MI-SPOL-13

Programový model nad distribuovanou pamětí: MPI (procesy, komunikátory, 2-bodové a skupinové komunikační operace, blokující a neblokující operace a jejich komunikační módy). Hybridní MPI+OpenMP model.

Message Passing Interface: standardizovaný a přenositelný systém zasílání zpráv mezi procesy paralelního programu
Libovolné komunikační sítě s distribuovanou pamětí
Standard definuje syntaxi a sémantiku knihovních funkcí (narozdíl od OpenMP neobsahuje direktivy)
Velké množství MPI knihoven

```
#include <mpi.h>
int main(int argc, char* argv[]){
    // Inicializace knihovny
    MPI_Init(&argc, &argv);

    int proc_num, num_procs;
    // Číslo procesu
    MPI_Comm_rank(MPI_COMM_WORLD, &proc_num);
    // Počet procesů
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    // ...
    // Ukončení práce s knihovnou
    MPI_Finalize();
}
```

OpenMP: přímá podpora překladačů -- přepínač `-fopenmp`
MPI: wrapper překladače -- `mpic++`
Konkrétní překladač lze zvolit

Speciální **spouštěcí nástroje:** `mpirun`
Počet procesů nastaven přepínačem `-np`
Procesy lze spustit na různých počítačích.

MPI procesy **nesdílí paměť**, komunikace výhradně **zasíláním zpráv**.
⇒ veškeré proměnné privátní

Hybrid OpenMP a MPI

Možnosti tvorby programů:

- **Pouze MPI:** na každém jádru/uzlu/procesoru pouze 1 nebo několik málo MPI procesů nedělených na vlákna
- **Kombinace OpenMP a MPI:** na každém uzlu/procesoru 1 nebo několik málo MPI procesů, každý se pomocí OpenMP dělí na vlákna běžící na jádrech

Typická architektura:

- **1 MPI proces na výpočetní uzel:** rozdělení procesu na vlákna odpovídá počtu jader uzlu
- **1 MPI proces na procesor:** 1 vlákno -- 1 jádro procesoru. Často lepší výkon

Inicializace: volání `MPI_Init_thread` s parametrem:

- `MPI_THREAD_SINGLE` : pouze MPI, žádná vlákna
- `MPI_THREAD_FUNNELED` : vícevláknové MPI procesy, pouze hlavní vlákno může volat MPI funkce
- `MPI_THREAD_SERIALIZED` : vícevláknové MPI procesy, v daném okamžiku smí MPI funkce volat pouze 1 vlákno (kritická sekce) -- **jednoportový model**
- `MPI_THREAD_MULTIPLE` : vícevláknové MPI procesy bez omezení -- **všeportový model**

```
int main(int argc, char* argv){
    int provided, required = MPI_THREAD_FUNNELED;
    MPI_Init_thread(&argc, &argv, required, &provided);
    if(provided < required){ /* error */ }
    // ...
}
```

Překlad: `mpic++ -fopenmp ...`
Spuštění: `mpirun -np 2 ./program`

Komunikátory

Každý MPI proces součástí aspoň jedné **skupiny procesů**
V rámci skupiny číslování 0 až $p-1$, v každé skupině má proces jiné číslo

Komunikátor: parametr každé MPI funkce určující **množinu procesů**, v níž komunikace probíhá

Implicitní komunikátor: `MPI_COMM_WORLD` (všechny procesy)

Intra-komunikátor: asociovaný s konkrétní skupinou procesů, komunikace v rámci skupiny
Inter-komunikátor: asociovaný s dvěma skupinami, komunikace mezi nimi

Komunikační operace

2-bodové komunikační operace: komunikace mezi dvěma MPI procesy
Kolektivní komunikační operace: komunikace mezi všemi procesy asociovanými s daným komunikátorem

Blokující komunikační operace: příslušná MPI funkce ukončena až po splnění určité podmínky
Neblokující komunikační operace: příslušná MPI funkce ukončena okamžitě. Dokončení operace je potřeba explicitně testovat

Typ přenášených dat: MPI definuje hodnoty `MPI_Datatype` pro základní C/C++ typy (`MPI_CHAR` , `MPI_INT` , ...)
Složitější typy: Nutno **vytvořit typ:** pomocí `MPI_Type_create...` (např. `MPI_Type_create_struct`)

Tag: Odlišení zpráv různého sémantického významu
Možnost použít `MPI_ANY_TAG`

Stavový objekt: Struktura obsahující položky `MPI_SOURCE` (číslo zdrojového procesu) a `MPI_TAG` (značku zprávy)
Pomocí `MPI_Get_count` lze získat velikost zprávy
Lze ignorovat (`MPI_STATUS_IGNORE`)

Návratové hodnoty MPI funkcí

Návratové hodnoty -- indikace úspěchu (vrácení `MPI_SUCCESS`)

Ošetření chyb: Před návratem při výskytu chyby volána **obsluha chyby** (error handler)
Pro `MPI_COMM_WORLD` obsluha předdefinovaná na `MPI_ERRORS_ARE_FATAL` (násilné ukončení celého programu)
Lze použít `MPI_ERRORS_RETURN` , která pouze navrátí kód chyby z MPI funkce (nastavení `MPI_Errhandler_set(MPI_COMM_WORLD, MPI_ERRORS_RETURN)`)

Blokující komunikační operace

Příslušná MPI funkce ukončena až po **splnění určité podmínky**.

