Vstup a výstup v MPI
PPP 06

Gabriela Nečasová

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
inecasova@fit.vut.cz

Úvod

- **Úkoly:** pracujeme s funkcemi **MPI_File***
  - Práce se vstupem a výstupem v MPI
  - Vytvoření souboru následované individuálními zápisy
  - Zápis matici distribuované po řádcích do souboru pomocí kolektivních komunikací
  - Dlaždicové načtení obrázku Lenny, aplikace gamma korekce a uložení obrázku zpět

- Kde počítat?
  - Barbora/Karolina/Merlin – možno vypracovat celé cvičení
  - Poznámka: potřebujeme 16 procesů, na Merlinovi je potřeba `--oversubscribe`

- Cvičení jsou dobrovolná, ovšem mohou výrazně usnadnit řešení projektu ☺
- **Nápověda**:
  - Přednáška PPP 6
  - Open MPI dokumentace: https://www-lb.open-mpi.org/doc/v4.1/
    - Seznam všech MPI rutin (MPI_Init,...)

Úvod

- Připojení na server?
  - `ssh [merlin|barbora|karolina]`
- Ověření, že je k dospozici MPI:
  - `mpicc --help`

- Pokud pracujete na Barboře/Karolině:
  - Karolina
    - `$ salloc -A DD-24-108 -p qcpu_exp -N 1 --ntasks-per-node 128 -t 01:00:00 --x11`
  - Barbora
    - `$ salloc -A DD-24-108 -p qcpu_exp -N 1 --ntasks-per-node 36 -t 01:00:00 --x11`
  - `ml OpenMPI`

- Zobrazení info o MPI instalaci:
  - `ompi_info`

Překlad a spuštění

- 1. možnost: cmake:
  - `$ cmake -Bbuild -S.`
  - `$ cmake --build build --config Release`
- 2. možnost: mpicc:
  - `mpic++ io.cpp -o one`
- Aplikace má 1 parametr – číslo úlohy (zde 1):
  - `mpirun -np 16 ./io 1`

Základní funkce

- **initMatrix()**
  - Inicializace: 100 * i + j
- **clearMatrix()**
- **printMatrix()**

ÚKOL 1: VYTVOŘENÍ SOUBORU A ZÁPIS Z JEDNOHO RANKU

1: Vytvoření souboru a zápis z jednoho ranku

Cílem tohoto příkladu je vyzkoušet si tvorbu a uzavření souboru (`MPI_File_open`, `MPI_File_close`) a zápis z jednoho ranku (`MPI_File_write`). Zadání se nachází ve funkci `main` pod sekcí `case 1`:

1. Nejprve si deklarujte objekt MPI souboru.
2. Následně soubor otevřete pro zápis.
3. Root rank poté zapíše do souboru text `Hello from rank #0`.
4. Na závěr soubor uzavřete.
5. Přeložte soubor.
6. Spusťte výslednou binárku a prohlédněte si obsah souboru:

```
$ mpiexec ./io 1
$ cat File1.txt
```

1: Vytvoření souboru a zápis z jednoho ranku

- // 1. Declare an MPI file.
- **MPI_File myFile**
- // 2. Open a file called "File1.txt" with write permission.
- **MPI_File_open()**
- // 3. Use a ROOT rank and write Hello into the file.
  - Vytvoříme si řetězec: std::string text = "Hello from rank #0\n";
- **MPI_File_write()**
- // 4. Close the file.
- **MPI_File_close()**

- Použití: logování

```
int MPI_File_open(MPI_Comm comm, ROMIO_CONST char *filename, int amode, MPI_Info info, MPI_File *
fh)
```

- Kolektivní operace!
- Pokud by chtěl soubor otevřít 1 rank, použijeme comm = MPI_COMM_SELF
- amode = MPI_MODE_[APPEND|CREATE|RDONLY|RDWR|WRONLY|DELETE_ON_CLOSE|EXCL|SEQUENTIAL]
- Info = MPI_INFO_NULL
  - ale pokud máme paralelní filesystem (Lustre), dá se nastavit tvar dat, sekvenční data/random přístup, jak je soubor rozdělen na stripes, jaká jsou Unix přístupová práva, atp.

```
int MPI_File_write(MPI_File fh, const void *buf, int count, MPI_Datatype datatype, MPI_Status
*status)
int MPI_File_close(MPI_File * fh)
```

