



Автоматизация разработки параллельных программ

Бахтин Владимир Александрович

*Ассистент кафедры системного программирования
факультета ВМК, МГУ им. М. В. Ломоносова*

*К.ф.-м.н., зав. сектором 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, 2011 г.



Содержание

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



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

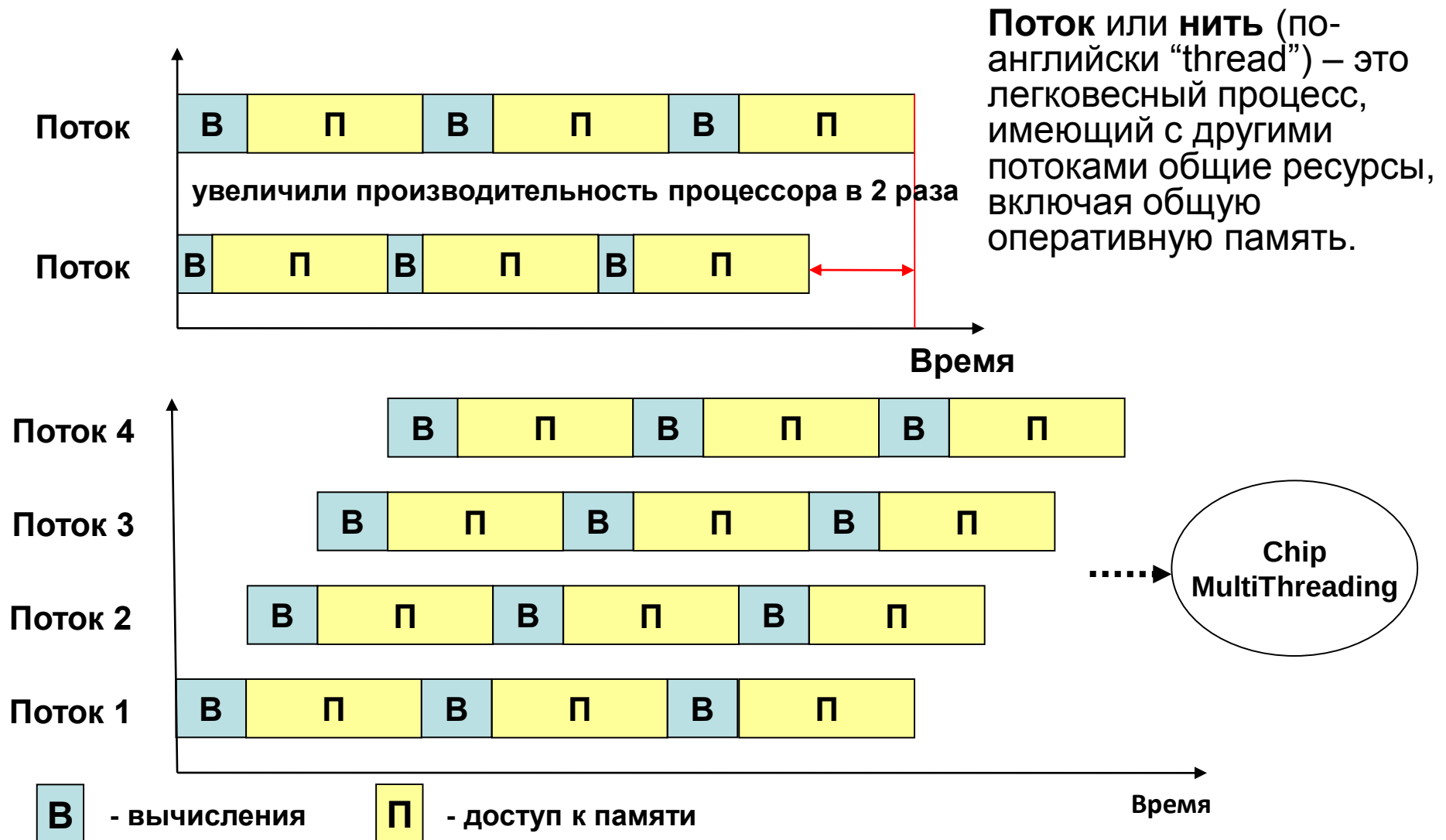
Latency to L3: 12 - 21 cycles

Latency to memory: 180 – 225 cycles

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



Тенденции развития современных процессоров





Тенденции развития современных процессоров

Суперкомпьютер Jaguar Cray XT5-HE Opteron Six Core 2.6 GHz

- ❑ Пиковая производительность - 2331 TFlop/s
- ❑ Число ядер в системе — 224 162
- ❑ Производительность на Linpack - 1759 TFlop/s (75.4% от пиковой)
- ❑ Энергопотребление комплекса - **6950.60 кВт**

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

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

Тенденции развития современных процессоров



AMD Opteron серии 6100 (Magny-Cours)

6176 SE 12 ядер @ 2,3 ГГц, 12 МБ L3 Cache

6136 8 ядер @ 2,4 ГГц, 12 МБ L3 Cache

встроенный контроллер памяти (4 канала памяти DDR3) до 42.7 GB/s

4 канала «точка-точка» с использованием HyperTransport 3.0 до 25.6 GB/s

Тенденции развития современных процессоров

Intel Xeon серии 5600 (Nehalem)

X5680 6 ядер @ 3,33 ГГц, 12 нитей, 12 МБ L3 Cache

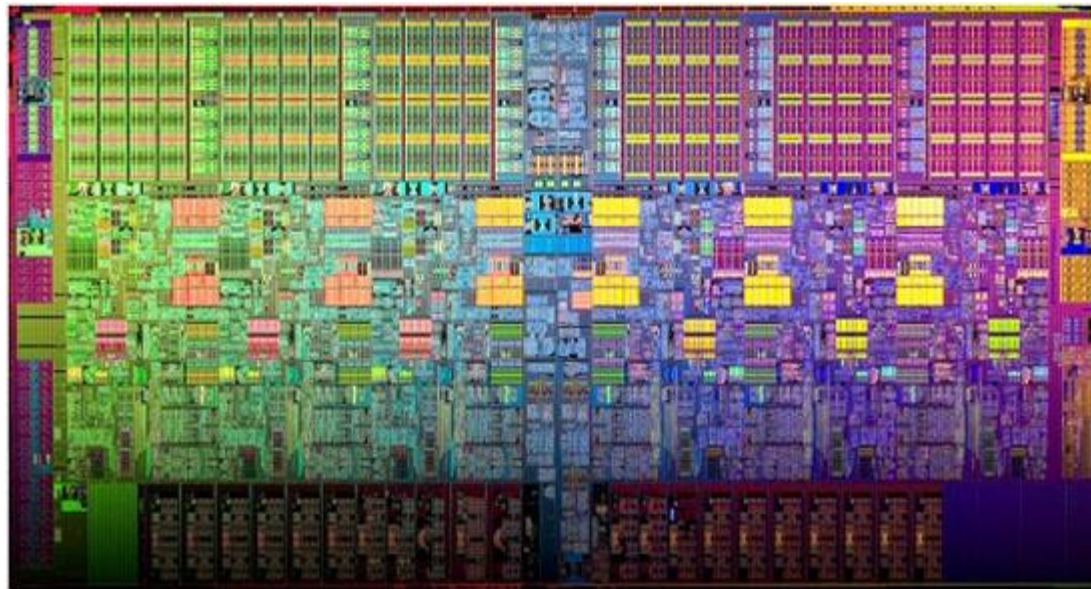
X5677 4 ядра @ 3,46 ГГц, 8 нитей, 12 МБ L3 Cache

