Programování architektur založených na posílání zpráv

Úvod

Standard MPI

Úvod

Programování architektur založených na posílání zpráv:

- synchronizace probíhá pomocí posílání zpráv
- je vhodné pro nepříliš úzce synchronní výpočty

Standard MPI

Standard pro posílání zpráv - MPI = Message Passing Interface
Dostupné implementace

- OpenMPI - `http://www.open-mpi.org/`
- LAM-MPI - `http://www.lam-mpi.org/`
- MPICH - `http://www-unix.mcs.anl.gov/mpi/mpich/`
- Intel, HP, ...

Zdroje na internetu:
`http://www-unix.mcs.anl.gov/mpi/`
Wikipedia

Základ kódu pro MPI

```
#include <mpi.h>
int main( int argc, char* argv[] )
{
  MPI_Init( argc, argv );
  ...
  MPI_Finalize();
}
```

Základ kódu pro MPI

```
#include <mpi.h>
int main( int argc, char* argv[] )
{
  MPI_Init( argc, argv );
  ...
  MPI_Finalize();
}
```

Funkce `MPI_Init` a `MPI_Finalize` musí být volány právě jednou a všemi procesy. Vrácená hodnota by měla být `MPI_SUCCES`.

Základ kódu pro MPI

```
#include <mpi.h>
int main( int argc, char* argv[] )
{
  MPI_Init( argc, argv );
  ...
  MPI_Finalize();
}
```

Funkce `MPI_Init` a `MPI_Finalize` musí být volány právě jednou a všemi procesy. Vrácená hodnota by měla být `MPI_SUCCES`.

```
shell$ mpicc -o foo foo.c
shell$ mpif77 -o foo foo.f
mpirun -v -np 2 foo
```

Komunikační skupiny

Komunikační skupiny určují, které procesy se budou účastnit zvolené operace.

Jde o tzv. COMMUNICATORS s typem `MPI_Comm`.

Všechny běžící procesy jsou obsaženy ve skupině `MPI_COMM_WORLD`.

Informace o procesech

Počet procesů (`size`) v dané skupině lze zjistit pomocí:

```
int MPI_Comm_size( MPI_Comm comm, int* size );
```

Identifikační číslo procesu (`rank`) vůči dané skupině lze zjistit pomocí:

```
int MPI_Comm_rank( MPI_Comm comm, int* rank );
```

Kostra jednoduché aplikace

```
#include <mpi.h>
int main( int argc, char* argv[] )
{
   int nproc, iproc;
   Config config;
   InputData input_data;
   OutputData output_data;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nproc);
   MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
   if(iproc == 0)
   {
      ParseConfigurationParameters(&config,&argc,&argv);
      GetInputData(&config,&input_data);
   }
   Broadcast(&config, 0);
   Scatter(&input_data, 0);
   Compute(&input_data, &output_data);
   Gather(&output_data, 0);
   if(iproc == 0) WriteOutput(&output_data);
   MPI_Finalize();
}
```

Funkce `send` a `receive` I.

```
int MPI_Send( void* buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm );
int MPI_Receive( void* buf, int count,
                 MPI_Datatype datatype, int source,
                 int tag, MPI_Comm comm,
                 MPI_Status* status );
```

Funkce `send` a `receive` I.

```
int MPI_Send( void* buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm );
int MPI_Receive( void* buf, int count,
                 MPI_Datatype datatype, int source,
                 int tag, MPI_Comm comm,
                 MPI_Status* status );
```

- `buf` ukazatel na pole dat typu `datatype` o velikosti `count`

Funkce `send` a `receive` I.

```
int MPI_Send( void* buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm );
int MPI_Receive( void* buf, int count,
                 MPI_Datatype datatype, int source,
                 int tag, MPI_Comm comm,
                 MPI_Status* status );
```

- `buf` ukazatel na pole dat typu `datatype` o velikosti `count`
- `datatype` může být: `MPI_CHAR`, `MPI_INT`, `MPI_FLOAT`, `MPI_DOUBLE`, ...