- lokální operace pro rank, není tu komunikátor

ÚKOL 2: INDIVIDUÁLNÍ ZÁPIS DO JEDNOHO SOUBORU

2: Individuální zápis do jednoho souboru

Cílem tohoto příkladu je vyzkoušet si individuální zápis do souboru na předem dané místo (`MPI_File_write_at`). Zadání se nachází ve funkci `main` pod sekcí `case 2`:

1. Nejprve si deklarujte objekt MPI souboru.
2. Následně soubor otevřete pro zápis.
3. Nyní pomocí operace exkluzivního scanu zjistěte, na které místo v souboru můžete daný rank zapsat svůj řetězec.
4. Zapište na dané místo požadovaný řetězec.
5. Na závěr soubor uzavřete.
6. Přeložte soubor.
7. Spusťte výslednou binárku:

```
$ mpiexec -np 16 --oversubscribe ./io 2
$ cat File2.txt
```

8. Vyhodnoťte správnost dat v souboru.

2: Individuální zápis do jednoho souboru

- // 1. Declare an MPI file: **MPI_File myFile;**
- // 2. Open a file called "File2.txt" with write permission.
- **MPI_File_open()**
- // Declare a string and write "I am %d from %d\n"
- constexpr std::size_t maxStrSize{128};
- std::array<char, maxStrSize> str{};
- const int strSize = std::**snprintf**(str.data(), maxStrSize, "I am %d from %d\n", mpiGetCommRank(MPI_COMM_WORLD), mpiGetCommSize(MPI_COMM_WORLD));
  - Jak zjistit na jakou pozici v souboru mám zapsat? Jak udělat správný seek()? Exkluzivní suma prefixů – je to redukce – redukuje hodnoty, které jsou přede mnou.
- **MPI_File_write()**
- // 3. Find positions where to write into the file. Use the Exscan operation.
- **MPI_Exscan()** // Díky této funkci bude výpis v souboru seřazen
- // 4. Use an individual mpi write call at an appropriate position.
- // Check the performance and then try to use a collective operation instead.
- **MPI_File_write_at()**
- // 5. Close the file: **MPI_File_close()**

Zkusme spustit např s 10 ranky. Proč to tak dlouho trvá?

Protože **MPI_File_wirte_at()** jsou individuální zápisy – je tu boj o zamykání části souboru pro zápis (rank načte celý stripe, zmodifikuje část, zapíše)

Nahradíme tuto operaci kolektivním zápisem (funkce končí „_all“)
MPI_File_write_at_all() – tzn. všichni píšou

```
int MPI_Exscan(const void *sendbuf, void *recvbuf, int count,MPI_Datatype datatype, MPI_Op op,
MPI_Comm comm)
• sendbuf – kolik bajtů zapisuji
• recv – kolik zapsaali ti přede mnou
• count – počet položek
• MPI_Op = MPI_SUM

int MPI_File_write_at(MPI_File fh, MPI_Offset offset, const void *buf,
    int count, MPI_Datatype datatype, MPI_Status *status)
• offset = o kolik bajtů se mám posunout
```

2: Individuální zápis do jednoho souboru

Měření času: **`time mpirun -np 10 ./io_solution 2`**

```
-----------------------------------------------------------------
                          PPP Lab 6
-----------------------------------------------------------------
 Example 2 - Write ranks and comm size in the file in ascending order
-----------------------------------------------------------------
 Task finished successfully!
-----------------------------------------------------------------

real    0m34.391s
user    4m15.891s
sys     0m3.892s
```

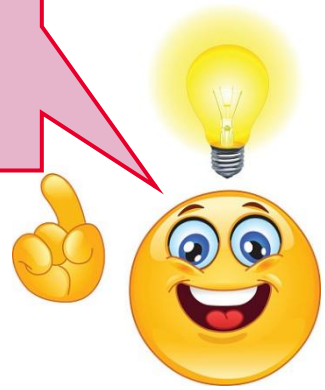
MPI_File_write_at()

```
-----------------------------------
                          PPP Lab 6
-----------------------------------
 Example 2 - Write ranks and comm size
-----------------------------------
 Task finished successfully!
-----------------------------------

real    0m2.410s
user    0m0.847s
sys     0m3.740s
```

MPI_File_write_at_all()

MPI I-O ví, že všichni budou zapisovat si řekne: Ha, mám tady např. 4 zápisy malých bločků, které jdou vedle sebe, **tak já to sesbírám**. Takže uvnitř se provede **gather**() do jednoho pole a provede se zápis **jedenkrát**.



