MAC 0219/5742 - Introdução à Computação Paralela e Distribuída

EP 2 - “Faça seu próprio MPI_Bcast em casa”

Alfredo Goldman e Vitor Terra

1. Introdução

O padrão MPI (*Message Passing Interface*) especifica, além das operações de comunicação ponto-a-ponto entre processos (`MPI_Send` e `MPI_Recv`), operações de comunicação coletivas, que envolvem simultaneamente todos os processos em um mesmo comunicador MPI. Uma das principais operações coletivas é `MPI_Bcast` (*broadcast*), cujo propósito é compartilhar dados presentes em um *buffer* `*buf` em processo `root` com todos os demais processos. Uma possível aplicação seria a leitura de um arquivo pelo processo `root` e transmissão para os demais processos, evitando múltiplas leituras simultâneas e eventuais gargalos de I/O. Sua assinatura é:

```
int MPI_Bcast(
     void *buf,
     int count,
     MPI_Datatype datatype,
     int root,
     MPI_Comm comm)
```

Apesar de o processo `root` e os demais terem papéis diferentes na execução, todos os processos realizam a mesma chamada de `MPI_Bcast`. Após o término da operação, os dados presentes na variável `*buf` do processo `root` estarão disponíveis na mesma variável em cada um dos demais processos (incluindo o próprio processo `root`). Para cada processo, são transmitidas `count` unidades do tipo `datatype`, ou seja, um total de `count*sizeof(datatype)` bytes. A figura a seguir ilustra a operação de *broadcast* entre quatro processos:

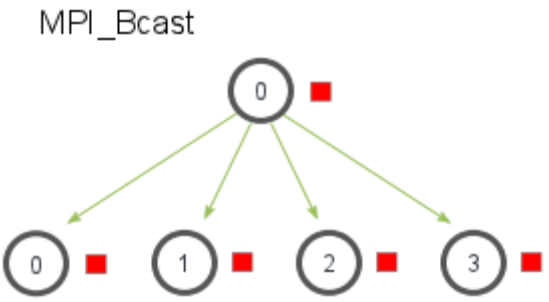

Figura 1: Esquema representando a operação `MPI_Bcast` com o processo de *rank* 0 como `root` e 4 processos no total.[1]

A partir da descrição anterior, poderia surgir a dúvida: qual a diferença entre chamar a função `MPI_Bcast` em vez de enviar o dado para cada processo usando `MPI_Send` e `MPI_Recv`? O objetivo deste EP é tentar responder essa pergunta, implementando uma função `custom_bcast` que apresente a mesma assinatura e comportamento que a função `MPI_Bcast`. Em outras palavras, deve ser possível trocar `MPI_Bcast` por `custom_bcast` em um determinado código e obter o mesmo resultado após a execução: transferir um bloco de dados do processo `root` para os demais processos.

2. Instalação do OpenMPI

Existem diversas implementações do padrão MPI disponíveis para uso. Neste EP, vamos usar OpenMPI, uma das principais implementações de código aberto. Para instalar OpenMPI no Linux, baixe e descompacte o arquivo *tarball* contendo a versão estável mais recente[2]:

```
$ wget https://download.open-mpi.org/release/open-mpi/v4.1/openmpi-4.1.5.tar.gz
```

```
$ tar -xzvf openmpi-4.1.5.tar.gz
```

Em seguida, configure o diretório de instalação em `--prefix` e instale:

```
$ cd openmpi-4.1.5
```

```
$ ./configure --prefix=/usr/local/
```

```
$ sudo make all install
```

[1] MPI Scatter, Gather, and Allgather · MPI Tutorial
[2] Disponível em Open MPI: Version 4.1

Para confirmar que a instalação ocorreu corretamente, verifique a versão instalada:

```
$ mpirun --version
```

Caso obtenha a seguinte mensagem de erro:

```
mpirun: error while loading shared libraries: libopen-rte.so.40:
cannot open shared object file: No such file or directory
```

Execute o seguinte comando[3]:

```
$ sudo ldconfig
```

3. Código fornecido

O código fornecido para o trabalho está disponível no GitHub[4] e contém os arquivos `broadcast.c` e `Makefile`. O arquivo `broadcast.c` gera um executável `broadcast`, que possui os seguintes parâmetros de entrada:

- `array_size`: tamanho do *array* de inteiros a ser transmitido do processo *root* para os demais;
- `root`: *rank* do processo *root* na operação *broadcast*, que deve ser um inteiro maior ou igual a zero e menor do que o número de processos. O valor padrão do parâmetro é zero;
- uma *flag* opcional `--custom`: quando fornecida, executa a função `custom_bcast` no lugar de `MPI_Bcast`.

O programa imprime em `stdout` o tempo transcorrido em segundos na execução da operação *broadcast*.

Para compilar o projeto, use o comando:

```
$ make
```

Para compilar em modo *debug*, no qual cada processo imprime em `stdout` os dados enviados ou recebidos, use o comando:

```
$ make debug
```

Entre cada execução de `make`/`make debug`, caso não haja alterações no código fonte, é necessário executar `make clean` para forçar a geração de um novo executável.

