Семинар 6. Дополнительные средства MPI. Параллельная сортировка.

1. Дополнительные средства MPI.

1) Группы, коммуникаторы, топологии обменов:

Группа – некоторое множество MPI-процессов приложения.

2) Функции работы с группами:

int MPI_Comm_group(MPI_Comm, MPI_Group *) – создание группы;

Лучше сделать переопределение:

#define MPI_Group_create MPI_Comm_group

int MPI_Group_free(MPI_Group *) – удаление группы;

int MPI_Group_size(MPI_Group group, int *) – определение размера группы;

int MPI_Group_rank(MPI_Group group, int *) – определение номера в группе;

int MPI_Group_translate_ranks (MPI_Group, int, int *, MPI_Group, int *) – трансляция номера из одной группы в другую;

int MPI_Group_incl(MPI_Group group, int, int *, MPI_Group *) – включение в группу новых членов;

int MPI_Group_excl(MPI_Group group, int, int *, MPI_Group *) – исключение членов из группы;

int MPI_Group_compare(MPI_Group, MPI_Group, int *) – сравнение групп;

int MPI_Group_union(MPI_Group, MPI_Group, MPI_Group *) – объединение двух групп в третью;

int MPI_Group_intersection(MPI_Group, MPI_Group, MPI_Group *) – пересечение двух групп в третью;

int MPI_Group_difference(MPI_Group, MPI_Group, MPI_Group *) – дополнение двух групп в третью;

int MPI_Group_range_incl(MPI_Group group, int, int [][3], MPI_Group *) – включение в группу;

int MPI_Group_range_excl(MPI_Group group, int, int [][3], MPI_Group *) – исключение из группы.

3) Коммуникатор – указатель на конкретную группу в функциях передачи сообщений.

Стандартные коммуникаторы:

MPI_COMM_WORLD – коммуникатор, объединяющий все процессы приложения;

MPI_COMM_NULL – ошибочный коммуникатор;

MPI_COMM_SELF – коммуникатор, включающий только вызывающей процесс;

MPI_GROUP_EMPTY – коммуникатор пустой группы;

MPI_GROUP_NULL – коммуникатор плохой группы;

Функции работы с коммуникаторами:

int MPI_Comm_create(MPI_Comm, MPI_Group, MPI_Comm *) – создание коммуникатора группы;

int MPI_Comm_dup(MPI_Comm, MPI_Comm *) – дублирование коммуникатора;

int MPI_Comm_split(MPI_Comm, int, int, MPI_Comm *) – разбиение коммуникатора на несколько новых;

int MPI_Comm_free(MPI_Comm *) – удаление коммуникатора (не работает!!!);

2) Стандарт MPI-2:

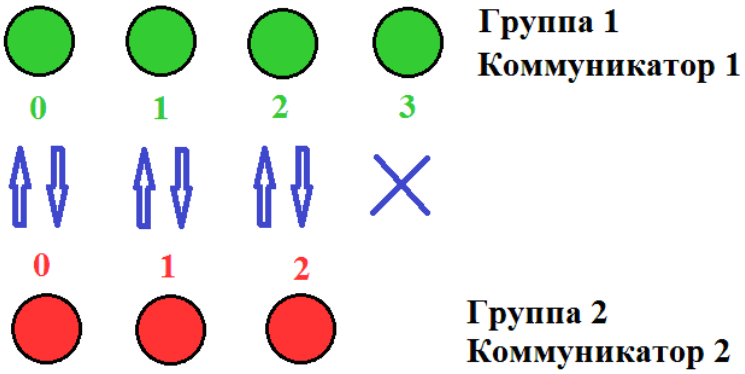
а) объединение локальных и удаленных процессов;

б) параллельный ввод-вывод.

Стандарты MPI-3, MPI-4.

2. Реализация примеров.

Пример 1 – Создание двух групп и обмен между членами с одинаковыми номерами внутри групп.