Intel® Turbo Boost

Intel® QuickPath

Intel® Hyper-Threading

Intel® Intelligent Power



Тенденции развития современных процессоров



Intel Core i7 980X (Gulftown) 3,33 ГГц

- ❑ 6 ядер
- ❑ 12 потоков с технологией Intel Hyper-Threading
- ❑ 12 МБ кэш-памяти Intel Smart Cache
- ❑ встроенный контроллер памяти (3 канала памяти DDR3 1066 МГц)
- ❑ технология Intel QuickPath Interconnect

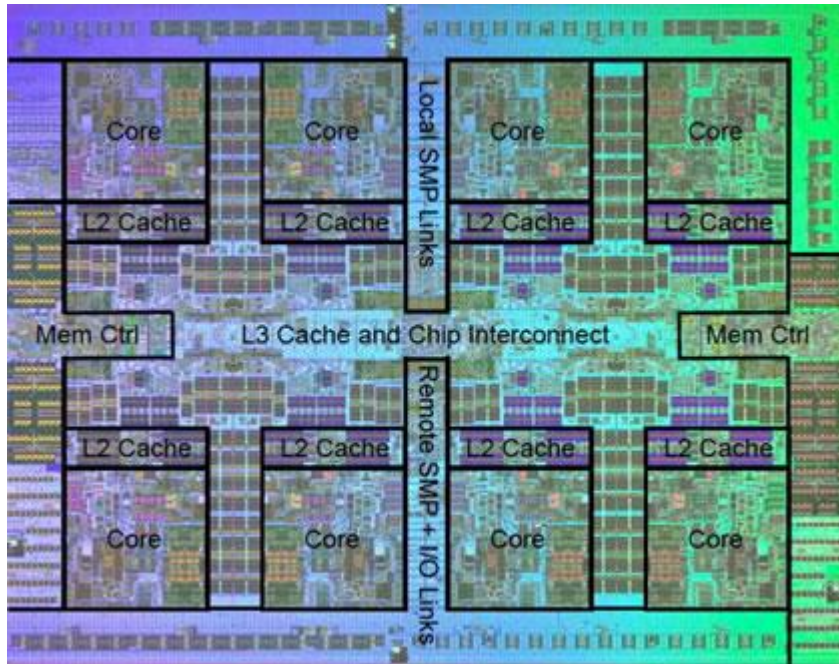
Тенденции развития современных процессоров



Intel Itanium 9350 (Tukwila) 1,73 ГГц

- ❑ 4 ядер
- ❑ 8 потоков с технологией Intel Hyper-Threading
- ❑ 24 МБ L3 кэш-памяти
- ❑ технология Intel QuickPath Interconnect
- ❑ технология Intel Turbo Boost

Тенденции развития современных процессоров

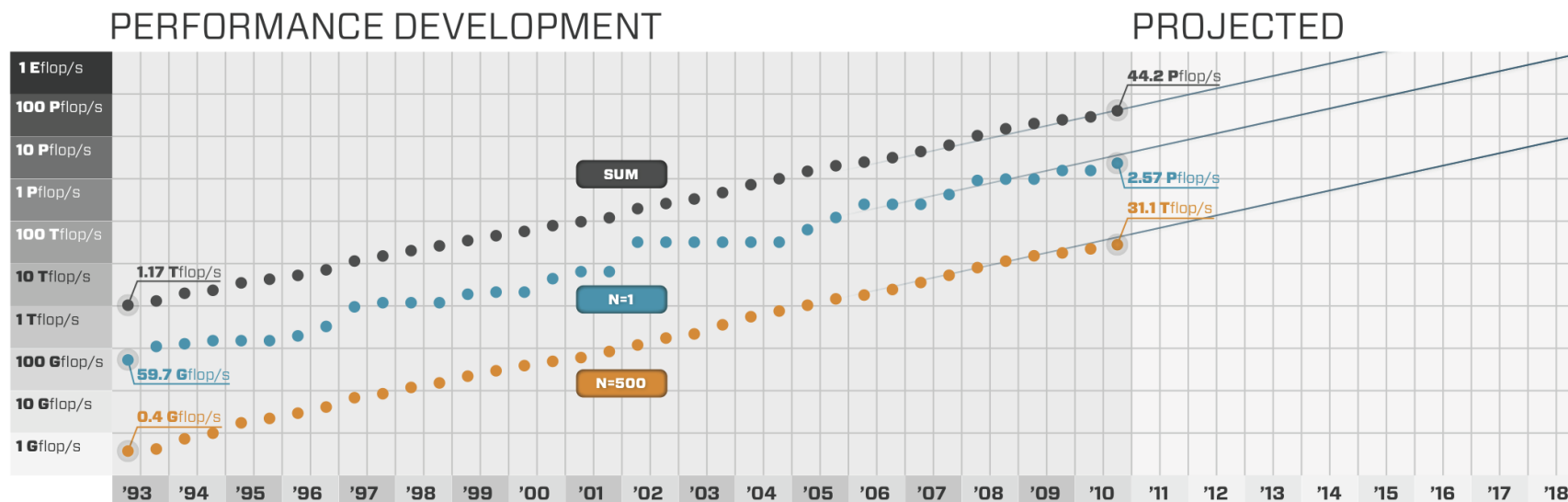


IBM Power7

- ❑ 3,5 - 4,0 ГГц
- ❑ 8 ядер x 4 нити Simultaneous MultiThreading
- ❑ L1 64КБ
- ❑ L2 256 КБ
- ❑ L3 32 МБ
- ❑ встроенный контроллер памяти

