Скорость работы программы

(2-е задание)

Перемножение

```
void mult_matrices(double **a, double **b, double **c, int n)
{
   int i, j, k;
   double sum;
  FORI
   {
    FORJ
      {
            FORK
         {
               c[i][j] += a[i][k] * b[k][j];
         }
     }
   }
}
```

Варианты циклов

первый цикл по i, второй по j, третий по k - кратко назовем ijk
первый цикл по i, второй по k, третий по j - кратко назовем ikj
первый цикл по j, второй по i, третий по k - кратко назовем jik
первый цикл по j, второй по k, третий по i - кратко назовем jki
первый цикл по k, второй по i, третий по j - кратко назовем kij
первый цикл по k, второй по j, третий по i - кратко назовем kji

Результаты.

Цикл	Произ. Мфлопс	Время
ijk	106.17	150.65
ikj	218.08	73.34
kij	218.41	73.23
jik	145.19	110.16
jki	92.78	172.40
kji	93.90	170.17

Оптимизация

Цикл	Произ Мфлопс	Время с.	Опция
kji	122.38	130.70	-O
kij	528.58	30.26	-O
kji(i)	135.13	118.37	-O
kij(I)	544.17	29.39	-O

Разворачивание циклов

```
int i, j, k;
 int i1,i2;
 double tmp,tmp1,tmp2;
 double temp;
 double *p,*ci,*ci1,*ci2;
 for(i=0, i1=1, i2=2; i < n; i+= 3,i1+=3, i2+= 3){
    for( j= 0;j < n; j++){
       p=&b[j][0];
       tmp= a[i][j];tmp1=a[i1][j];tmp2=a[i2][j];
       ci=&c[i][0]; ci1=&c[i1][0]; ci2=&c[i2][0];
       for( k=0; k < n; k++){
          temp=p[k];
          ci[k]+= tmp*temp;ci1[k]+=tmp1*temp; ci2[k]+=tmp2*temp;
       }
    }
 }
```

Результаты.

Произ. Мфлопс	Время с.	Опции
485.33	32.95	
2062.52	7.75	-O
2496.55	6.40	-O

Разворачивание циклов

```
int i,j,k;
double *q,*b1,*b2,*b3,*b4,*b5,*b6,*b7,*b8,*b9,*b10;
double a1,a2,a3,a4,a5,a6,a7,a8,a9,a10;

for(k = 0; k < n; k+= 10) {
  b1=&b[k][0]; b2=&b[k+1][0]; b3=&b[k+2][0]; b4=&b[k+3][0];
  b5=&b[k+4][0]; b6= &b[k+5][0];b7=&b[k+6][0]; b8=&b[k+7][0];
  b9= &b[k+8][0];b10=&b[k+9][0];
  for(i = 0; i < n; i++) {
    q=&c[i][0];
    a1 = a[i][k]; a2 = a[i][k+1]; a3 = a[i][k+2]; a4 = a[i][k+3];
    a5 = a[i][k+4]; a6 = a[i][k+5]; a7 = a[i][k+6]; a8 = a[i][k+7];
    a9 = a[i][k+8]; a10 = a[i][k+9];
    for(j = 0;j < n; j++) {
      q[j] += a1*b1[j]+a2*b2[j]+a3*b3[j]+a4*b4[j]+a5*b5[j]+
           a6*b6[j]+a7*b7[j]+a8*b8[j]+a9*b9[j]+a10*b10[j];
    }
  }
}
```

Результат

Произ. Мфлопс	Время с.	Опции
1006.65	15.89	
3155.78	5.06	-O
3435.85	4.65	-O

dgemm

```
 SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
•.. Scalar Arguments
• DOUBLE PRECISION ALPHA,BETA
•INTEGER K,LDA,LDB,LDC,M,N
•CHARACTER TRANSA,TRANSB
•* ..
•* .. Array Arguments .
•.
•DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
•* ..
•*
•* Purpose
•* =======
•*
•* DGEMM performs one of the matrix-matrix operations
•*
•* C := alpha*op( A )*op( B ) + beta*C,
```

Си

```
extern void dgemm_(char *tra, char *trb, int *m, int *n, int *k, double
*alpha, double *a, int *lda,double *b, int *ldb, double *beta, double
*c, int *ldc);
```

```
transA='T';
transB='T';
alpha=1.0;
zero=0.0;
// mult_matrices(a, b, c,n);

dgemm_(&transA,&transB,&n,&n,&n,&alpha,a,&n,b,&n,
&zero,c,&n);
```

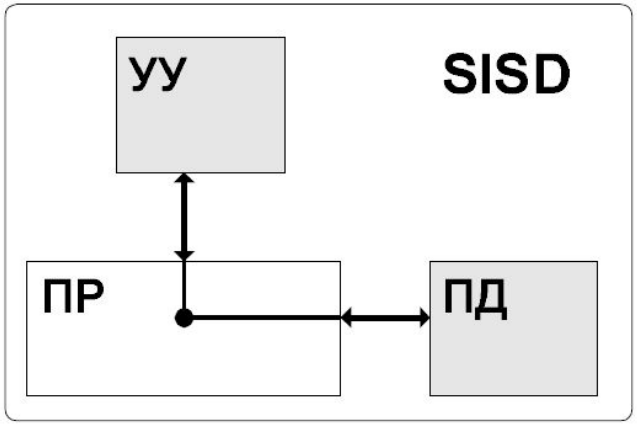
Компиляция

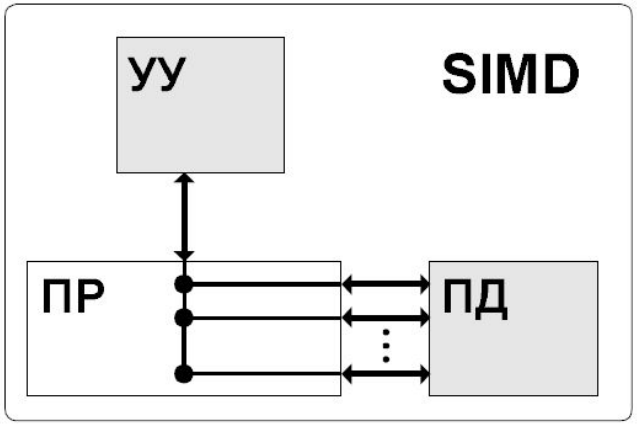
cc -g -o matmul matmuldot.o -lm -lgoto -lpthread -lgfortran

