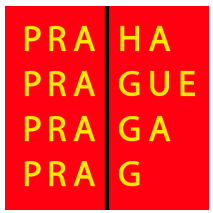
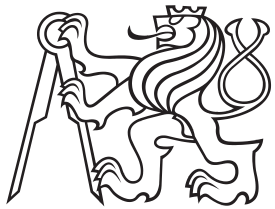
Úvod do paralelního programování 2 – MPI

Jakub Šístek

Ústav technické matematiky FS

9.1.2007

Osnova

Osnova

Paralelizace

Příklad - paralelizace skalárního součinu dvou vektorů

- rozsekání vektorů na menší kusy
- paralelní výpočet dílčích skalárních součinů
- sečtení dílčích výsledků

Základní modely paralelního počítače

- sdílená paměť *(shared memory)*
- distribuovaná paměť *(distributed memory)*
- hybridní uspořádání – dnes nejčastější

Filosofie paralelizace

- rozvětvení výpočtu v jeho kritických bodech (násobebí matic, eliminace,. . .) na více procesorů – počítače se sdílenou pamětí – knihovna funkcí **OpenMP**
- spuštění stejného programu na více procesorech – sdílení dat zajištěno přes “posílání zpráv s daty” mezi procesory – počítače s distribuovanou pamětí – knihovna **Message Passing Interface (MPI)**

paralelní knihovny

OpenMP

- jednoduchost
- malé zásahy do sériového programu
- podmíněný překlad pro vytvoření sériové/paralelní verze
- vyžaduje počítač se sdílenou pamětí – značné omezení

MPI

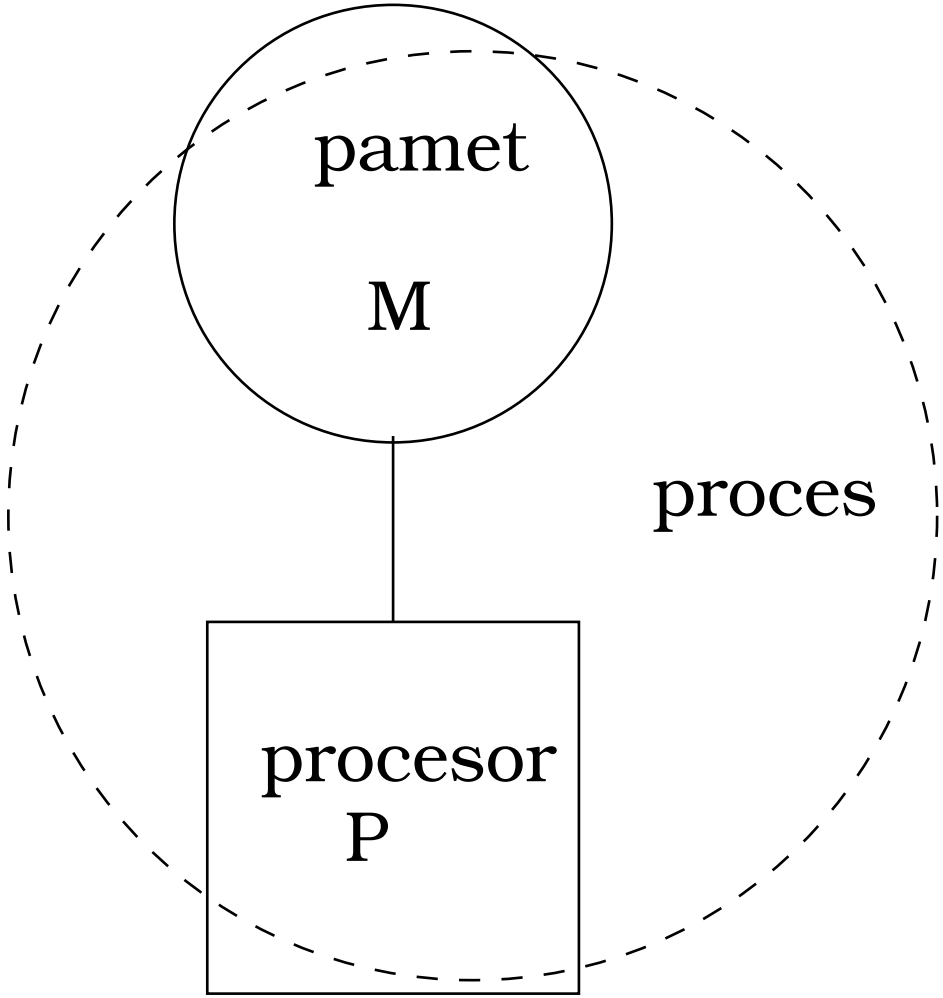
- od počátku vývoje paralelní program
- náročnější na představivost
- univerzálnější – nezávislost na architektuře

