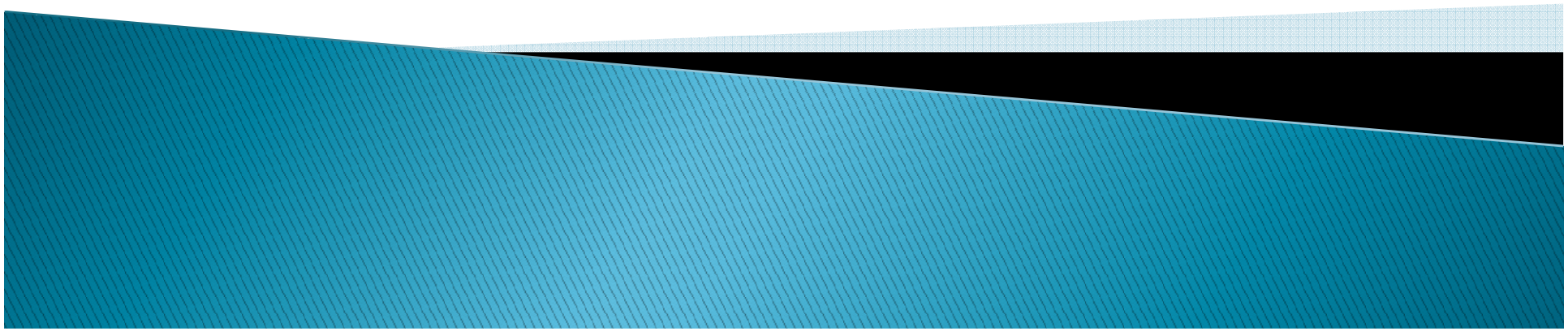
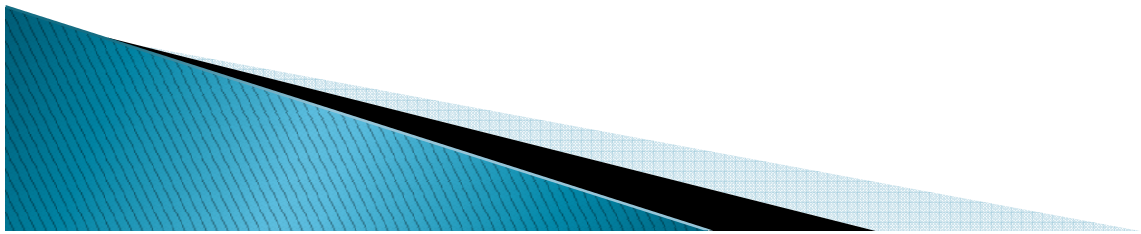
MPI 2.0 paralelní I/O & dynamická správa procesů

Jaroslav Hořejší – hor412



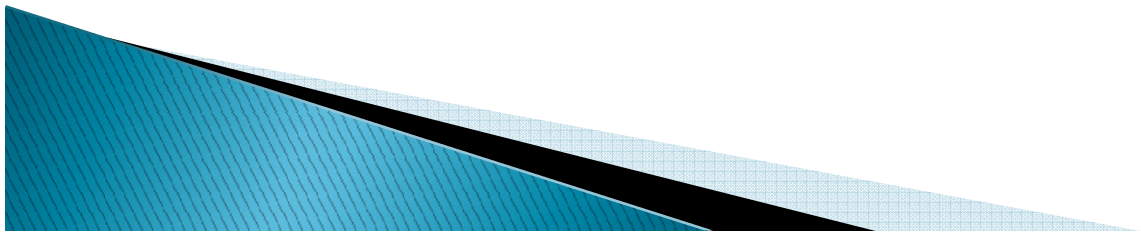
Obsah

- Proč vzniklo MPI?
- Paralelní I/O v MPI
- Dynamická správa procesů
- Funkce
- Závěr



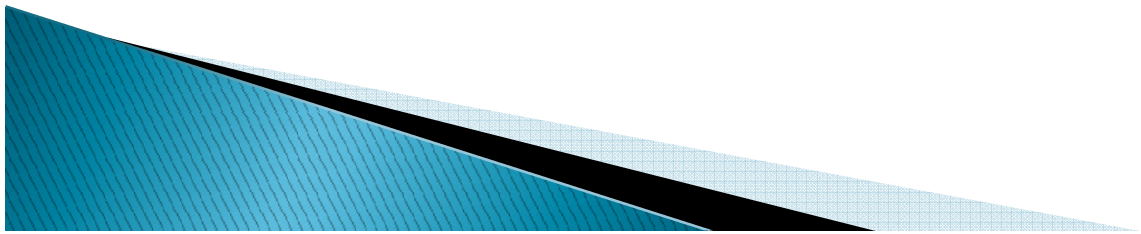
Proč vzniklo MPI?

