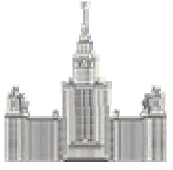


Гибридная модель программирования MPI/OpenMP

Бахтин Владимир Александрович
Доцент кафедры системного программированния факультета ВМК, МГУ им. М. В. Ломоносова
К.ф.-м.н., зав. сектором Института прикладной математики им М.В.Келдыша РАН

Москва, 2021 г.

Содержание

- ❑ Современные вычислительные системы
- ❑ Технология Intel Cluster OpenMP
- ❑ Гибридная модель MPI/OpenMP

TOP 500 The List.

PRESENTED BY

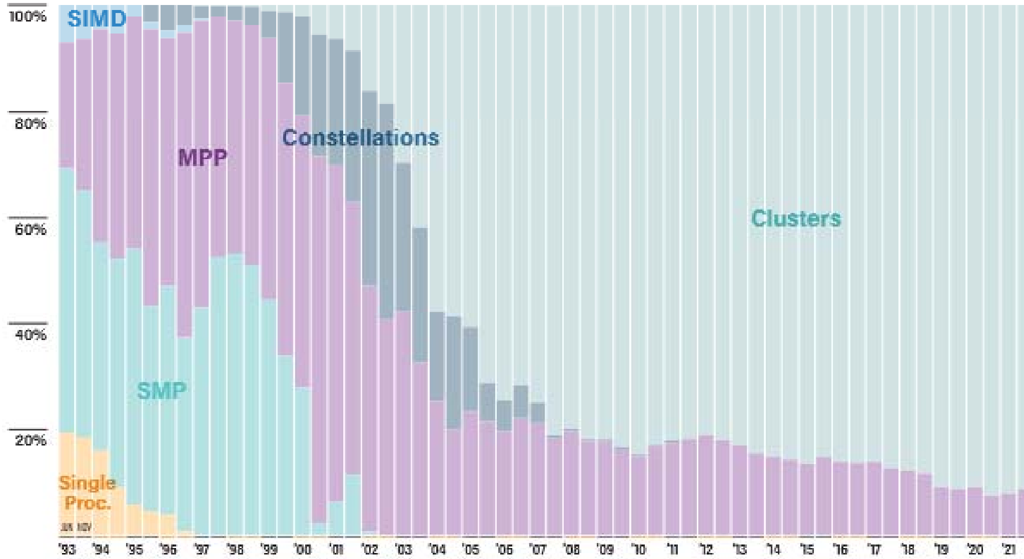
FIND OUT MORE AT

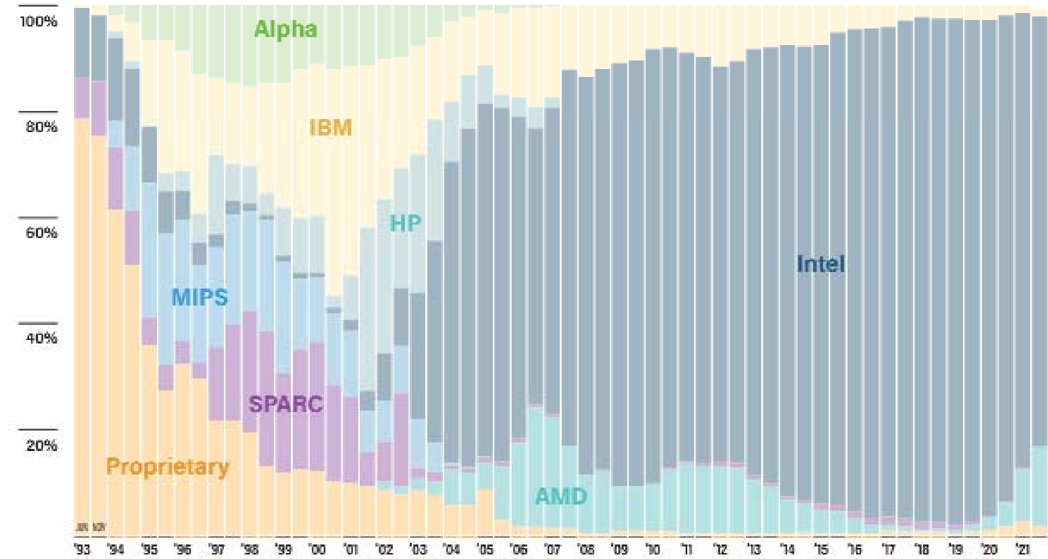
top500.org

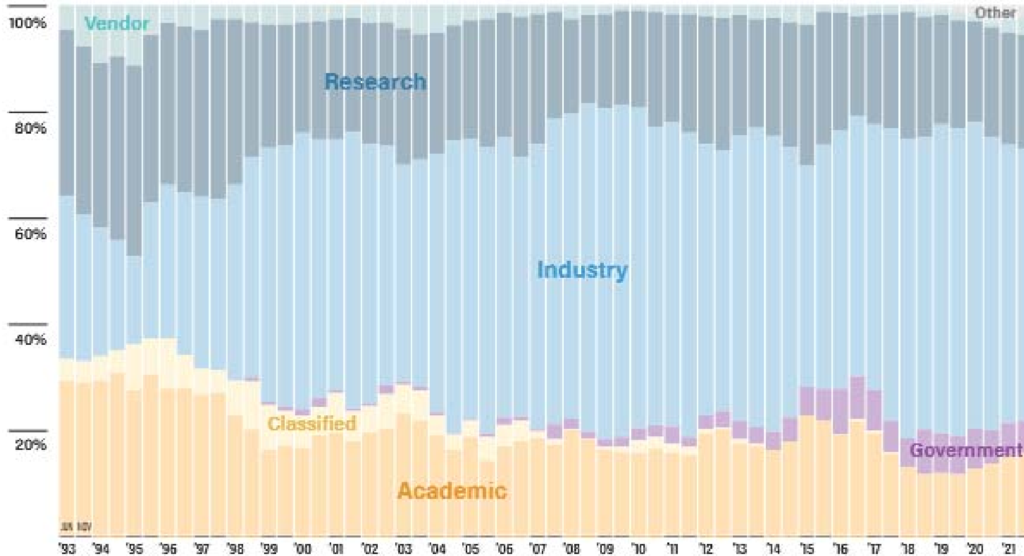
NOVEMBER 2021

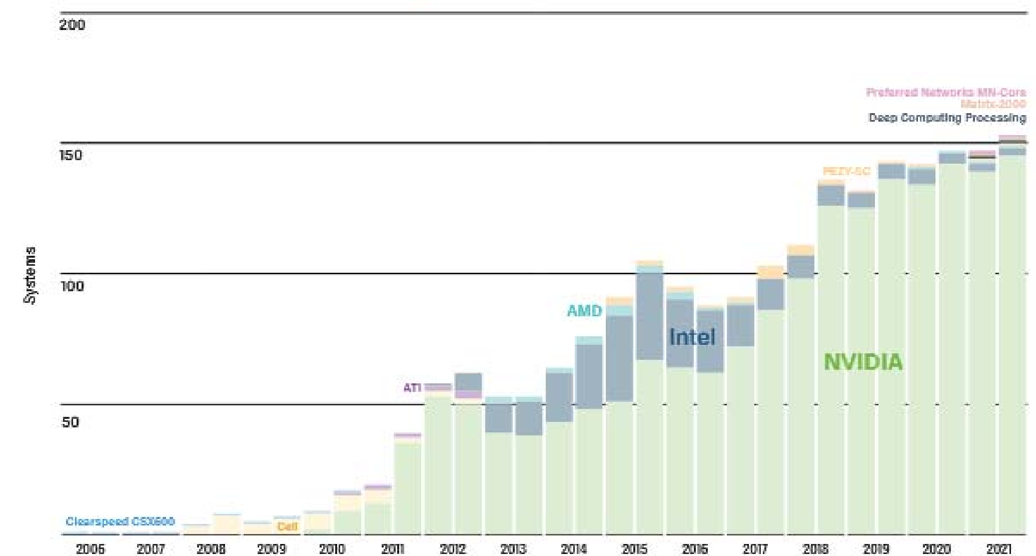
			SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	Fugaku	Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D	RIKEN R-CCS	Japan	7,630,848	442.0	29.9
2	Summit	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,414,592	148.6	10.1
3	Sierra	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/NNSA/LLNL	USA	1,572,480	94.6	7.44
4	Sunway TaihuLight	Shenwei SW26010 (260C, 1.45 GHz) Custom Interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
5	Perlmutter	HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10 (274 GB)	LBNL	USA	761,856	70.9	2.58

Performance Development

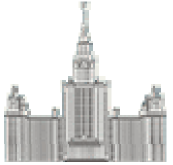






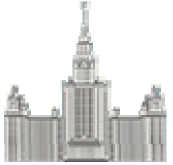


HPLINPACK *A Portable Implementation of the High Performance Linpack Benchmark for Distributed Memory Computers* ▸ MORE INFO AT **http://icl.utk.edu/hpl/**



Содержание

- Современные вычислительные системы
- Технология Intel Cluster OpenMP
- Гибридная модель MPI/OpenMP



Технология Intel Cluster OpenMP

- В 2006 году в Intel® компиляторах версии 9.1 появилась поддержка Cluster OpenMP.
- Технология Cluster OpenMP поддерживает выполнение OpenMP программ на нескольких вычислительных узлах, объединенных сетью.
- Базируется на программной реализации DSM (Thread Marks software by Rice University).

