Praktické paralelní programování (PPP 2023)
Počítačové cvičení č. 2: Kolektivní komunikace v MPI

Jiří Jaroš (jarosjir@fit.vutbr.cz)

1 ÚVOD

Cílem tohoto cvičení je vyzkoušet si práci s kolektivními komunikacemi v MPI. Cvičení je rozděleno do dvou tématických celků: 1. Broadcast a jeho implementace, 2. Skalární součin vektoru pomocí Scatter + Reduce.
Cvičení je možné vypracovat na superpočítačích Barbora nebo Karolina v plném rozsahu. Pokud budete pracovat na domácích počítačích nebo na merlinu, body 3.4 a 4.3 vynechte.

2 BROADCAST A JEHO IMPLEMENTACE

Soubor `bcast.cpp` obsahuje zadání tří dílčích problémů jejich cílem je rozhlásit vaše jméno přes všechny ranky. První problém využívá pouze funkce `MPI_Bcast`, druhý se pak snaží o implementaci Bcastu pomocí blokujících komunikací a třetí pomocí neblokujících.
Pokud používáte cluster Karolina:

1. Nejprve si zažádejte o jeden uzel v interaktivním módu

   ```
   $ salloc -p qcpu_exp -N 1 --ntasks-per-node 128 -t 01:00:00 --x11
   ```

2. Natáhněte modul s OpenMPI

   ```
   $ ml OpenMPI
   ```

Pokud používáte cluster Barbora:

1. Nejprve si zažádejte o jeden uzel v interaktivním módu

   ```
   $ salloc -p qcpu_exp -N 1 --ntasks-per-node 36 -t 01:00:00 --x11
   ```

2. Natáhněte modul s OpenMPI

   ```
   $ ml OpenMPI
   ```

2.1 Překlad

Vygenerujte překladový skript pomocí `cmake` a spusťte překlad:

```
$ cmake -Bbuild -S.
$ cmake --build build
```

2.2 Příklad 1. - Využití funkce MPI_Bcast

Před začátkem práce přepište proměnnou `studentLogin` vlastním loginem.

1. Zadání se nachází pod sekcí `case 1:` ve funkci `main`.
2. Aby bylo možné řetězec rozeslat, musíte použít dva broadcasty. Nejprve rozešlete velikost řetězce, poté si rezervujte v bufferu (funkce `reserve`) dostatek místa a následně rozešlete řetězec.
3. Použijte datové typy `MPI_INT` a `MPI_CHAR`. Řetězec můžete předat adresou na první prvek nebo metodou `data()`. Nepoužívejte metodu `c_str()`, jelikož vrací const pointer.
4. Přeložte soubor.
5. Spusťte výslednou binárku:

   ```
   $ mpiexec ./bcast 1
   ```

2.3 Příklad 2. - Implementace pomocí dvou blokujících P2P transferů

1. Zadání se nachází pod sekcí `case 2:` ve funkci `main`.
2. Nyní si definujte 2 MPI tagy (jeden pro velikost řetězce a druhý pro samotný řetězec).
3. Rozdělte výpočet na část, kde pracuje root, a kde ostatní. Můžete využít připravených funkcí `mpiGetCommRank(const MPI_Comm& comm)`, `mpiGetCommSize(cosnt MPI_Comm& comm)` a konstanty `MPI_ROOT_RANK`.
4. Nyní v cyklu rozešlete velikost a řetězec všem partnerům, kromě sebe pomocí `MPI_Send`.

5. U ostatních partnerů nejprve přijměte velikost řetězce a následně řetězec pomocí `MPI_Recv`.
6. Použijte datové typy `MPI_INT` a `MPI_CHAR`.
7. Přeložte soubor.
8. Spusťte výslednou binárku

```
$ mpiexec ./bcast 2
```