- Model předávaní zpráv – 90 letech uznáván
- MPI forum – 1992 sdružení asi 140 organizací a firem
- MPI 1.0 – 1994
- MPI 1.1 – 1995
- MPI 1.2 – 1995 – 1996
- MPI 2.0 – 1997
- MPI 3.0 – 2010?



Paralelní I/O v MPI

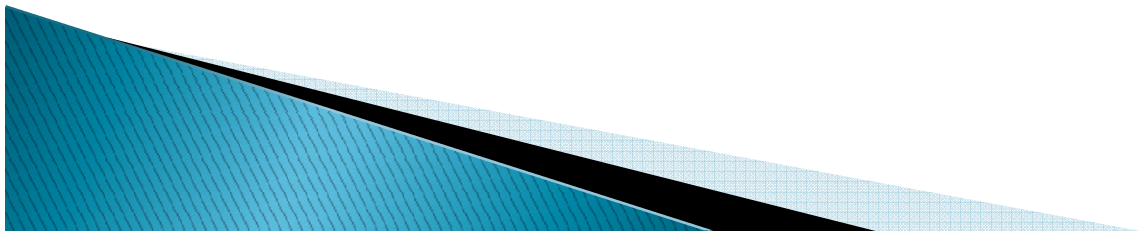
- Standard POSIX s jeho přenositelností a optimalizací nelze použít paralelní I/O v MPI. Místo toho musí implementovat rozhraní , které podporuje dělení dat souborů mezi procesy a sdružené rozhraní pro přesun globálních dat struktur mezi pamětí procesu a soubory. MPI poskytuje vzory pro přístup k sdíleným souborům(broadcast atd) Toto rozhraní použití technik „collective buffer“ .



Dynamické správa procesu

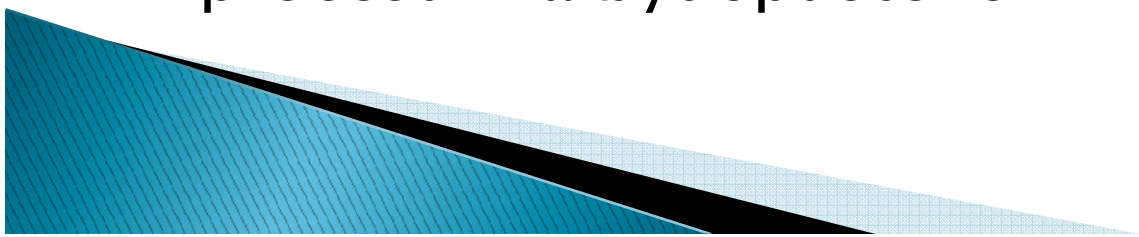
- Ve verzi MPI 1.1 neexistovala, ale protože bylo nutno spravovat procesy z důvodu , že důležité třídy potřebovaly řízení procesů a jejich výpočet a typ před spuštěním, aby mohli použiti pokročilé metody paralelního zpracování úloh např seriové programy s paralelními moduly

- Dále zjednodušilo přechod mezi PVM a MPI, protože byla doplněna funkce MPI_Spawn, která obdobou PVM_Spawn



Důležité funkce

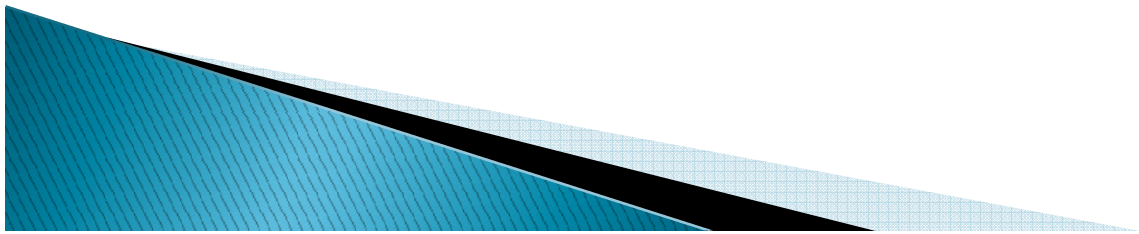
- MPI_COMM_WORLD vrací komunikátor sdružující všechny procesy
- MPI_COMM_GET_PARENT(parent) – vrátí rodiče ve workerovi
- MPI_Init (&argc,&argv); funkce, která musí být volána před spuštěním jakékoliv jiné funkce. MPI_FINALIZE(); ukončuje provádění MPI. Tato funkce musí být volaná jako poslední.
- MPI_COMM_SIZE zjistí počet procesů ve skupině, která je asociována s komunikátorem
- MPI_UNIVERSE_SIZE je funkce, která zjistí kolik procesů má být spuštěno.



