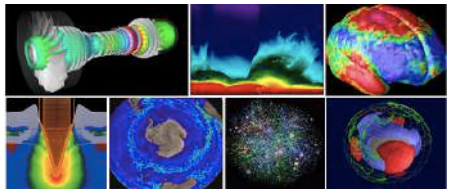
Tính toán song song và phân tán

PGS.TS. Trần Văn Lăng
langtv@vast.vn
langtv@gmail.com



Lập trình với MPI

1. Giới thiệu về MPI
2. Đặc điểm
3. Cài đặt
4. Sử dụng
5. Cách lập trình
6. Áp dụng

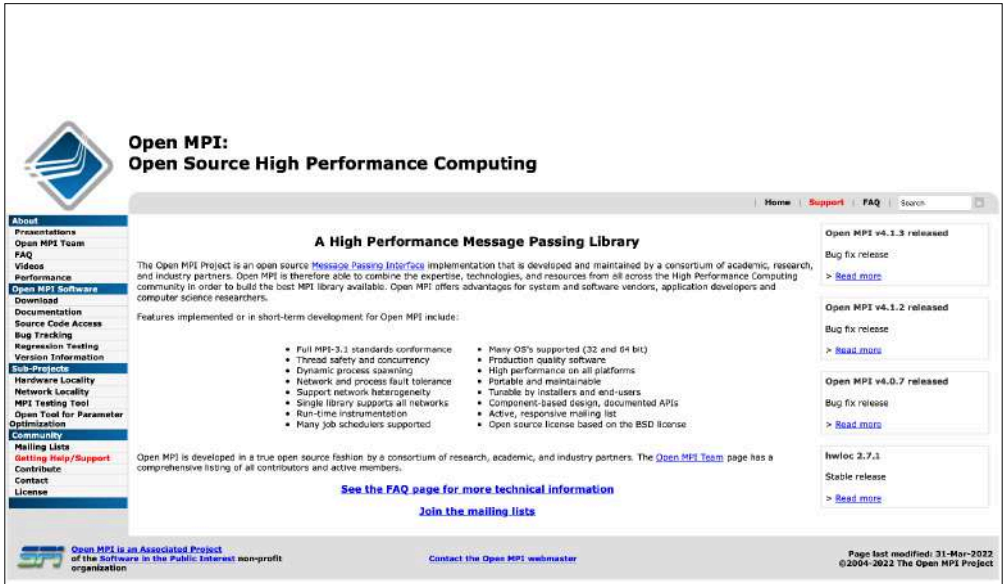
Message – Passing Interface (MPI)

- Khác với PVM được phát triển trong một dự án nghiên cứu,
- Còn MPI ra đời bởi một ủy ban trong một diễn đàn Hội nghị với trên 60 chuyên gia tứ 40 tổ chức khác nhau.

- Do mỗi nhà sản xuất MPP (Massively Parallel Processor) đã tạo ra các API trao đổi thông điệp riêng,
- Nên khó viết những chương trình tính toán song song có tính cơ động (portable)
- MPI là một hệ thống giao tiếp chuyển thông điệp được dùng trên mạng phân tán.

Một số phiên bản hiện thực

- MPICH của Argonne National Lab/Mississippi State University: http://www-unix.mcs.anl.gov/mpi/mpich/download.html
- CHIMP của Edinburgh Parallel Computing Centre ftp://epcc.ed.ac.uk/pub/chimpk
- OpenMPI: https://www.open-mpi.org: The Open MPI project has many members, contributors, and partners (https://www.open-mpi.org/about/members/)



- LAM/MPI của Lab. for Scientific Computing, Dep. of Computer Science and Engineering (University of Notre Dame), http://www.lam-mpi.org
- HP MPI của Hewlett-Packard Company , http://h21007.www2.hp.com/dspp/tech/tech_TechSoftwareDetailPage_IDX/1,1703,1238,00.html

Lịch sử MPI

- Những chuyên gia xây dựng hệ thống ban đầu tứ những trường đại học, các viện nghiên cứu, các Phòng thí nghiệm quốc gia, các xí nghiệp sản xuất máy tính song song.
- Họ cùng nhau mô tả và định nghĩa tập hợp các thư viện giao tiếp chuẩn trong chuyển thông điệp. Từ đó chuẩn hóa các giao thức chuyển thông điệp.
- https://cvw.cac.cornell.edu/mpi/history