Osnova

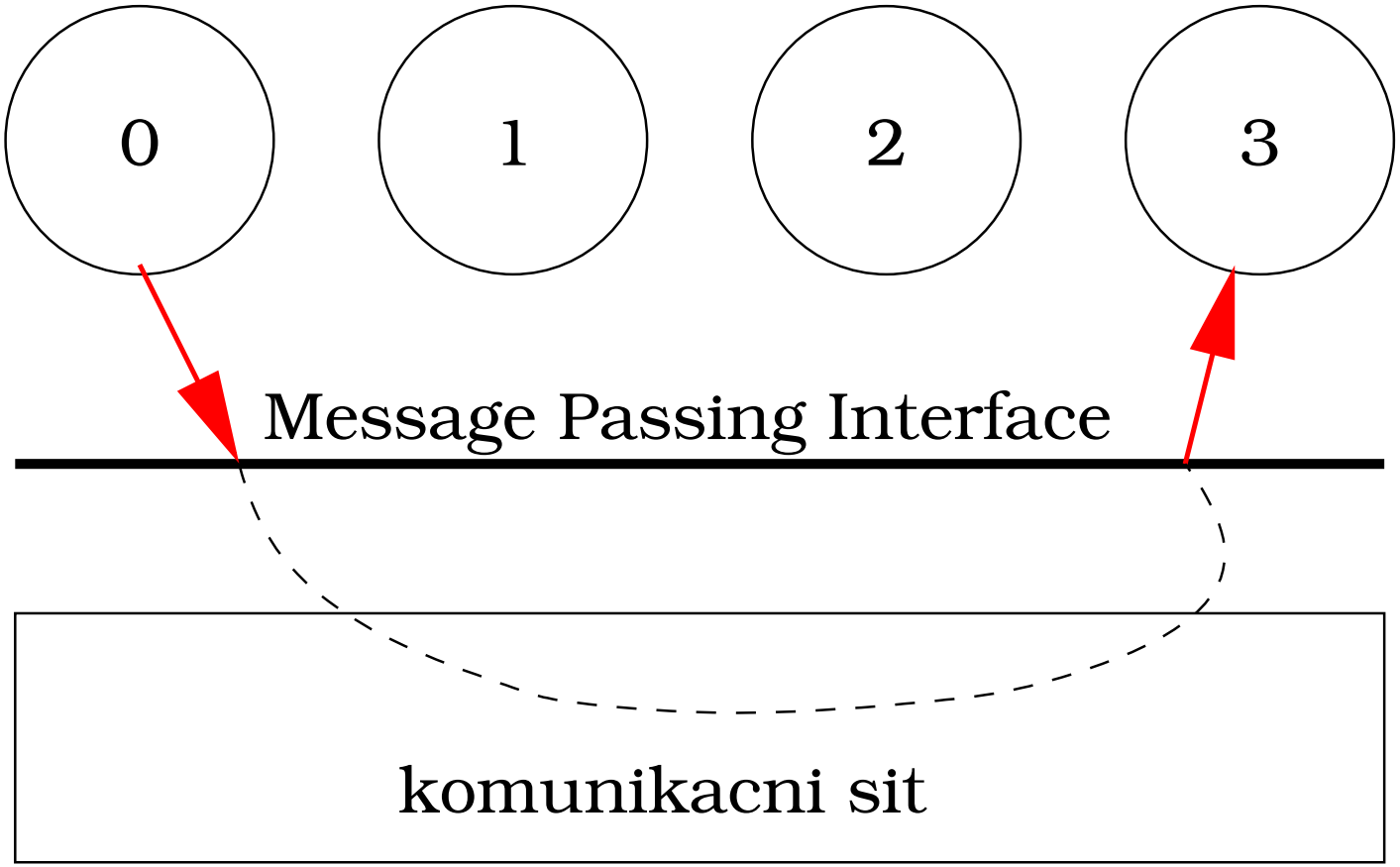
proč se učit MPI?