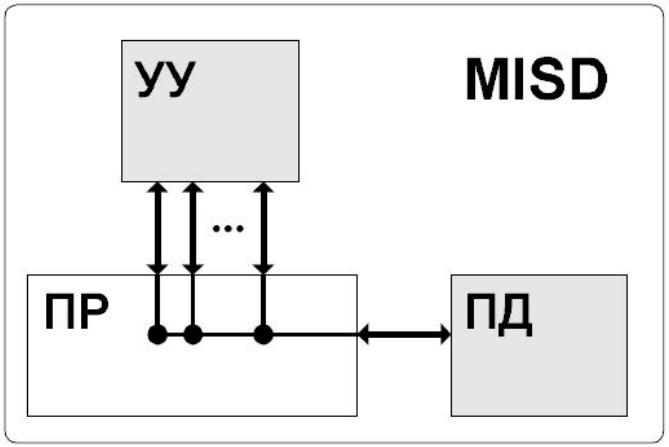
Makefile

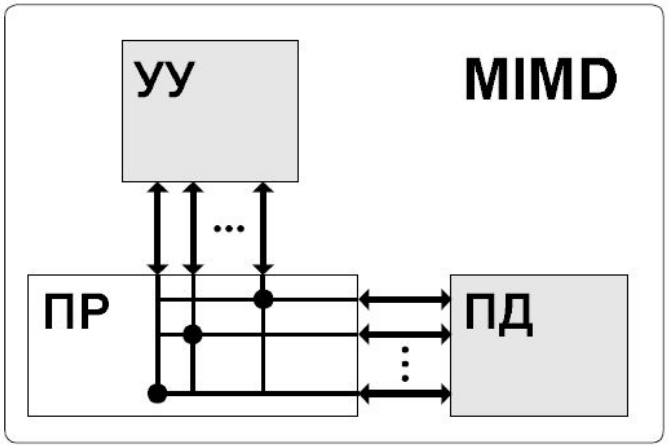
```
PROG=matmul

TESTOBJ=matmuldot.o
CFLAGS=-g
CLFLAGS=-g
LIBS=-lm -lgoto -lpthread -lgfortran

all:$(TESTOBJ)
	$(CC) $(CLFLAGS) -o $(PROG) $(TESTOBJ) $(LIBS)
clean:
	rm *.o $(PROG)

%.o: %.c $(CC) -c -o $@ $< $(CFLAGS)
```

Классификация Флина.









Openmp

OMP_DYNAMIC

Включает/отключает режим, в котором количество создаваемых нитей при входе в параллельную область может меняться динамически.

Начальное значение: Если компилятор не поддерживает данный режим, то false. Иначе – зависит от реализации.

export OMP_DYNAMIC=true

void omp_set_dynamic(int dynamic_threads);

Узнать значение переменной можно:

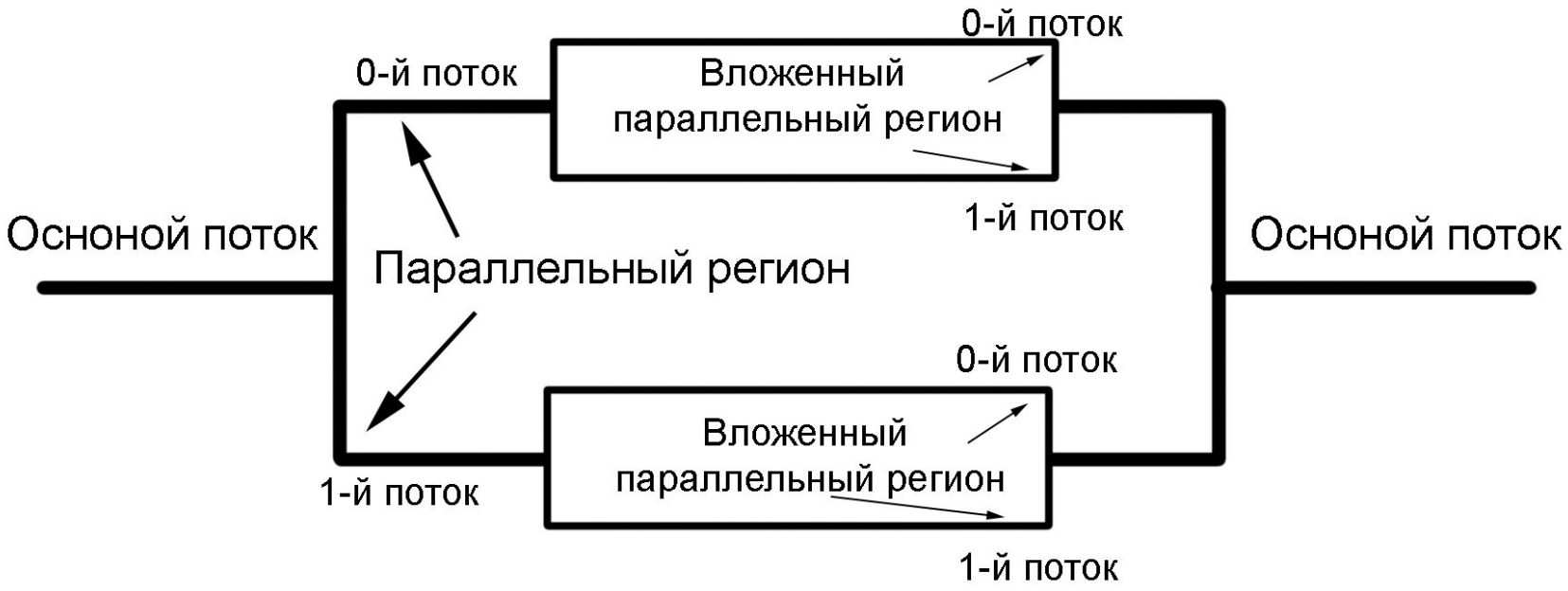
int omp_get_dynamic(void);

```
#include <stdio.h>
#include <omp.h>
int n;
#pragma omp threadprivate(n)
int main(int argc, char *argv[])
{
int num;
n=1;
#pragma omp parallel private (num)
{
num=omp_get_thread_num(); printf("Значение n на нити %d (на входе): %d\n", num,
n);
n=omp_get_thread_num() + 1; printf("Значение n на нити %d (на выходе): %d\n", num,
n);
}
printf("Значение n (середина): %d\n", n);
#pragma omp parallel private (num)
{
num=omp_get_thread_num(); printf("Значение n на нити %d (ещё раз): %d\n", num,
n);
}
n=1;
#pragma omp parallel copyin(n)
{
printf("Значение n: %d\n", n);
}
```