Для компилятора Intel® C++:

- icc -cluster-openmp *options source-file*
- icc -cluster-openmp-profile *options source-file*

Для компилятора Intel® Fortran:

- ifort -cluster-openmp *options source-file*
- ifort -cluster-openmp-profile *options source-file*

kmp_cluster.ini

--hostlist=home,remote --process_threads=2

Модель памяти в OpenMP

Общая память

Private-переменные
Кэш общих переменных
Нить
Threadprivate-переменные

Private-переменные
Кэш общих переменных
Нить
Threadprivate-переменные

Private-переменные
Кэш общих переменных
Нить
Threadprivate-переменные

Модель памяти в OpenMP

Общая память

static int i = 0;

i = 1

… = i + 3;

#pragma omp flush (i)

#pragma omp flush (i)

i = 0

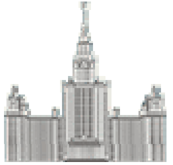
i = 1;

Нить 0

i = 1

Нить 1

… = i + 2; // ?



Консистентность памяти в OpenMP

Корректная последовательность работы нитей с переменной:

- Нить0 записывает значение переменной - write(var)
- Нить0 выполняет операцию синхронизации – flush (var)
- Нить1 выполняет операцию синхронизации – flush (var)
- Нить1 читает значение переменной – read (var)

Директива flush:

#pragma omp flush *[(list)]* - для Си

!$omp flush *[(list)]* - для Фортран

Консистентность памяти в OpenMP

1. Если пересечение множеств переменных, указанных в операциях flush, выполняемых различными нитями не пустое, то результат выполнения операций flush будет таким, как если бы эти операции выполнялись в некоторой последовательности (единой для всех нитей).
2. Если пересечение множеств переменных, указанных в операциях flush, выполняемых одной нитью не пустое, то результат выполнения операций flush, будет таким, как если бы эти операции выполнялись в порядке определяемом программой.
3. Если пересечение множеств переменных, указанных в операциях flush, пустое, то операции flush могут выполняться независимо (в любом порядке).

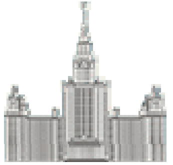
Консистентность памяти в OpenMP

#pragma omp flush *[(список переменных)]*

По умолчанию все переменные приводятся в консистентное состояние (**#pragma omp flush**):

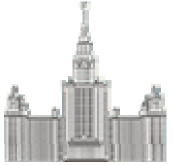
- при барьерной синхронизации;
- при входе и выходе из конструкций **parallel**, **critical** и **ordered;**
- при выходе из конструкций распределения работ (**for**, **single**, **sections**, **workshare**), если не указана клауза **nowait;**
- при вызове **omp_set_lock** и **omp_unset_lock;**
- при вызове **omp_test_lock**, **omp_set_nest_lock**, **omp_unset_nest_lock** и **omp_test_nest_lock**, если изменилось состояние семафора.

При входе и выходе из конструкции **atomic** выполняется **#pragma omp flush(x)**, где x – переменная, изменяемая в конструкции **atomic.**



Преимущества Intel Cluster OpenMP

- Упрощает распределение последовательного или OpenMP кода по узлам.
- Позволяет использовать одну и ту же программу для последовательных, многоядерных и кластерных систем.
- Требует совсем незначительного изменения кода программы, что упрощает отладку.
- Позволяет слегка измененной OpenMP-программе выполняться на большем числе процессоров без вложений в аппаратную составляющую SMP.
- Представляет собой альтернативу MPI, которая может быть быстрее освоена и применена.



Директива SHARABLE

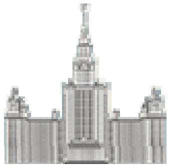
#pragma intel omp sharable (variable *[, variable …]*) – для Си и Си++

!dir$ omp sharable (variable *[, variable …]*) - для Фортрана

- определяют переменные, которые должны быть помещены в Distributed Virtual Shared Memory

В компиляторе существуют опции, которые позволяют изменить класс переменных, не изменяя текст программы:

- [-no]-clomp-sharable-argexprs
- [-no]-clomp-sharable-commons
- [-no]-clomp-sharable-localsaves
- [-no]-clomp-sharable-modvars



Работа с SHARABLE- переменными

Node 0

Pages: A B C

Write A[1]
Write C[1]

OMP Barrier
notices received and pro-pagaded by master thread
WriteNotice(0A,2A,2B,0C)
WriteNotice(0A,1B,0C)

by additional service thread
Calculate Diffs(A,TwinA)

Node 1

A B C

Write B[2]

OMP Barrier
WriteNotice(1B)
node page

Read A[1]

Page Fault
Diff Request(A)

Re-Read A[1]

Node 2

A B C

Write A[2]
Write B[1]

page A starts read-only
Page Fault
allocate (TwinA)
memcpy (TwinA := A)
Re-Write A[2]

OMP Barrier
WriteNotice(2A,2B)

by additional service thread
Calculate Diffs(A,TwinA)

Использование Intel Cluster OpenMP

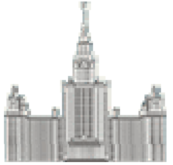
Целесообразно:

- если программа дает хорошее ускорение при использовании технологии OpenMP:

  Speedup = Time(1thread)/Time (n threads) = ~n
- если программа требует малой синхронизации
- данные в программе хорошо локализованы

Наиболее целесообразно для задач (RMS - recognition, mining, and synthesis):

- Обработка больших массивов данных
- Рендеринг в графике
- Поиск
- Распознавание образов
- Выделение последовательностей в генетике



Использование Intel Cluster OpenMP

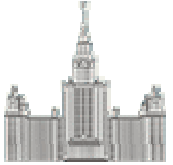
Cluster OMP % of Open MP
with speedups 16 procs

Percentage: 0, 20, 40, 60, 80, 100, 120, 140
Speedup: 0, 2, 4, 6, 8, 10, 12, 14

Particle Sim
m hd
cfd
structure
graph
crystallog

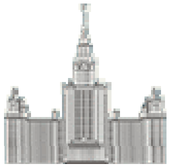
CLOMP % of OMP
OMP speedup
CLOMP speedup

1. a particle-simulation code
2. a magneto-hydro-dynamics code
3. a computational fluid dynamics code
4. a structure-simulation code
5. a graph-processing code
6. a linear solver code
7. an x-ray crystallography code

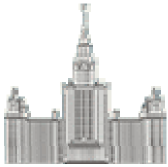


Содержание

- Современные вычислительные системы
- Технология Intel Cluster OpenMP
- Гибридная модель MPI/OpenMP



Гибридная модель MPI/OpenMP

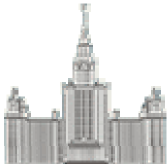


Достоинства использования в узлах OpenMP вместо MPI

- Возможность инкрементального распараллеливания.
- Упрощение программирования и эффективность на нерегулярных вычислениях, проводимых над общими данными.
- Ликвидация или сокращение дублирования данных в памяти, свойственного MPI-программам.
- Дополнительный уровень параллелизма на OpenMP реализовать проще, чем на MPI.