Код программы (ex09a.c)

```
#include "stdio.h"  #include <math.h>  #include "mynet.h"
static int np, mp, nl;  static char pname[MPI_MAX_PROCESSOR_NAME];
static MPI_Status status;  static double tick, t1, t2, t3;
int main(int argc, char *argv[])
{
  MPI_Group group, group1, group2;
  int np1, np2, mp1, mp2, mp3, ranks[128], i, a[4], b[4];
  MyNetInit(&argc,&argv,&np,&mp,&nl,pname,&tick);
  if (np<2) mpierr("Too small processes for communications",1); sleep(1);
  MPI_Comm_group(MPI_COMM_WORLD, &group);
  np1 = np/2; np2 = np - np1;
  for (i=0; i<np1; i++) ranks[i] = i;
  MPI_Group_incl(group, np1, ranks, &group1);
  MPI_Group_excl(group, np1, ranks, &group2);
  MPI_Group_rank(group1, &mp1);
  MPI_Group_rank(group2, &mp2);
  if (mp1 == MPI_UNDEFINED) {
    if (mp2<np1) MPI_Group_translate_ranks(group1, 1, &mp2, group, &mp3);
    else mp3 = MPI_UNDEFINED;
  }
  else
    MPI_Group_translate_ranks(group2, 1, &mp1, group, &mp3);
  a[0] = mp; a[1] = mp1; a[2] = mp2; a[3] = mp3;
  if (mp3 != MPI_UNDEFINED) {
    MPI_Sendrecv(a, 4, MPI_INT, mp3, MY_TAG,
                 b, 4, MPI_INT, mp3, MY_TAG,
                 MPI_COMM_WORLD, &status);
    fprintf(stderr,"%02d <-> %02d\n",mp,mp3);
  }
  for (i=0; i<4; i++) fprintf(stderr,"mp=%d i=%d a=%d b=%d\n",mp,i,a[i],b[i]);
  MPI_Group_free(&group);
  MPI_Group_free(&group1);
  MPI_Group_free(&group2);
  MPI_Finalize();
  return 0;
}
```

Трансляция:

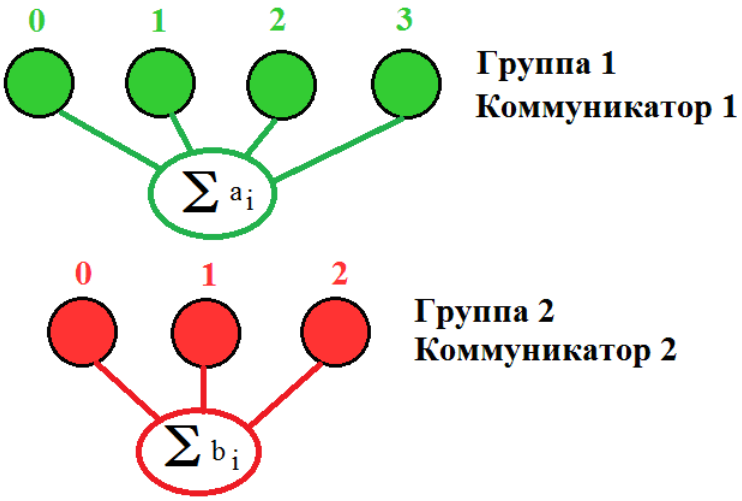
```
>mpicc -o ex09a.px -O2 ex09a.c mynet.c -lm
```

Результаты выполнения:

```
>mpirun -np 2 -nolocal -machinefile hosts ex09a.px  (запуск в классе)
>mpirun -np 2 ex09a.px                              (запуск на сервере)
00 <-> 01
mp=0 i=0 a=0 b=1
mp=0 i=1 a=0 b=-32766
mp=0 i=2 a=-32766 b=0
mp=0 i=3 a=1 b=0
01 <-> 00
mp=1 i=0 a=1 b=0
mp=1 i=1 a=-32766 b=0
mp=1 i=2 a=0 b=-32766
mp=1 i=3 a=0 b=1
```

```
>mpirun -np 3 -nolocal -machinefile hosts ex09a.px (запуск в классе)
>mpirun -np 3 ex09a.px                             (запуск на сервере)
mp=2 i=0 a=2 b=0
mp=2 i=1 a=-32766 b=-1078937432
mp=2 i=2 a=1 b=-1078938104
mp=2 i=3 a=-32766 b=134513195
00 <-> 01
mp=0 i=0 a=0 b=1
mp=0 i=1 a=0 b=-32766
mp=0 i=2 a=-32766 b=0
mp=0 i=3 a=1 b=0
01 <-> 00
mp=1 i=0 a=1 b=0
mp=1 i=1 a=-32766 b=0
mp=1 i=2 a=0 b=-32766
mp=1 i=3 a=0 b=1
>mpirun -np 4 -nolocal -machinefile hosts ex09a.px (запуск в классе)
>mpirun -np 4 ex09a.px                             (запуск на сервере)
00 <-> 02
mp=0 i=0 a=0 b=2
mp=0 i=1 a=0 b=-32766
mp=0 i=2 a=-32766 b=0
mp=0 i=3 a=2 b=0
02 <-> 00
mp=2 i=0 a=2 b=0
mp=2 i=1 a=-32766 b=0
mp=2 i=2 a=0 b=-32766
mp=2 i=3 a=0 b=2
01 <-> 03
mp=1 i=0 a=1 b=3
mp=1 i=1 a=1 b=-32766
mp=1 i=2 a=-32766 b=1
mp=1 i=3 a=3 b=1
03 <-> 01
mp=3 i=0 a=3 b=1
mp=3 i=1 a=-32766 b=1
mp=3 i=2 a=1 b=-32766
mp=3 i=3 a=1 b=3
```