- Tiến trình chuẩn hóa được bắt đầu từ Workshop on Standards for Message Passing in a Distributed Memory Environment được tổ chức vào hai ngày 29 – 30/4/1992 ở Williamsburg - Virginia.
- Hội thảo này được tài trợ bởi "Center for Research on Parallel Computing",

- Phiên bản 1.0 ra đời vào tháng 5/1994,
- Sau đó vào tháng 3/1995, diễn đàn MPI lại nhóm họp, và phiên bản 1.1 được đưa ra vào 6/1995.
- MPI-2 bắt đầu vào tháng 4/1994 và sau đó cứ 6 tuần, các nhà thiết kế lại gặp nhau một lần cho đến ngày 25/4/1997 MPI-2 được thông qua.
- MPI-3 xuất hiện từ năm 2008 với các thư viện cho ngôn ngữ FORTRAN

- MPI-2 có thêm nhiều hàm so với MPI-1 và có nhiều phương pháp truyền thông hơn. Chẳng hạn,
  - **MPI_Spawn()** để khởi động các MPI process
  - Các hàm truyền thông 1 chiều như put, get
  - Các hàm truyền thông dạng non-blocking

Đặc điểm của MPI

- Những thuận lợi cơ bản MPI là có những chương trình chuyển thông điệp cấp thấp trên hệ thống phân tán,
- Các chương trình này đã được chuẩn hóa để người sử dụng thuận tiện khi dùng các ngôn ngữ cấp cao như C/C++, FORTRAN.

- MPI được tập hợp bởi trí tuệ của hầu hết các chuyên gia hàng đầu trong lĩnh vực này (kể cả các chuyên gia thiết kế PVM), nên các giao thức giao tiếp thuận lợi và tốt hơn.

- PVM là máy ảo, là tập hợp các máy không đồng nhất được kết nối với nhau để người sử dụng dùng như một máy song song lớn đơn giản.
- MPI với ý định tạo ra các đặc tả chuyển thông điệp chuẩn để những nhà sản xuất bộ xử lý song song quy mô (Massively Parallel Processors) cài đặt trên hệ thống của họ.

- MPI có một tập khá lớn các hàm truyền thông điểm đến điểm (point-to-point communication)
- Có nhiều hàm truyền thông tập thể (collective communication)
- Có khả năng xác định các sơ đồ truyền thông (communication topologies)

Cài đặt OpenMPI

- Download: https://download.open-mpi.org/release/open-mpi/v4.1/openmpi-4.1.3.tar.gz
- Install: chẳng hạn, install ở $HOME
  - tar xzvf openmpi-4.1.3.tar.gz
  - cd openmpi-4.1.3
  - ./configure --prefix=$HOME/openmpi
  - make all
  - make install # Để có folder $HOME/openmpi

- Trên macOS có thể install qua brew
  - $~> brew install openmpi
- Hoặc
  - $~> brew instll open-mpi
- Trên Ubuntu
  - $~>apt install openmpi-bin

- Để dịch (bằng ngôn ngữ C) và thực thi
  - mpicc -o example mpitest.c
  - mpirun -np 2 mpitest
- Để thực thi trên nhiều máy, phải
  - Copy source file đến máy để biên dịch lại

  scp mpitest.c lang@Ubuntu:/home/lang/work/MPI
  - Lần sau, có thể đồng bộ bằng lệnh

  rsync -arv mpitest.c lang@Ubuntu:/home/lang/work/MPI

Sử dụng MPI

- Để biên dịch các chương trình viết bằng ngôn ngữ C và C++, sử dụng các shell script đã có sẵn như mpicc, mpiCC. Chẳng hạn,

  $ mpicc -o mpitest mpitest.c

  $ mpiCC -o mpitest mpitest.cc

```c
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv){
  int rank;
  char hostname[256];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  gethostname(hostname,255);
  printf("Hello from process %3d on host %s\n",
  rank, hostname);
  MPI_Finalize();
  return 0;
}
```


- Sau khi đã biên dịch, để thực thi chương trình example với hai process, chúng ta viết

  $ mpirun -np 2 mpitest
- Có thể chỉ định đường dẫn của tập tin thi hành qua tham số -wdir

  $ mpirun -np 2 -wdir folder mpitest

- Hoặc lưu trữ tập tin thi hành này trong tham số PATH. Chẳng hạn, bổ sung thêm vị trí lưu trữ này:

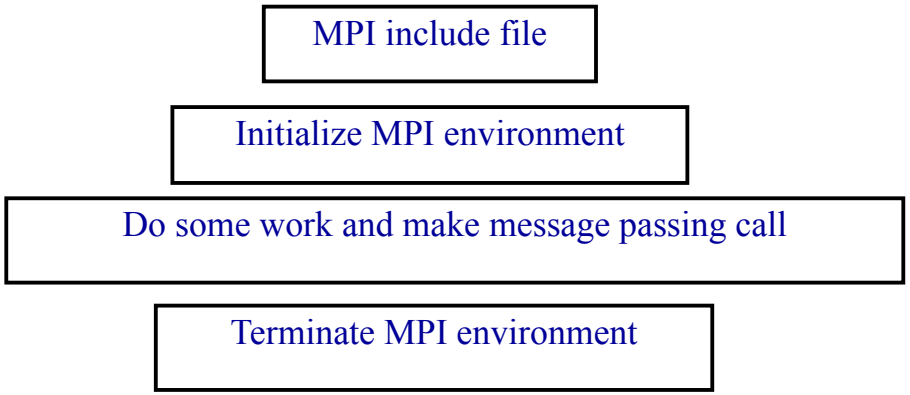
  echo "export PATH=$PATH:$HOME/work/MPI" >> $HOME/.bash_profile
- Cũng lưu ý rằng, để các node liên kết được với nhau (trao đổi thông điệp với nhau) thì trước đó phải thiết lập cơ chế remote shell. Chẳng hạn qua rsh hoặc ssh.

- Thông thường, dùng tham số -hostfile để chỉ định số tiến trình trên mỗi máy (mỗi node) thông qua tập tin lưu trữ
- Chẳng hạn với tập tin myhosts, có các dòng
  - Ubuntu slots=3
  - Ubuntu slots=4
- Khi đó với lệnh
  - mpirun -hostfile myhosts mpitest

- Kết quả

```
lang@Ubuntu:~$ mpirun -hostfile myhosts mpitest
Hello from process   2 on host Ubuntu
Hello from process   1 on host Ubuntu
Hello from process   0 on host Ubuntu
Hello from process   4 on host Ubuntu2
Hello from process   6 on host Ubuntu2
Hello from process   5 on host Ubuntu2
Hello from process   3 on host Ubuntu2
lang@Ubuntu:~$ _
```


Cách lập trình MPI cơ bản



- MPI sử dụng những đối tượng gọi là cơ cấu truyền thông (**communicator**) và nhóm (**group**) hay sơ đồ kết nối (**topology**) để tập hợp các process giao tiếp với nhau.
- Hầu hết các hàm MPI đòi hỏi phải chỉ định **communicator** như là một đối số.

- Trong một **communicator** mỗi process có một giá trị duy nhất là số tự nhiên (gọi là tác vụ ID hay process ID),
- Giá trị này tăng liên tục từ 0 cho đến **NPROC-1**, trong đó **NPROC** là số tác vụ của **communicator**.

- MPI không sử dụng khái niệm máy ảo như PVM, nhưng cung cấp một sự trừu tượng cao hơn bằng sơ đồ trao đổi thông điệp (Message - Passing Topology).
- MPI sử dụng cơ cấu truyền thông và sơ đồ kết nối làm cho MPI khác với các hệ thống truyền thông điệp khác.

- Cơ cấu truyền thông là tập hợp các process có thể trao đổi với nhau. Cơ cấu truyền thông là một miền truyền thông (comm. domain) để định nghĩa tập các process cho phép giao tiếp với nhau.
- Sơ đồ kết nối là một cấu trúc trên các process của một cơ cấu truyền thông, cho phép xác định các process theo nhiều cách khác nhau.

- Có hai loại cơ cấu truyền thông:
  - *Intracommunicator*: truyền thông trong cùng nhóm
  - *Intercommunicator*: truyền thông giữa các nhóm
- Một process có một vị trí trong nhóm, cũng có thể là thành viên của nhiều nhóm.
- Chuẩn là cơ cấu intracommunicator. Khi một chương trình khởi động, các process được cấp cho cùng cơ cấu truyền thông sẽ khởi động theo.

