Введение в программирование с использованием MPI

MPI стандарт для построения параллельных программ для вычислительных систем с распределенной памятью

Основы MPI

- MPI реализует модель параллельного программирования для систем с распределенной памятью
- Не требует специальных компиляторов, реализуется в виде библиотек
- Содержит более 300 функций

Модель MPI

- Параллельная программа состоит из процессов, процессы могут быть многопоточными.
- MPI реализует передачу сообщений между процессами.
- Основная схема взаимодействия между 2-мя процессами: схема «рукопожатия» – процессы согласовывают передачу .
- Межпроцессное взаимодействие предполагает:
  - синхронизацию
  - перемещение данных из адресного пространства одного процесса в адресное пространство другого процесса.

6 основных функций MPI

- **Как стартовать/завершить параллельное выполнение**
  - MPI_Init
  - MPI_Finalize
- **Кто я (и другие процессы), сколько нас**
  - MPI_Comm_rank
  - MPI_Comm_size
- **Как передать сообщение коллеге (другому процессу)**
  - MPI_Send
  - MPI_Recv

Замер времени MPI_Wtime

- Время замеряется в секундах
- Выделяется интервал в программе

```
double MPI_Wtime(void)
```

Пример.

```
double start, finish, time ;
start=-MPI_Wtime;
MPI_Send(…);
finish = MPI_Wtime();
time= start+finish;
```

Сумма элементов вектора с использованием MPI_Send и MPI_Recv

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define N_LOCAL 1024
int main(int argc, char *argv[])
{ double sum_local, sum_global, start, finish;
  double a[N_LOCAL];
  int  i, n =N_LOCAL;
  int size, myrank;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Сумма элементов вектора с использованием MPI_Send и MPI_Recv

```
If (!myrank)  /* process-root */
{ start = MPI_Wtime(); sum_global=0;
  for (i=1; i<size; i=+) {
  MPI_Recv( &sum_local, 1, MPI_DOUBLE, MPI_ANY_SOURCE,
  MPI_ANY_TAG, MPI_COMM_WORLD, &status);
  sum_global += sum_local; }
  printf (“ Sum of vector elements = %f \n Time =%f\n”, sum_global,
MPI_Wtime() - start);
}
 else { /* processes: 1 .. size */
sum_local =0;
for(i=0; i<n; i++) sum_local+= a[i];
 MPI_Send (&sum_local, 1 ,  MPI_DOUBLE, 0,0, MPI_COMM_WORLD);
}
MPI_Finalize();
 exit (0); }
```

MPI_Probe

int MPI_Probe(int *source*, int *tag*, MPI_Comm *comm*, MPI_Status* *status*)

Проверка статуса операции приема сообщения.

Параметры аналогичны функции MPI_Recv

Deadlocks

- Процесс 0 посылает большое сообщение процессу 1
  - Если в принимающем процессе недостаточно места в системном буфере, процесс 0 должен ждать пока процесс 1 не предоставит необходимый буфер.
  - Что произойдет:

Process 0	Process 1
Send(1)	**Send(0)**
Recv(1)	**Recv(0)**

- Называется “unsafe” потому, что зависит от системного буфера.

Пути решения «ubsafe» передач

- Упорядочить передачи:

Process 0	Process 1
`Send(1)`	`Recv(0)`
`Recv(1)`	`Send(0)`

- Использовать неблокирующие передачи:

Process 0	Process 1
`Isend(1)`	`Isend(0)`
`Irecv(1)`	`Irecv(0)`
`Waitall`	`Waitall`

12

Неблокирующие коммуникации

Цель – уменьшение времени работы параллельной программы за счет совмещения вычислений и обменов.

Неблокирующие операции завершаются, не дожидаясь окончания передачи данных. В отличие от аналогичных блокирующих функций изменен критерий завершения операций – немедленное завершение.

Проверка состояния передач и ожидание завершение передач выполняются специальными функциями.

Форматы неблокирующих функций

```
MPI_Isend(buf,count,datatype,dest,tag,comm,request)

MPI_Irecv(buf,count,datatype,source,tag,comm,
request)
```

request - “квитанция» о завершении передачи.
Тип: MPI_Requset
MPI_REQUEST_NULL - обнуление

MPI_Wait() ожидание завершения.
MPI_Test() проверка завершения. Возвращается флаг, указывающий на результат завершения.