ÚKOL 3: KOLEKTIVNÍ ZÁPIS MATICE DISTRIBUOVANÉ PO ŘÁDCÍCH

3: Kolektivní zápis matice distribuované po řádcích

Cílem tohoto příkladu je vyzkoušet si kolektivní zápis do souboru pomocí dvou funkcí `MPI_File_set_view` a `MPI_File_write_all`. Matice je vygenerovaná na root ranku a je nutné ji nejprve rozptýlit mezi ostatní ranky. Následně je matice kolektivně zapsána do souboru. Poslední část programu soubor sekvenčně otevře a porovná zapsaná data. Zadání se nachází ve funkci `main` pod sekcí `case 3`:

1. Vytvořte datový typ pro distribuci řádků matice.
2. Rozptylte matici po blocích mezi jednotlivé ranky.
3. Otevřete soubor pro zápis.
4. Nastavte pohled do souboru tak, aby každý rank viděl svoji část (zde stačí pracovat s hodnotou `displacement`).
5. Zapište matici kolektivním zápisem.
6. Uzavřete soubor a uvolněte vytvořený datový typ.
7. Přeložte soubor.
8. Spusťte výslednou binárku:

```
$ mpiexec -np 2 ./io 3
$ mpiexec -np 4 ./io 3
$ mpiexec -np 8 ./io 3
$ mpiexec -np 16 --oversubscribe ./io 3
```

9. Porovnejte výsledky všech operací.

- **Matice 16x4**
- **filename = File3.dat**
- Matice distribuována **po řádcích**
- Každý rank zapíše nějaký počet řádků
- **Globální matice** (root)
- **Lokální matice** – pro každý rank
- MPI neumí dělat recreate, takže se může pak stát že část souboru obsahuje nová data a část stará, proto je dobré soubor smazat **MPI_File_delete()**
- Root vytvoří globální pole a vytiskne ho

3: Kolektivní zápis matice distribuované po řádcích

- // 1. Create a datatype for Matrix row.
- **MPI_Type_contiguous()**
- **MPI_Type_commit()**

- // 2. Scatter gMatrix into lMatrix.
- **MPI_Scatter()**

- // 3. Declare an MPI file.
- **MPI_File file**

- // 4. Open a file with "filename" with write permission.
- **MPI_File_open()**

```
int MPI_File_set_view(MPI_File fh, MPI_Offset disp,
MPI_Datatype etype, MPI_Datatype filetype, const char
*datarep, MPI_Info info)
```

- disp – v bajtech, tzn. Kde začíná řádek?
- etype – elementární dat. typ, zde **MPI_INT**
- filetype – co chceme vykousnout ze souboru – chceme blok dat který odpovídá řádku, klidně také **MPI_INT**
- datarep – **„native"** – tzn. data se uloží v nativním formátu daného stroje
- info – **MPI_INFO_NULL**

```
int MPI_File_write_all(MPI_File fh, const void *buf,
int count, MPI_Datatype datatype, MPI_Status *status)
```

- // 5. Set a file view. Hint - use displacement to move at appropriate position in the file.
- **MPI_File_set_view()**

- // 6. Use a collective write.
- **MPI_File_write_all()**
-
- // 7. Close the file.
  MPI_File_close()
- // 8. Delete the matrix row datatype.
  MPI_Type_free()

MPI_File_write_at() je sice fajn, ale co když nepůjde o souvislá data – co když chci zapsat sloupec nebo dlaždici? Tzn. Data jsou v paměti s nějakým rozestupem a museli bychom po každé provádět seek() a to nechceme.

Chceme aby funkce vybere jen určitá data a vytvoří kontinuální blok dat, která když zapíšu do souboru, tak se to umístí na ty správné pozice.

MPI_File_set_view() – popisujeme tvar dat z pohledu souboru

MPI_File_write_all() – popisujeme co zapisujeme (co je v paměti)

- od funkce MPI_File_write_at() se liší jen tím, **že tu není offset**, protože funkce MPI_File_set_view() už nastaví pozici v souboru, takže pak už provedeme pouze zápis.