Mô hình tính toán SPMD

```
main( int argc, char *argv[] ){
 MPI_Init(&argc, &argv);
 ............
 MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
 if ( myrank == 0 )
  master();
 else
  slave();
 ............
 MPI_Finalize();
}
```


Hàm thư viện MPI

- MPI cơ bản (MPI-1)có 128 hàm thư viện.
- Trường hợp đơn giản, chỉ cần 6 hàm thư viện như sau:
  - `MPI_Init()`
  - `MPI_Comm_rank()`
  - `MPI_Comm_size()`
  - `MPI_Send()`
  - `MPI_Recv()`
  - `MPI_Finalize()`

`MPI_Init(int*,char**)`

- Để khởi tạo môi trường thực thi MPI.
- Hàm này được gọi chỉ một lần trong chương trình MPI và được gọi trước bất kỳ hàm MPI nào.
- Hai tham số được chuyển từ tham số của hàm `main(int argc,char* argv[])`. Tham số này luôn luôn xuất hiện mặc dù trong chương trình không dùng đến.

MPI_Comm_rank(MPI_Comm,int*)

- Hàm được dùng để xác định ID của tác vụ, có giá trị trong khoảng **[0,NPROC-1]**.
- Tham số thứ nhất là **communicator**, nếu giá trị là **MPI_COMM_WORLD** sẽ cho biết tất cả các process sẽ chạy khi ứng dụng bắt đầu thực thi.
- Tham số thứ hai lưu trữ tác vụ ID được trả về, nếu là chương trình master, giá trị 0.

MPI_Comm_size(MPI_Comm,int*)

- Hàm này cho biết số process trong nhóm được liên kết.
- Thông thường dùng với cơ cấu **MPI_COMM_WORLD** để xác định số process bởi chương trình ứng dụng.

MPI_Send(void* buf,int count,MPI_Datatype data,int dest,int tag,MPI_Comm comm)

- Dùng để truyền dữ liệu tại địa chỉ **buf** đến process định danh là **dest**, số dữ liệu được truyền là **count** với kiểu của mỗi dữ liệu là **data**.
- Còn **tag** và **comm** là nhãn thông điệp và cơ cấu truyền thông.

MPI_Datatype	C/C++ datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

```
MPI_Recv(void* buf,int count, MPI_Datatype
data,int src,int tag,MPI_Comm comm,MPI_Status
*status)
```

- Tương tự với **MPI_Send()**, **MPI_Recv()** thực hiện công việc ngược lại để nhận dữ liệu.
- Hàm trả về giá trị **status** thuộc kiểu **MPI_status** là **struct** gồm 3 thành phần **MPI_SOURCE**, **MPI_TAG** và **MPI_ERROR** qua đó có thể nhận biết kết qủa.

MPI_Finalize()

- Hàm được gọi sau tất cả các hàm MPI khác để kết thúc môi trường thực thi MPI.

Ví dụ: Gửi thông điệp từ slave

- MPI_Init() // khoi dong MPI
- MPI_Comm_rank() // tim thu tu cua tien trinh
- MPI_Comm_size() // tim so tien trinh
- MPI_Get_processor_name() // ten host

- if (rank)
  - **sprintf(msg, "Greetings from process %d on %s!", rank, hostname)**
  - **MPI_Send()**
- else
  - **for (source = 1; source < size; source++)**
    - **MPI_Rcev();**
    - **printf("%s\n", msg)**
- MPI_Finalize();

Ví dụ: Đo thời gian truyền 1 dữ liệu

- double data[100];
- if (rank == 0)
  - double t0 = MPI_Wtime()
  - MPI_Send(data)
  - MPI_Recv(data)
  - double t1 = MPI_Wtime()
  - printf("Time for transfer a data: %10.6f\n", (t1-t0)/2/100)
- else if(rank == 1)
  - MPI_Recv(data)
  - MPI_Send(data)
- MPI_Finalize()

Ví dụ: Thời gian truyền dữ liệu của từng tiến trình

- double t[size];
- if (rank == 0)
  - double t0 = MPI_Wtime()
  - MPI_Send(t0)
  - for (src = 1; src < size; src++)
    - MPI_Recv(t[src])
  - for (src = 1; src < size; src++)
    - printf("%f\n", t[src]);
- else
  - MPI_Recv(t0)
  - double t1 = MPI_Wtime()
  - MPI_Send(t1-t0)
- MPI_Finalize()