Funkce `send` a `receive` I.

```
int MPI_Send( void* buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm );
int MPI_Receive( void* buf, int count,
                 MPI_Datatype datatype, int source,
                 int tag, MPI_Comm comm,
                 MPI_Status* status );
```

- `buf` ukazatel na pole dat typu `datatype` o velikosti `count`
- `datatype` může být: `MPI_CHAR`, `MPI_INT`, `MPI_FLOAT`, `MPI_DOUBLE`, ...
- `dest` ID příjemce

Funkce `send` a `receive` I.

```
int MPI_Send( void* buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm );
int MPI_Receive( void* buf, int count,
                 MPI_Datatype datatype, int source,
                 int tag, MPI_Comm comm,
                 MPI_Status* status );
```

- `buf` ukazatel na pole dat typu `datatype` o velikosti `count`
- `datatype` může být: `MPI_CHAR`, `MPI_INT`, `MPI_FLOAT`, `MPI_DOUBLE`, ...
- `dest` ID příjemce
- `source` ID odesílatele

Funkce `send` a `receive` I.

```
int MPI_Send( void* buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm );
int MPI_Receive( void* buf, int count,
                 MPI_Datatype datatype, int source,
                 int tag, MPI_Comm comm,
                 MPI_Status* status );
```

- `buf` ukazatel na pole dat typu `datatype` o velikosti `count`
- `datatype` může být: `MPI_CHAR`, `MPI_INT`, `MPI_FLOAT`, `MPI_DOUBLE`, ...
- `dest` ID příjemce
- `source` ID odesílatele
- `tag` určuje typ zprávy

Funkce `send` a `receive` I.

```
int MPI_Send( void* buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm );
int MPI_Receive( void* buf, int count,
                 MPI_Datatype datatype, int source,
                 int tag, MPI_Comm comm,
                 MPI_Status* status );
```

- `buf` ukazatel na pole dat typu `datatype` o velikosti `count`
- `datatype` může být: `MPI_CHAR`, `MPI_INT`, `MPI_FLOAT`, `MPI_DOUBLE`, ...
- `dest` ID příjemce
- `source` ID odesílatele
- `tag` určuje typ zprávy
- `comm` udává komunikační skupinu

Struktura `MPI_Status`

```
typedef struct MPI_Status {
          int MPI_SOURCE;
          int MPI_TAG;
          int MPI_ERROR;
};
```

- `MPI_SOURCE` = odesílatel
- `MPI_SOURCE` = typ zprávy
- `MPI_ERROR` = chybové hlášení

Funkce `send` a `receive` II.

U odesílatele i příjemce musí být `count`, `datatype` a `tag` stejné.

Funkce `send` a `receive` II.

U odesílatele i příjemce musí být `count`, `datatype` a `tag` stejné.

Funkce

```
int MPI_Get_count( MPI_Status* status,
                   MPI_Datatype datatype,
                   int* count )
```

udává skutečný počet přijatých dat.

Funkce `send` a `receive` II.

U odesílatele i příjemce musí být `count`, `datatype` a `tag` stejné.

Funkce

```
int MPI_Get_count( MPI_Status* status,
                   MPI_Datatype datatype,
                   int* count )
```

udává skutečný počet přijatých dat.