Funkce – MPI_SPAWN

MPI_COMM_SPAWN (command, argv, maxprocs, info, root, comm, intercomm, array_of_errcodes)

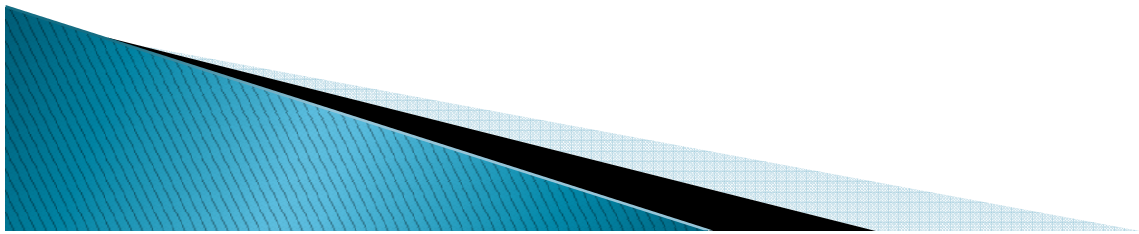
- IN command jméno programu, který bude vytvořen
- IN argv argument commandu je pole stringů ,který obsahují argumenty, které prochází programem
- IN maxprocs maximální počet procesů, který bude vytvořen a spuštěn
- IN info nastavuje kličovou dvojici hodnot, které říkají runtime systému, kde a jak má být proces zahájen (NULL)
- IN root číslo procesu ve kterém jsou předchozí argumenty otestovány
- IN comm je intrakomunikátor obsahující skupinu vytvořených procesů (handle)
- OUT intercomm interkomunikátor mezi současnou skupinou procesů a nově vytvořenou skupinou procesů (handle)
- OUT array_of_errcodes jeden kód přes proces (pole integerů) délka argumentu maxprocs



Další funkce

- MPI_COMM_SPAWN_MULTIPLE (count, array_of_commands, array_of_argv, array_of_maxprocs, array_of_info, root, comm, intercomm, array_of_errcodes)

- MPI_COMM_MODIFY (comm, maxprocs, intercomm)



Příklad

```
/*manager*/
#include “mpi.h”
int main(int argc, char *argv[])
{
        int world_size, universe_size, *universe_sizep, flag ;
MPI_Comm everyone ;           /*intercommunicator */
char worker_program[100] ;

MPI_Init (&argc, &argv) ;
MPI_Comm_size(MPI_COMM_WORLD, &world_size) ;
If (world_size  != 1)                                   error (“Top heavy with
management”);

MPI_Comm_get_attr(MPI_COMM_WORLD, MPI_UNIVERSE_SIZE,
                                                        &universe_sizep, &flag) ;
if (!flag) {
        printf (“This MPI does not support UNIVERSE_SIZE> How many\n\
   processes total?”) ;
                scanf (“%d” , &universe_size);
        } else universe_size = *universe_sizep;
        if (universe_size == 1 ) error (“No room to start workers”) /*



choose_worker_program (worker_program);
MPI_Comm_spawn(worker_program, MPI_ARGV_NULL, universe_size-1,
                MPI_INFO_NULL, 0, MPI_COMM_SELF, &everyone,
                MPI_ERRCODES_IGNORE) ;

MPI_Finalize()
Return 0;
}
```

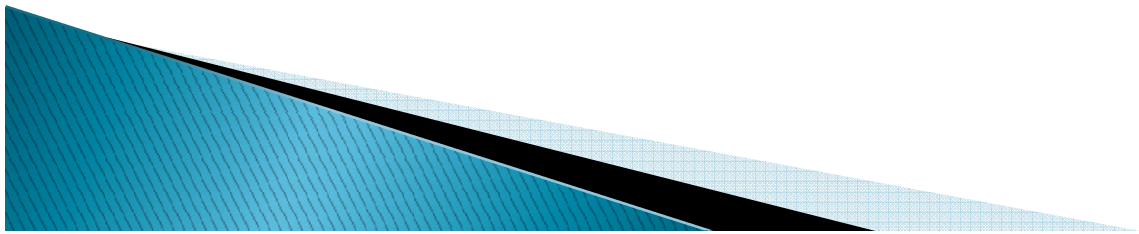
master

```
#include „mpi.h“
int main (int argc, char &argv [])
{
        int size;
        MPI_Comm parent;
        MPI_Init (&argc, &argv);
MPI_Comm_get_parent (&parent) ;
If  (parent == MPI_COMM_NULL) error ( „No parent!“) ;
MPI_Comm_repote_size (parent, &size);
 If (size != 1 ) error („Something is wrong with the parent „)

/*
 * Paralel code here.
 * The manager is represented as the process with rank 0 in ( the remote
 * group of) the parent communicator. It the workers need to communicate
 * among themselves, they can use MPI_COMM_WORLD.
 */

MPI_Finalize ();
Return 0;
```

worker

Zdroje

- Dynamic Process Management in an MPI Setting, Wiliam Gropp, Ewing Lusk, Mathematics and Computer Science Division, Argone National Laboratory
- MPI: A Message-Passing Interface Standard Version 2.1, Message Passing Interface Forum, June 23.2008
- http://www.cs.vsb.cz/jakl/pa/volny/



Závěr

Děkuji za pozornost.