Преимущества OpenMP для многоядерных процессоров

- Объемы оперативной памяти и кэш памяти, приходящиеся в среднем на одно ядро, будут сокращаться – присущая OpenMP экономия памяти становится очень важна.
- Ядра используют общую Кэш-память, что требуется учитывать при оптимизации программы.



National Institute for Computational Sciences. University of Tennessee

Суперкомпьютер Kraken Cray XT5-HE Opteron Six Core 2.6 GHz

http://nics.tennessee.edu

Пиковая производительность – 1173.00 TFlop/s

Число ядер в системе — 112 800

Производительность на Linpack – 919.1 TFlop/s (78% от пиковой)

Updrage: замена 4-х ядерных процессоров AMD Opteron на 6-ти ядерные процессоры AMD Opteron

Результат: 6-ое место в TOP500 в июне 2009 - 3-ье место в TOP500 в ноябре 2009

https://www.youtube.com/watch?v=iTe8vb0QUNs

National Institute for Computational

Межведомственный Суперкомпьютерный Центр Российской Академии Наук

Суперкомпьютер MVS-100K

http://www.jscc.ru/

Пиковая производительность – 227,94 TFlop/s

Число ядер в системе — 13 004

Производительность на Linpack – 119,93 TFlop/s (52.7% от пиковой)

Updrage: замена 2-х ядерных процессоров Intel Xeon 53xx на 4-х ядерные процессоры Intel Xeon 54xx

Результат: 57-ое место в TOP500 в июне 2008 - 36-ое место в TOP500 в ноябре 2008

Oak Ridge National Laboratory

Суперкомпьютер Jaguar Cray XT5-HE Opteron Six Core 2.6 GHz

http://computing.ornl.gov

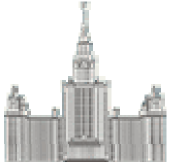
Пиковая производительность - 2627 TFlop/s

Число ядер в системе — 298 592

Производительность на Linpack - 1941 TFlop/s (73.87% от пиковой)

Updrage: замена 4-х ядерных процессоров AMD Opteron на 6-ти ядерные процессоры AMD Opteron

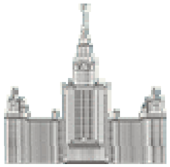
Результат: 2-ое место в TOP500 в июне 2009 - 1-ое место в TOP500 в ноябре 2009



Oak Ridge National Laboratory

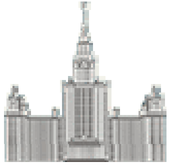
Jaguar Scheduling Policy

MIN Cores	MAX Cores	MAXIMUM WALL-TIME (HOURS)
120 000		24
40 008	119 999	24
5004	40 007	12
2004	5003	6
1	2 003	2



Cray MPI: параметры по умолчанию

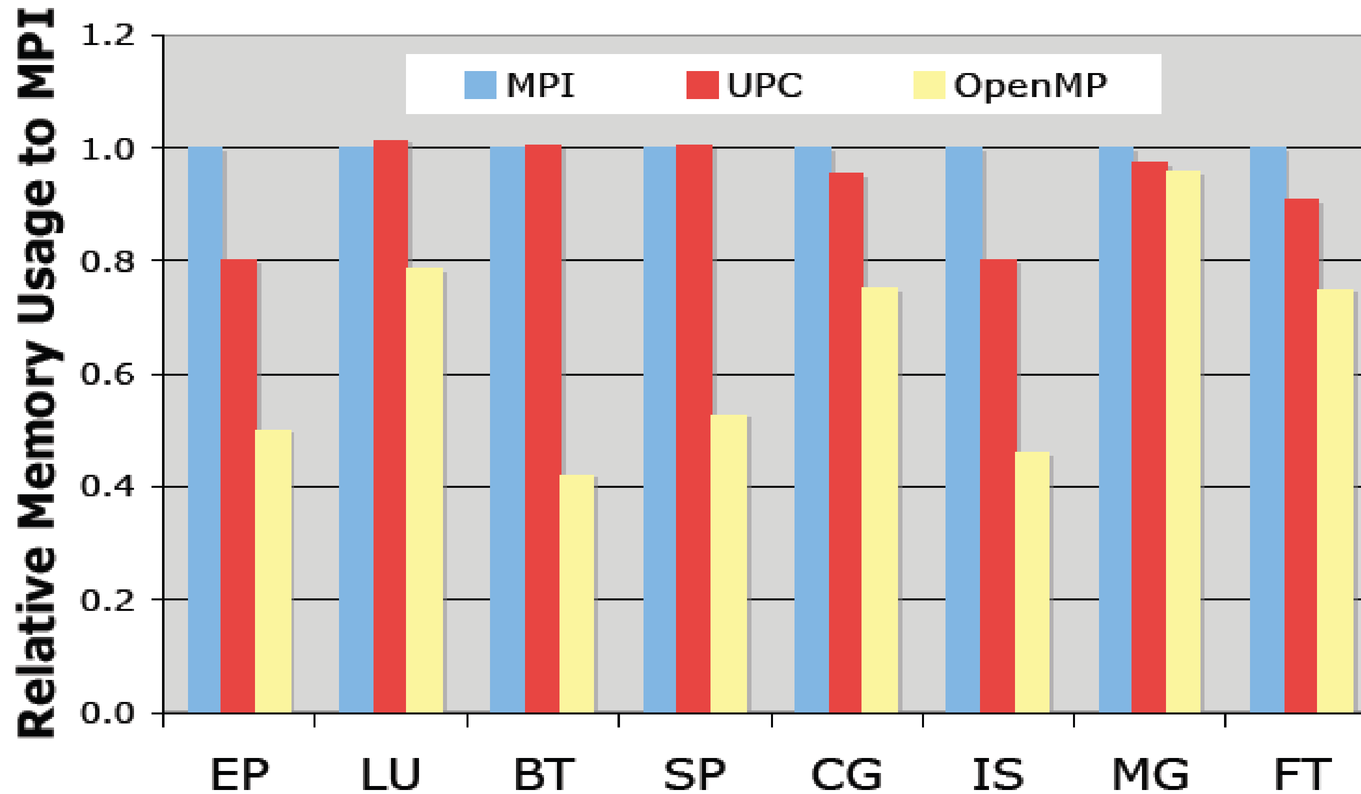
MPI Environment Variable Name	1,000 PEs	10,000 PEs	50,000 PEs	100,000 Pes
MPICH_MAX_SHORT_MSG_SIZE (This size determines whether the message uses the Eager or Randervous protocol)	128,000 Bytes	20,480	4096	2048
MPICH_UNEX_BUFFER_SIZE (The buffer allocated to hold the unexpected Eager data)	60 MB	60 MB	150 MB	260 MB
MPICH_PTL_UNEX_EVENTS (Portals generates two events for each unexpected message received)	20,480 events	22,000	110,000	220,000
MPICH_PTL_OTHER_EVENTS (Portals send-side and expected events)	2048 events	2500	12,500	25,000



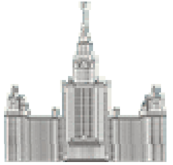
Тесты NAS

BT	3D Навье-Стокс, метод переменных направлений
CG	Оценка наибольшего собственного значения симметричной разреженной матрицы
EP	Генерация пар случайных чисел Гаусса
FT	Быстрое преобразование Фурье, 3D спектральный метод
IS	Параллельная сортировка
LU	3D Навье-Стокс, метод верхней релаксации
MG	3D уравнение Пуассона, метод Multigrid
SP	3D Навье-Стокс, Beam-Warning approximate factorization