2.4 Příklad 3. - Implementace pomocí jednoho neblokujícího P2P transferů

1. Zadání se nachází pod sekcí `case 3:` ve funkci `main`.
2. Nyní si definujte pouze 1 MPI tag pro samotný řetězec.
3. Rozdělte výpočet na část, kde pracuje root, a kde ostatní. Můžete využít připravených funkcí `mpiGetCommRank(const MPI_Comm& comm)`, `mpiGetCommSize(const MPI_Comm& comm)` a konstanty `MPI_ROOT_RANK`.
4. Ve smyčce rozešlete řetězec všem parterům, kromě sebe pomocí `MPI_Isend`. Pozor na správné vytvoření pole requestů a vyjmutí roota ze skupiny na niž se čeká. Po odeslání všech zpráv zajistěte dokončení pomocí `MPI_Waitall`.
5. U ostatních partnerů nejprve zjistěte status zprávy, které dorazila pomocí `MPI_Probe`. Z tohoto statusu zjistěte velikost zprávy pomocí `MPI_Get_count`. Následně alokujte buffer a zprávy přijměte pomocí `MPI_Recv`.
6. Použijte datové typ `MPI_CHAR`.
7. Přeložte soubor.
8. Spusťte výslednou binárku

```
$ mpiexec ./bcast 3
```

2.5 Instrumentace a trasování

Tuto část provádějte pouze na superpočítači Barbora/Karolina. Po zobrazení trasovacích dat (trace) zazoomujte na konec časové osy (začátek představuje inicializace MPI) a algoritmu. Identifikujte část, kde se nachází broadcast. Ověřte, že broadcast nemá barierové chování.

```
$ ml purge
$ ml Scalasca Vampir

$ cmake -Bbuild_prof -S. -DCMAKE_CXX_COMPILER=scorep-mpicxx
```

```
$ cmake --build build_prof

$ scalasca -analyze  -t mpiexec -np 32  ./bcast.scorep 1

$ vampir scorep_bcast_32_trace/traces.otf2
```

3 Skalární součin vektoru pomocí Scatter a Redukce

Soubor `scatter.cpp` obsahuje zadání dvou dílčích problémů jejichž úkolem je rozptýlit dva vektory dané velikosti mezi jednotlivé ranky, spočítat dílčí skalární součiny a výsledek redukovat do rootu. V příkladu č. 1 uvažujte, že velikost pole je dělitelná počtem procesů, v příkladu č. 2 tomu tak již není.

1. Natáhněte modul s OpenMPI

```
$ ml purge
$ ml OpenMPI
```

3.1 Příklad 4. - Skalární součin nad polem soudělné velikosti

1. Zadání se nachází pod sekcí `case 1:` ve funkci `main`.
2. Vytvořte proměnou pro dílčí výsledek daného ranku.
3. Určete množství prvků v každém procesu. Vycházejte ze skutečnosti, že celkovou velikost `size` znáte v každém ranku.
4. Vytvořte lokální pole `a` a `b` a rezervujte v nich dostatek prostoru. Použijte `std::vector`.
5. Pomocí dvou volání `MPI_Scatter` rozptylte pole `a` a `b`. Pro předání adres bufferů použijte funkci `data()` typu `std::vector`.
6. Spočtěte lokální skalární součin.
7. Pomocí jednoho volání `MPI_Reduce` zredukujte výslednou hodnotu do ranku 0.
8. Přeložte soubor.
9. Spusťte výslednou binárku

```
$ mpiexec -np 32 ./scatter 1
```

3.2 Příklad 5. - Skalární součin nad polem nesoudělné velikosti

1. Zadání se nachází pod sekcí `case 2:` ve funkci `main`.
2. Vycházejte z předchozího příkladu.
3. Jelikož velikost pole a počet ranků nejsou soudělné, je nutné rozdělit práci nerovnoměrně. Nejprve si spočteme kolik by měl mít každý rank, pokud by byla velikost soudělná. Následně určíme kolik prvků zbývá a ty přiřadíme několika prvním rankům tak, aby každý z nich měl o prvek navíc.

4. Vytvoříme pole `sendCounts` - kolik každému a `displacements` - kde začíná porce pro každého.
5. Následně použijeme funkce `MPI_Scatterv`.
6. Zbytek už je stejný jako u minulého příkladu.
7. Přeložte soubor.
8. Spusťte výslednou binárku

```
$ mpiexec ./scatter 2
```

3.3 Instrumentace a trasování

```
$ ml purge
$ ml Scalasca Vampir

$ cmake -Bbuild_prof -S. -DCMAKE_CXX_COMPILER=scorep-mpicxx
$ cmake --build build_prof

$ scalasca -analyze  -t mpiexec -np 32  ./scatter.scorep 1
$ vampir
```