TOP500[®] NOVEMBER 2010

	NAME/MANUFACTURER/COMPUTER	LOCATION	COUNTRY	CORES	R _{max} Pflop/s
1	Tianhe-1A NUDT 6-core Intel X5670 2.93 GHz + Nvidia M2050 GPU w/custom interconnect	NUDT/NSCC/Tianjin	China	186,368	2.57
2	Jaguar Cray XT-5 6-core AMD 2.6 GHz w/custom interconnect	DOE/SC/ORNL	USA	224,162	1.76
3	Nebulae Dawning TC3600 Blade Intel X5650 2.67 GHz, NVidia Tesla C2050 GPU w/ Iband	NSCS	China	120,640	1.27
4	Tsubame 2.0 HP Proliant SL390s G7 nodes (Xeon X5670 2.93GHz) , NVIDIA Tesla M2050 GPU w/Iband	TiTech	Japan	73,278	1.19
5	Hopper Cray XE-6 12-core AMD 2.1 GHz w/custom interconnect	DOE/SC/LBNL	USA	153,408	1.05





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;
        }
}
```

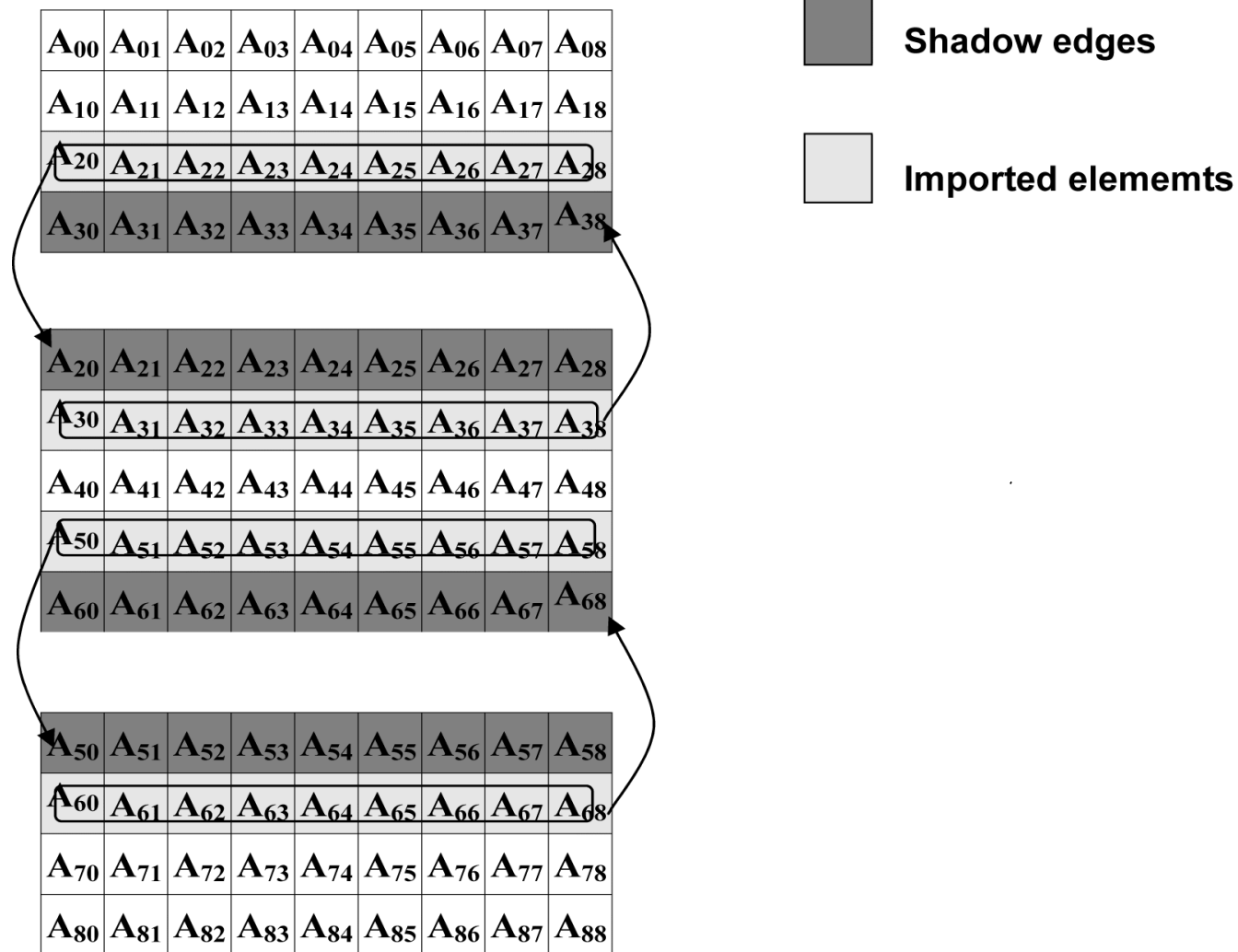


Алгоритм Якоби. Последовательная версия

```
/****** 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





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]);
```

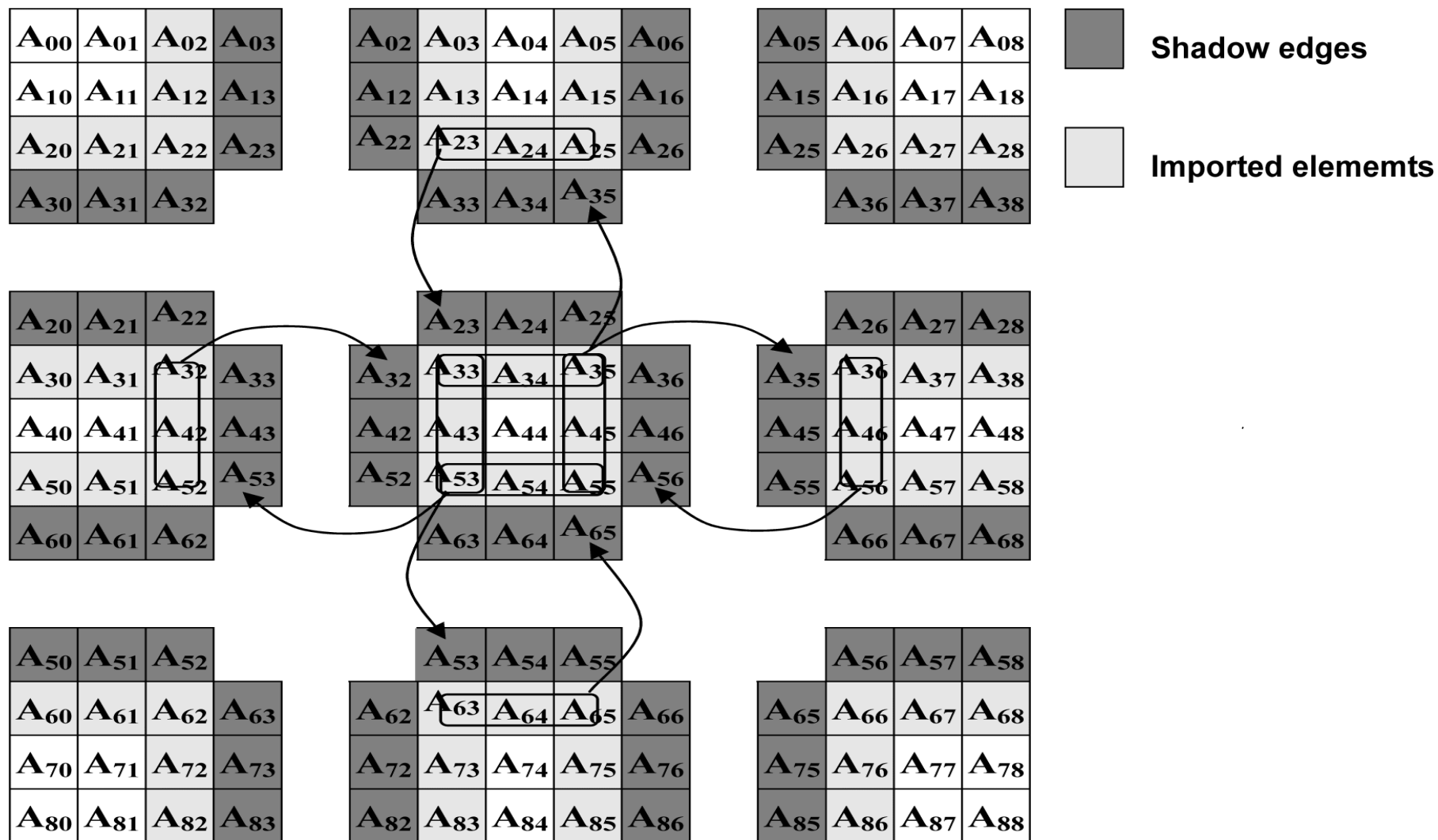


Алгоритм Якоби. MPI-версия