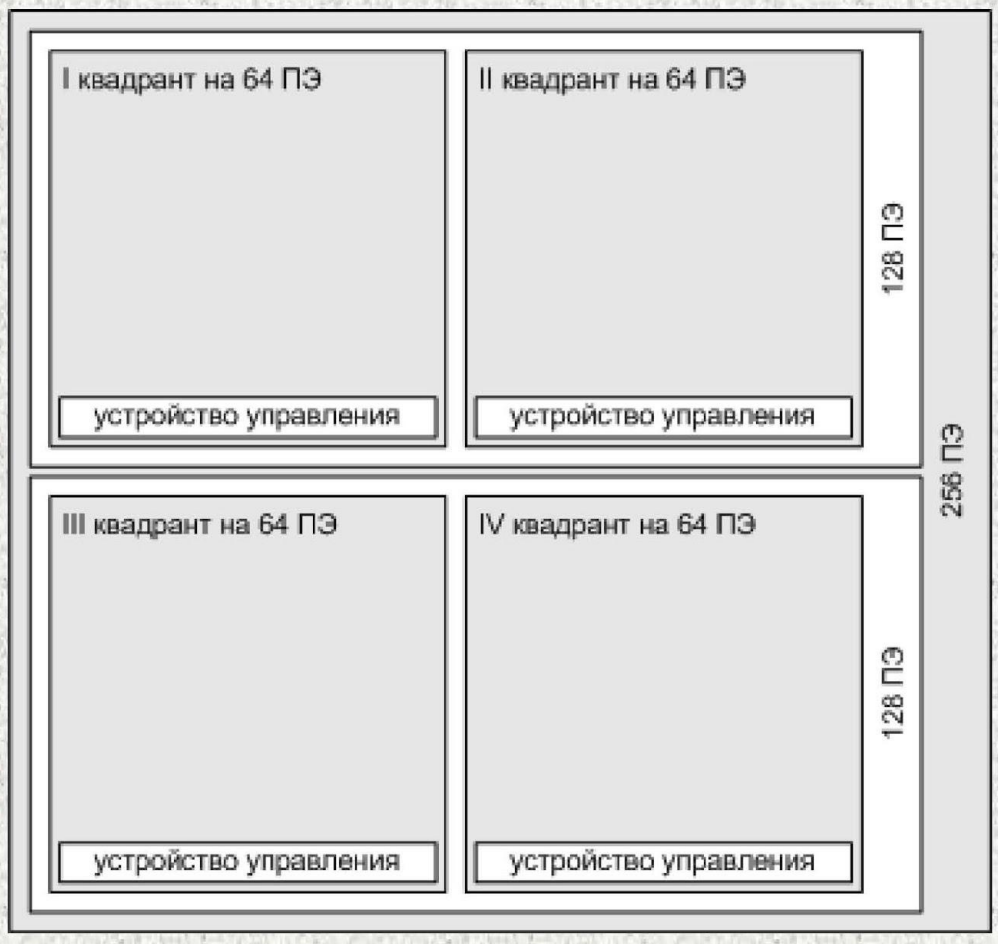
Вложенные регионы

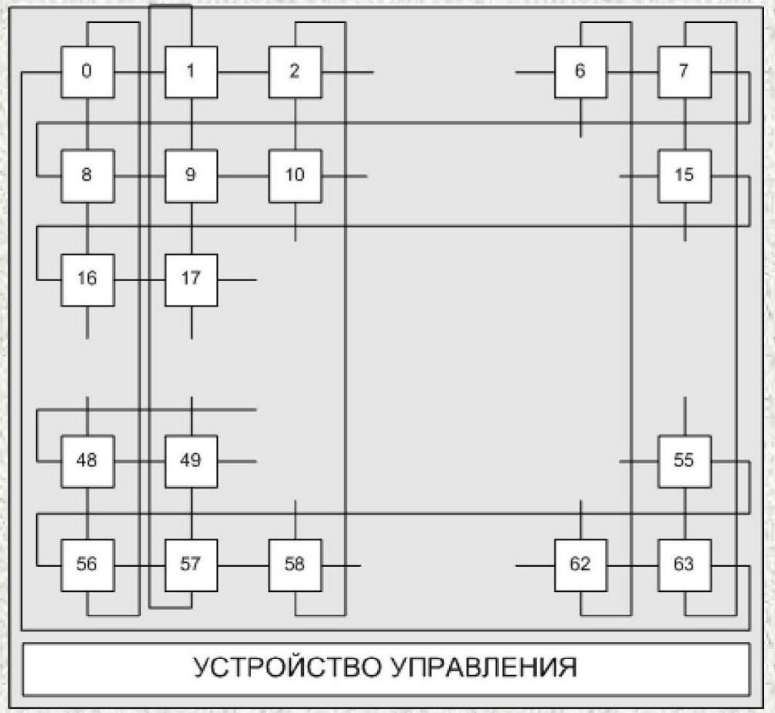
```
int main(){
#pragma omp parallel       // параллельный регион 1
{
   #pragma omp parallel
   {

   }
}
```