Po ukončení funkce `MPI_xSend` lze buffer s daty **bezpečně modifikovat**, po ukončení `MPI_Recv` jsou přijatá data **uložena v bufferu**

2-bodové blokující operace:

- **Odeslání:** `MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_COMM comm)`
- **Přijetí:** `MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`
  Zdroj může být číslo, nebo `MPI_ANY_SOURCE`

Komunikační módy:
Funkce `MPI_Send` používá **standardní mód**: funkce ukončena, když jsou data:

- buď **odeslána** cílovému procesu (ukončení funkce po přijetí dat cílovým procesem)
- nebo **překopírována do odčasného systémového bufferu** pro pozdější odeslání (ukončení funkce ještě před přijetím)

Volba z možností standardního módu je na MPI knihovně

Buffered mode: funkce `MPI_Bsend` -- ukončení funkce **zaručeně nezávisí** na přijetí dat cílem

Synchronous mode: funkce `MPI_Ssend` -- ukončení funkce až po iniciaci přijetí dat cílem

Ready mode: funkce `MPI_Rsend` -- pokud nebyl dosud iniciován příjem dat cílem, funkce ukončena s chybou

Neblokující komunikační operace

Neblokující funkce `MPI_Isend`, `MPI_Ibsend`, `MPI_Issend` a `MPI_Irsend` iniciují odslání dat a ihned **ukončí svou činnost**

Buffer vstupních dat **nelze modifikovat**, dokud není otestováno dokončení operace

Neblokující funkce `MPI_Irecv` iniciuje přijem dat, ale buffer je možno použít až po otestování dokončení operace

Všechny neblokující funkce mají **dodatečný parametr** typu `MPI_Request` (místo parametru `status`, ten se získá až z testování)

```
MPI_Request request;
MPI_Isend(&c, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
// ...
int flag;
MPI_Status status;
MPI_Test(&request, &flag, &status);
// Proměnnou c lze modifikovat v závislosti na hodnotě flag
// ...
MPI_Wait(&request, &status);
// Proměnnou c lze nyní modifikovat bezpodmínečně
```

Funkce `MPI_Test` a `MPI_Wait` testují/čekají na ukončení **jedné** operace.
Funkce `MPI_Testany` a `MPI_Waitany` testují/čekají na **libovolnou** neblokující operaci z množiny (parametr `MPI_Request[]`)
Funkce `MPI_Testall` a `MPI_Waitall` testují/čekají na všechny neblokující operace z určité množiny

Komunikační módy:
Ukončení neblokujících funkcí nezávisí na splnění žádných podmínek (jsou ukončeny hned). Na splnění obdobných podmínek jako u blokujících funkcí závisí **ukončení funkcí `MPI_Test` a `MPI_Wait`** (ukončení za podmínek podle toho, jestli je volána `MPI_Ibsend`, `MPI_Issend`, ...)

Skupinové komunikační operace

Všechny mají své blokjící i neblokující verze

All-to-one broadcast:

- `MPI_Bcast(void* data, int count, MPI_Datatype dtatype, int root, MPI_Comm comm)`
- Proces `root` vysílá tutéž zprávu celému `comm`

All-to-one gather:

- `MPI_Gather(const void* sendbuf, int sendcount, MPI_Datatype st, void* recvbuf, int recvcount, MPI_Datatype rt, int root, MPI_Comm comm)`
- Proces `root` posbírá data od celého `comm` (včetně sebe sama)

- Ostatní procesy ignorují `recvbuf, recvcount, rt`
- `MPI_Gatherv` : Proces `root` sbírá od každého procesu **různý počet dat** (počety předány v poli `recvcounts[]` , jejich ofsety v `recvbuf` jsou v parametru `displs[]`)

All-to-all gather/broadcast:

- `MPI_Allgather`
- Podobné jako `MPI_Gather` , ale sbírají všichni

One-to-all scatter:

- `MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype st, void *recvbuf, int recvcount, MPI_Datatype rt, int root, MPI_Comm comm)`
- Proces `root` distribuuje data ostatním procesům včetně sebe sama
- od AOG se liší pouze směrem
- `MPI_Scatterv` : nutno určit, kolik dat se každému procesu pošle (ty to musí předem vědět)

All-to-all scatter:

- `MPI_Alltoall(const void* sendbuf, int sendcount, MPI_Datatype st, void* recvbuf, int recvcount, MPI_Datatype rt, MPI_Comm comm)`
- Symetrická operace

Další MPI funkce

- `MPI_Sendrecv` : přijetí i odeslání zprávy zároveň
  - `MPI_Sendrecv_replace` : stejný buffer pro vstup a výstup
- `MPI_Probe` : testování příchodu zprávy bez vlastního přijetí
  - Ukončena když přijde zpráva odpovídající parametrům
  - `MPI_Iprobe` : ukončena okmažitě, nastaven příznak příchodu zprávy