```
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);
```

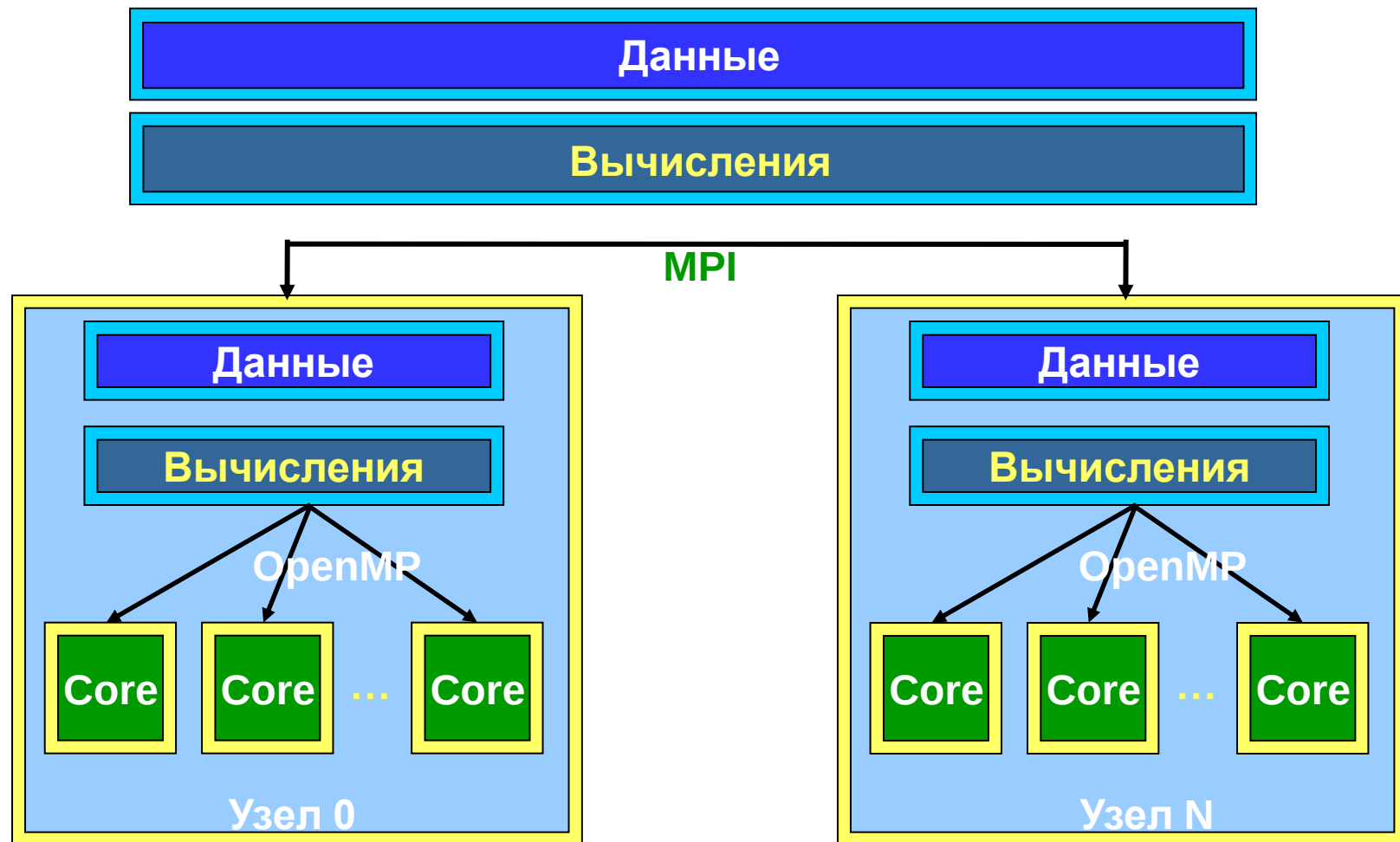


Алгоритм Якоби. MPI-версия

```
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;
}
```



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]);
```