- obě funkce jsou vždy blokující
- `send` může být implementováno bufferově (nelze s tím počítat, pozor na deadlock)

Funkce `send` a `receive` - příklad **s deadlockem**

```
int a[ 10 ], b[ 10 ], myrank;
MPI_Status status;
...
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
if( myrank == 0 ) {
      MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
      MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
}
else if( myrank == 1 ) {
      MPI_Recv(b, 10, MPI_INT, 0, 2, MPI_COMM_WORLD);
      MPI_Recv(a, 10, MPI_INT, 0, 1, MPI_COMM_WORLD);
}
```

Funkce `send` a `receive` - příklad **bez deadlocku**

```
int a[ 10 ], b[ 10 ], myrank;
MPI_Status status;
...
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
if( myrank == 0 ) {
      MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
      MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
}
else if( myrank == 1 ) {
      MPI_Recv(a, 10, MPI_INT, 0, 1, MPI_COMM_WORLD);
      MPI_Recv(b, 10, MPI_INT, 0, 2, MPI_COMM_WORLD);
}
```

Simultání send a receive

Za účelem zabránění deadlocku je často lepší použít funkce pro simultání `send/receive`.

```
int MPI_Sendrecv( void* sendbuf, int sendcoutn,
            MPI_Datatype senddatatype, int dset,
            int sendtag, void* recvbuf,
            int recvcount, MPI_Datatype recvdatatype,
            int source, int recvtag,
            MPI_Comm comm, MPI_Status* status )
```

```
int MPI_Sendrecv_replace( void* buf, int count,
            MPI_Datatype datatype, int dest,
            int sendtag, int source,
            int recvtag, MPI_Comm comm,
            MPI_Status* status )
```

V případě `MPI_Sendrecv_replace` jsou odeslaná data přepsána přijatými.

Neblokující send a receive I.

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

int MPI_Irecv( void* buf, int count,
               MPI_Datatype datatype, int source,
               int tag, MPI_Comm comm,
               MPI_Request* request )
```

Obě funkce vracejí řízení programu dříve, než jsou data skutečně přenesena.

- `request` - slouží k ověření, zda byla data již přenesena

Neblokující send a receive II.

Funkce pro ověření stavu přenosu dat při neblokujícím send a receive.

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

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

- `MPI_Test` vrací v proměnné `flag` nenulovou hodnotu, pokud již operace skončila
- `MPI_Wait` čeká na ukončení operace

Hromadné komunikační operace I.

- Bariéra

```
int MPI_Barrier( MPI_Comm comm )
```

Hromadné komunikační operace I.

- Bariéra

```
int MPI_Barrier( MPI_Comm comm )
```

- One-to-all broadcast

```
int MPI_Bcast( void* buf, int count, MPI_Datatype,
               int source, MPI_Comm comm )
MPI_Bcast( &x, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
```

Hromadné komunikační operace I.

- Bariéra

```
int MPI_Barrier( MPI_Comm comm )
```

- One-to-all broadcast

```
int MPI_Bcast( void* buf, int count, MPI_Datatype,
               int source, MPI_Comm comm )
MPI_Bcast( &x, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
```

- All-to-one reduction

```
int MPI_Reduce( void* sendbuf, void* recvbuf,
                int count, MPI_Datatype datatype,
                MPI_Op op, int target, MPI_Comm comm)
```

Hromadné komunikační operace I.

- Bariéra

```
int MPI_Barrier( MPI_Comm comm )
```

- One-to-all broadcast

```
int MPI_Bcast( void* buf, int count, MPI_Datatype,
               int source, MPI_Comm comm )
MPI_Bcast( &x, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
```

- All-to-one reduction

```
int MPI_Reduce( void* sendbuf, void* recvbuf,
                int count, MPI_Datatype datatype,
                MPI_Op op, int target, MPI_Comm comm)
```

  - op = operace: `MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD`

Hromadné komunikační operace I.

- Bariéra

```
int MPI_Barrier( MPI_Comm comm )
```

- One-to-all broadcast

```
int MPI_Bcast( void* buf, int count, MPI_Datatype,
               int source, MPI_Comm comm )
MPI_Bcast( &x, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
```

- All-to-one reduction

```
int MPI_Reduce( void* sendbuf, void* recvbuf,
                int count, MPI_Datatype datatype,
                MPI_Op op, int target, MPI_Comm comm)
```

  - op = operace: `MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD`
  - výsledek se uloží do `recvbuf` procesu `target`

Hromadné komunikační operace I.

- Bariéra