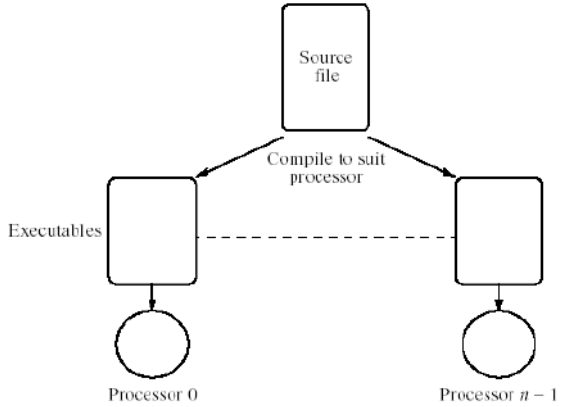
Ví dụ: Tính tổng (theo PVM đã có)

- if (rank == 0)
  - for (p = 1; p < size; p++)
    - MPI_Send(r)
    - MPI_Send(data[r*p])
  - for (i = 0; i < r; i++)
    - sum += data[i]
  - for (i = r*size; i < N; i++)
    - sum += data[i]
  - for (p = 1; p < size; p++)
    - MPI_Recv(s)
    - sum += s
  - prinf("%d", sum)
- else
  - MPI_Recv(r)
  - MPI_Recv(data)
  - for (i = 0; i < r; i++)
    - s += data[i]
  - MPI_Send(s)
- MPI_Finalize()

Ứng dụng

- Nhắc lại: có hai bước cơ bản cho các tính toán truyền thông điệp:
  - Tạo ra các tiến trình, tác vụ (process, task) riêng để thực thi trên các máy tính khác nhau.
  - Thực thi các việc liên quan đến gửi và nhận thông điệp.

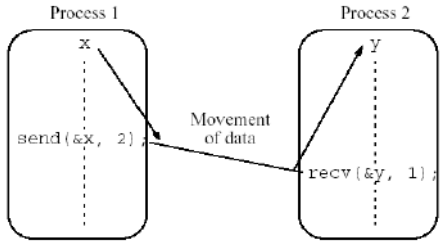
- SPMD (Single Program Multiple Data):
  - Các tiến trình khác nhau được hòa vào trong một chương trình.
  - Trong khi chương trình thực hiện, các phần khác nhau sẽ được chọn lựa cho mỗi task



- MPMD (Multiple Program Multiple Data):
  - Các chương trình riêng biệt được viết cho mỗi processor.
  - Một processor nào đó thi hành một tiến trình chủ (master process) và những tiến trình khác được bắt đầu từ bên trong tiến trình chủ.

Thủ tục gửi và nhận

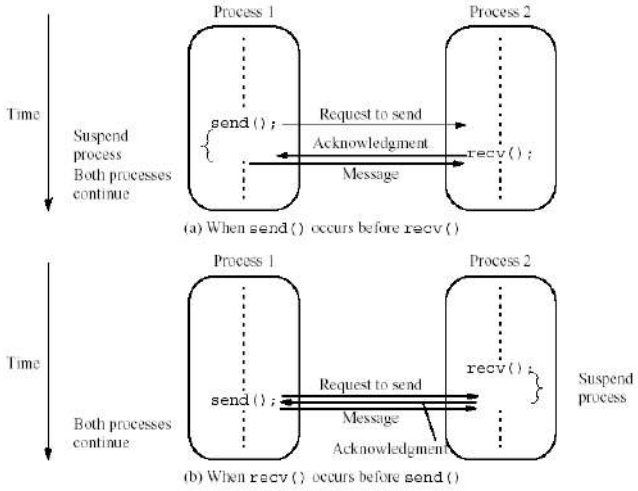
- Việc chuyển thông điệp giữa các tiến trình thực chất là sử dụng các hàm thư viện send() và receive().



Chuyển thông điệp đồng bộ

- Một hàm được gọi là chuyển đồng bộ (Synchronous Passing) nếu nó kết thúc (return) khi việc truyền thông điệp vừa hoàn thành.
- Hàm được gọi là gởi đồng bộ (synchronous send) nếu nó đợi cho đến khi thông điệp được chấp nhận bởi tiến trình nhận.

