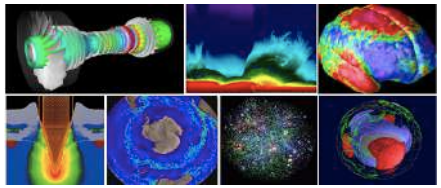
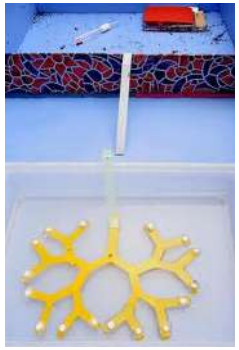
Tính toán song song và phân tán

PGS.TS. Trần Văn Lăng
langtv@vast.vn
lang@lhu.edu.vn
http://staff.vast-hcm.ac.vn/~lang
http://fair.conf.vn/~lang



PVM trong việc lập trình song song

1. Môi trường truyền thông điệp
2. Hệ thống truyền thông điệp
3. PVM – Parallel Virtual Machine
4. Kiến trúc PVM
5. Cài đặt PVM
6. Sử dụng PVM
7. Lập trình trong PVM



Môi trường truyền thông điệp

- Để thực hiện tính toán song song và phân tán, cần có môi trường truyền thông điệp với 3 yếu tố:
  - Multiple processors (Cho các trạm làm việc)
  - Network (Liên kết giữa các trạm)
  - Môi trường tạo và quản lý việc xử lý song song
    - Hệ điều hành
    - Môi trường giao tiếp (PVM, MPI, ...)
    - Thư viện truyền thông điệp

- Để viết chương trình song song:
  - Phân ly thuật giải hoặc dữ liệu thành các phần riêng.
  - Phân bổ những phần công việc này như các task làm việc đồng thời trên các bộ xử lý.
  - Hợp tác và trao đổi giữa các bộ xử lý.

- Để hiện thực một chương trình song song, có thể sử dụng, hoặc:
  - Một ngôn ngữ song song chuyên biệt
  - Ngôn ngữ cấp cao với các cú pháp và từ khóa liên quan đến song song.
  - Ngôn ngữ cấp cao thông dụng với các hàm thư viện liên quan đến song song.

- Theo ba cách tiếp cận trên:
  - *occam* là ngôn ngữ lập trình song song chuyên biệt, dùng trên máy gọi là *transputer*
  - Một vài ngôn ngữ xử lý song song cấp cao như **CC++** (Compositional **C++**); FORTRAN M (Argonne National Laboratory), FORTRAN 90, HPF (High Performance FORTRAN, extended from FORTRAN 90), ...

  - Sử dụng những hàm thư viện về truyền thông điệp (chẳng hạn PVM và MPI) với ngôn ngữ C/C++, FORTRAN.

Hệ thống truyền thông điệp

- Hệ thống truyền thông điệp tạo ra môi trường cho phép người lập trình cài đặt chương trình tính toán song song.
- Môi trường cài đặt này có thể hoạt động trên nhiều chủng loại máy tính khác nhau (máy PC với bộ xử lý thuộc họ Intel, các kiến trúc Sparc, Alpha, HP, ...)

- Hầu hết các ứng dụng song song đều được cài đặt trên hệ điều hành UNIX như Solaris, AIX, Linux, ...
- Chính vì vậy, các máy với những hệ điều hành này đều có thể tạo ra hệ thống truyền thông điệp.
- Có hai hệ thống chuyển thông điệp phổ biến:
  - Hệ thống PVM (*Parallel Virtual Machine*)
  - Môi trường MPI *(Message-Passing Interface)*

Parallel Virtual Machine

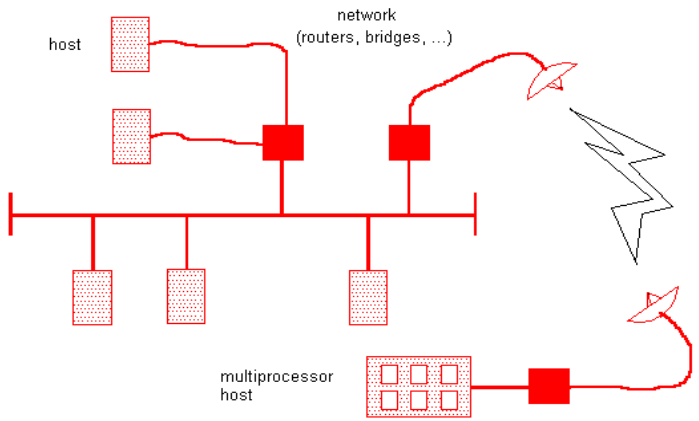
- PVM – Parallel Virtual Machine (*máy ảo song song*) được dùng để chỉ một máy tính logic có bộ nhớ phân tán
- PVM cung cấp các thủ tục để khởi tạo các task trên máy ảo (*virtual machine*) và cho phép các task này trao đổi với nhau.
- *Task* trên hệ thống PVM được coi là một đơn vị tính toán, có ý nghĩa như một *UNIX process*.