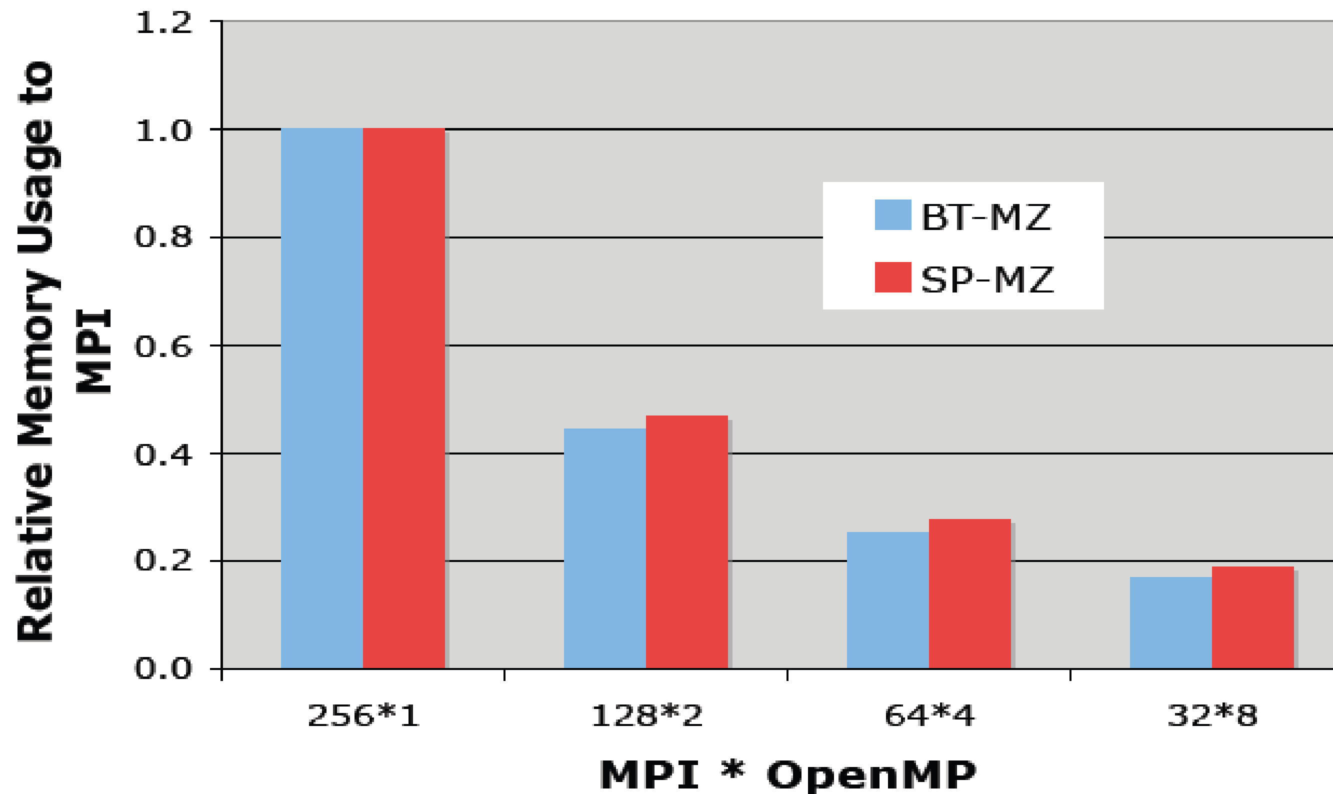
Тесты NAS



Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platforms
https://www.nersc.gov/assets/NERSC-Staff-Publications/2010/Cug2010Shan.pdf

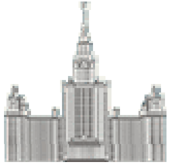


Тесты NAS

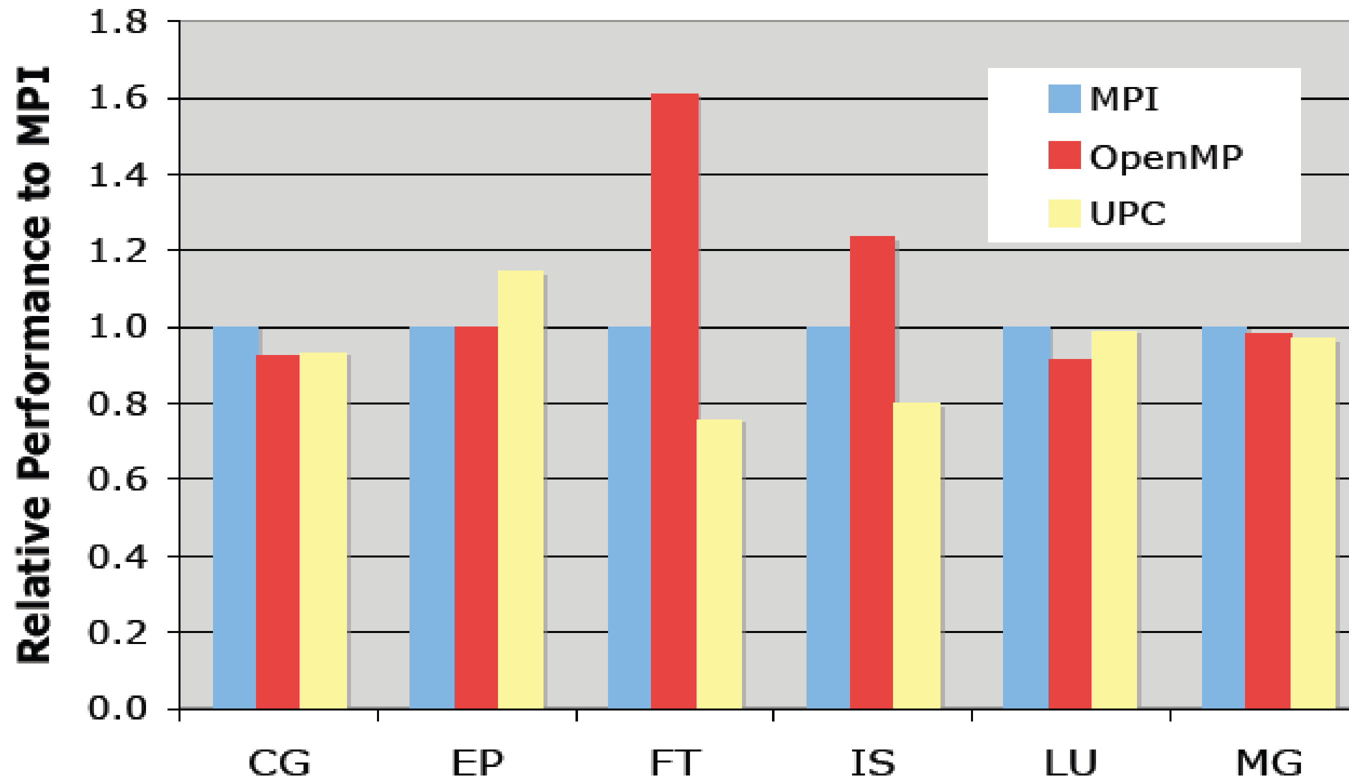


Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platforms

https://www.nersc.gov/assets/NERSC-Staff-Publications/2010/Cug2010Shan.pdf

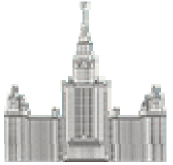


Тесты NAS



Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platforms

https://www.nersc.gov/assets/NERSC-Staff-Publications/2010/Cug2010Shan.pdf



Тесты NAS (класс B)

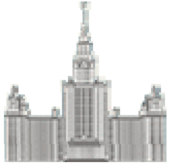
Время выполнения, в сек

	CG	EP	FT	IS	LU	MG
MPI	55,91	5,27	36,84	2,1	82,27	7,81
OpenMP	52,02	6,92	19,14	1,13	64,82	8,8

Суперкомпьютер MVS-100K

mvapich 1.2

Intel compiler, v. 10.1, -O3



Тесты NAS (класс B)

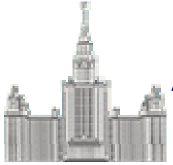
Время выполнения, в сек

	CG	EP	FT	IS	LU	MG
MPI	47,96	5,08	31,6	1,93	63,45	5,13
OpenMP	42,48	5,08	15,44	1,04	47,44	5,14

Суперкомпьютер СКИФ-МГУ «Чебышев»

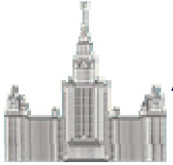
mvapich 1.2

Intel compiler, v. 11.1, -O3



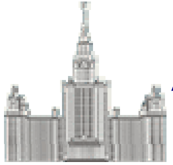
Алгоритм Якоби. Последовательная версия

```
/* 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

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}

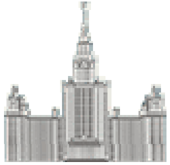
A_{50}	A_{51}	A_{52}	A_{53}	A_{54}	A_{55}	A_{56}	A_{57}	A_{58}
A_{60}	A_{61}	A_{62}	A_{63}	A_{64}	A_{65}	A_{66}	A_{67}	A_{68}
A_{70}	A_{71}	A_{72}	A_{73}	A_{74}	A_{75}	A_{76}	A_{77}	A_{78}
A_{80}	A_{81}	A_{82}	A_{83}	A_{84}	A_{85}	A_{86}	A_{87}	A_{88}

Shadow edges

Imported elememts

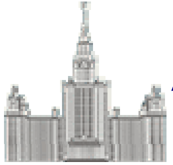
Алгоритм Якоби. MPI-версия