- knihovna MPI je nejdůležitější “kus programu” ve světě paralelního programování
- všechny největší superpočítače na světě jsou programovány pomocí MPI
- psaní paralelních programů pomocí MPI je zábava!

běžný – sekvenční – program



role MPI



charakteristiky MPI

- každý proces má své vlastní proměnné
- procesy komunikují pomocí volání speciálních připravených (standardních) funkcí
- MPI je první standard v komunikaci paralelních počítačů
- MPI se na uživatele (programátora) tváří všude stejně, ale jeho chování závisí na počítači – **přenositelnost & rychlost**
- program je obvykle v běžném programovacím jazyce - C nebo Fortran
- na každém procesoru běží **stejný** program samostatně
- *Single Program Multiple Data* $\approx$ stejný program, různá data
- procesory si posílají “zprávy” s daty

jak může každý dělat něco jiného?

- procesory jsou odlišeny pouze svým **pořadovým číslem**
- **Př.**: proces č.0 přičte číslo 5 do proměnné i, proces č.1 ne

```
if (moje_cislo == 0) i = i + 5;
```

moje_cislo: 0	**moje_cislo: 1**
i = 10	i = 10
i = i + 5	
i = 15	i = 10

zprávy

- zprávy jsou balíčky (pakety) dat vyměňované mezi procesy
- systém pro výměnu zpráv potřebuje následující informace
  - posílající proces(y)
  - umístění dat v odesílajícím procesu (adresa)
  - datový typ (integer, float, double,. . .)
  - počet prvků (1 – jednoduchá proměnná, délka pole)
  - přijímající proces(y)
  - umístění dat v přijímajícím procesu
  - velikost zásobníku pro příjem zpráv

obecná pravidla

MPI je jako poštovní společnost

- proces musí zaregistrovat zprávu do MPI $\approx$ poštovní schránka
- zpráva musí mít identifikaci kam má být doručena $\approx$ poštovní adresa
- příjemce zprávy musí být schopen zprávy přijímat $\approx$ poštovní schránka u domu

dva základní typy komunikace v MPI

point–to–point komunikace

- komunikace jednoho s jedním
- nejjednodušší forma posílání zpráv
- více druhů podle potřeb (odesílatel čeká/nečeká na doručení své zprávy,. . .)

globální (kolektivní) komunikace

- synchronizace procesorů
- komunikace všech se všemi
- komunikace jeden všem/všichni na jednoho
- více druhů podle potřeb

Osnova

náležitosti MPI programu

hlavičkový soubor

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

volání funkcí

```
error = MPI_Xxxxxxx(parametr, ...)
```

případně

```
MPI_Xxxxxxx(parametr, ...)
```

inicializace MPI

```
error = MPI_Init(NULL,NULL)
```

ukončení MPI – poslední funkce

```
error = MPI_Finalize()
```

co to je komunikátor?