Вложенные регионы



Компьютер ILLIAC IV

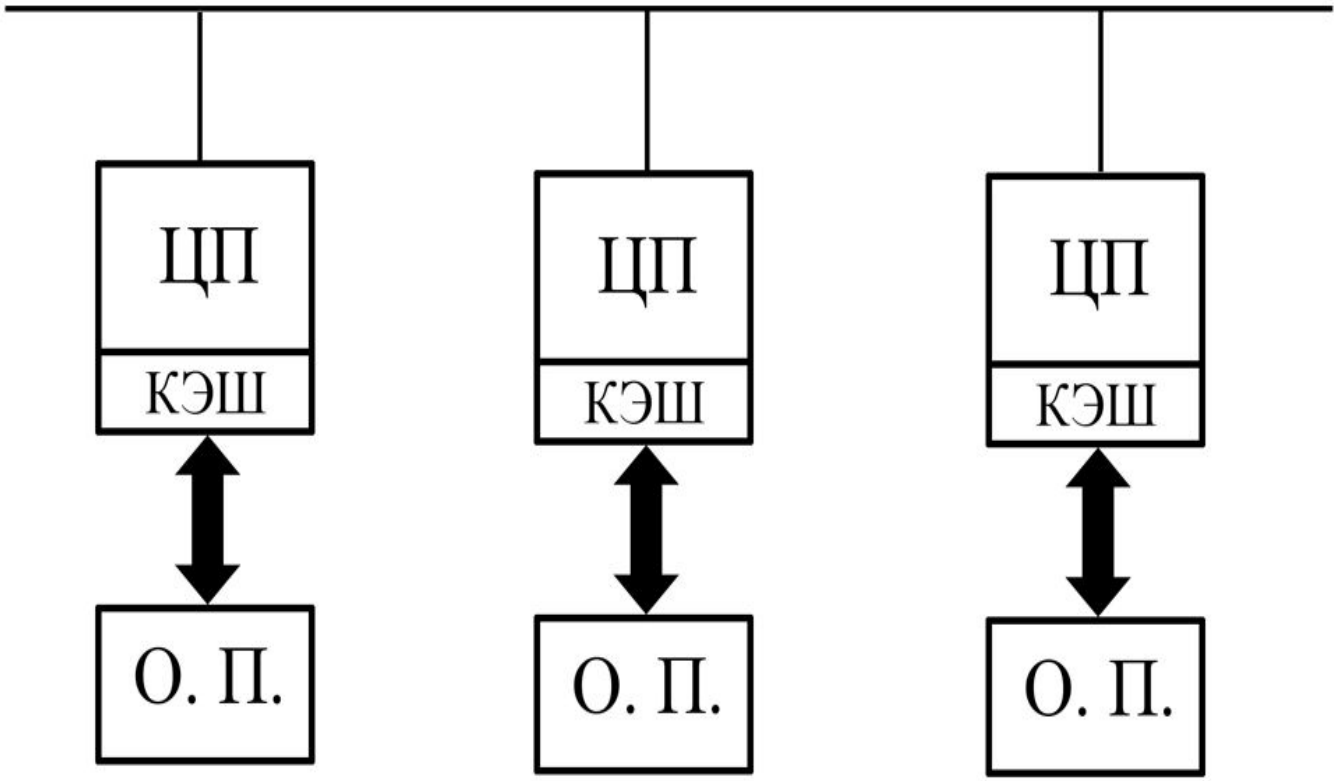




Архитектура Fermi

Системы с разделенной памятью
MPP- системы(Массово-параллельная архитектура)

Системы с массовым параллелизмом (MPP)



Особенности MPP.

Достоинство

- Хорошая масштабируемость.

Недостатки

- Сложности межпроцессорного взаимодействия
- Разработка программ.

Кластера

Кластеры высокой доступности
Обозначаются аббревиатурой HA (англ. High Availability — высокая доступность). Создаются для обеспечения высокой доступности сервиса, предоставляемого кластером. Избыточное число узлов, входящих в кластер, гарантирует предоставление сервиса в случае отказа одного или нескольких серверов. Типичное число узлов — два, это минимальное количество, приводящее к повышению доступности. Создано множество программных решений для построения такого рода кластеров. В частности, для GNU/Linux, FreeBSD и Solaris существует проект бесплатного ПО Linux-HA.

Кластеры распределения нагрузки
Принцип их действия строится на распределении запросов через один или несколько входных узлов, которые перенаправляют их на обработку в остальные, вычислительные узлы. Первоначальная цель такого кластера — производительность, однако, в них часто используются также и методы, повышающие надежность. Подобные конструкции называются серверными фермами. Программное обеспечение (ПО) может быть как коммерческим (OpenVMS Cluster, Platform LSF HPC, Sun Grid Engine, Moab Cluster Suite, Maui Cluster Scheduler), так и бесплатным (Linux Virtual Server, Mosix).

Кластеры повышенной производительности
Обозначаются англ. аббревиатурой HPC (High performance cluster). Позволяют увеличить скорость расчетов, разбивая задание на параллельно выполняющиеся потоки. Используются в научных исследованиях. Одна из типичных конфигураций — набор серверов с установленной на них операционной системой Linux, такую схему принято называть кластером Beowulf. Для HPC создается специальное ПО, способное эффективно распределять задачу между узлами.
Эффективные связи между серверами в кластере позволяют им поддерживать связь и оперативно обмениваться данными, поэтому такие кластеры хорошо приспособлены для выполнения процессов, использующих общие данные.

