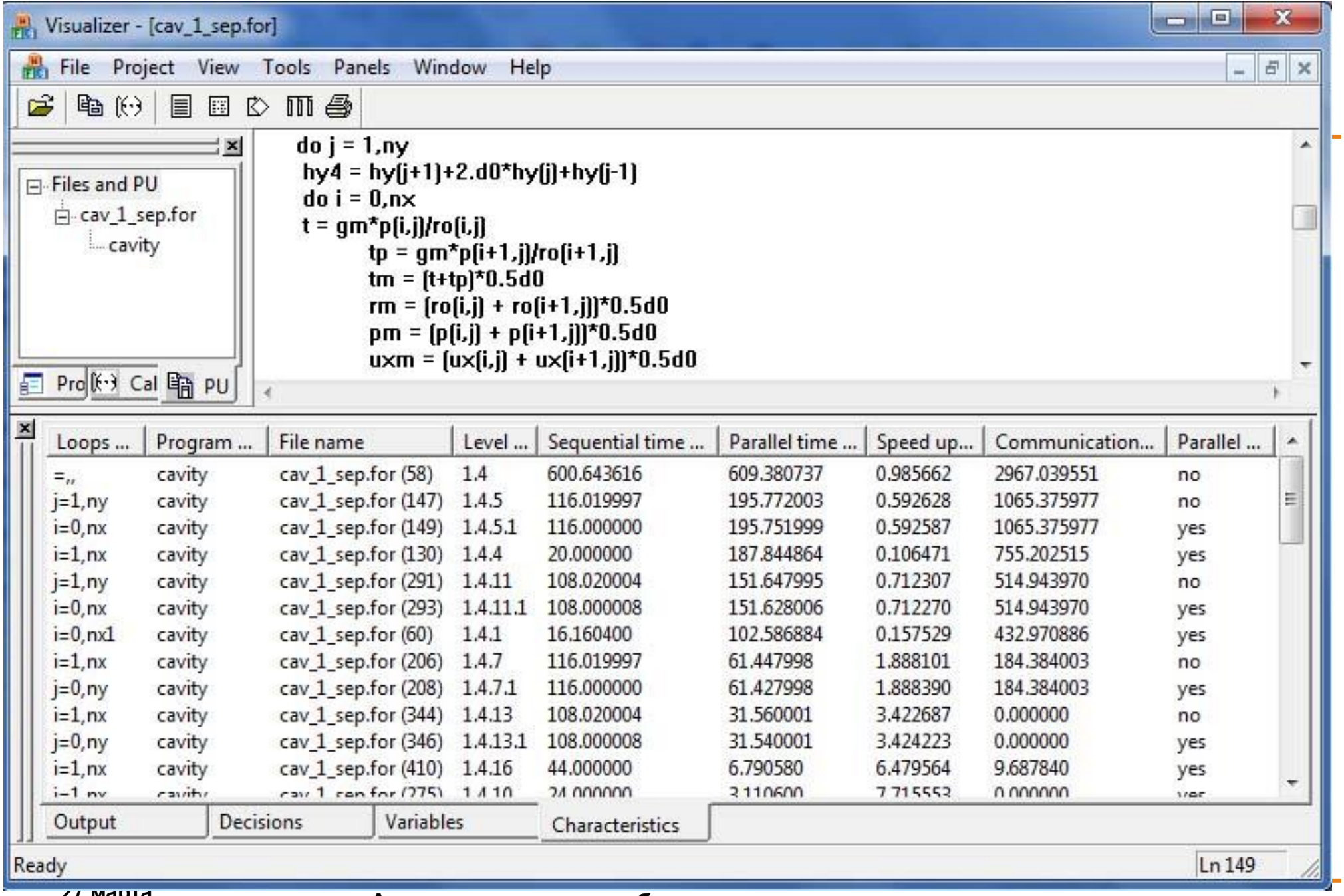
- тестах NAS LU, BT, SP
- программе MHPDV (трехмерного моделирования сферического взрыва во внешнем магнитном поле с помощью решения уравнений идеальной магнитогидродинамики)
- программе ZEBRA (расчет нейтронных полей атомного реактора в диффузионном приближении)