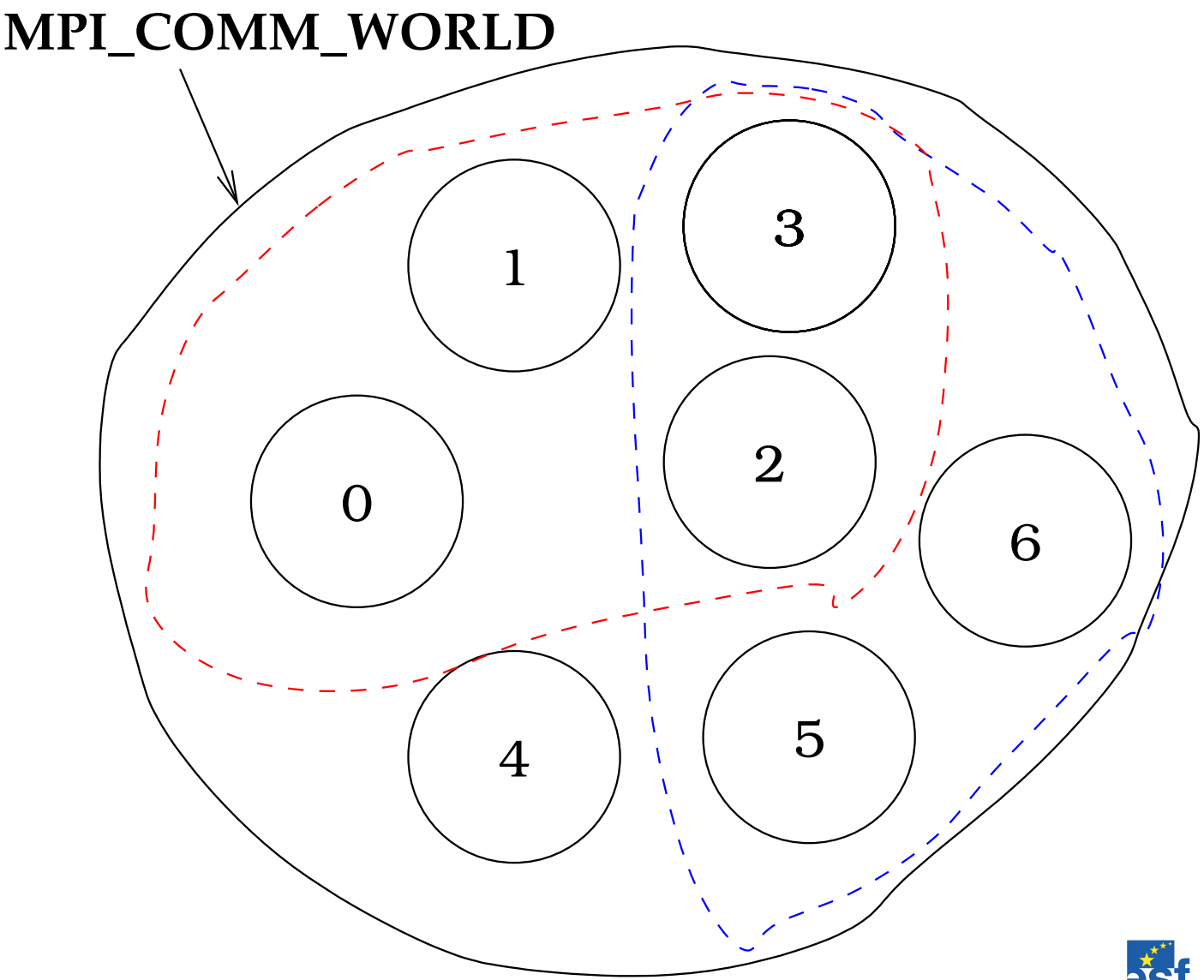
- komunikační prostředí, které obsahuje komunikující procesory (všechny nebo část)

MPI_COMM_WORLD

- základní komunikátor
- zaveden při inicializaci MPI
- obsahuje všechny procesory, na kterých je program spuštěn
- pro časté použití je praktické přiřadit nějaké proměnné

```
comm = MPI_COMM_WORLD
```

co to je komunikátor?



jak se orientovat v komunikátoru?

identifikace – číslo

- počítání procesů od 0 (0, 1,. . . ,velikost-1)

```
MPI_Comm_rank(comm, &moje_cislo)
```

počet procesů v komunikátoru – velikost

- počítání procesů od 0 (0, 1,. . .)

```
MPI_Comm_size(comm, &velikost)
```

Minimální program v MPI

```
#include <stdio.h>
#include "mpi.h"
main()
{
/* komunikator, pocet procesoru a cislo procesoru */
int comm, size, rank;

comm = MPI_COMM_WORLD; /* cislo pro komunikator */
MPI_Init(NULL,NULL); /* inicializace MPI */
MPI_Comm_size(comm, &size); /* pocet procesoru */
MPI_Comm_rank(comm, &rank); /* cislo procesoru */

printf("myid = %d: Hello world! \n",rank); /* tisk */

MPI_Finalize(); /* konec MPI */
}
```