Пример 2 – Создание двух групп и соответствующих коммуникаторов.



Код программы (ex09b.c)

```
#include "stdio.h"  #include <math.h>  #include "mynet.h"
static int np, mp, nl;  static char pname[MPI_MAX_PROCESSOR_NAME];
static MPI_Status status;  static double tick, t1, t2, t3;
int main(int argc, char *argv[])
{
  MPI_Group gr, gr1, gr2; MPI_Comm cm1, cm2, cm3;
  int i, np1, np2, mp1, mp2, ranks[128]; double s, p;
  MyNetInit(&argc,&argv,&np,&mp,&nl,pname,&tick);
  if (np<2) mpierr("Too small processes for communications",1); sleep(1);
```

```
  MPI_Comm_group(MPI_COMM_WORLD, &gr);
  np1 = np/2; np2 = np - np1;
  for (i=0; i<np1; i++) ranks[i] = i;
  MPI_Group_incl(gr, np1, ranks, &gr1);
  MPI_Group_excl(gr, np1, ranks, &gr2);
  MPI_Group_rank(gr1, &mp1);
  MPI_Group_rank(gr2, &mp2);
  MPI_Comm_create(MPI_COMM_WORLD, gr1, &cm1);
  MPI_Comm_create(MPI_COMM_WORLD, gr2, &cm2);
  if (cm1>0) cm3 = cm1; else if (cm2>0) cm3 = cm2; else cm3 = 0;
  s = 5.0 * (mp+1); p = 0.0;
  MPI_Reduce(&s, &p, 1, MPI_DOUBLE, MPI_SUM, 0, cm3);
  fprintf(stderr,"mp=%d np1=%d np2=%d mp1=%d mp2=%d s=%le p=%le\n",
    mp,np1,np2,mp1,mp2,cm1,cm2,cm3,s,p);
  MPI_Group_free(&gr);
  MPI_Group_free(&gr1);
  MPI_Group_free(&gr2);
  MPI_Finalize();
  return 0;
}
Трансляция:
>mpicc -o ex09b.px -O2 ex09b.c mynet.c -lm
Результаты выполнения:
>mpirun -np 2 -nolocal -machinefile hosts ex09b.px (запуск в классе)
>mpirun -np 2 ex09b.px                             (запуск на сервере)
mp=0 np1=1 np2=1 mp1=0 mp2=-32766 cm1=135 cm2=0 cm3=135 s=5.000000e+00 p=5.000000e+00
mp=1 np1=1 np2=1 mp1=-32766 mp2=0 cm1=0 cm2=135 cm3=135 s=1.000000e+01 p=1.000000e+01
>mpirun -np 3 -nolocal -machinefile hosts ex09b.px (запуск в классе)
>mpirun -np 3 ex09b.px                             (запуск на сервере)
mp=0 np1=1 np2=2 mp1=0 mp2=-32766 cm1=135 cm2=0 cm3=135 s=5.000000e+00 p=5.000000e+00
mp=1 np1=1 np2=2 mp1=-32766 mp2=0 cm1=0 cm2=135 cm3=135 s=1.000000e+01 p=2.500000e+01
mp=2 np1=1 np2=2 mp1=-32766 mp2=1 cm1=0 cm2=135 cm3=135 s=1.500000e+01 p=0.000000e+00
>mpirun -np 4 -nolocal -machinefile hosts ex09b.px (запуск в классе)
>mpirun -np 4 ex09b.px                             (запуск на сервере)
mp=0 np1=2 np2=2 mp1=0 mp2=-32766 cm1=135 cm2=0 cm3=135 s=5.000000e+00 p=1.500000e+01
mp=1 np1=2 np2=2 mp1=1 mp2=-32766 cm1=135 cm2=0 cm3=135 s=1.000000e+01 p=0.000000e+00
mp=2 np1=2 np2=2 mp1=-32766 mp2=0 cm1=0 cm2=135 cm3=135 s=1.500000e+01 p=3.500000e+01
mp=3 np1=2 np2=2 mp1=-32766 mp2=1 cm1=0 cm2=135 cm3=135 s=2.000000e+01 p=0.000000e+00
```