Автоматически распараллеливающий компилятор

Варианты	Число проц 1	Число проц 8	Число проц 64	Число проц 256	Число проц 1024
BT-авто	мало памяти	1255.97	182.70	54.64	**21.36**
BT-ручное	мало памяти	817.88	128.01	30.27	**7.19**
LU-авто	**3482.40**	1009.49	148.78	40.33	25.55
LU-ручное	**2103.14**	858.26	122.61	34.99	19.97
SP-авто	**1982.00**	-	-	-	-
SP-ручное	**2601.85**	-	-	-	-
MHPDV-авто	3703.23	500.78	89.32	34.75	12.78
MHPDV-ручное	3574.29	486.74	79.63	32.15	10.98
ZEBRA-авто	75.09	11.13	1.96	-	-
ZEBRA-ручное	75.62	10.18	1.85	-	-

	BT	LU	SP	MHPDV	ZEBRA
Время работы DVM-эксперта на ПЭВМ (сек)	1855	96	879	41	753
Количество строк	10442	3635	5950	1878	2426
Количество циклов	504	424	**499**	116	49
Количество параллелизуемых циклов DVM-экспертом	481	410	**293**	115	28
Количество обращений к массивам	30530	6638	11015	4643	2680
Количество массивов	34	30	37	33	40
Суммарное число измерений	75	64	70	78	49
Распределение массивов по измерениям (начиная с одномерных массивов)	16 5 7 3 2 1	22 3 0 5 4	21 3 9 4	16 0 6 11	36 0 3 1
Количество групп измерений	6	5	5	5	6
Количество построенных схем	16	16	16	16	64
Среднее время построения одной схемы (сек)	38,56	3,93	2,62	1,43	0,29
Среднее время оценки эффективности одной схемы (сек)	77,39	2,06	52,31	1,13	11,47
Общее количество перебранных процессорных решеток	127	127	182	144	543


Visualizer - [cav_1_sep.for]
File Project View Tools Panels Window Help
Files and PU
cav_1_sep.for
cavity
Pro Cal PU
do j = 1,ny
hy4 = hy(j+1)+2.d0*hy(j)+hy(j-1)
do i = 0,nx
t = gm*p(i,j)/ro(i,j)
tp = gm*p(i+1,j)/ro(i+1,j)
tm = (t+tp)*0.5d0
rm = (ro(i,j) + ro(i+1,j))*0.5d0
pm = (p(i,j) + p(i+1,j))*0.5d0
uxm = (ux(i,j) + ux(i+1,j))*0.5d0
Loo... Progr... File name dvm_xp_0 Decisions Writ
=,, cavity cav_1_sep.for (58) наличие операторов ввода-вывода
i=1,nx cavity cav_1_sep.for (71) данные для записи витка ro(i,0) и ro(i,ny1) находятся на разных процессорах
j=1,ny cavity cav_1_sep.for (85) данные для записи витка ro(0,j) и ro(nx1,j) находятся на разных процессорах
j=1,ny cavity cav_1_sep.for (147) данные для записи витка tmp1(:,j) находятся на разных процессорах
i=1,nx cavity cav_1_sep.for (206) данные для записи витка tmp1(i,:) находятся на разных процессорах
j=1,ny cavity cav_1_sep.for (291) данные для записи витка tmp1(:,j) находятся на разных процессорах
i=1,nx cavity cav_1_sep.for (344) данные для записи витка tmp1(i,:) находятся на разных процессорах
i=1,nx cavity cav_1_sep.for (454) данные для записи витка ro(i,0) и ro(i,ny1) находятся на разных процессорах
j=1,ny cavity cav_1_sep.for (468) данные для записи витка ro(0,j) и ro(nx1,j) находятся на разных процессорах
j=1,ny1 cavity cav_1_sep.for (527) наличие операторов ввода-вывода
i=1,nx1 cavity cav_1_sep.for (529) наличие операторов ввода-вывода
Output Decisions Variables Characteristics
Ready
Ln 147