```
/* 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];
```



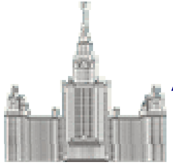
Алгоритм Якоби. MPI-версия

```
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");
```



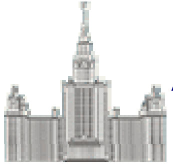
Алгоритм Якоби. MPI-версия

```
/* 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;
        }
```



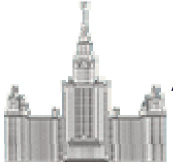
Алгоритм Якоби. MPI-версия

```
/******  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

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	

A_{02}	A_{03}	A_{04}	A_{05}	A_{06}
A_{12}	A_{13}	A_{14}	A_{15}	A_{16}
A_{22}	A_{23}	A_{24}	A_{25}	A_{26}
	A_{33}	A_{34}	A_{35}	

A_{05}	A_{06}	A_{07}	A_{08}
A_{15}	A_{16}	A_{17}	A_{18}
A_{25}	A_{26}	A_{27}	A_{28}
	A_{36}	A_{37}	A_{38}

A_{20}	A_{21}	A_{22}	
A_{30}	A_{31}	A_{32}	A_{33}
A_{40}	A_{41}	A_{42}	A_{43}
A_{50}	A_{51}	A_{52}	A_{53}
A_{60}	A_{61}	A_{62}	

	A_{23}	A_{24}	A_{25}	
A_{32}	A_{33}	A_{34}	A_{35}	A_{36}
A_{42}	A_{43}	A_{44}	A_{45}	A_{46}
A_{52}	A_{53}	A_{54}	A_{55}	A_{56}
	A_{63}	A_{64}	A_{65}	

	A_{26}	A_{27}	A_{28}
A_{35}	A_{36}	A_{37}	A_{38}
A_{45}	A_{46}	A_{47}	A_{48}
A_{55}	A_{56}	A_{57}	A_{58}
	A_{66}	A_{67}	A_{68}

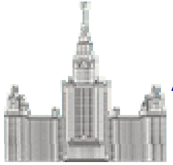
A_{50}	A_{51}	A_{52}	
A_{60}	A_{61}	A_{62}	A_{63}
A_{70}	A_{71}	A_{72}	A_{73}
A_{80}	A_{81}	A_{82}	A_{83}

	A_{53}	A_{54}	A_{55}	
A_{62}	A_{63}	A_{64}	A_{65}	A_{66}
A_{72}	A_{73}	A_{74}	A_{75}	A_{76}
A_{82}	A_{83}	A_{84}	A_{85}	A_{86}

	A_{56}	A_{57}	A_{58}
A_{65}	A_{66}	A_{67}	A_{68}
A_{75}	A_{76}	A_{77}	A_{78}
A_{85}	A_{86}	A_{87}	A_{88}

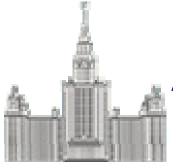
Shadow edges

Imported elememts



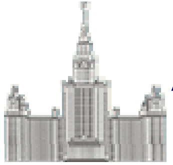
Алгоритм Якоби. MPI-версия

```
/*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];
```



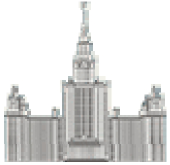
Алгоритм Якоби. MPI-версия

```
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 */
```



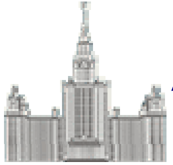
Алгоритм Якоби. MPI-версия

```
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);
```



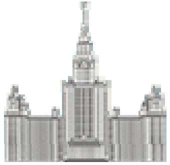
Алгоритм Якоби. MPI-версия

```
/* 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));
```



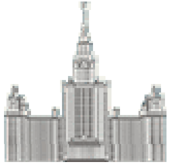
Алгоритм Якоби. MPI-версия

```
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);
```

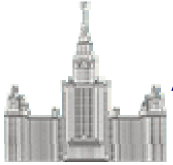


Алгоритм Якоби. MPI-версия

```
        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;
}
```

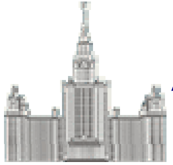
Алгоритм Якоби. MPI/OpenMP-версия

```
/* 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];
```



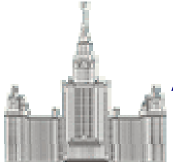
Алгоритм Якоби. MPI/OpenMP-версия

```
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");
```



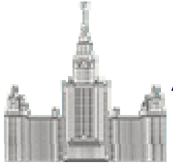
Алгоритм Якоби. MPI/OpenMP-версия

```
/* 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];
                    }
            }
```

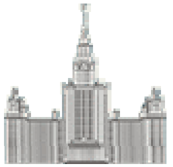


Алгоритм Якоби. MPI/OpenMP-версия

```
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;
}
```



Гибридная модель DVM/OpenMP

Данные

DVM

DVM

Данные

Вычисления

OpenMP

Core Core … Core

Узел 0

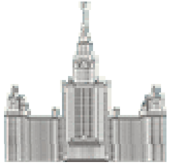
Данные

Вычисления

OpenMP

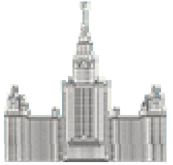
Core Core … Core

Узел N



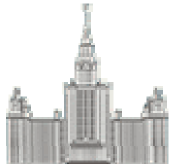
Алгоритм Якоби. DVM/OpenMP-версия

```
          PROGRAM    JAC_OpenMP_DVM
          PARAMETER    (L=1000,  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/OpenMP-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
```

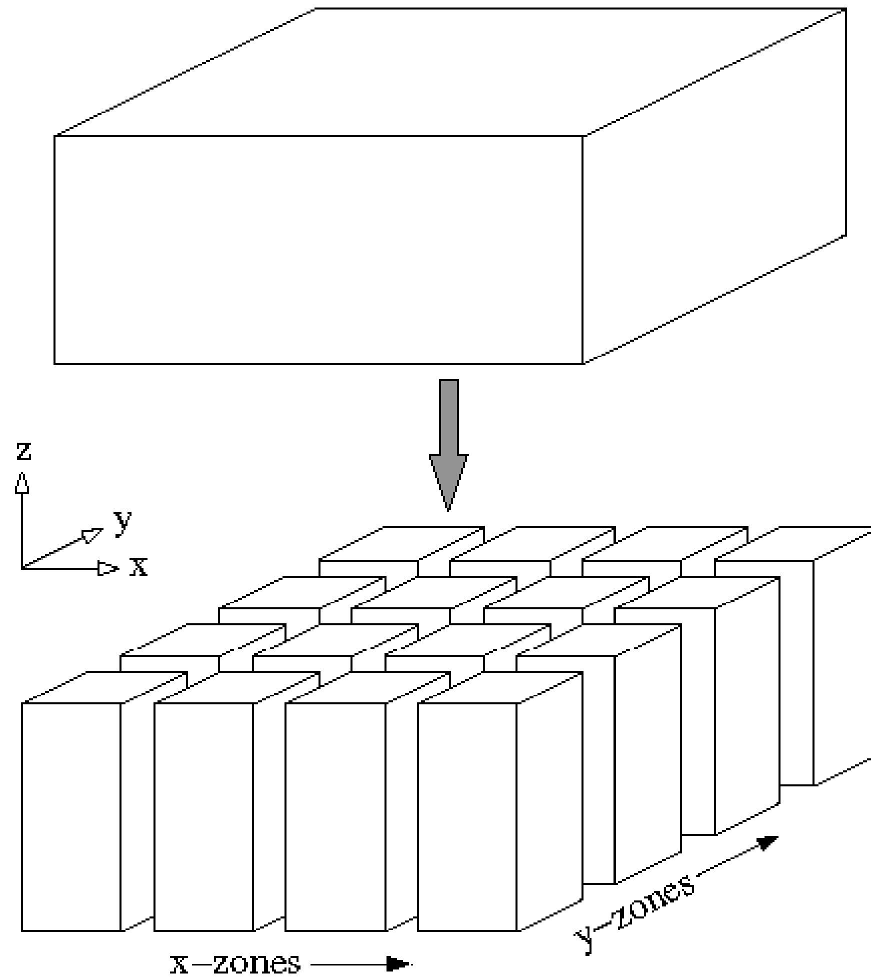


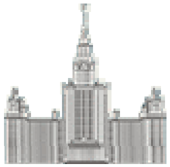
Тесты NAS MultiZone

BT (Block Tridiagonal Solver) 3D Навье-Стокс, метод переменных направлений
LU (Lower-Upper Solver) 3D Навье-Стокс, метод верхней релаксации
SP (Scalar PentadiagonalSolver) 3D Навье-Стокс, Beam-Warning approximate factorization

