# MPI – 2 parallel I/O & dynamic process management

1. Proč MPI vzniklo

   Přestože **„model předávání zpráv“** byl do začátku 90 let minulého století široce přijímán, byly systémy předávání zpráv vzájemně nekompatibilní, proto bylo nutno vytvořit standard, s ohledem na rozšiřující komunitu uživatelů víceprocesorových systémů. Proto v roce 1992 vznikla standardizační pracovní skupina, která ještě v tomto roce iniciovala vznik MPI fóra. Do práce na standardu se zapojilo 175 odborníků ze 40 významných institucí, zahrnující výrobce paralelních počítačů, vývojáře, akademiky a vědce. V roce 1994 vznikla první verze MPI 1.0. To co vzniklo nebyl program ale dokument, který popisuje jednotlivé funkce v MPI , aby vývojáři mohli tento systém naimplementovat. Práce na vylepšování tohoto standardu pokračovali a už v červnu roku 1995 vznikla verze MPI 1.1 a následně vznikla verze MPI1.2. Později vznikla verze MPI – 2, a v dnešní době se pracuje na MPI – 3.0. Tento standard má být vydán v roce 2010.

2. Popis paralelního vstupu a výstupu
   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 Paralelní I/O vzory pro přístup k sdíleným souborům(broadcast atd) Toto rozhraní umožňuje použití technik sdruženého bufferování .

3. Dynamické řízení procesů

Ve verzi MPI 1.1 neexistovala, ale protože bylo nutno zpravovat procesy z důvodu , že důležité třídy potřebovaly řízení procesů a jejich výpočet a typ před spuštěním, aby mohli používat 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

Ve verzi MPI 1.1 neexistovaly funkce MPI_COMM_SPAWN a další funkce pro dynamickou zprávu procesů. Ale protože bylo nutné nějak zpravovat procesy, na druhou stranu se zjednodušilo přechod mezi PVM a MPI a bylo potřeba zavést paralelní techniky. Důležité třídy potřebovaly například používat seriové aplikace s paralelními moduly. Potřebovaly zjistit počet a typ procesů, které mají být spuštěny.Ve verzi MPI 1.1 nebylo schopno MPI navrhnout rozhraní, které by bylo přenositelné mezi širokým spektrem existujících nebo potencionálních zdrojů a řízení procesů.
Verze MPI 2 má kromě funkce MPI_COMM_SPAWN ještě další důležitou funkci MPI_COMM_SPAWN_MULTIPLE, která bude níže popsaná. Dále existuje funkce MPI_COMM_WORLD, která vrací komunikátor sdružující všechny procesy. Přes komunikátor v dětském procesu, můžeme získat pomoci funkce MPI_COMM_GET_PARENT rodiče.

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

| | | |
|---|---|---|
| IN | command | jméno programu, který bude vytvořen |

| | | |
|---|---|---|
| IN | argv | argumenty commandu nebo-li programu, který bude vytvořen |
| IN | maxprocs | maximální počet procesů, který bude vytvořen a spuštěn |
| IN | info | množina dvojic hodnot, které říkají runtime systému, kde a jak má být proces zahájen |
| 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) |

**MPI_COMM_MODIFY (comm, maxprocs**, **intercomm)**

| | | |
|---|---|---|
| IN | comm | je intrakomunikátor obsahující skupinu vytvořených procesů (handle) |
| IN | maxprocs | maximální počet procesů, který bude vytvořen a spuštěn |
| OUT | intercomm | interkomunikátor mezi současnou skupinou procesů a nově vytvořenou skupinou procesů (handle) |

Nové procesy dostanou vyšší čísla než stávající procesy.

MPI_COMM_SPAWN_MULTIPLE spouští několik odlišných úloh, které spouští stejnou funkci MPI_COMM_WORLD.

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

| | | |
|---|---|---|
| IN | count | počet příkazů (kladné celé číslo) |
| IN | array_of_commands | názvy programů, které mají být provedeny |
| IN | array_of_argv | argumenty pro příkazy (pole stringů) |
| IN | array_of_maxprocs | maximální počet procesů, které mají být spuštěny |
| IN | array_of_info | pole objektů které říkají runtime systému, kde a jak mají být spuštěny procesy |
| IN | root | číslo procesu na kterém budou 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)

MPI_Init (&argc,&argv); funkce, která musí být volána před spuštěním jakékoliv jiné funkce, parametry funkce jsou parametry ve funkci main.
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.

**4. Příklad**

Manager-worker Příklad pomoci MPI_COMM_SPAWN.

```
/*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”)

    /*
     *  Now spawn the workers. Note that there is a run-time determination
     * of what type of worker to spawn, and presumably this calculation must
     *  be done at run time and cannot be calculated before starting
     *  the program. It everything is known when the application is
```

```
        *  first started, it is generally better to start them all at once
        *  in a single MPI_COMM_WORLD.
        * /

        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) ;
        /*
        *  Parallel code here. The communicator "everyone" can be used
        *  to communicate with the spawned processes, which have ranks O,…
        *  MPI_UNIVERSE_SIZE-1 in the remote group of the intercommunicator
        *  "everyone" .
        */

        MPI_Finalize()
        Return 0;
    }
    /* worker */

    #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;
}
```

5. 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/

6. Závěr