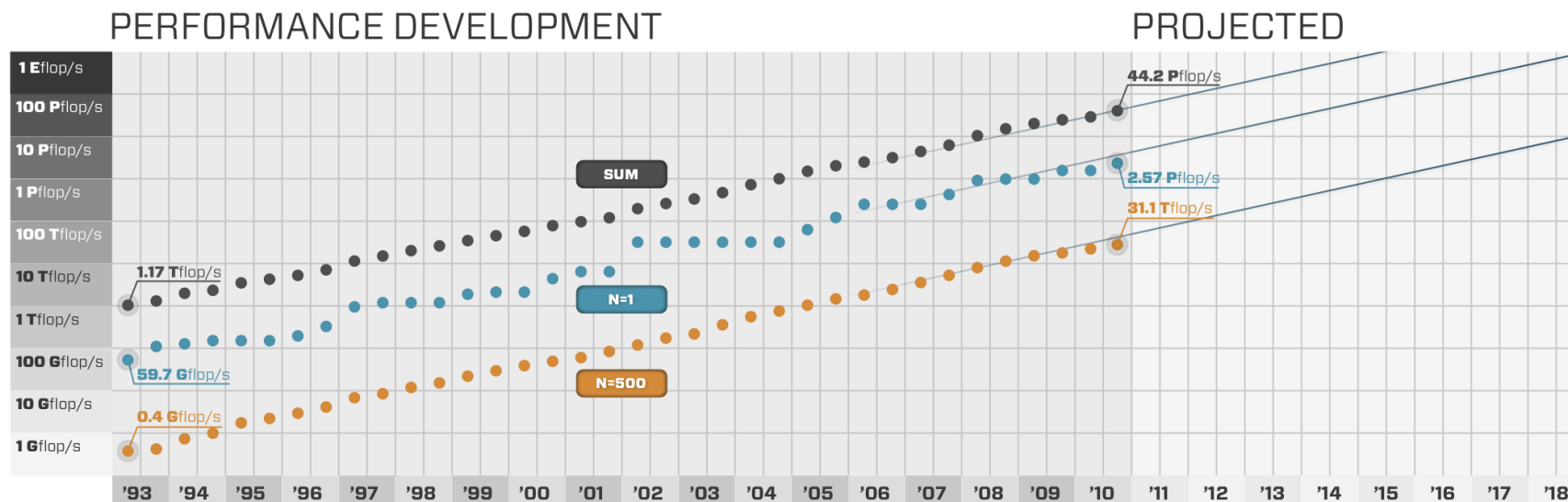


Алгоритм Якоби. MPI/OpenMP-версия

```
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;
}
```

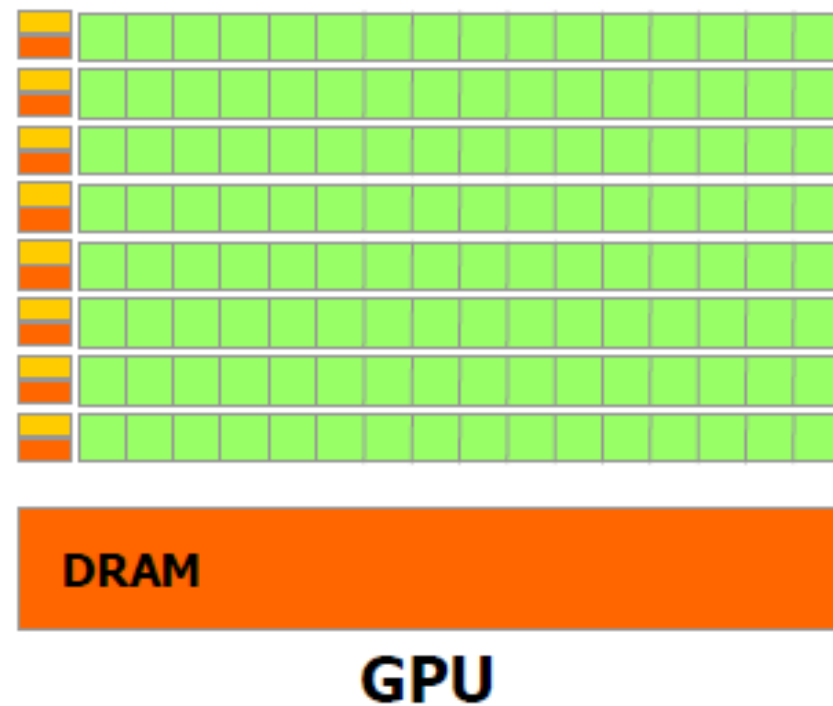
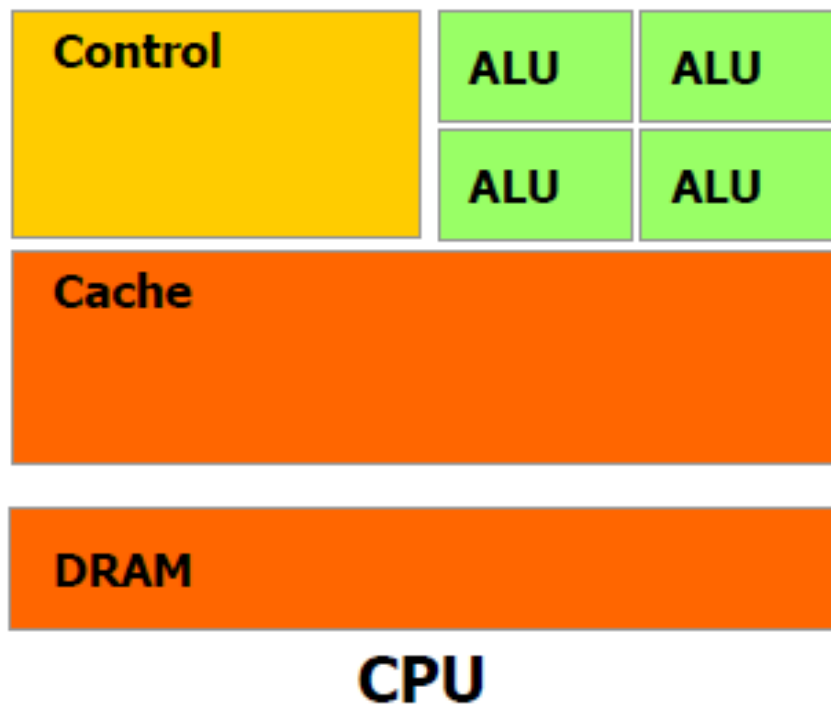
TOP500[®] NOVEMBER 2010

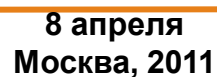
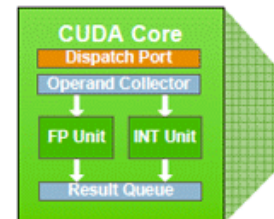
	NAME/MANUFACTURER/COMPUTER	LOCATION	COUNTRY	CORES	R _{max} Pflop/s
1	Tianhe-1A NUDT 6-core Intel X5670 2.93 GHz + Nvidia M2050 GPU w/custom interconnect	NUDT/NSCC/Tianjin	China	186,368	2.57
2	Jaguar Cray XT-5 6-core AMD 2.6 GHz w/custom interconnect	DOE/SC/ORNL	USA	224,162	1.76
3	Nebulae Dawning TC3600 Blade Intel X5650 2.67 GHz, NVidia Tesla C2050 GPU w/ Iband	NSCS	China	120,640	1.27
4	Tsubame 2.0 HP Proliant SL390s G7 nodes (Xeon X5670 2.93GHz) , NVIDIA Tesla M2050 GPU w/Iband	TiTech	Japan	73,278	1.19
5	Hopper Cray XE-6 12-core AMD 2.1 GHz w/custom interconnect	DOE/SC/LBNL	USA	153,408	1.05