http://www.nas.nasa.gov/News/Techreports/2003/PDF/nas-03-010.pdf

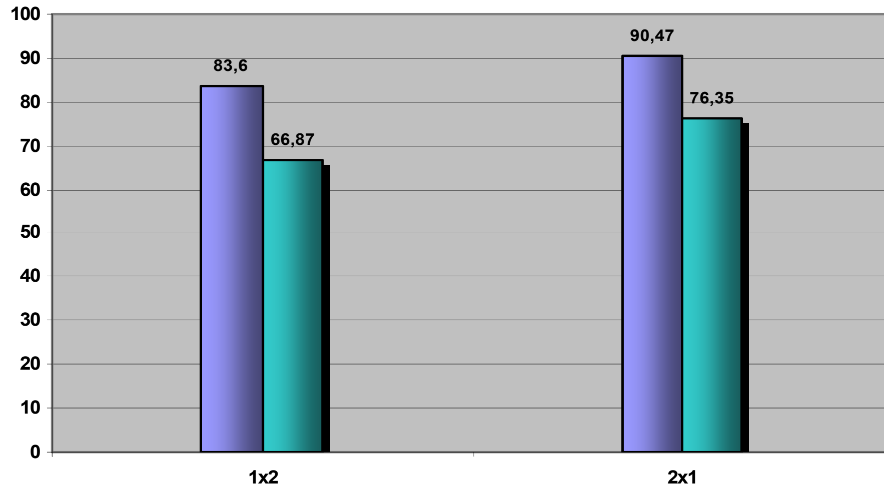
Тесты NAS MultiZone



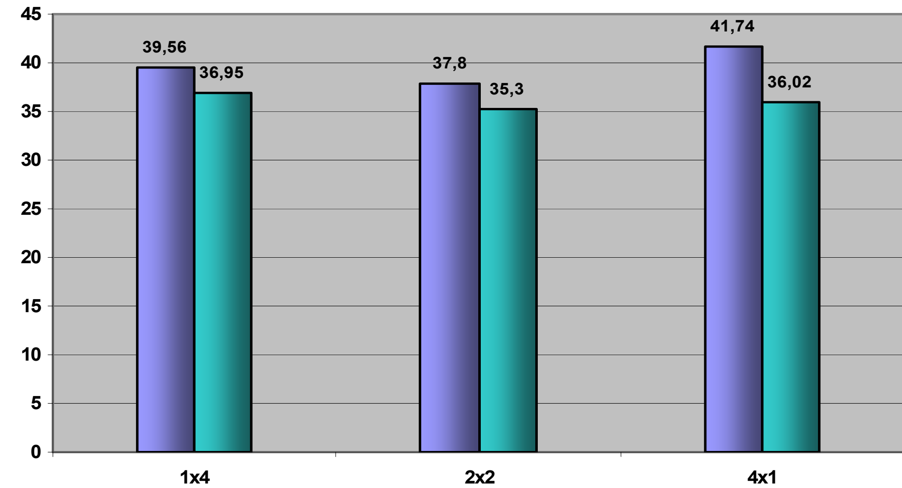


Тест SP-MZ (класс A) на IBM eServer pSeries 690 Regatta

2 процессора



4 процессора

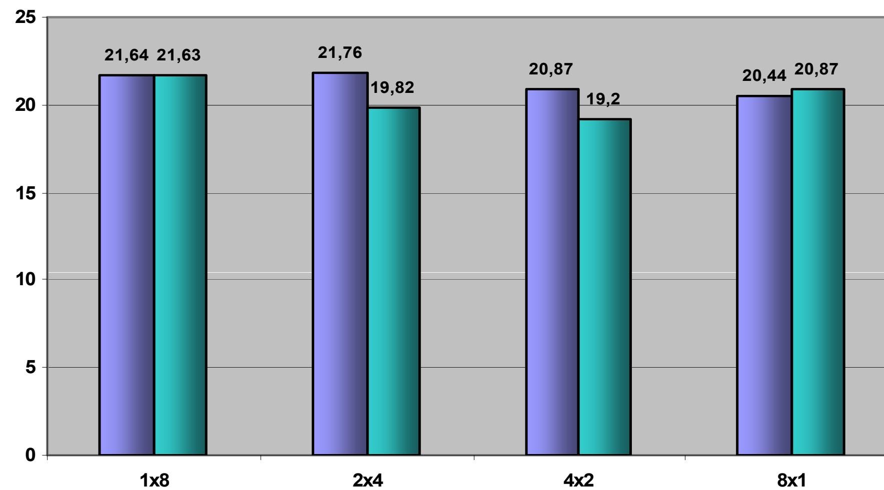


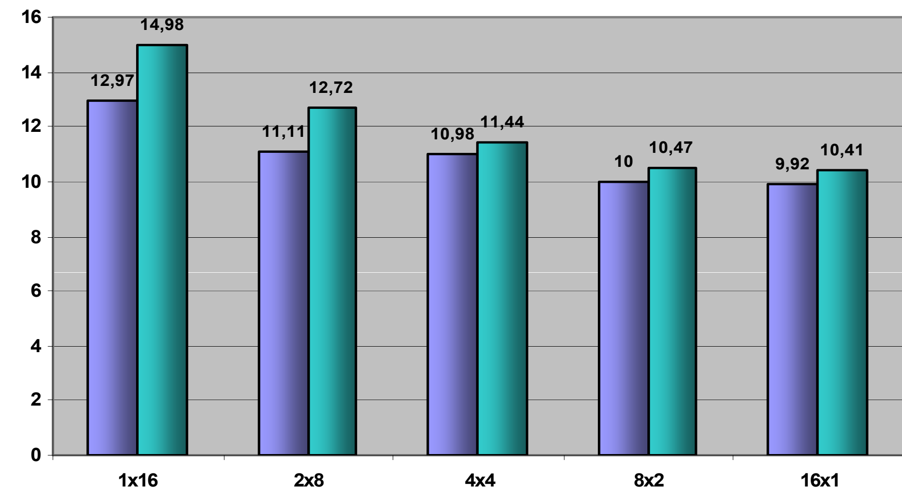
8 процессов

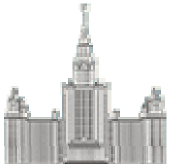
MPI

DVM

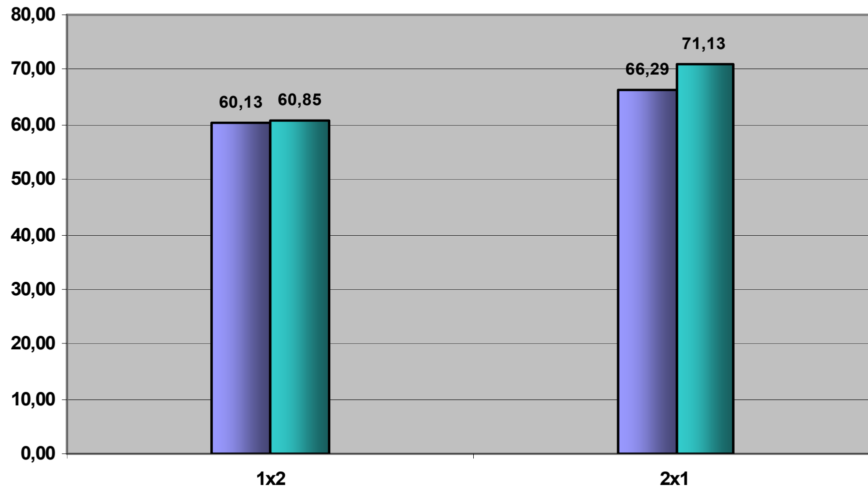
16 процессоров

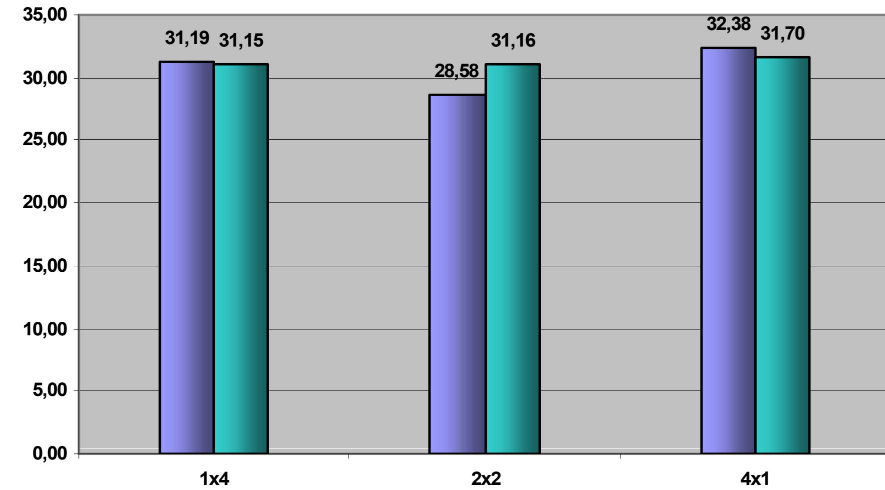






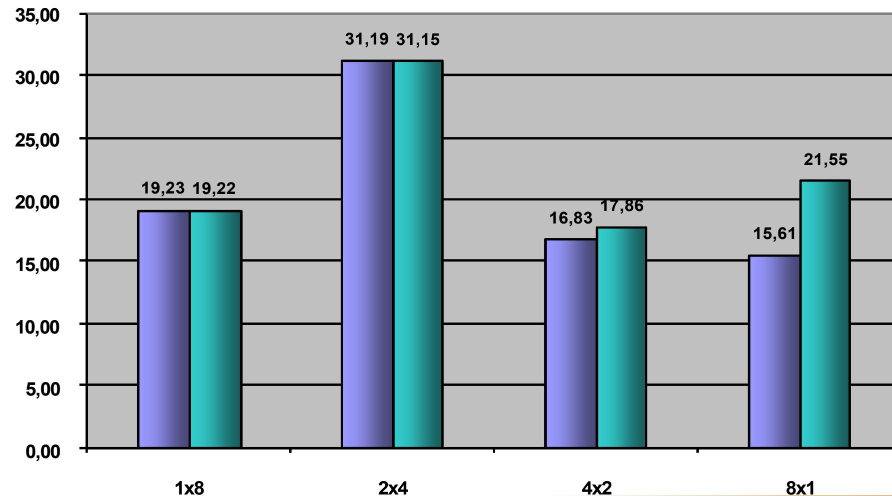
Тест LU-MZ (класс A) на IBM eServer pSeries 690 Regatta