Асинхронные передачи

*int MPI_Isend(void *buf,int count, MPI_Datatype datatype,int dest, int tag, MPI_Comm comm, MPI_Requset request)*

*int MPI_Irecv(void *buf,int count, MPI_Datatype datatype,int source, int tag, MPI_Comm comm, MPI_Status *status, MPI_Requset request)*

*int MPI_Iprobe(int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status);*

*int MPI_Wait(MPI_Request *request, MPI_Status *status)*

*int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)*

flag : true, если передача завершена

Ожидание завершения асинхронных передач

int **MPI_Waitall (int count, MPI_Request array_of_requests[],**
MPI_Status array_of_statuses[])
// waits for all given communications to finish and fills in the statuses
Int **MPI_Waitany (int count, MPI_Request array_of_requests[], int *index,**
MPI_Status *status)
// waits for one of the given communications to finish, sets the index to indicate
// which one and fills in the status
int **MPI_Waitsome (int incount, MPI_Request array_of_requests[],**
int *outcount, int array_of_indices[], MPI_Status array_of_statuses[])
// waits for at least one of the given communications to finish, sets the number
// of communication requests that have finished, their indices and status

Проверка состояния передач

int **MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)**

// tests if the communication is finished. Sets flag to 1 and fills in the status if
// finished or sets the flag to 0 if not finished.

int **MPI_Testall(int count, MPI_Request array_of_requests[], int *flag,**
MPI_Status array_of_statuses[])

// test whether all given communications are finished. Sts flag to 1 and fills in
// the status aray if all are finished or sets the flag to 0 if not all are finished.

int **MPI_Testany(int count, MPI_Request array_of_requests[], int *index,**
int *flag, MPI_Status *status)

// test whether one of the given communications is finished. Sets flag to 1 and fills
// in the index and status if one finished or sets the flag to 0 if none is finished.

int **MPI_Testsome(int incount, MPI_Request array_of_requests[], int *outcount,**
int array_of_indices[], MPI_Status array_of_statuses[])

// tests whether some of the given communications is finished, sets the number
// of communication requests that have finished, their indices and statuses.

int **MPI_Cancel(MPI_Request *request)**

Коллективные передачи

- Передача сообщений между группой процессов
- Вызываются ВСЕМИ процессами в коммуникаторе
- Примеры:
  - Broadcast, scatter, gather (рассылка данных)
  - Global sum, global maximum, и т.д. (Коллективные операции)
  - Барьерная синхронизация

Характеристики коллективных передач

- Коллективные операции не являются помехой операциям типа «точка-точка» и наоборот
- Все процессы коммуникатора должны вызывать коллективную операцию
- Синхронизация не гарантируется (за исключением барьера).
  Завершение операции – локально в процессе
- Нет тэгов
- Принимающий буфер должен точно соответствовать размеру отсылаемого буфера
- Асинхронные коллективные передачи - в MPI-3

Функции коллективной передачи

- MPI_Bcast() – Broadcast (one to all)
- MPI_Reduce() – Reduction (all to one)
- MPI_Allreduce() – Reduction (all to all)
- MPI_Scatter() – Distribute data (one to all)
- MPI_Gather() – Collect data (all to one)
- MPI_Alltoall() – Distribute data (all to all)
- MPI_Allgather() – Collect data (all to all)