Visualizer - [cav_1_sep.for]
File Project View Tools Panels Window Help
Files and PU
cav_1_sep.for
cavity
Pro Cal PU
do j = 1,ny
hy4 = hy(j+1)+2.d0*hy(j)+hy(j-1)
do i = 0,nx
t = gm*p(i,j)/ro(i,j)
tp = gm*p(i+1,j)/ro(i+1,j)
tm = (t+tp)*0.5d0
rm = (ro(i,j) + ro(i+1,j))*0.5d0
pm = (p(i,j) + p(i+1,j))*0.5d0
uxm = (ux(i,j) + ux(i+1,j))*0.5d0
Loops ... Program ... File name Level ... Sequential time ... Parallel time ... Speed up... Communication... Parallel ...
=,, cavity cav_1_sep.for (58) 1.4 600.643616 609.380737 0.985662 2967.039551 no
j=1,ny cavity cav_1_sep.for (147) 1.4.5 116.019997 195.772003 0.592628 1065.375977 no
i=0,nx cavity cav_1_sep.for (149) 1.4.5.1 116.000000 195.751999 0.592587 1065.375977 yes
i=1,nx cavity cav_1_sep.for (130) 1.4.4 20.000000 187.844864 0.106471 755.202515 yes
j=1,ny cavity cav_1_sep.for (291) 1.4.11 108.020004 151.647995 0.712307 514.943970 no
i=0,nx cavity cav_1_sep.for (293) 1.4.11.1 108.000008 151.628006 0.712270 514.943970 yes
i=0,nx1 cavity cav_1_sep.for (60) 1.4.1 16.160400 102.586884 0.157529 432.970886 yes
i=1,nx cavity cav_1_sep.for (206) 1.4.7 116.019997 61.447998 1.888101 184.384003 no
j=0,ny cavity cav_1_sep.for (208) 1.4.7.1 116.000000 61.427998 1.888390 184.384003 yes
i=1,nx cavity cav_1_sep.for (344) 1.4.13 108.020004 31.560001 3.422687 0.000000 no
j=0,ny cavity cav_1_sep.for (346) 1.4.13.1 108.000008 31.540001 3.424223 0.000000 yes
i=1,nx cavity cav_1_sep.for (410) 1.4.16 44.000000 6.790580 6.479564 9.687840 yes
Output Decisions Variables Characteristics
Ready
Ln 149

Вопросы, замечания?

Спасибо!

Литература

- DVM-система. http://www.keldysh.ru/dvm
  http://www.dvm-system.org
- OpenMP Application Program Interface Version 3.0, May 2008.
  http://www.openmp.org/mp-documents/spec30.pdf
- MPI: A Message-Passing Interface Standard Version 2.2, September 2009. http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf

- Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие.-М.: Изд-во МГУ, 2009.
  http://parallel.ru/info/parallel/openmp/OpenMP.pdf
- Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие.-М.: Изд-во МГУ, 2004.
  http://parallel.ru/tech/tech_dev/MPI/mpibook.pdf
- Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.
- Э. Таненбаум, М. ван Стеен. Распределенные системы. Принципы и парадигмы. – СПб. Питер, 2003

Автор

Бахтин Владимир Александрович, кандидат физико-математических наук, заведующий сектором Института прикладной математики им. М.В. Келдыша РАН, доцент кафедры системного программирования факультета ВМК, МГУ им. М. В. Ломоносова
bakhtin@keldysh.ru

Инициализация и завершение MPI программ

Первой вызываемой функцией MPI должна быть функция:

int MPI_Init (int *agrc, char ***argv)

Для инициализации среды выполнения MPI-программы. Параметрами функции являются количество аргументов в командной строке и текст самой командной строки.

Последней вызываемой функцией MPI обязательно должна являться функция:

int MPI_Finalize (void)

Обратно

Определение количества и ранга процессов

Определение количества процессов в выполняемой параллельной программе осуществляется при помощи функции:

int MPI_Comm_size (MPI_Comm comm, int *size).

Для определения ранга процесса используется функция:

int MPI_Comm_rank (MPI_Comm comm, int *rank).