---

[3] Comando ldconfig no Linux (índice de bibliotecas) [Guia Básico]
[4] https://github.com/vitortterra/MAC0219-5742-EP2-2023

Para executar o programa, deve-se adicionar `mpirun -np <num_procs>` antes do executável `./broadcast`, onde `num_procs` é o número de processos MPI.

```
$ mpirun –np <num_procs> ./broadcast ––array_size <array_size>
                    [--root <root>] [––custom]
```

Cada processo MPI é associado a um *slot*, sendo o número de *slots* disponíveis limitado, em geral, pelo número de *cores* físicos disponíveis. Caso a quantidade de processos solicitada ultrapasse o número de *slots* disponíveis, ocorre o seguinte erro:

```
--------------------------------------------------------------------------

There are not enough slots available in the system to satisfy the 4 slots that
were requested by the application:

  ./broadcast

Either request fewer slots for your application, or make more slots available
for use.

A "slot" is the Open MPI term for an allocatable unit where we can launch a
process.  The number of slots available are defined by the environment in which
Open MPI processes are run:

  1. Hostfile, via "slots=N" clauses (N defaults to number of processor cores
if not provided)

  2. The --host command line parameter, via a ":N" suffix on the hostname (N
defaults to 1 if not provided)

  3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)

  4. If none of a hostfile, the --host command line parameter, or an RM is
present, Open MPI defaults to the number of processor cores

In all the above cases, if you want Open MPI to default to the number of
hardware threads instead of the number of processor cores, use the
--use-hwthread-cpus option.

Alternatively, you can use the --oversubscribe option to ignore the number of
available slots when deciding the number of processes to launch.

--------------------------------------------------------------------------
```

Assim, caso seja necessário, os experimentos podem ser executados adicionando as *flags* `--use-hwthread-cpus` (habilita o uso de *hyperthreading*[5]) ou `--oversubscribe` (executa mais de um processo por *slot*). Ao reportar tempos de execução no relatório, indique caso alguma dessas opções seja utilizada.

Para remover o arquivo gerado na compilação, use o comando

```
$ make clean
```

4. Tarefas

Você e seu grupo deverão implementar a sua própria versão da *operação broadcast* na função `custom_bcast`, inicialmente vazia. Ela deve ter a mesma assinatura que `MPI_Bcast`, além de apresentar o mesmo comportamento: enviar dados do processo *root* para cada um dos demais processos. Note que:

- assim como `MPI_Bcast`, todos os processos chamam `custom_bcast`, tanto o root quanto os demais;
- o valor de retorno das funções MPI indica o sucesso ou o tipo de erro da operação.

Verifique que seu programa está correto, variando o número de processos, o tamanho do *array* e o *rank* do processo *root*, executando em modo *debug* (ver seção anterior) e com a *flag* `--custom` na execução.

Em seguida, execute experimentos para verificar como varia o tempo de execução em função do número de processos (np) e da quantidade de dados transferidos entre processos (`array_size`), para cada uma das versões do *broadcast* (`MPI_Bcast` e `custom_bcast`). Para medição de tempo, não é necessário fornecer o parâmetro `--root`, usando assim o processo de *rank* zero como padrão. Sugerimos o uso de um *script* simples para automatizar a realização dos experimentos e as chamadas a `mpirun (...) ./broadcast (...)` com os devidos parâmetros.

Obtenham o tempo médio de execução combinando os seguintes valores dos parâmetros de entrada, tanto com quanto sem a *flag* `--custom`:

```
array_size = 2^10, 2^11, 2^12, 2^13, 2^14, 2^15, 2^16, 2^17

np = 1, 2, 4, 8, 16, 32
```

[5] Hyper-threading – Wikipédia, a enciclopédia livre

Vocês devem fazer um certo número de medições e analisar a variação dos valores obtidos. Sugerimos pelo menos 10 medições para cada experimento, e também que vocês usem a média e o intervalo de confiança das 10 medições nos seus gráficos. Caso observem variabilidade muito grande nas medições, resultando num intervalo de confiança muito grande, vocês podem realizar mais medições, sempre apresentando a média e o intervalo de confiança. Não é recomendado fazer menos de 10 medições.

Depois de realizar os experimentos, vocês deverão elaborar gráficos que evidenciem o comportamento das duas versões de *broadcast* com relação à variação dos parâmetros descritos anteriormente. Os gráficos deverão ser claros e legíveis, com eixos nomeados. Deverão apresentar a média e o intervalo de confiança das (pelo menos) 10 execuções para cada cenário experimental. A automação dos experimentos e da visualização dos dados gerados é fundamental para a pesquisa em Ciência da Computação, pois permite gerar e analisar grandes conjuntos de dados sem muito esforço manual.

Incluam os gráficos em um relatório, comparando as duas versões do *broadcast*, analisando como ambas se comportam com a variação do tamanho da entrada e do número de processos, e explicando por que isso ocorre.

5. Entrega

Vocês deverão entregar no e-Disciplinas um único arquivo .zip por grupo, com os nomes dos integrantes, contendo:

- Um relatório em .pdf com as análises e gráficos;
    - Relatórios em .doc, .docx ou .odt não serão aceitos.
- O arquivo `broadcast.c` contendo a implementação da função `custom_bcast`;
- Um arquivo .csv com as medições feitas.

A nota do EP2 vai de **0.0** a **10.0**, e a avaliação será feita da seguinte maneira:

- Implementação: **5.0**
    - Código compila sem erros e *warnings*: **2.0**
    - Código executa sem erros e produz o resultado correto: **2.5**
    - Boas práticas de programação e clareza do código: **0.5**
- Relatório: **5.0**
    - Apresentação e análise dos experimentos: **4.5**
    - Clareza do texto e figuras: **0.5**

Em caso de dúvidas, use o fórum de discussão do e-Disciplinas ou entre em contato diretamente com o monitor (vitortterra@ime.usp.br) ou o professor (gold@ime.usp.br).