3. Параллельная сортировка.

Параллельная сортировка распределенного массива.

Топология: линейка процессоров. Синхронные обмены.

Алгоритмы сортировки – пирамидальная ($8N\log_2 N$), улучшенная пирамидальная ($6N\log_2 N$), Бэтчера (параллельная $N(\log_2 N)^2$ на одном вычислителе), сортировка слиянием.

Общая задача – отсортировать данные в распределенном массиве, собираемом в разных удаленных друг от друга источниках. Результат должен храниться в этих же источниках (см. Рис. 1).

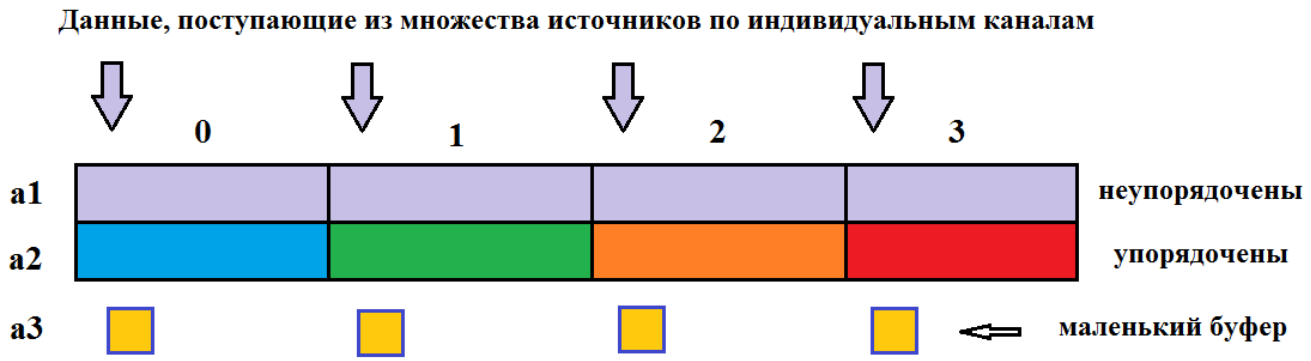


Рис. 1. Иллюстрация к задаче сортировки данных в распределенном массиве.

При слиянии двух массивов a1 и a2 необходимо, чтобы все минимальные значения попали в массив a1, все максимальные – в массив a2. При этом важное значение имеет размер буфера a3 (см. Рис. 2).

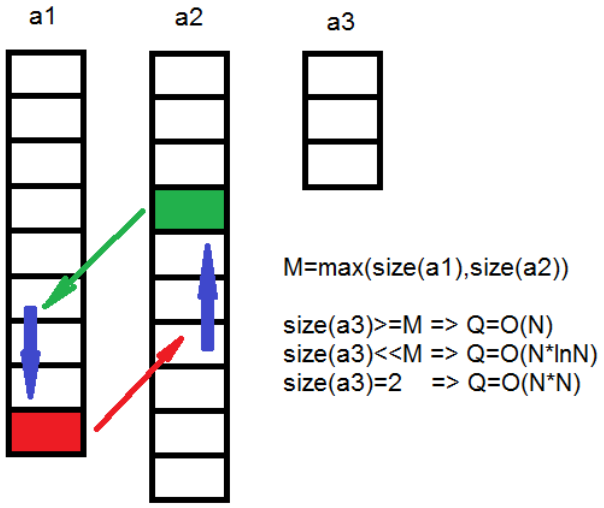


Рис. 2. Иллюстрация к проблеме слияния двух упорядоченных массивов.