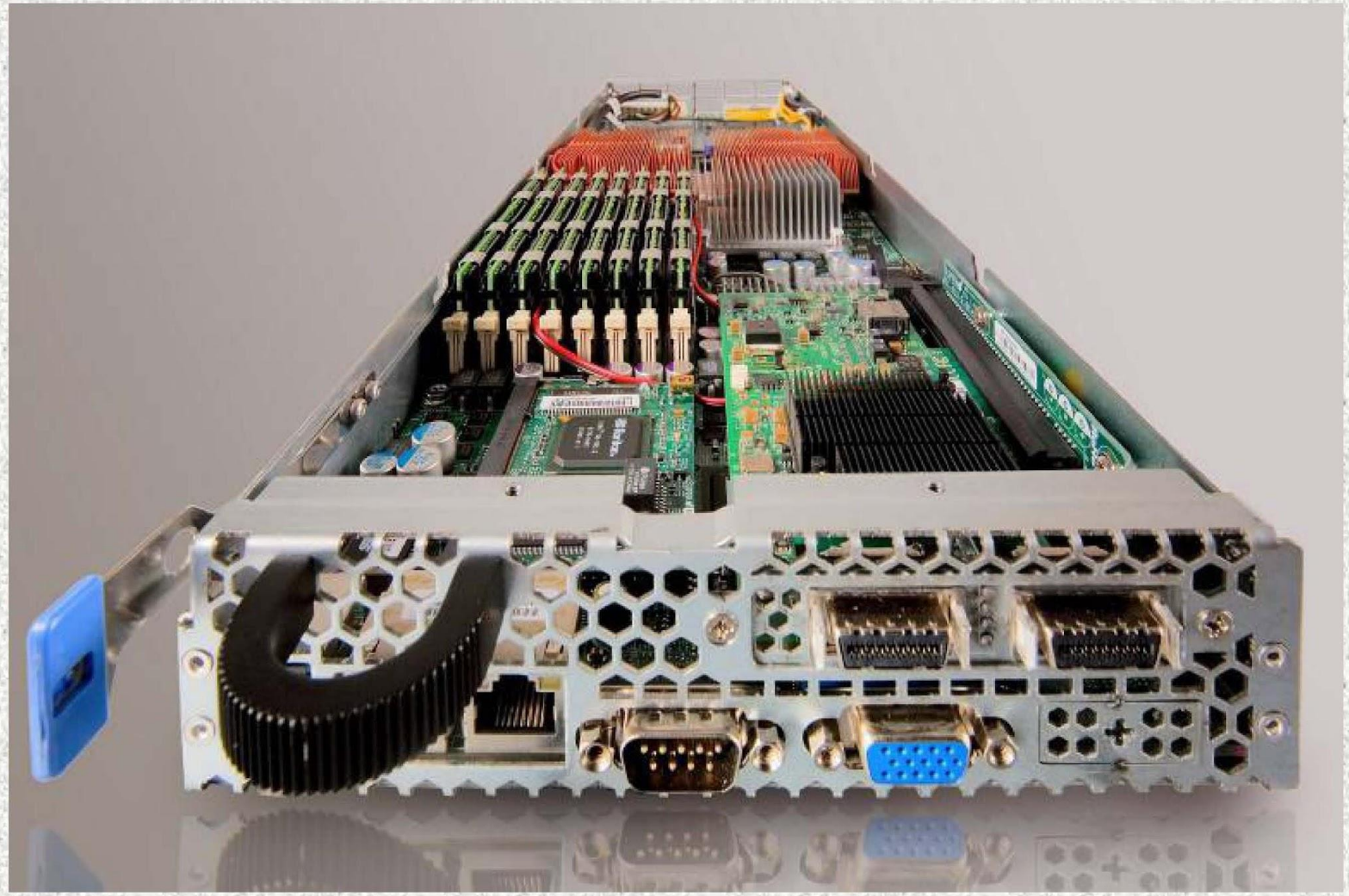
Системы распределенных вычислений (grid)
Такие системы не принято считать кластерами, но их принципы в значительной степени сходны с кластерной технологией. Их также называют grid-системами. Главное отличие — низкая доступность каждого узла, то есть невозможность гарантировать его работу в заданный момент времени (узлы подключаются и отключаются в процессе работы), поэтому задача должна быть разбита на ряд независимых друг от друга процессов. Такая система, в отличие от кластеров, не похожа на единый компьютер, а служит упрощенным средством распределения вычислений. Нестабильность конфигурации, в таком случае, компенсируется большим числом узлов.

Вычислительные ресурсы ЮГИНФО

- LINUX-кластер (10 узлов, 2003 г., CPU P4 2.4 Ггц, память 512 Мб)
- INFINI-кластер (21 узел, 2005 г ., CPU P4 3.4 Ггц, память 2 Гб.)
- IBMX-кластер (12 узлов, 2008 г. CPU Xeon 3.0 Ггц, память 8

Суперкомпьютер СКИФ МГУ “Чебышев”

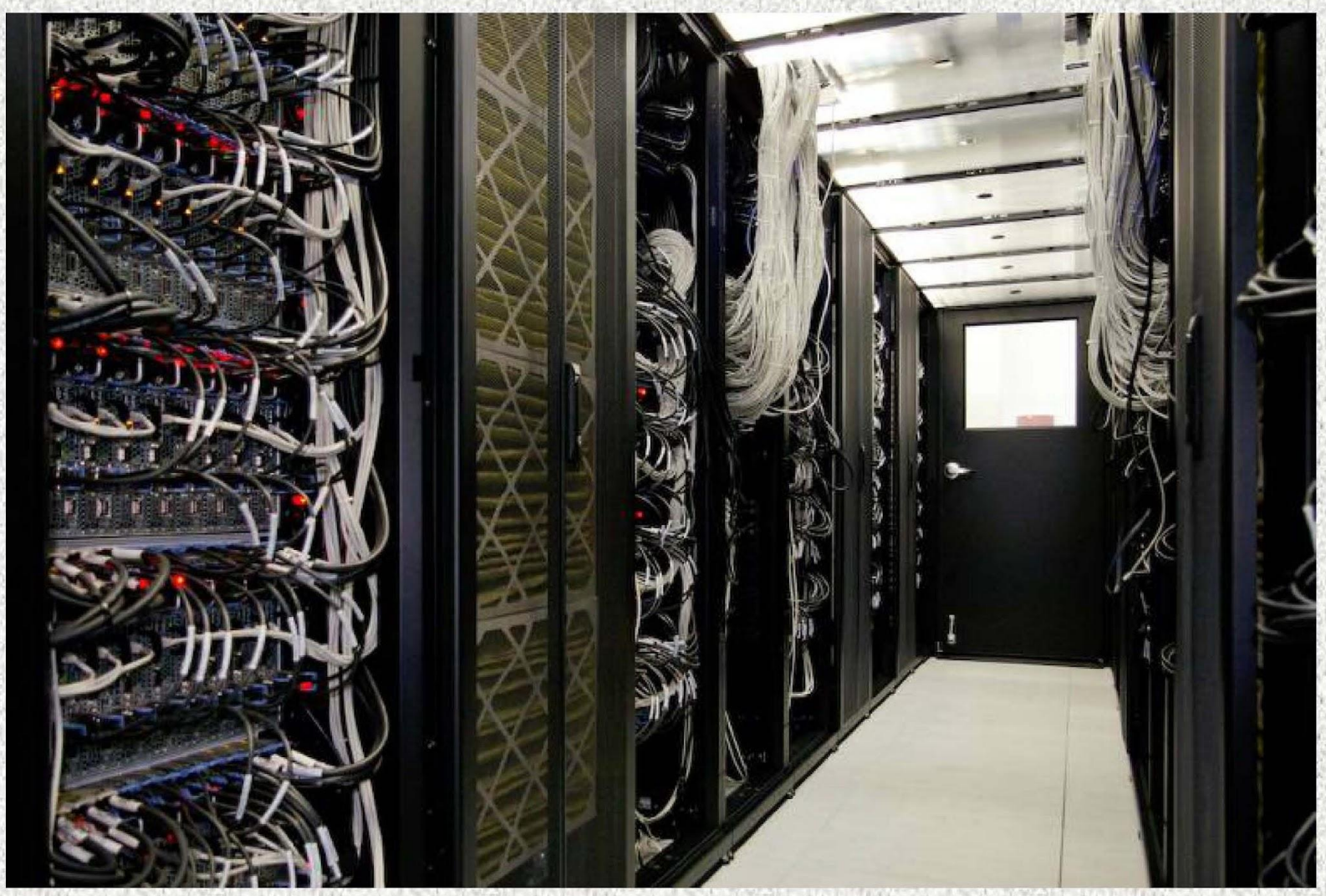
Суперкомпьютер СКИФ МГУ “Чебышев”