- Hàm nhận đồng bộ sẽ đợi cho đến khi thông điệp mà nó mong đợi đi đến.
- Thực chất của các hàm đồng bộ là thực hiện hai hành động: truyền dữ liệu và đồng bộ hóa các tiến trình, nói cách khác, có sự phối hợp với nhau trong việc gởi và nhận.



Blocking và Nonblocking

- Blocking được dùng để mô tả những hàm sẽ trả về khi dữ liệu truyền hoàn tất
- Blocking và Synchronous là đồng nghĩa.
- Nonblocking và Asynchronous là đồng nghĩa
- Trong PVM, các hàm gởi là asynchronous.
- Còn các hàm nhận là có thể là synchronous hoặc asynchronous.

Ví dụ: Tính tích phân

- Cho hàm số y = f(x), tìm tích phân trên đoạn [a,b] của hàm số này.
- Ta có với h khá nhỏ,

$$\int_a^{a+h} f(x)dx = \frac{h}{2}\left[f(a) + f(a+h)\right]$$

- Trường hợp đoạn [a,b] khá lớn, ta có thể phân hoạch ra thành các đoạn con khá nhỏ với:

$$a = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = b$$

- Khi đó,

$$\int_{x_i}^{x_{i+1}} dx = \frac{x_{i+1} - x_i}{2}\left[f(x_{i+1}) + f(x_i)\right] = \frac{h}{2}\left[f(x_{i+1}) + f(x_i)\right]$$

- Từ đây suy ra

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \ldots + \int_{x_{n-1}}^{x_n} f(x)dx$$

$$= \frac{h}{2}\left[f(x_0) + f(x_n)\right] + h\left[f(x_1) + \ldots + f(x_{n-1})\right]$$

Thuật toán tuần tự

```
float f(x)
h = (b-a)/n
Seq(a,b,n,h)
   integral = (f(a)+f(b))/2
   x = a
   for ( i = 0; i < n ; i++ )
   x += h
   integral += f(x)
   integral *= h
```

Thuật toán song song

- if (rank == 0)
  - h = (b-a)/n
  - local_n = n/p
- local_a = a + rank*local_n*h
- local_b = local_a + local_n*h
- local_integral = Seq(local_a,local_b,local_n, h)
- if (rank == 0)
  - integral = local_integral
  - for (p = 1; p < size; p++)
    - MPI_Recv(local_integral)
    - integral += local_integral
  - printf(integral)

- else
  - MPI_Send(local_integral)
- float Seq(local_a,local_b,local_n,h)
  - integral = (f(local_a)+f(local_b))/2
  - x = local_a
  - for (i=1; i <local_n; i++)
    - x += h
    - integral += f(x)
  - integral *= h
  - return integral

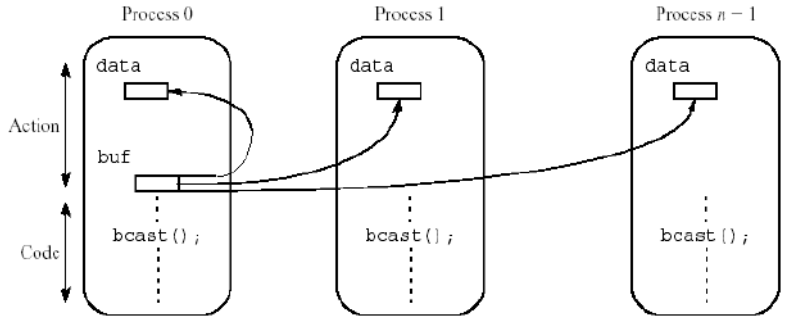
Xem tập tin integral.c

Cách thức truyền dữ liệu

- Phát tán, truyền rộng khắp (broadcast)
- Phân phối, tán xạ (scatter)
- Thu thập (gather)
- Rút gọn (reduce)

Broadcast (phát tán)

- Gởi cùng 1 thông điệp đến tất cả các process liên quan đến bài toán

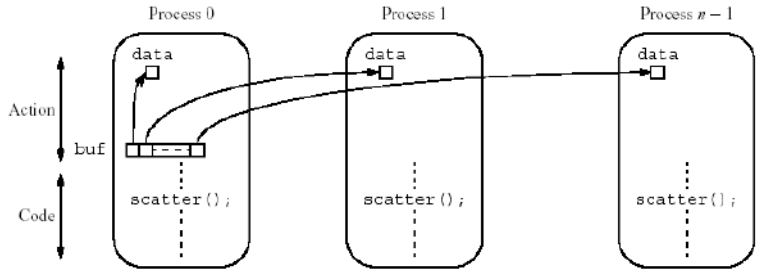


Ví dụ: Tính tích phân

- Có thể cải tiến tính tích phân trong đoạn gửi các giá trị
  - h = (b-a)/n
  - local_n
- Đến tất cả các tiến trình bằng cách dùng hàm MPI_Bcast()