Реализации MPI на суперкомпьютере Харизма

```
[mkhalilov@sms ~]$ module avail
```

```
BEAST/v1.10.4                                OpenBlas/v0.3.18                          (D)  gnu/5.4.0                                 matlab/r2020b                          (D)
BEAST/v2.6.2                                 OpenPose/1.6                                   gnu10/10.1                                nvidia_sdk/nvhpc-byo-compiler/20.7
BEAST/v2.6.3                          (D)    Python/Anaconda_v10.2019                       gnu7/7.3.0                                nvidia_sdk/nvhpc-byo-compiler/21.9 (D)
CUDA/10.0                                    Python/Anaconda_v11.2020                       gnu8/8.2.0                                nvidia_sdk/nvhpc-nompi/20.7
CUDA/10.2                                    Python/Miniconda_v4.7.12.1                     gnu8/8.3.0                         (L,D)  nvidia_sdk/nvhpc-nompi/21.9            (D)
CUDA/11.0                                    Python/Miniconda_v4.9.2                   (D)  gnu9/9.3                                  nvidia_sdk/nvhpc/20.7
CUDA/11.2                                    QuantumEspresso/v6.38_gnu                      gperf/3.1                                 nvidia_sdk/nvhpc/21.9_no_cuda
CUDA/11.4                             (D)    QuantumEspresso/v6.38_pgi_mkl_NO_mpi-gpu-aware graphics_magick/v1.3.34                   nvidia_sdk/nvhpc/21.9                  (D)
EasyBuild/3.7.1                              QuantumEspresso/v6.38_pgi_mkl             (D)  hpcx/hpcx-cuda-ompi                       ohpc                                   (L)
GROMACS/2019.6                               R/v3.6.1                                       hpcx/hpcx-cuda                            openmpi/4.0.1
GROMACS/2020.6_plumed_v2.7.2                 R/v4.0.2                                       hpcx/hpcx-mt-ompi                         openmpi/4.0.5                          (D)
GROMACS/xdrfile/v1.1.4                       R/v4.0.3                                  (D)  hpcx/hpcx-mt                              papi/5.6.0
INTEL/oneAPI_2021_u2_env                     angsd/v0.933                                   hpcx/hpcx-ompi                            plumed/v2.7.2
INTEL/oneAPI_2021_u2                         autotools                                 (L)  hpcx/hpcx-prof-ompi                       pmix/2.2.2
INTEL/opencl_sdk                             beagle/v3.1.2                                  hpcx/hpcx-prof                            primme/v3.1.1
INTEL/parallel_studio_xe_2018_u2_ce          blas/v3.10.0                                   hpcx/hpcx-stack                           prun/1.2
INTEL/parallel_studio_xe_2020_ce             charliecloud/0.9.2                             hpcx/hpcx                         (D)  prun/1.3                               (L,D)
INTEL/parallel_studio_xe_2020_u1_ce          clustershell/1.8.2                             hwloc/v2.6.0                              singularity/2.6.0
INTEL/parallel_studio_xe_2020_u4_ce   (D)    cmake/3.9.4                                    hwloc/1.11.10                     (D)  singularity/3.2.0                      (D)
IQ-Tree/v2.0.4                               cmake/3.12.2                                   jags/v4.3.0                               sratoolkit/v2.10.8
MPLUS/v8.4                                   cmake/3.21.3                              (D)  java/v1.8                                 tools/AdmixTools/7.0.1
Octave/v5.2.0                                cnpy_lib/1.0                                   julia/v1.5.3                              tools/git/v2.33
OpenBUGS/v3.2.3                              fftw/v3.3.8                                    libs/gsl/2.6                              valgrind/3.13.0
OpenBlas/v0.3.7                              fltk/v1.3.5                                    matlab/r2020a
```

```
[mkhalilov@sms ~]$ which mpirun
/opt/ohpc/pub/mpi/openmpi3-gnu8/3.1.4/bin/mpirun
```

Запуск MPI-программ на суперкомпьютере Харизма

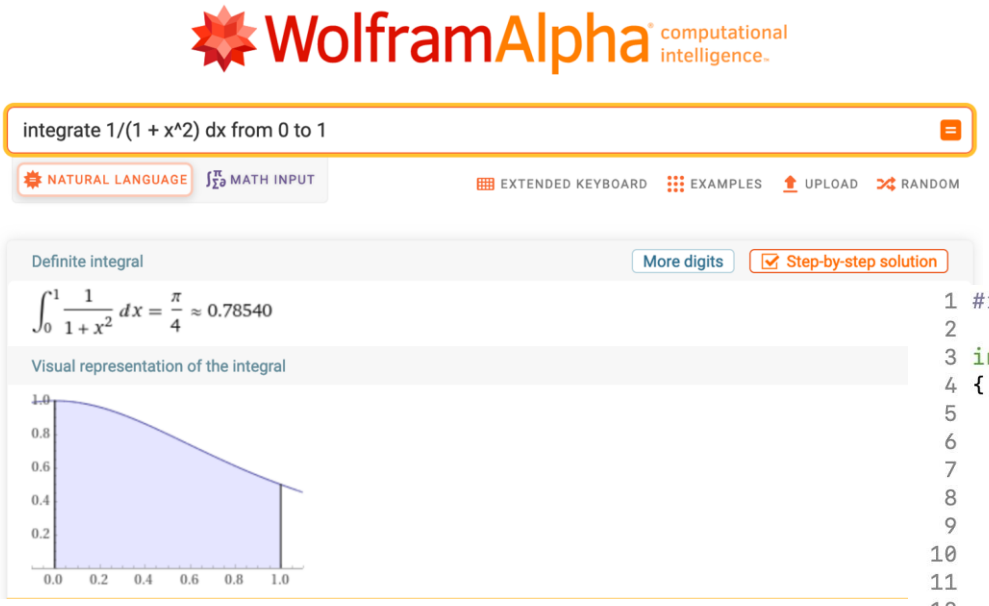
```cpp
#include <iostream>
#include <mpi.h>

int main(int argc, char **argv) {
    int my_rank, total_ranks;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &total_ranks);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    std::cout << "Hello from rank " << my_rank << " out of " << total_ranks << std::endl;

    MPI_Finalize();
}
```

```
[mkhalilov@sms ~]$ mpicxx hello_mpi.c -o hello_mpi
[mkhalilov@sms ~]$ srun -N 4 --ntasks-per-node=1 ./hello_mpi
Hello from rank 2 out of 4
Hello from rank 1 out of 4
Hello from rank 0 out of 4
Hello from rank 3 out of 4
[mkhalilov@sms ~]$ salloc -N 4 --ntasks-per-node=1
salloc: Granted job allocation 407718
salloc: Waiting for resource configuration
salloc: Nodes cn-[034-037] are ready for job
[mkhalilov@sms ~]$ ssh cn-034
[mkhalilov@cn-034 ~]$ mpirun --host cn-034:1,cn-035:1,cn-036:1,cn-037:1 ./hello_mpi
Warning: Permanently added 'cn-036,192.168.1.36' (ECDSA) to the list of known hosts.
Warning: Permanently added 'cn-037,192.168.1.37' (ECDSA) to the list of known hosts.
Warning: Permanently added 'cn-035,192.168.1.35' (ECDSA) to the list of known hosts.
Hello from rank 0 out of 4
Hello from rank 1 out of 4
Hello from rank 2 out of 4
Hello from rank 3 out of 4
```