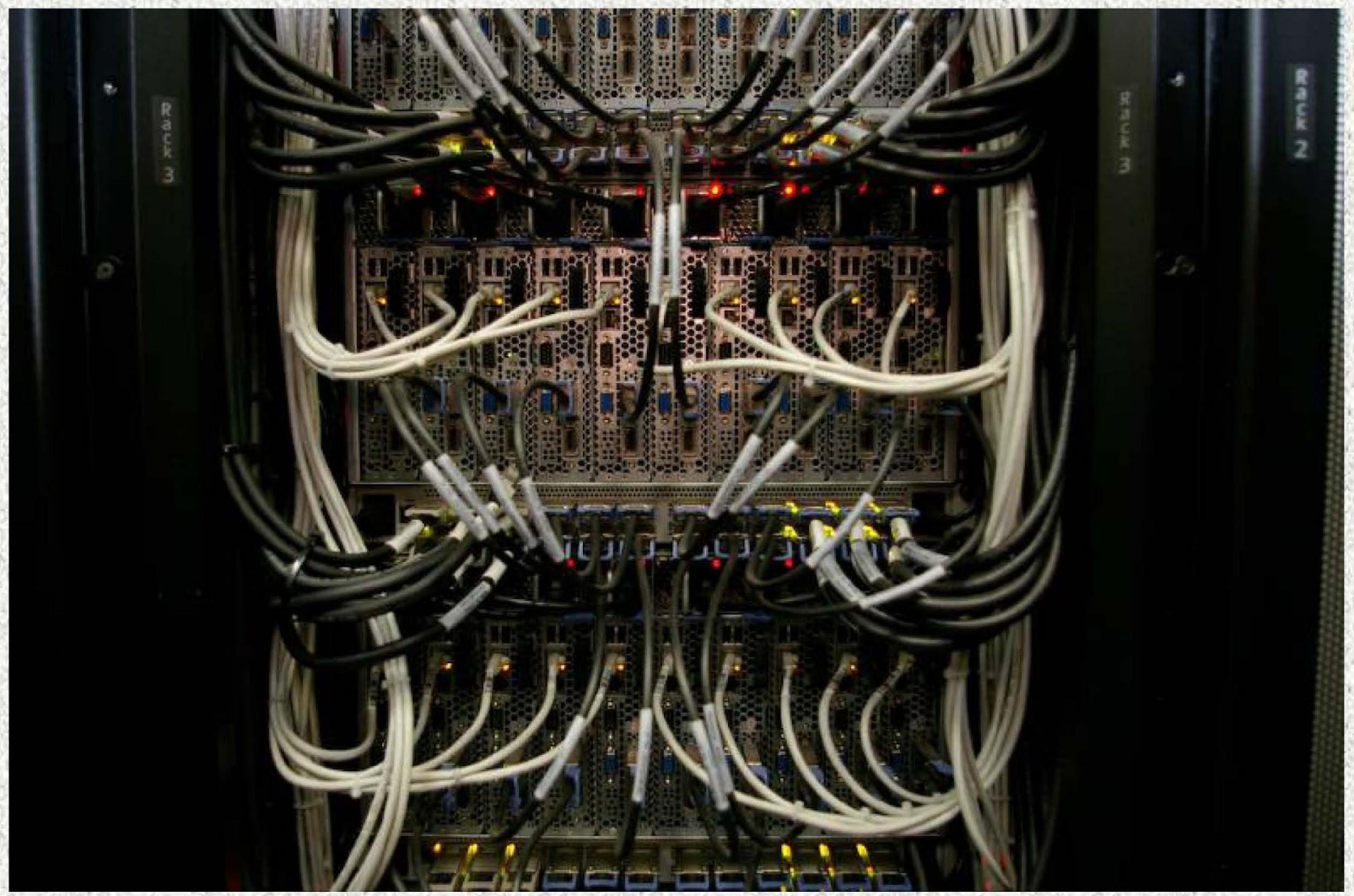
Суперкомпьютер СКИФ МГУ “Чебышев”

60 Tflop/s, 1250 процессоров Intel Xeon (*4 ядра)

Суперкомпьютер СКИФ МГУ “Чебышев”



Суперкомпьютер СКИФ МГУ “Чебышев”



Софт

- INTEL 11.0
  - icc
  - icpc
  - ifort
- GCC
  - gcc
  - g++
  - gfortran
- MPI

Системы управления заданиями

- PBS (Portable Batch System) – система управления ресурсами и загрузкой кластеров. Может работать на большом числе различных платформ: Linux, FreeBSD, NetBSD, Digital Unix, Tru64, HP-UX, AIX, IRIX, Solaris. В настоящее время существует свободная и обладающая более широкими возможностями реализация PBS, называемая Torque.
- LSF (Load Sharing Facility) – система, аналогичная PBS. Разработана компанией Platform Computing. Также способна работать на множестве платформ.
- NQE (Network Queuing Environment) – продукт компании Cray Research, использующийся чаще всего как менеджер ресурсов на суперкомпьютерах, кластерах и системах Cray, хотя может работать и на других платформах.
- LoadLeveler – продукт компании IBM, управляющий балансом загрузки крупных кластеров. Используется в основном на кластерах IBM.
- Condor – свободно доступный менеджер ресурсов, разработанный в основном студентами различных университетов Европы и США. Аналогичен вышеперечисленным. Работает на различных платформах UNIX и Windows NT
- Easy-LL – совместная разработка IBM и Cornell Theory Center, предназначенная для управления крупным кластером IBM в этом центре.

PBS.

- Оригинальный opensource проект OpenPBS разработанный в 1998 году MRJ.(на данный момент не поддерживается)
- TORQUE (*Terascale Open-Source Resource and QUEue Manager*) - проект основанный на OpenPBS и поддерживаемый Adaptive Computing Enterprises, Inc
- PBS Professional (PBS Pro) - коммерческая система предлагаемая Altair Engineering.