Scatter (phân phối)

- Gởi từng phần của dữ liệu đến từng process riêng



Tính tổng của một dãy

- Giả sử số phần tử cần tính tổng chia hết cho số tiến trình.
- Dùng hàm MPI_BCast() để chuyển số phần tử cần tính tổng trên mỗi tiến trình cho các tiến trình
- Dùng hàm MPI_Scatter() để phân phối các giá trị cần tính cho từng tiến trình

Thuật toán của chương trình chính

- initializing MPI
- If (rank == 0)
  - initialize() /* công việc phải làm ban đầu để có dữ liệu */
- MPI_Bcast(r=N/P)
- if (rank != 0)
  - d = new float[r]
- MPI_Scater(data, d, root)
- dowork(s) //tính các tổng riêng

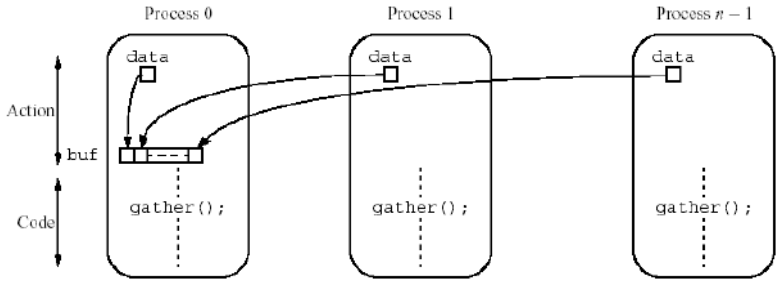
- if (rank == 0)
  - sum = s
  - for (i = 1; i < P; i++)
    - MPI_Recv(s)
    - sum += s
- else
  - MPI_Send(s)
- MPI_Barrier() /* đồng bộ hóa các công việc ở phần trước */

- if (rank == 0)
  - cout << sum
- MPI_Finalize()

Xem tập tin sumMPI01.cc

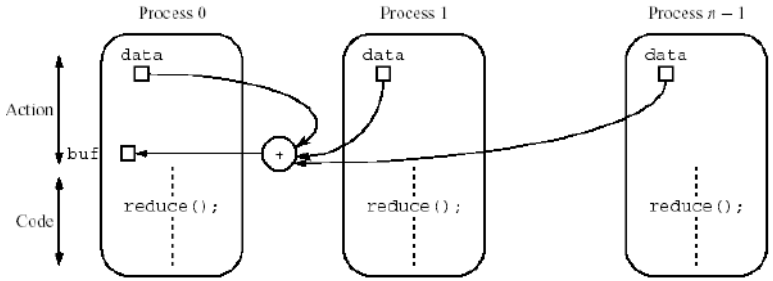
Gather (thu thập)

- Thu thập các giá trị riêng từ tập những process.



Reduce (rút gọn về)

- Thực chất là thu thập (gather) nhưng kết hợp với những phép toán số học, luận lý nào đó



Ví dụ: Tính tích phân

- Có thể tính tổng tích phân (integral) từ các tổng riêng (local_integral) bằng cách dùng hàm rút gọn MPI_Reduce() với bộ lọc MPI_SUM

Ví dụ: tính tổng có dùng reduce

- Thay cho các lệnh:
  - if (rank == 0)
    - sum = s
    - for (i = 1; i < P; i++)
      - MPI_Recv(s)
      - sum += s
  - else
    - MPI_Send(s)
  - MPI_Barrier() /* đồng bộ hóa các công việc ở phần trước */

- Có thể viết
  - sum = 0
  - MPI_Reduce(&s, &sum, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD)
- Để cộng dồn các giá trị trong s vào sum của tiến trình 0