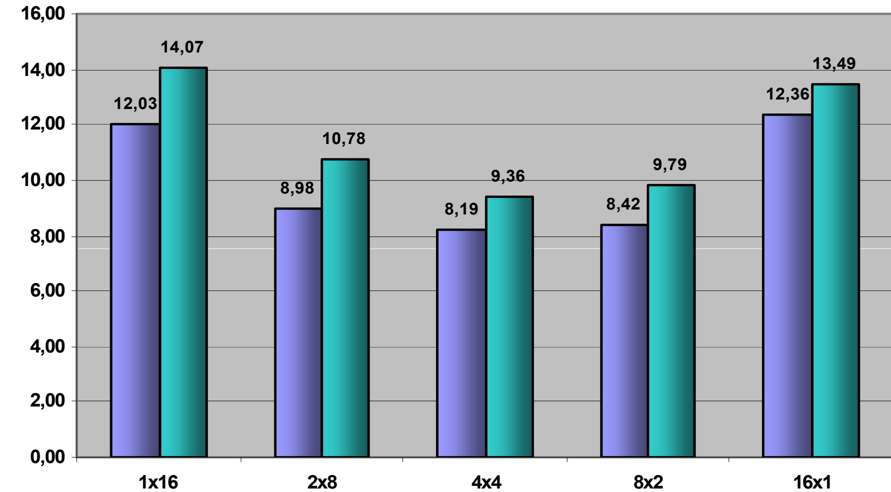


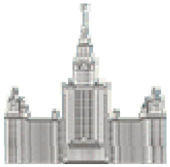


MPI

DVM

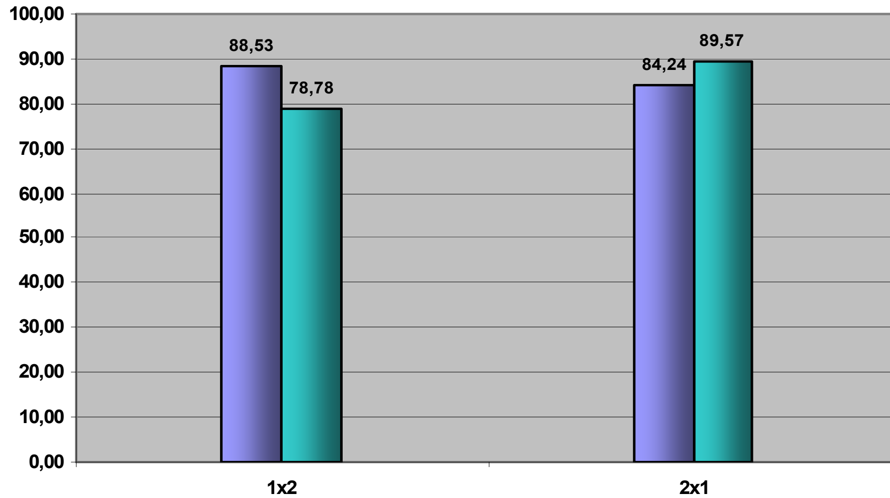


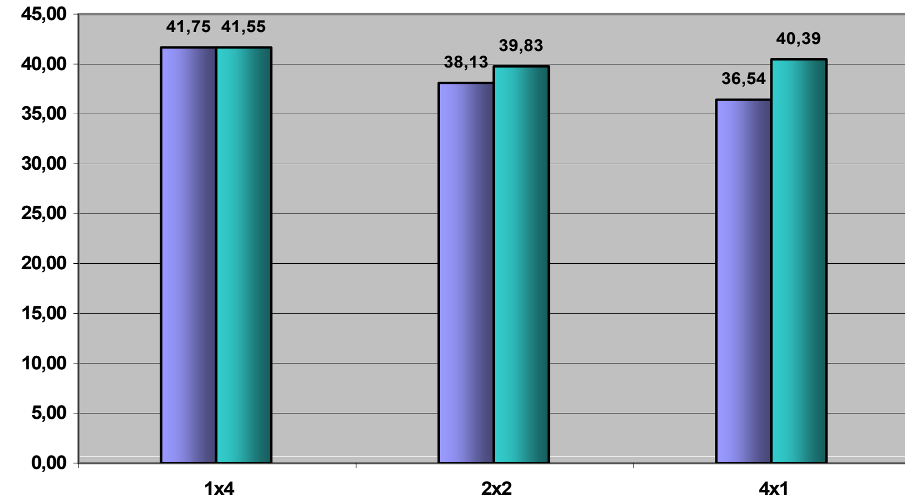




Тест BT-MZ (класс A) на IBM eServer pSeries 690 Regatta

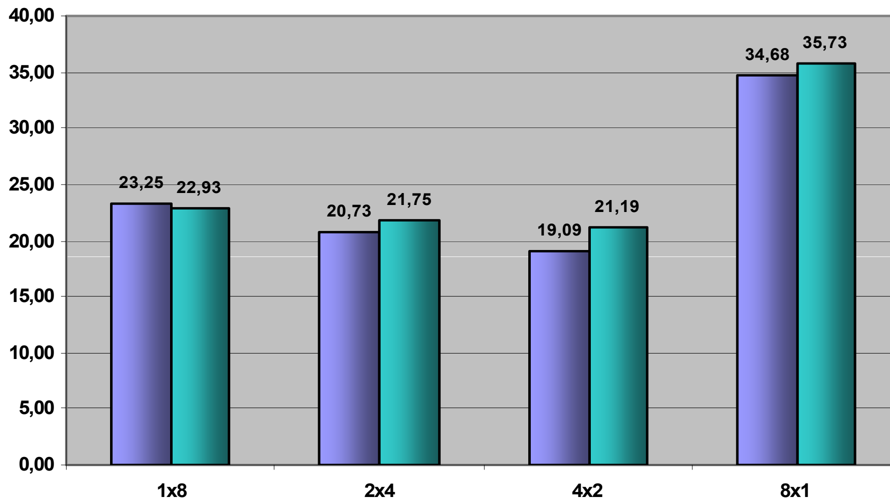
зоны от 13 x 13 x 16 и до 58 x 58 x 16

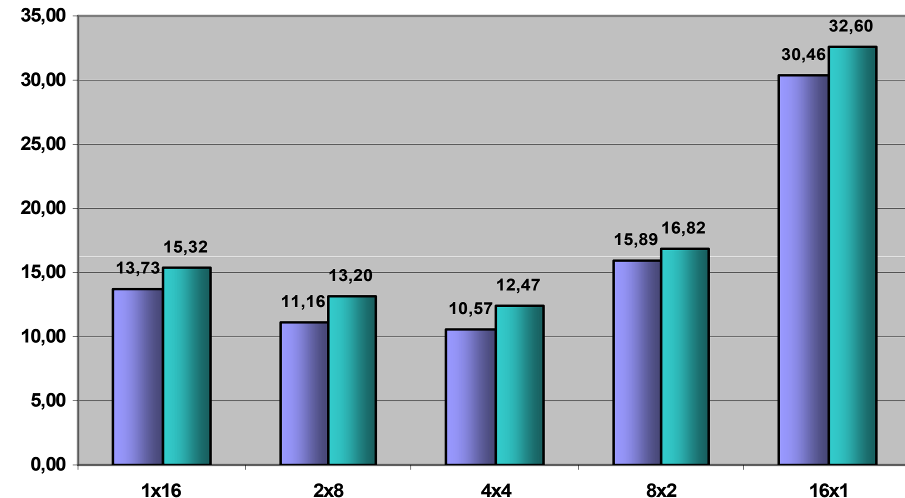


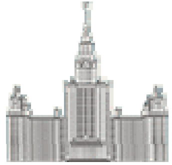


MPI

DVM

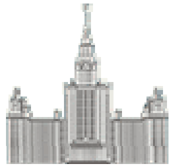






Расчет дозвукового обтекания летательного аппарата

Задача 810 областей	средняя загрузка	Max загрузка
75 процессоров	19258	19296
128 процессоров	11284	11648
256 процессоров	5642	11648
384 процессоров	3761	11648
512 процессоров	2821	11648

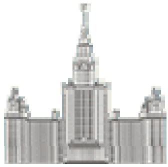


Преимущества гибридной модели MPI/OpenMP

Ликвидация или сокращение дублирования данных в памяти узла.

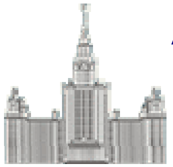
Дополнительный уровень параллелизма на OpenMP реализовать проще, чем на MPI (например, когда в программе есть два уровня параллелизма – параллелизм между подзадачами и параллелизм внутри подзадачи).

Улучшение балансировки на многоблочных задачах при меньшей трудоемкости реализации еще одного уровня параллелизма.



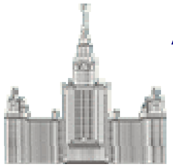
Алгоритм Якоби. Оптимизированная MPI/OpenMP-версия

```
/******  iteration loop  *************************/
t1=MPI_Wtime();
#pragma omp parallel default(none) private(it,i,j) shared (A,B,myrank,
        nrow,ranksize,ll,shift,req,status)
 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 for nowait
                        for(j=1; j<=L-2; j++)
                        {
                                    A[i][j] = B[i-1][j];
                        }
            }
```



Алгоритм Якоби. Оптимизированная MPI/OpenMP-версия

```
#pragma omp barrier
#pragma omp single
{
        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. Оптимизированная MPI/OpenMP-версия

```
        for(i=1; i<=nrow; i++)
        {
            if (((i==1)&&(myrank==0))||((i==nrow)&&(myrank==ranksize-1))) continue;
            #pragma omp for nowait
            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;
}
```

Привязка процессов к ядрам

Intel MPI

export I_MPI_PIN_DOMAIN=omp (node)

mpirun …

или

mpirun –env I_MPI_PIN_DOMAIN omp ...

OpenMPI

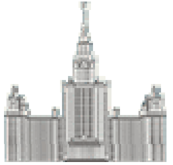
mpirun –bind-to-none ....

MVAPICH

mpirun VIADEV_USE_AFFINITY=0 …

Привязка процессов к ядрам

```
#include <sched.h>
#include <omp.h>
void SetAffinity (int rank) {