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
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
```



DVM-система

DVM-система состоит из следующих компонент:

- Компилятор Fortran-DVM/OpenMP
- Компилятор C-DVM
- Библиотека поддержки LIB-DVM
- 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-DVM = Язык Си + специальные макросы

Fortran-DVM/OpenMP = Язык Фортран 95 + специальные комментарии

- Специальные комментарии и макросы 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)
CDVM$ DISTRIBUTE (BLOCK, BLOCK) :: A
CDVM$ ALIGN B(I,J) WITH A(I,J)
PRINT *, '***** TEST_JACOBI *****'
DO IT = 1, ITMAX
CDVM$ 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
ENDDO
```



Алгоритм Якоби. DVM-версия

```
CDVM$  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
END
```

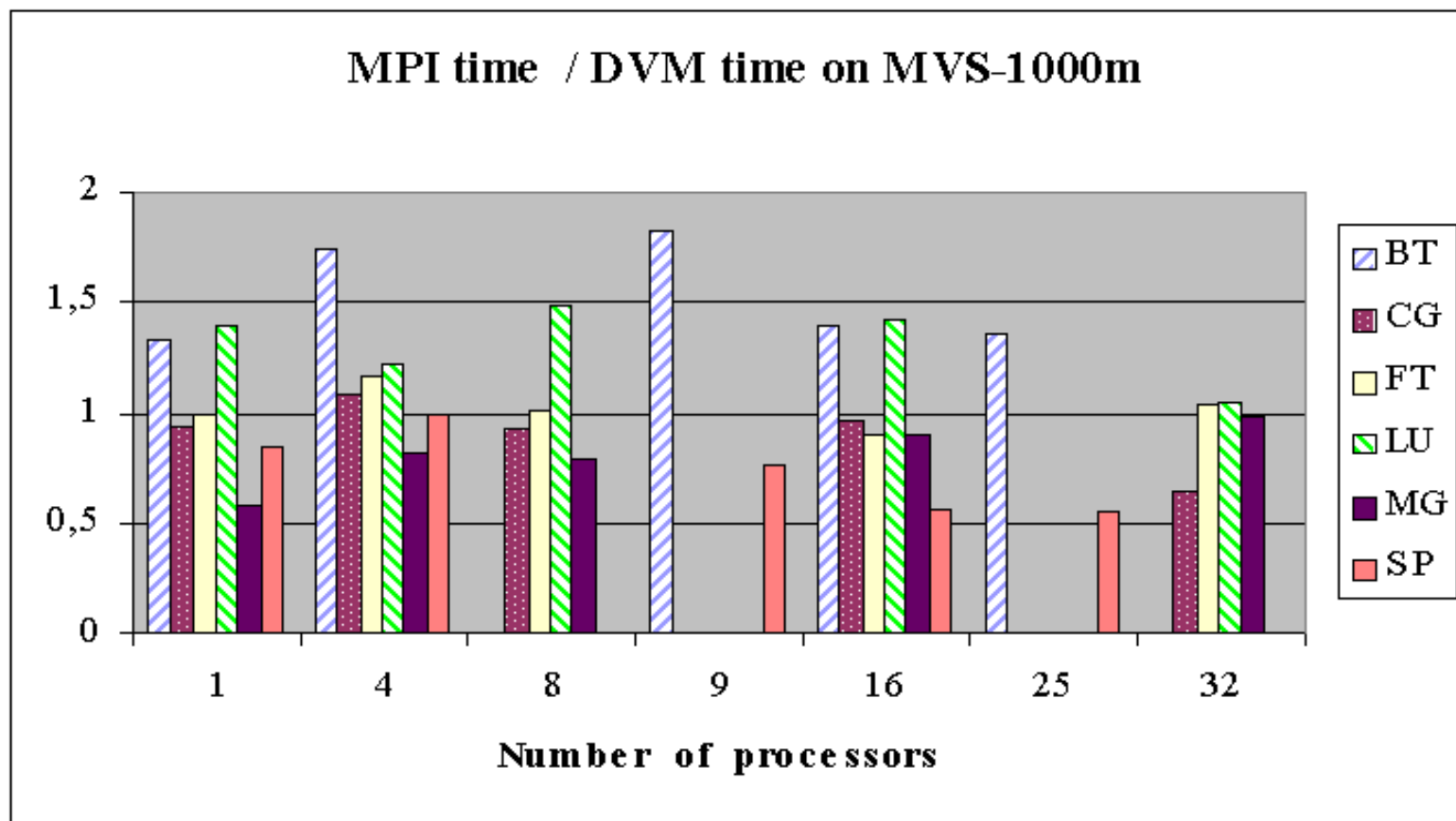


Тесты NAS

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

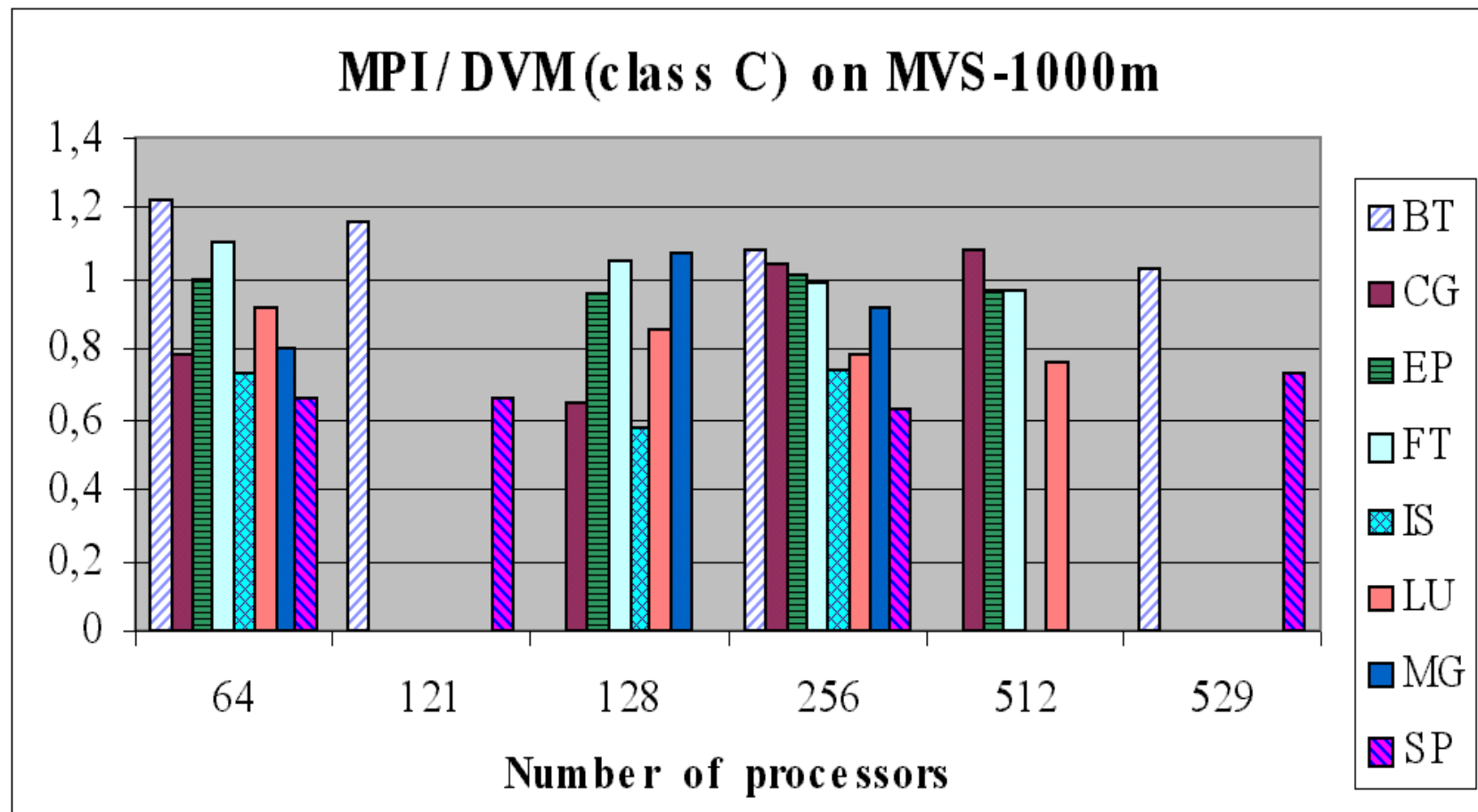