```
int MPI_Barrier( MPI_Comm comm )
```

- One-to-all broadcast

```
int MPI_Bcast( void* buf, int count, MPI_Datatype,
               int source, MPI_Comm comm )
MPI_Bcast( &x, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
```

- All-to-one reduction

```
int MPI_Reduce( void* sendbuf, void* recvbuf,
                int count, MPI_Datatype datatype,
                MPI_Op op, int target, MPI_Comm comm)
```

  - op = operace: `MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD`
  - výsledek se uloží do `recvbuf` procesu `target`
  - `MPI_Reduce( &x, &max_x, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD );`

Hromadné komunikační operace II.

- All-to-all reduction

```
int MPI_Allreduce( void* sendbuf, void* recvbuf,
                   int count, MPI_Datatype datatype,
                   MPI_Op op, MPI_Comm comm )
```

Hromadné komunikační operace II.

- All-to-all reduction

```
int MPI_Allreduce( void* sendbuf, void* recvbuf,
                   int count, MPI_Datatype datatype,
                   MPI_Op op, MPI_Comm comm )
```

- Prefix sum

```
int MPI_Scan( void* sendbuf, void* recvbuf,
              int count, MPI_Datatype datatype,
              MPI_Op op, MPI_Comm comm )
```

Hromadné komunikační operace III.

- Scatter

```
int MPI_Scatter( void* sendbuf, int sendcount,
                 MPI_Datatype senddatatype,
                 void* recvbuf, int recvcount,
                 MPI_Datatype recvdatatype,
                 int source, MPI_Comm comm )
```

Hromadné komunikační operace III.

- Scatter

```
int MPI_Scatter( void* sendbuf, int sendcount,
                 MPI_Datatype senddatatype,
                 void* recvbuf, int recvcount,
                 MPI_Datatype recvdatatype,
                 int source, MPI_Comm comm )
```

  - `sendcount = recvcount` udává počet prvků posílaných jednomu procesu

Hromadné komunikační operace III.

- Scatter

```
int MPI_Scatter( void* sendbuf, int sendcount,
                 MPI_Datatype senddatatype,
                 void* recvbuf, int recvcount,
                 MPI_Datatype recvdatatype,
                 int source, MPI_Comm comm )
```

  - `sendcount = recvcount` udává počet prvků posílaných jednomu procesu
- Scatter vektorově - každý proces dostane jiný objem dat

```
int MPI_Scatterv( void* sendbuf, int* sendcounts,
                  int* displs,
                  MPI_Datatype senddatatype,
                  void* recvbuf, int recvcount,
                  MPI_Datatype recvdatatype,
                  int source, MPI_Comm comm )
```

Hromadné komunikační operace III.

- Scatter

```
int MPI_Scatter( void* sendbuf, int sendcount,
                 MPI_Datatype senddatatype,
                 void* recvbuf, int recvcount,
                 MPI_Datatype recvdatatype,
                 int source, MPI_Comm comm )
```

  - `sendcount = recvcount` udává počet prvků posílaných jednomu procesu
- Scatter vektorově - každý proces dostane jiný objem dat

```
int MPI_Scatterv( void* sendbuf, int* sendcounts,
                  int* displs,
                  MPI_Datatype senddatatype,
                  void* recvbuf, int recvcount,
                  MPI_Datatype recvdatatype,
                  int source, MPI_Comm comm )
```

  - `sendcounts` ukazatel na pole udávájící počet prvků posílaných danému procesu

Hromadné komunikační operace III.

- Scatter

```
int MPI_Scatter( void* sendbuf, int sendcount,
                 MPI_Datatype senddatatype,
                 void* recvbuf, int recvcount,
                 MPI_Datatype recvdatatype,
                 int source, MPI_Comm comm )
```

  - `sendcount = recvcount` udává počet prvků posílaných jednomu procesu
- Scatter vektorově - každý proces dostane jiný objem dat