Пример-заготовка (ex10a.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mynet.h"
#include "myrand.h"
static int np, mp, nl;
static char pname[MPI_MAX_PROCESSOR_NAME];
static MPI_Status status;
static double tick, t1;
int main(int argc, char *argv[])
{
  int ns, ncm, nc, nn, i1, i2, i, ii; double amin, amax;
  int* ind = NULL; double* a1 = NULL; double* a2 = NULL;
  MyNetInit(&argc,&argv,&np,&mp,&nl,pname,&tick);
  if (argc<1) ns = 10000; else sscanf(argv[1],"%d",&ns); if (ns<np) ns = 10000;
  ind = (int*)(malloc(sizeof(int)*3*np));
  myindexes(np,ns,ind);
  i1 = ind[3*mp]; i2 = ind[3*mp+1]; nc = ind[3*mp+2];
  fprintf(stderr,"mp=%d ns=%d i1=%d i2=%d nc=%d\n",mp,ns,i1,i2,nc);
  t1 = MPI_Wtime();
  if (np==1) {
    a1 = (double*)(malloc(sizeof(double)*nc));
    for (i=0; i<nc; i++) a1[i] = myrand();
    qsort(a1, nc, sizeof(double), mycomp);
  }
  else {
    ncm = ns / np + 1;
    a1 = (double*)(malloc(sizeof(double)*ncm));
    a2 = (double*)(malloc(sizeof(double)*ncm));
    if (mp==0) {
      for (i=0; i<nc; i++) a1[i] = myrand();
      for (ii=1; ii<np; ii++) {
        nn = ind[3*ii+2];
        for (i=0; i<nn; i++) a2[i] = myrand();
        MPI_Send(a2,nn,MPI_DOUBLE,ii,MY_TAG,MPI_COMM_WORLD);
      }
    }
    else
      MPI_Recv(a1,nc,MPI_DOUBLE,0,MY_TAG,MPI_COMM_WORLD,&status);
    qsort(a1,nc,sizeof(double),mycomp);
    for (ii=0; ii<1; ii++){
      if (mp%2==0){
        if (mp+1<np){
          MPI_Sendrecv(a1,ncm,MPI_DOUBLE,mp+1,0,
                       a2,ncm,MPI_DOUBLE,mp+1,0,
```

```
                        MPI_COMM_WORLD,&status);
          //... Здесь нужно вызвать функцию слияния a1, a2
        }
        if (mp>0){
          MPI_Sendrecv(a1,ncm,MPI_DOUBLE,mp-1,0,
                       a2,ncm,MPI_DOUBLE,mp-1,0,
                       MPI_COMM_WORLD,&status);
          //... Здесь нужно вызвать функцию слияния a1, a2
        }
      }
      else{
        if (mp>0){
          MPI_Sendrecv(a1,ncm,MPI_DOUBLE,mp-1,0,
                       a2,ncm,MPI_DOUBLE,mp-1,0,
                       MPI_COMM_WORLD,&status);
          //... Здесь нужно вызвать функцию слияния a1, a2
        }
        if (mp+1<np){
          MPI_Sendrecv(a1,ncm,MPI_DOUBLE,mp+1,0,
                       a2,ncm,MPI_DOUBLE,mp+1,0,
                       MPI_COMM_WORLD,&status);
          //... Здесь нужно вызвать функцию слияния a1, a2
        }
      }
    }
  }
  amin = a1[0]; amax = a1[nc-1];
  t1 = MPI_Wtime() - t1;
  fprintf(stderr,"mp=%d amin=%le amax=%le t1=%le\n",mp,amin,amax,t1);
  MPI_Finalize();
  return 0;
}
```

Трансляция:

```
>mpicc -o ex10a.px -O2 ex10a.c mynet.c myrand.c -lm
```

Результаты выполнения:

```
>mpirun -np 1 -nolocal -machinefile hosts ex10a.px
mp=0 ns=10000 i1=0 i2=9999 nc=10000
mp=0 amin=1.536087e+00 amax=3.276779e+04 t1=7.759000e-03
>mpirun -np 2 -nolocal -machinefile hosts ex10a.px
mp=0 ns=10000 i1=0 i2=4999 nc=5000
mp=1 ns=10000 i1=5000 i2=9999 nc=5000
mp=0 amin=1.536087e+00 amax=3.276779e+04 t1=1.320900e-02
mp=1 amin=2.633987e+00 amax=3.275803e+04 t1=9.561000e-03
```