Пересылки точка-точка

```cpp
#include <iostream>
#include <mpi.h>

int main( int argc, char *argv[] )
{
    int world_rank;
    int world_size;
    int number = 0;
    MPI_Status status;

    MPI_Init(&argc, &argv );
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    if (world_rank == 0) {
        number = 42;
        std::cout << "Process 0 sending number " << number << " to process 1" << std::endl;
        MPI_Send(&number, 1, MPI_INT, 1, 123, MPI_COMM_WORLD);
    } else if (world_rank == 1) {
        std::cout << "Number = " << number << " on process 1 before receiving" << std::endl;
        MPI_Recv(&number, 1, MPI_INT, 0, 123, MPI_COMM_WORLD, &status);
        std::cout << "Process 1 received number " << number << "from process 0" << std::endl;
    }

    MPI_Finalize();
}
```

Численная аппроксимация числа Пи




```cpp
#include <iostream>

int main(int argc, char **argv)
{
    size_t nsteps = 0;
    double x = 0.0, sum = 0.0, step = 0.0, pi = 0.0;

    std::cin >> nsteps;

    step = 1.0 / static_cast<double>(nsteps);
    for (size_t i = 0; i < nsteps; i++) {
        x = (i + 0.5) * step;
        sum = sum + 4.0 / (1.0 + x*x);
    }
    pi = sum * step;

    std::cout.precision(25);
    std::cout << "pi sum: " << pi << std::endl;

    return 0;
}
```

Численная аппроксимация числа Пи. Параллельная реализация.

```cpp
#include <iostream>
#include <mpi.h>

int main(int argc, char **argv)
{
    size_t nsteps = 0;
    double x = 0.0, sum = 0.0, step = 0.0, pi = 0.0;
    int my_rank = 0, total_ranks = 0;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &total_ranks);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    if (my_rank == 0) {
        std::cin >> nsteps;
    }

    MPI_Bcast(&nsteps, 1, MPI_INT, 0, MPI_COMM_WORLD);

    step = 1.0 / static_cast<double>(nsteps);
    for (size_t i = my_rank; i < nsteps; i += total_ranks) {
        x = (i + 0.5) * step;
        sum = sum + 4.0 / (1.0 + x*x);
    }
    sum = sum * step;

    MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE, MPI_SUM,
               0, MPI_COMM_WORLD);

    if (my_rank == 0) {
        std::cout.precision(25);
        std::cout << "pi sum: " << pi << std::endl;
    }

    MPI_Finalize();
    return 0;
}
```

Литература

- “Суперкомпьютерное моделирование и технологии. Технология параллельного программирования MPI”, ВМК МГУ, Н.Н. Попова
- *Антонов А. С.* Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В.А.Садовничий. – Издательство Московского университета М.:, 2012. – С. 344.