   int MPI_PROCESSES_PER_NODE =
omp_get_num_procs()/omp_get_max_threads ();
#pragma omp parallel
  {
   cpu_set_t mask;
   CPU_ZERO(&mask);
   int cpu = (rank% MPI_PROCESSES_PER_NODE)*omp_get_num_threads() +
omp_get_thread_num ();
   CPU_SET(cpu,&mask);
   sched_setaffinity ((pid_t)0, sizeof(cpu_set_t),&mask);
  }
}
```



Инициализация MPI с поддержкой нитей

```
int main(int argc, char **argv) {
    int required = MPI_THREAD_FUNNELED;
    int mpi_rank, mpi_size, mpi_err, provided;
    MPI_Comm comm = MPI_COMM_WORLD;
    mpi_err = MPI_Init_thread(&argc, &argv, required, &provided);
    mpi_err = MPI_Comm_rank(comm, &mpi_rank);
    if (mpi_rank == 0) {
        switch (provided) {
            case MPI_THREAD_SINGLE: /* */ break;
            case MPI_THREAD_FUNNELED: /* */ break;
            case MPI_THREAD_SERIALIZED: /* */ break;
            case MPI_THREAD_MULTIPLE: /* */ break;
            default: /* */ break;
        }
    }
}
```

Инициализация MPI с поддержкой нитей

MPI_THREAD_SINGLE

MPI-процесс исполняет единственную нить

MPI_THREAD_FUNNELED

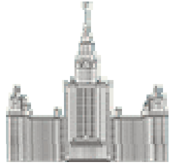
MPI-процесс может быть многонитевым, но делать MPI-вызовы разрешено только тому процессу, который проводил инициализацию MPI

MPI_THREAD_SERIALIZED

MPI-процесс может быть нитевым, но в любой момент времени MPI-вызов делает лишь одна нить

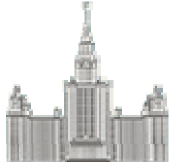
MPI_THREAD_MULTIPLE

MPI-процесс может быть многонитевым, и несколько нитей могут вызывать MPI-функции одновременно



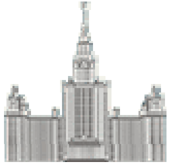
Литература

- OpenMP Application Program Interface Version 3.1, July 2015.
  http://www.openmp.org/mp-documents/OpenMP3.1.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.



Автор

Бахтин Владимир Александрович, кандидат физико-математических наук, заведующий сектором Института прикладной математики им. М.В. Келдыша РАН, доцент кафедры системного программированния 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

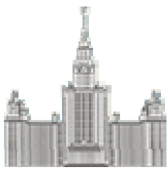
Определение количества процессов в выполняемой параллельной программе осуществляется при помощи функции:

```
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.

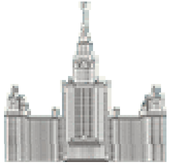
Обратно

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_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

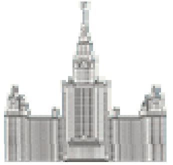
Определение декартовых координат процесса по его рангу:

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