Архитектура PBS.

Сервер (pbs_server) который является центром PBS, именно сервер принимает задания от пользователей, удаляет задания, изменяет задания и выполняет другие функции.
Планировщик (pbs_sched) который управляет расстановкой задач в очереди и определяет, какая задача будет запущенна на выполнение.
Mom (machine oriented miniserver) (pbs_mom) исполняет пользовательский скрипт и следит, чтобы он работал отведенное время. На всех компьютерах, где должны выполнятся задания должен быть запущен pbs_mom.

PBS наиболее часто используемые команды

- qsub – команда для запуска задачи
- qstat – для просмотра состояния очереди
- pbsnodes(pestat) – состояния узлов в очереди

qsub

- **qsub [options]** ***PBS_script***

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l nodes=2:LINUX
cd $PBS_O_WORKDIR
mpirun -np 2 ping_DELLE
```

Результат работы программы без каких либо дополнительных усилий с Вашей стороны будет записываться в файл и помещаться в тот каталог, из которого было запущено задание. Имя выходного файла формируется автоматически следующим образом:
<имя скрипта>.o<номер задания>, а в файл
<имя скрипта>.e<номер задания> - будет записываться стандартный канал диагностики (ошибок).

опции команды qsub

- **q** - название очереди пакетной обработки
- **-l** - набор технических параметров, набираемых через "**,**":
  - **walltime** - максимальное время выполнения задачи,
  - **nodes** - требуемое количество процессоров (после указания количества процессоров после "**:**" следует указывать название очереди)
- **-m** - события, происходящие в процессе пакетной обработки задачи, о которых следует извещать по *e-mail*: **b** - начало, **e** - завершение, **a** - прекращение работы по ошибке;
- **-M** - *e-mail* адрес, на который будут направляться все служебные сообщения о состоянии задачи
- **-r** - (*y/n*) (т.е. *да* или *нет*) следует ли восстанавливать задачу при перезагрузке узлов

Команда qstat

```
rsufs.cc.rsu.ru:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
36822.rsufs.cc. alper    DELL     m13+_sp10    2516   1   1    --  336:0 R 27:20
36722.rsufs.cc. data     IBMX     sukharina    4130   2   1    --  300:0 R 126:3
36723.rsufs.cc. data     IBMX     sukharina    4096   2   1    --  300:0 R 126:3
36743.rsufs.cc. alper    IBMX     bd33_so_de   4263   6   1    --  336:0 R 120:5
36789.rsufs.cc. mihas    IBMX     mg          16926   1   1    --  300:0 R 94:09
36790.rsufs.cc. mihas    IBMX     Si           7821   2   1    --  300:0 R 94:08
36804.rsufs.cc. mevs     IBMX     marina         --   1   1    --  300:0 Q   --
36805.rsufs.cc. mevs     IBMX     marina         --   1   1    --  300:0 Q   --
36806.rsufs.cc. mevs     IBMX     marina         --   1   1    --  300:0 Q   --
36807.rsufs.cc. mevs     IBMX     marina         --   2   1    --  300:0 Q   --
36808.rsufs.cc. mevs     IBMX     marina         --   2   1    --  300:0 Q   --
36565.rsufs.cc. tversu   INFINI   C5H11NO2_b  25640   5   1    --  336:0 R 170:0
36846.rsufs.cc. mvl      INFINI   vika        13362   1   1    --  336:0 R 23:03
36848.rsufs.cc. mvl      INFINI   vika         9694   1   1    --  336:0 R 22:58
36849.rsufs.cc. mvl      INFINI   vika        10920   1   1    --  336:0 R 22:56
36855.rsufs.cc. tversu   INFINI   C6H13NO2_b  11735   5   1    --  336:0 R 04:06
36835.rsufs.cc. tversu   LINUX    NO2_CH4_NO  11991   5   1    --  336:0 R 24:07
36738.rsufs.cc. agud     WSD      N2opt.pbs    3522   1   1    --  320:0 R 120:4
```

Описание вывода qstat

- **Job id** - уникальный идентификатор задачи
- **Name** - имя исполняемой задачи
- **User** - имя владельца задачи
- **Time Use** - общее процессорное время, использованное задачей на данный момент
- **S** - состояние задачи
  - **Q** - находится в очереди
  - **R** - вычисляется
  - **E** - произошла ошибка при выполнении
- **Queue** - название очереди, в которой запущена задача

команда pestat

node	props	state	load	pmem	ncpu	mem	resi	usrs	jobs	jobids
rsudl	SERVER	free	2.11*	3829	2	0	0	0/0	0	
dl01	DELLE	free	0.16	3829	2	0	0	0/0	0	
dl02	DELLE	free	0.12	3829	2	0	0	0/0	0	
dl03	DELLE	free	0.11	3829	2	0	0	0/0	0	
dl04	DELLE	excl	0.43*	3829	2	0	0	2/1	1	15897
dl05	DELLE	excl	0.47*	3829	2	0	0	1/1	1	15897
dl06	DELLE	free	0.11	3829	2	0	0	0/0	0	
dl07	DELLE	excl	0.98	3829	2	0	0	1/1	1	15897
dl08	DELLE	free	0.11	3701	2	0	0	1/1	0	

qdel

- qdel - удаление задания

qdel 36807

Распараллеливания программ

- SPMD (Single Program Multiple Date) - на всех процессорах выполняются копии одной программы, обрабатывающие разные блоки данных;
- MPMD (Multiple Program Multiple Date) - на процессорах выполняются разные программы, обрабатывающие разные данные.

Методологический подхода(Фостера) к решению задачи на многопроцессорной системе

- разбиение задачи на минимальные независимые подзадачи (partitioning);
- установление связей между подзадачами (communication);
- объединение подзадач с целью минимизации коммуникаций (agglomeration);
- распределение укрупненных подзадач по процессорам таким образом, чтобы обеспечить равномерную загрузку процессоров (mapping).

Общая схема распараллеливания