Обратно

Неблокирующие обмены данными между процессорами

Для передачи сообщения процесс-отправитель должен выполнить функцию:
int MPI_Isend(void *buf, int count, MPI_Datatype type, int dest,
int tag, MPI_Comm comm, MPI_Request *request),
где
• buf - адрес буфера памяти, в котором располагаются данные отправляемого сообщения,
• count - количество элементов данных в сообщении,
• type - тип элементов данных пересылаемого сообщения,
• dest - ранг процесса, которому отправляется сообщение,
• tag - значение-тег, используемое для идентификации сообщений,
• comm - коммуникатор, в рамках которого выполняется передача данных.
Для приема сообщения процесс-получатель должен выполнить функцию:
int MPI_Irecv(void *buf, int count, MPI_Datatype type, int source,
int tag, MPI_Comm comm, MPI_Status *status, MPI_Request *request),
где
•buf, count, type - буфер памяти для приема сообщения, назначение каждого отдельного параметра соответствует описанию в MPI_Send,
•source - ранг процесса, от которого должен быть выполнен прием сообщения,
•tag - тег сообщения, которое должно быть принято для процесса,
•comm - коммуникатор, в рамках которого выполняется передача данных,
•status - указатель на структуру данных с информацией о результате выполнения операции приема данных.

Обратно

MPI_Waitall

Ожидание завершения всех операций обмена осуществляется при помощи функции:

```
int MPI_Waitall(
int count,
        MPI_Request array_of_requests[],
        MPI_Status array_of_statuses[])
```

Обратно

MPI_Cart_create

Создание декартовой топологии (решетки) в MPI:

```
int MPI_Cart_create(MPI_Comm oldcomm, int ndims, int *dims, int *periods,
  int reorder, MPI_Comm *cartcomm),
```

где:

- **oldcomm - исходный коммуникатор,**
- **ndims - размерность декартовой решетки,**
- **dims - массив длины ndims, задает количество процессов в каждом измерении решетки,**
- **periods - массив длины ndims, определяет, является ли решетка периодической вдоль каждого измерения,**
- **reorder - параметр допустимости изменения нумерации процессов,**
- **cartcomm - создаваемый коммуникатор с декартовой топологией процессов.**

Обратно

MPI_Cart_shift

Функция:
int MPI_Card_shift(MPI_Comm comm, int dir, int disp, int *source, int *dst)
для получения номеров посылающего(source) и принимающего (dst) процессов в декартовой топологии коммуникатора (comm) для осуществления сдвига вдоль измерения dir на величину disp.

Обратно

MPI_Card_coords

Определение декартовых координат процесса по его рангу:

int MPI_Card_coords(MPI_Comm comm,int rank,int ndims,int *coords),

где:
•comm - коммуникатор с топологией решетки,
• rank - ранг процесса, для которого определяются декартовы координаты,
• ndims - размерность решетки,
• coords - возвращаемые функцией декартовы координаты процесса.

Обратно

MPI_Type_vector

Для снижения сложности в MPI предусмотрено несколько различных способов конструирования производных типов:

- ***Непрерывный*** **способ позволяет определить непрерывный набор элементов существующего типа как новый производный тип,**
- ***Векторный*** **способ обеспечивает создание нового производного типа как набора элементов существующего типа, между элементами которого существуют регулярные промежутки по памяти. При этом, размер промежутков задается в числе элементов исходного типа,**
- ***Индексный*** **способ отличается от векторного метода тем, что промежутки между элементами исходного типа могут иметь нерегулярный характер,**
- ***Структурный*** **способ обеспечивает самое общее описание производного типа через явное указание карты создаваемого типа данных.**

int MPI_Type_vector(int count, int blocklen, int stride, MPI_Data_type oldtype, MPI_Datatype *newtype),

где

- **count - количество блоков,**
- **blocklen - размер каждого блока,**
- **stride - количество элементов, расположенных между двумя соседними блоками**
- **oldtype - исходный тип данных,**
- **newtype - новый определяемый тип данных.**

Обратно

MPI_Type_commit

Перед использованием производный тип должен быть объявлен при помощи функции:

int MPI_Type_commit (MPI_Datatype *type)

При завершении использования производный тип должен быть аннулирован при помощи функции:

int MPI_Type_free (MPI_Datatype *type).

Обратно