3: Kolektivní zápis matice distribuované po řádcích

```
-------------------------------------------------------------------
 Example 3 - Write a matrix distributed over rows
-------------------------------------------------------------------
Original array:
 - Rank  0, row  0 = [       0,        1,        2,        3]
 - Rank  0, row  1 = [     100,      101,      102,      103]
 - Rank  0, row  2 = [     200,      201,      202,      203]
 - Rank  0, row  3 = [     300,      301,      302,      303]
 - Rank  0, row  4 = [     400,      401,      402,      403]
 - Rank  0, row  5 = [     500,      501,      502,      503]
 - Rank  0, row  6 = [     600,      601,      602,      603]
 - Rank  0, row  7 = [     700,      701,      702,      703]
 - Rank  0, row  8 = [     800,      801,      802,      803]
 - Rank  0, row  9 = [     900,      901,      902,      903]
 - Rank  0, row 10 = [    1000,     1001,     1002,     1003]
 - Rank  0, row 11 = [    1100,     1101,     1102,     1103]
 - Rank  0, row 12 = [    1200,     1201,     1202,     1203]
 - Rank  0, row 13 = [    1300,     1301,     1302,     1303]
 - Rank  0, row 14 = [    1400,     1401,     1402,     1403]
 - Rank  0, row 15 = [    1500,     1501,     1502,     1503]

File File3.dat:
 - Rank  0, row  0 = [       0,        1,        2,        3]
 - Rank  0, row  1 = [     100,      101,      102,      103]
 - Rank  0, row  2 = [     200,      201,      202,      203]
 - Rank  0, row  3 = [     300,      301,      302,      303]
 - Rank  0, row  4 = [     400,      401,      402,      403]
 - Rank  0, row  5 = [     500,      501,      502,      503]
 - Rank  0, row  6 = [     600,      601,      602,      603]
 - Rank  0, row  7 = [     700,      701,      702,      703]
 - Rank  0, row  8 = [     800,      801,      802,      803]
 - Rank  0, row  9 = [     900,      901,      902,      903]
 - Rank  0, row 10 = [       0,        0,        0,        0]
 - Rank  0, row 11 = [       0,        0,        0,        0]
 - Rank  0, row 12 = [       0,        0,        0,        0]
 - Rank  0, row 13 = [       0,        0,        0,        0]
 - Rank  0, row 14 = [       0,        0,        0,        0]
 - Rank  0, row 15 = [       0,        0,        0,        0]
-------------------------------------------------------------------
```

Gamma
Korekce
(zesvětlení)

ÚKOL 4: APLIKACE GAMMA KOREKCE NA OBRÁZEK LENNY

4: Aplikace gamma korekce na obrázek Lenny

Cílem tohoto příkladu je vyzkoušet si práci s hlavičkami a dlaždicovou dekompozici dat při načítání a ukládání do souboru. Root rank nejprve otevře vstupní soubor ve formátu PGM a přečte z něj hlavičku. Tu následně zpracuje a zjistí velikost uloženého obrázku. Nyní všechny ranky kolektivně načtou svoji dlaždici obrázku. Následně se aplikuje gamma korekce a provede se zpětné uložení (hlavička i data).
Zadání se nachází ve funkci `main` pod sekcí `case 4`:

1. Otevřete vstupní soubor pro čtení.
2. Smažte výsledný soubor.
3. Root rank načte a zpracuje hlavičku.
4. Vytvořte datový typ pro distribuci informací z hlavičky mezi jednotlivé ranky a proveďte rozhlášení.
5. Vytvořte datový typ pro čtvercovou dlaždici v souboru.
6. Nastavte pohled do vstupního souboru a přečtete obrázek. Nezapomeňte přeskočit hlavičku.
7. Aplikujte gamma korekci.
8. Zapište hlavičku do výstupního souboru.
9. Nastavte pohled a zapište výsledný obrázek.
10. Přeložte soubor.
11. Spusťte výslednou binárku:

```
$ mpiexec -np 4  ./io 4
$ mpiexec -np 16 --oversubscribe ./io 4
```