```
if (proc_id == 0) {
  task1();
}
if (proc_id == 1) {
  task2();
}
resutl=reduce(result_task1, result_task2, ...)
```

Общая организация MPI

Коммуникационная библиотека MPI стала общепризнанным стандартом в параллельном программировании для систем с распределенной памятью с использованием механизма передачи сообщений. Полное и строгое описание среды программирования MPI можно найти в авторском описании разработчиков MPI: The Complete Reference

http://rsusu1.rnd.runnet.ru/parallel/mpi/index.html

http://parallel.ru/tech/tech_dev/mpi.html

MPI программа представляет собой набор независимых процессов, каждый из которых выполняет свою собственную программу.

Процессы MPI программы взаимодействуют друг с другом посредством вызова коммуникационных процедур.

Общая организация MPI (продолжение)

Для идентификации наборов процессов вводится понятие *группы*, объединяющей все или какую-то часть процессов.

Каждая группа образует *область связи*, с которой связывается специальный объект - *коммуникатор* области связи.

При инициализации MPI создается предопределенная область связи, содержащая все процессы MPI-программы, с которой связывается предопределенный коммуникатор MPI_COMM_WORLD.

Процессы внутри группы нумеруются целым числом в диапазоне 0..groupsize-1.

Структура MPI

Около 130 функций

- функции инициализации и закрытия MPI процессов;
- функции, реализующие коммуникационные операции типа точка-точка;
- функции, реализующие коллективные операции;
- функции для работы с группами процессов и коммуникаторами;
- функции для работы со структурами данных;
- функции формирования топологии процессов.

Характеристики функций

- *Локальная функция* – выполняется внутри вызывающего процесса. Ее завершение не требует коммуникаций.
- *Нелокальная функция* – для ее завершения требуется выполнение MPI-процедуры другим процессом.
- *Глобальная функция* – процедуру должны выполнять все процессы группы. Несоблюдение этого условия может приводить к зависанию задачи.
- *Блокирующая функция* – блокирует выполнение процесса до полного завершения коммуникационной операции. Возврат управления из процедуры гарантирует возможность повторного использования параметров, участвующих в вызове.
- *Неблокирующая функция* – возврат из процедуры происходит немедленно, без ожидания окончания операции и до того, как будет разрешено повторное использование параметров, участвующих в запросе. Завершение неблокирующих операций осуществляется специальными функциями.

Особенности MPI в Си

Все процедуры являются функциями, и большинство из них возвращает код ошибки.

При использовании имен подпрограмм и именованных констант необходимо строго соблюдать регистр символов.

Массивы индексируются с 0.

Логические переменные представляются типом int (true соответствует 1, a false – 0).

Определение всех именованных констант, прототипов функций и определение типов выполняется подключением файла *mpi.h*.

Соответствие между MPI-типами и типами языка C

тип MPI	тип языка C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Особенности MPI в Фортране

- большинство MPI процедур являются подпрограммами (вызываются с помощью оператора CALL),
- код ошибки возвращают через дополнительный последний параметр процедуры.
- Несколько процедур, оформленных в виде функций, код ошибки не возвращают.
- Не требуется строгого соблюдения регистра символов в именах подпрограмм и именованных констант.
- Массивы индексируются с 1.
- Объекты MPI, которые в языке C являются структурами, в языке FORTRAN представляются массивами целого типа.
- Определение всех именованных констант и определение типов выполняется подключением файла *mpif.h*

Соответствие между MPI-типами и типам языка FORTRAN

Тип MPI	**Тип языка FORTRAN**
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

Базовые функции MPI

Функция инициализации MPI_Init

C/C++:

```
int MPI_Init(int *argc, char ***argv)
```

Fortran:

```
MPI_INIT(IERROR)
INTEGER IERROR
```

Базовые функции MPI

Функция завершения MPI программ MPI_Finalize.

C/C++:

int MPI_Finalize(void)

Fortran:

MPI_FINALIZE(IERROR)

INTEGER IERROR

Базовые функции MPI

Функция определения числа процессов в области связи MPI_Comm_size.

C/C++:

int MPI_Comm_size(MPI_Comm comm, int *size)

Fortran:

MPI_COMM_SIZE(COMM, SIZE, IERROR)

INTEGER COMM, SIZE, IERROR

IN comm - коммуникатор

OUT size - число процессов в области связи коммуникатора comm.

Базовые функции MPI

. Функция определения номера процесса MPI_Comm_rank

C/C++:

int MPI_Comm_rank(MPI_Comm comm, int *rank)

Fortran:

MPI_COMM_RANK(COMM, RANK, IERROR)

INTEGER COMM, RANK, IERROR

IN comm - коммуникатор

OUT rank - номер процесса, вызвавшего функцию.

Базовые функции MPI

Функция передачи сообщения MPI_Send.

C/C++:

```
int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int
   tag, MPI_Comm comm)
```

Fortran:

```
MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR
```

IN buf - адрес начала расположения пересылаемых данных
IN count - число пересылаемых элементов
IN datatype - тип посылаемых элементов
IN dest - номер процесса-получателя в группе, связанной с коммуникатором comm
IN tag - идентификатор сообщения
IN comm - коммуникатор области связи

Базовые функции MPI

Функция приема сообщения MPI_Recv.

C/C++:

```
int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source,
    int tag, MPI_Comm comm, MPI_Status *status)
```

Fortran:

```
MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM,
 STATUS, IERROR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM,
 STATUS(MPI_STATUS_SIZE), IERROR
```

OUT	buf	- адрес начала расположения принимаемого сообщения
IN	count	- максимальное число принимаемых элементов
IN	datatype	- тип элементов принимаемого сообщения
IN	source	- номер процесса-отправителя
IN	tag	- идентификатор сообщения
IN	comm	- коммуникатор области связи
OUT	status	- атрибуты принятого сообщения

Базовые функции MPI

Функция отсчета времени (таймер) MPI_Wtime.

C/C++:

double MPI_Wtime(void)

Fortran:

DOUBLE PRECISION MPI_WTIME()

Функция опроса разрешения таймера MPI_Wtick.

C/C++:

double MPI_Wtick(void)

Fortran:

DOUBLE PRECISION MPI_WTICK()

Компиляция программ mpi.

```
#include <stdio.h>
#include <mpi.h>


int main (int argc, char* argv[])
 {
int rank, size;

MPI_Init (&argc, &argv); /* starts MPI */
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);
printf( "Hello world from process %d of %d\n", rank, size );
MPI_Finalize();
return 0;
}
```

mpicc helloworld.c

mpicc helloworld.c –o hello

mpicc helloworld.c –o hello -fopenmp