```
int MPI_Scatterv( void* sendbuf, int* sendcounts,
                  int* displs,
                  MPI_Datatype senddatatype,
                  void* recvbuf, int recvcount,
                  MPI_Datatype recvdatatype,
                  int source, MPI_Comm comm )
```

  - `sendcounts` ukazatel na pole udávájící počet prvků posílaných danému procesu
  - `displs` ukazatel na pole udávající pozici dat pro daný proces

Hromadné komunikační operace IV.

- Gather

```
int MPI_Gather( void* sendbuf, int sendcount,
                MPI_Datatype senddatatype,
                void* recvbuf, int recvcount,
                MPI_Datatype recvdatatype,
                int target, MPI_Comm comm )
```

Hromadné komunikační operace IV.

- Gather

```
int MPI_Gather( void* sendbuf, int sendcount,
                MPI_Datatype senddatatype,
                void* recvbuf, int recvcount,
                MPI_Datatype recvdatatype,
                int target, MPI_Comm comm )
```

  - `recvcount` udává počet prvků získaných od daného procesu

Hromadné komunikační operace IV.

- Gather

```
int MPI_Gather( void* sendbuf, int sendcount,
                MPI_Datatype senddatatype,
                void* recvbuf, int recvcount,
                MPI_Datatype recvdatatype,
                int target, MPI_Comm comm )
```

  - `recvcount` udává počet prvků získaných od daného procesu

- Gather vektorově - každý proces dostane jiný objem dat

```
int MPI_Scatterv( void* sendbuf, int sendcount,
                  MPI_Datatype senddatatype,
                  void* recvbuf, int* recvcount,
                  int* displs,
                  MPI_Datatype recvdatatype,
                  int target, MPI_Comm comm )
```

Hromadné komunikační operace IV.

- Gather

```
int MPI_Gather( void* sendbuf, int sendcount,
                MPI_Datatype senddatatype,
                void* recvbuf, int recvcount,
                MPI_Datatype recvdatatype,
                int target, MPI_Comm comm )
```

  - `recvcount` udává počet prvků získaných od daného procesu

- Gather vektorově - každý proces dostane jiný objem dat

```
int MPI_Scatterv( void* sendbuf, int sendcount,
                  MPI_Datatype senddatatype,
                  void* recvbuf, int* recvcount,
                  int* displs,
                  MPI_Datatype recvdatatype,
                  int target, MPI_Comm comm )
```

  - `recvcounts` ukazatel na pole udávájící počet prvků získaných od daného procesu

Hromadné komunikační operace IV.

- Gather

```
int MPI_Gather( void* sendbuf, int sendcount,
                MPI_Datatype senddatatype,
                void* recvbuf, int recvcount,
                MPI_Datatype recvdatatype,
                int target, MPI_Comm comm )
```

  - `recvcount` udává počet prvků získaných od daného procesu

- Gather vektorově - každý proces dostane jiný objem dat

```
int MPI_Scatterv( void* sendbuf, int sendcount,
                  MPI_Datatype senddatatype,
                  void* recvbuf, int* recvcount,
                  int* displs,
                  MPI_Datatype recvdatatype,
                  int target, MPI_Comm comm )
```

  - `recvcounts` ukazatel na pole udávájící počet prvků získaných od daného procesu
  - `displs` ukazatel na pole udávající pozici dat od daného procesu

Hromadné komunikační operace V.

- All-to-all personalized communication

```
int MPI_Alltoall( void* sendbuf, int sendcount,
                  MPI_Datatype senddatatype,
                  void* recvbuf, int recvcount,
                  MPI_Datatype recvdatatype,
                  MPI_Comm comm)
int MPI_Alltoallv( void* sendbuf, int* sendcounts,
                   int* sdispls,
                   MPI_Datatype senddatatype,
                   void* recvbuf, int* recvcounts,
                   int* rdispls,
                   MPI_Datatype recvdatatype,
                   MPI_Comm comm)
```