Xem tập tin sumMPI02.cc

Ví dụ: Tích vô hướng

- Cho $X = (x_1, x_2, \ldots, x_n)^T, Y = (y_1, y_2, \ldots, y_n)^T$
- Tính $\langle X, Y \rangle = x_1y_1 + x_2y_2 + \ldots + x_ny_n$
- Thuật toán tuần tự:
  - float Sequence(float* x, float* y, int n)
    - float s = 0
    - for (i = 0; i < n; i++)
      - s += x[i]*y[i]
    - return s

Thuật toán song song

- float Parallel(float* lx, float* ly, int nb)
  - float ldot = Sequence(lx,ly,nb)
  - MPI_Reduce(&ldot,&dot,1,MPI_FLOAT,MPI_SUM,0,MPI_COMM_WORLD)
  - return dot

Ví dụ: Tính tích ma trận với vector

- Cho ma trận $A = (a_{ij})$ cấp $m \times n$ và $X = (x_1, x_2, \ldots, x_n)^T$.
- Tìm $Y = (y_1, y_2, \ldots, y_m)^T$ sao cho $Y = AX$
- Các giá trị y_i được tính bởi công thức như sau: $y_i = a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n$

- Sử dụng giao tiếp gather thông qua hàm MPI_Gather() để thu thập tất cả các tổng riêng y_i về tiến trình thứ nhất.

- Khởi tạo MPI
- if (rank == 0)
  - initialize();
- MPI_Bcast(row)
- MPI_Bcast(col);
- if (rank != 0)
  - A = new float[row*col/P]
  - Y = new float[row/P]
  - X = new float[col]
- MPI_Bcast(X, col)
- MPI_Scatter(A,row*col/P, A, row*col/P)
- mult(A,X,Y,row/P,col)
- MPI_Gather(Y,row/P,Y,row/P)
- if (rank == 0)
  - print(Y)
- MPI_Finalize()

- void initialize()
  - A = new float[row*col]
  - read Matrix A
  - X = new float[col]
  - Y = new float[row]
  - read Vector X
- void mult(a,x,y,r,c)
  - for (i = 0; i < r; i++)
    - t = 0
    - for (j = 0; j < c; j++)
      - t += a[i*c+j]*x[j]
    - y[i] = t

Chi phí của thuật giải

- Xét bài toán tính tổng của n số nguyên.
- Giả sử có P task, trong đó task thứ nhất là vai trò master.
- $P-1$ task còn lại tính tổng của $\frac{N}{P}$ số
- Task đầu tiên tính tổng của $N-\frac{N}{P(P-1)}$.
- $T_p = T_{comp} + T_{comm}$

- Khi đó thuật giải gồm các bước:
  1. Master gởi $\frac{N}{P}$ số đến $P-1$máy slave
  2. Tất cả $P-1$ máy đồng thời tính tổng của $\frac{N}{P}$ số
  3. Master tính tổng của $N-\left[\frac{N}{P}\right]+\frac{N}{P}$
  4. $P-1$ máy slave gởi $P-1$ tổng riêng về cho master
  5. Master tính tổng của P tổng riêng

- Thời gian tính toán là (bước 2, 3 và 5)
  $T_{comp} = \frac{N}{P} + P - 1$
- Thời gian truyền là (bước 1 và 4)
  $T_{comm} = (P-1)\left(t_0 + \frac{N}{P}t_d\right) + (P-1)(t_0 + t_d)$

- Khi đó chi phí của thuật toán song song là
$T_p = T_{comp} + T_{comm}$

$$= \frac{N}{P} + P - 1 + (P-1)\left(2t_0 + \frac{N}{P}t_d + t_d\right)$$

$$= \frac{N}{P} + P + 2t_0P + t_dP - \frac{N}{P}t_d - 1 + Nt_d - 2t_0 - t_d$$

$$= (1-t_d)\frac{N}{P} + (1 + 2t_0 + t_d)P - (1 + 2t_0 - Nt_d + t_d)$$

- Để chi phí đạt cực tiểu, ta cần $\frac{\partial T_p}{\partial P} = 0$
- Hay $\frac{N}{P^2}(1-t_d) = 1 + 2t_0 + t_d$

- Từ đây suy ra số tiến trình cần thiết là

$$P = \sqrt{\frac{N(1-t_d)}{1 + 2t_0 + t_d}}$$

- Chẳng hạn, trên mạng 10Mbps = 10.485.760bps, thì thời gian để truyền 1 số nguyên (có 32 bit) là 0.0000030517578125 sec.