Тесты NAS





Тесты NAS





Алгоритм Якоби. DVM/OpenMP-версия

```
PROGRAM JAC_OpenMP_DVM
PARAMETER (L=4096, ITMAX=100)
REAL A(L,L), B(L,L)
CDVM$ DISTRIBUTE (BLOCK, BLOCK) :: A
CDVM$ ALIGN B(I,J) WITH A(I,J)
PRINT *, '***** TEST_JACOBI *****'
DO IT = 1, ITMAX
CDVM$ PARALLEL (J,I) ON A(I, J)
C$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/ОpenMP-версия

```
CDVM$  PARALLEL (J,I) ON B(I, J), SHADOW_RENEW (A)
C$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
END
```



Алгоритм Якоби. DVM/GPU-версия

```
PROGRAM  JAC_GPU_DVM
PARAMETER  (L=4096, ITMAX=100)
REAL  A(L,L), B(L,L)

CDVM$  DISTRIBUTE  ( BLOCK, BLOCK) :: A
CDVM$  ALIGN B(I,J) WITH A(I,J)

PRINT *, '***** TEST_JACOBI *****'

C$ACC  DATA REGION COPYOUT(B), LOCAL (A)
DO IT = 1, ITMAX

C$ACC  REGION
CDVM$  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
ENDDO
```



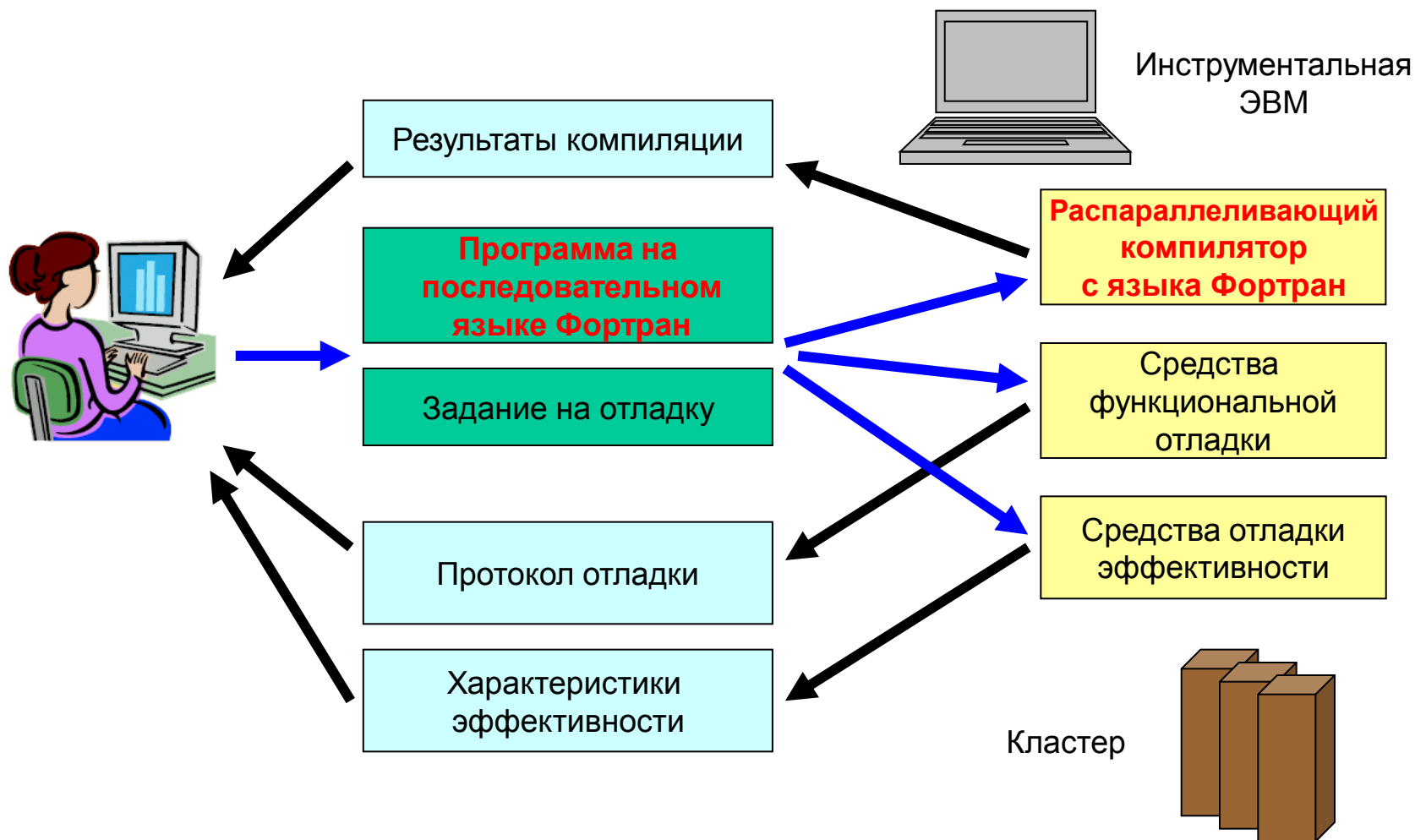
Алгоритм Якоби. DVM/GPU-версия