Osnova

základní poslání zprávy

```
error = MPI_Send(x,lx,MPI_DOUBLE, 0, 0, comm)
```

kde

- `x` adresa kde začíná pole pro poslání
- `lx` počet prvků pole, které chceme poslat
- `MPI_DOUBLE` typ posílané proměnné
- `0` proces, kterému data posíláme $\approx$ **adresát**
- `0` “tag” neboli značka – další identifikace zprávy
- `comm` číslo komunikátoru

přijetí zprávy

```
error = MPI_Recv(x,lx,MPI_DOUBLE, 1, 0, comm,
&status)
```

kde

- `x` adresa kde začíná pole pro uložení
- `lx` počet prvků pole, které chceme uložit
- `MPI_DOUBLE` typ přijímané proměnné
- `1` proces, od kterého data přijímáme $\approx$ **odesílatel**
- `0` “tag” neboli značka – další identifikace zprávy
- `comm` číslo komunikátoru
- `status` pomocné proměnné

co je třeba pro úspěch poslání zprávy

- odesílatel musí uvést existujícího příjemce
- příjemce musí specifikovat platného odesílatele
- předpokládaný odesílatel odesílá a předpokládaný příjemce přijímá
- odesílatel a příjemce musí být ve stejném komunikátoru
- shodují se tagy
- typy dat se shodují
- příjemce specifikoval dostatek místa pro přijetí dat

další možnosti příjmu zprávy

- příjemce může uvolnit shodu
- odesílatel `MPI_ANY_SOURCE` – libovolný odesílatel
- tag `MPI_ANY_TAG` – libovolný tag

měření času

```
cas1 = MPI_Wtime()
⋮
cas2 = MPI_Wtime()
doba = cas2 - cas1
```

- čas je měřen v sekundách

nestandardní situace

- pokud jeden proces posílá data a adresát je přijíma – OK
- pokud jeden proces posílá data, ale adresát nic nepřijíma – program končí chybou
- pokud jeden proces přijímá data, ale odesílatel nic neposílá – program končí chybou
- oba posílají a nikdo nepřijímá – program stojí – **deadlock**
- oba přijímají a nikdo neposílá – program stojí – **deadlock**

a na co se můžeme spolehnout?

- zprávy mezi dvěma procesy se nikdy “nepředběhnou” – podmínka MPI

Osnova

úloha

- funkce volány všemi procesory v komunikátoru
- použití pro skupinu procesorů
- synchronizace, celková suma po paralelním výpočtu (skalární součin,. . .)

výběr funkcí

synchronizace

- zabrzdění běhu procesů v komunikátoru tak, aby se dostaly na stejnou úroveň (měření času,...)

```
MPI_Barrier(comm)
```

kopie z jednoho procesu na všechny

```
MPI_Bcast(x,lx,MPI_DOUBLE,0,comm)
```

kde

- `x` adresa kde začíná pole pro poslání
- `lx` počet prvků pole, které chceme poslat
- `MPI_DOUBLE` typ posílané proměnné
- `0` proces, ze kterého se bude kopírovat hodnota
- `comm` číslo komunikátoru

sečtení hodnoty ze všech procesů

`MPI_Reduce(xold,xnew,lx,MPI_DOUBLE,MPI_SUM,0,comm)`
kde

- `xold` adresa kde začíná pole pro poslání
- `xnew` adresa kde začíná pole pro uložení na procesu **root**
- `lx` počet prvků pole, které chceme poslat
- `MPI_DOUBLE` typ posílané proměnné
- `MPI_SUM` typ operace – sečtení
- `0` proces, ze kterého se bude kopírovat hodnota – **root**
- `comm` číslo komunikátoru

sečtení hodnoty ze všech procesů a uložení na všechny

`MPI_Allreduce(xold,xnew,lx,MPI_DOUBLE,MPI_SUM,comm)`
kde

- `xold` adresa kde začíná pole pro poslání
- `xnew` adresa kde začíná pole pro uložení
- `lx` počet prvků pole, které chceme poslat
- `MPI_DOUBLE` typ posílané proměnné
- `MPI_SUM` typ operace – sečtení
- `comm` číslo komunikátoru
- není nikdo **root**