12. Porovnejte výsledný obraz, zda-li neobsahuje artefakty.

4: Aplikace gamma korekce na obrázek Lenny

- Každý rank si načte část obrázku **lena.pgm**
- -> může to být řádek / sloupec / **dlaždice** (16 ranků – dlaždice 4x4)
- Provedeme gamma korekci (zesvětlení)
- Paralelní zápis zpět do souboru
- Hlavička (lena.pgm)
  - **512 512** – velikost v pixelech
  - **255** – počet barev
- Pomocná struktura **Header** – cílem je parsovat hlavičku souboru a nastavit 3 proměnné:
  - **int nRows**
  - **int nCols**
  - **int size - velikost hlavičky (ne obrázku)**
- Strukturu **Header** lze poslat jako **MPI_Type_contiguous() // 3 x MPI_INT**
- **inFilename** – vstupní soubor
- **outFilename** – výstupní soubor
- **lRows** – lokální počet řádků
- **lCols** – lokální počet sloupců

```
P5
# Created by IrfanView
512 512
255
```

4: Aplikace gamma korekce na obrázek Lenny

- **// 1. Open input file.**
- **MPI_File_open() // MPI_MODE_RDONLY**

- **// 2. Delete the output file and the create a new one.**
- **MPI_File_delete(outFilename.data(), MPI_INFO_NULL)**
- **MPI_File_open() // MPI_MODE_CREATE | MPI_MODE_WRONLY**
- // Read Header by the root rank, expect maximum size 500 chars and write the same one to the output file.
- // Pouze root zparsuje hlavičku, uloží do výstupního souboru a broadcastem rozešle
- Header fileHeader{};
- **if (mpiGetCommRank(MPI_COMM_WORLD) == MPI_ROOT_RANK)** {
  constexpr std::size_t maxBufferSize{500};
  std::array<char, maxBufferSize> buffer{};
  // 3. Read the header string from the input file.
  MPI_File_read()
  fileHeader = Header(buffer.data());

  // 4. Write the header into the output file, use fileHeader.size and fileHeader.toString().c_str().
  MPI_File_write()
  }
  // Header can be sent as a contiguous block.
  static_assert(sizeof(Header) == 3 * sizeof(int));
- **// 5. Broadcast the header structure. Header is POD, can be sent as a contiguous block.**
- **MPI_Datatype headerType{MPI_DATATYPE_NULL}**
- **MPI_Type_contiguous()**
- **MPI_Type_commit(&headerType)**
- **MPI_Bcast() // nebo pošlu natvrdo 3 MPI_INT: MPI_Bcast(&fileHeader, 3, MPI_INT....)**

4: Aplikace gamma korekce na obrázek Lenny

- **// 6. Create a subarray to read/write a tile from/to the input/output file.**
- std::vector<unsigned char> image(lRows * lCols);

- std::array **imageSize** = {fileHeader.nRows, fileHeader.nCols};
- std::array **tileSize** = {lRows, lCols};
- std::array **tileStart** = {...}; **// každý rank vidí jinou dlaždici**
- **MPI_Type_create_subarray**(2, imageSize.data(), tileSize.data(), tileStart.data(), MPI_ORDER_C, MPI_UNSIGNED_CHAR, &tileType);
- **MPI_Type_commit()**

MPI_Type_create_subarray()
- Počet dimenzí = **2**
- **imageSize**: velikost původního obrázku
- **tileSize**: velikost dlaždice
- **tileStart**: kde začíná daná dlaždice

- **// 7. Set file view in the input and output files. - obrázek je v pohledu popsaný pomocí dlaždice tileType**
- mpiPrintf(MPI_ROOT_RANK, " Reading input file...\n");
- **MPI_File_set_view(inFile, fileHeader.size, MPI_UNSIGNED_CHAR, tileType, "native", MPI_INFO_NULL)**
- **MPI_File_set_view(outFile, fileHeader.size, MPI_UNSIGNED_CHAR, tileType, "native", MPI_INFO_NULL)**
- **// 8. Read input image.**
- **MPI_File_read_all()**
- // Run gamma correction kernel.
- ...

$$\text{tileStart}_x = \frac{\text{Rank}}{\sqrt{16}} \times lRows$$

$$\text{tileStart}_y = (\text{Rank} \mod \sqrt{16}) \times lCols$$

- **// 9. Write final image.**
- **MPI_File_write_all()**
- **// 10. Free created datatypes.**
- **MPI_Type_free()**
- **// 11. Close input and output file.**
- **MPI_File_close()**

Děkuji vám za pozornost!