```
CDVM$  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
C$ACC  END REGION
ENDDO
C$ACC  END DATA REGION
END
```

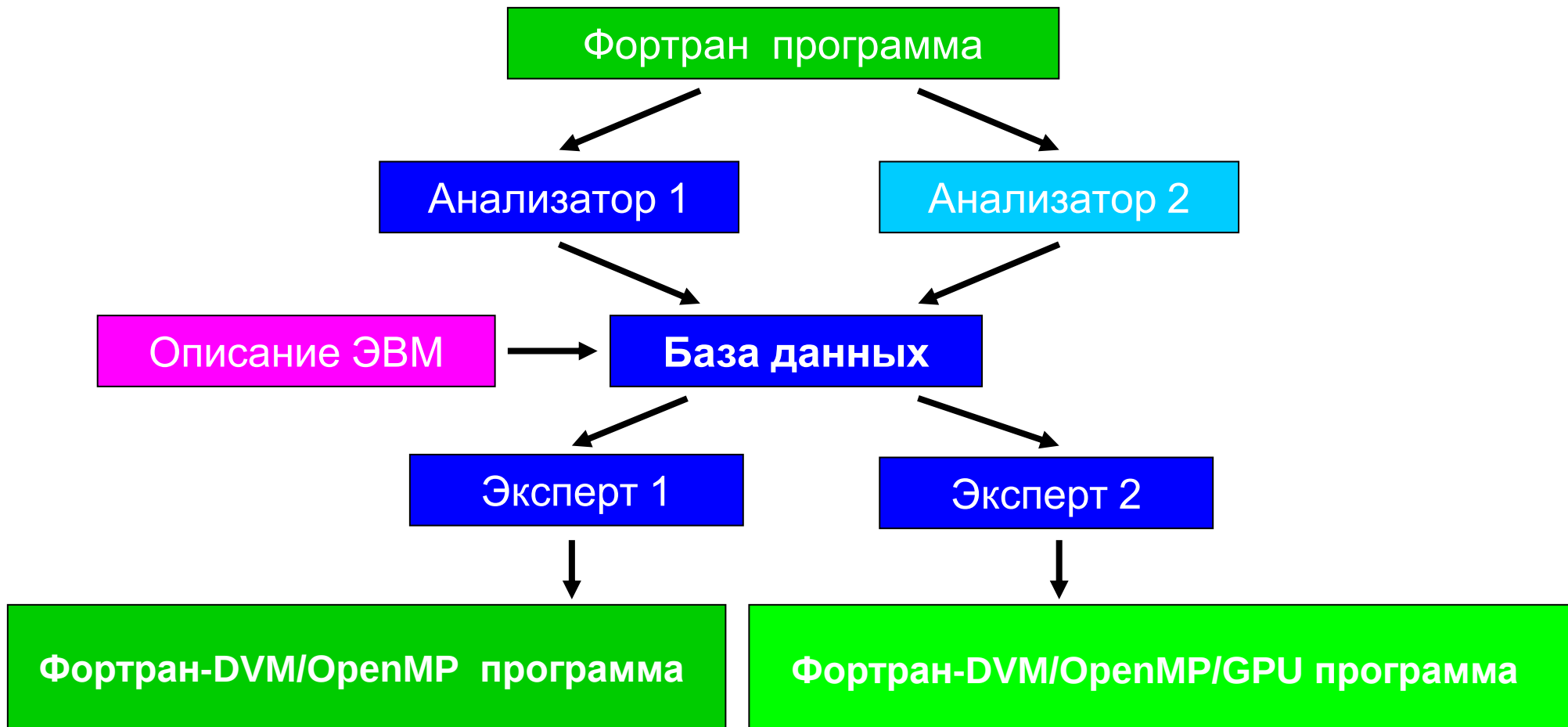


Система САПФОР





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	16 5 7 3 2 1	22 3 0 5 4	21 3 9 4	16 0 6 11	36 0 3 1
(начиная с одномерных массивов)					
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 - [luC_u.for]

FileViewToolsOptionsPanelsWindowHelp

Files and PU

luC_u.for

ludv2

read_input

domain

setcoeff

setbv

setiv

exact

Program units

```

do k = nz-1,2,-1
  do j = jend,jst,-1
    do i = iend,ist,-1

      r43 = ( 4.0d+00 / 3.0d+00 )
      c1345 = c1 * c3 * c4 * c5
      c34 = c3 * c4

      c
      c form the block daigonal
    
```

Loops / PU	Pro...	File name	Level ...	Seque...	Parall...	Speed...	Com...	Dependency	Read variab
k=2,nz-1	setiv	luC_u.for (594)	6.1	151.1...	19.128...	7.9024...	0.0000...		
l=1,isz3	ssor	luC_u.for (1334)	9.1	106.2...	13.450...	7.9024...	0.0000...		
k=2,nz-1	blts	luC_u.for (2200)	11.1	65.53...	8.6304...	7.5935...	0.5590...	REG (rsd);	rsd(1,;j-1,k)
k=nz-1,2,-(1)	buts	luC_u.for (2797)	12.1	63.10...	8.7710...	7.1951...	0.1269...	REG (rsd);	rsd(1,;j+1,k)
k=2,nz0-1	l2no...	luC_u.for (3380)	13.2	61.43...	9.7717...	6.2875...	8.4581...		
k=2,nz-1	error	luC_u.for (3432)	14.2	45.05...	5.7729...	7.8046...	1.1276...		
k=1,nz	rhs	luC_u.for (1569)	10.1	42.51...	5.3800...	7.9024...	0.0000...		
k=2,nz-1	rhs	luC_u.for (1608)	10.3	40.95...	5.2480...	7.8048...	0.0000...		
k=2,nz-1	rhs	luC_u.for (1793)	10.12	40.95...	5.2480...	7.8048...	0.3179...		
k=2,nz-1	rhs	luC_u.for (1979)	10.21	40.95...	5.2480...	7.8048...	0.0630...		



Вопросы, замечания?

Спасибо!



Литература

- ❑ DVM-система. <http://www.keldysh.ru/dvm>
- ❑ 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/mmpi-2.2/mmpi22-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,
ассистент 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.

[Обратно](#)

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.



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_Cart_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](#)