Kiến trúc PVM

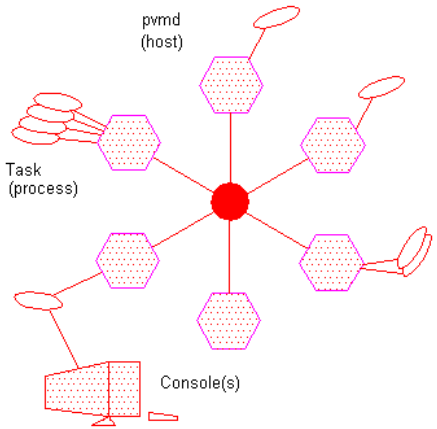
- Ứng dụng trên PVM có thể viết bằng ngôn ngữ C/ C++ hoặc FORTRAN 77.
- Thuật giải có thể song song hóa bằng cách dùng các cấu trúc truyền thông điệp với các hàm thư viện như **`pvm_send(),pvm_recv()`** để gởi và nhận dữ liệu.

Các hàm nầy là một bộ phận thứ hai của PVM, bên cạnh *pvmd* như là một PVM *daemon process*

Kiến trúc vật lý của PVM



Mô hình logic của PVM



Cài đặt PVM

- Có nhiều tập tin dưới dạng nén khác nhau của PVM, chúng ta có thể sử dụng tập tin **pvm3.4.6.tgz (**http://www.netlib.org/pvm3/pvm3.4.6.tgz**)**
- Đây là bản mới nhất được cập nhập vào 02/02/2009,
- Hiện nay PVM đã ổn định nên không có phiên bản mới hơn.

PVM trên Internet

- Giới thiệu về pvm3 và tải các tiện ích và tập tin pvm3.4.6.tgz để sử dụng (http://www.netlib.org/pvm3/)
- C++ Interface to PVM: http://www.informatik.uni-stuttgart.de/ipvr/bv/cppvm/
- Về XPVM: http://www.netlib.org/utk/icl/xpvm/xpvm.html

- Thông thường PVM được cài đặt để nhiều người cùng sử dụng, hoặc cho nhiều đề án khác nhau của cùng một người,
- Trong cả hai trường hợp PVM đều có mục tiêu sử dụng chung.

- Giả sử cần install PVM trên user có tên gọi lang của hệ điều hành LINUX
- Các bước sau đây cần tiến hành (giả sử tập tin **pvm3.4.6.tgz** đã có trên **$HOME**

```
$ tar xvfz pvm3.4.6.tgz
```

- Khi đó trên **$HOME** có thư mục **pvm3**

- Giả sử đang sử dụng Bash shell, cần đặt đường dẫn và biến môi trường sau đây trong tập tin **$HOME/.bash_profile**

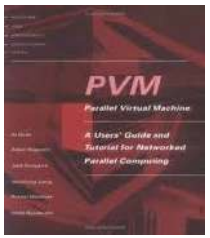
```
export PVM_ROOT=$HOME/pvm3
export PVM_RSH=/usr/bin/ssh
export PVM_ARCH=`$PVM_ROOT/lib/pvmgetarch`
export PVM_DPATH=$PVM_ROOT/lib/pvmd
PATH=$PATH:$PVM_ROOT/lib:$PVM_ROOT/bin/$PVM_ARCH
```


- Sau khi biến môi trường và đường dẫn đã được kích hoạt, biên dịch PVM bằng các lệnh để cài đặt PVM lên máy:

```
cd $PVM_ROOT
make
```

- Nếu thành công, gọi pvm

```
pvm
pvm>
```

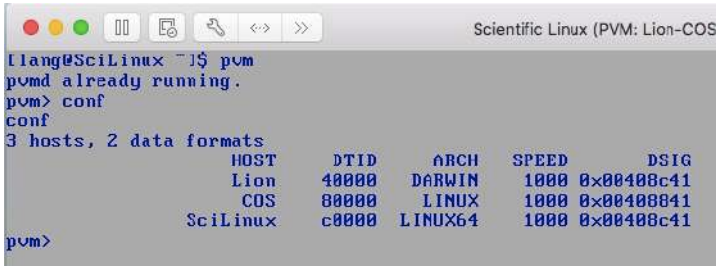


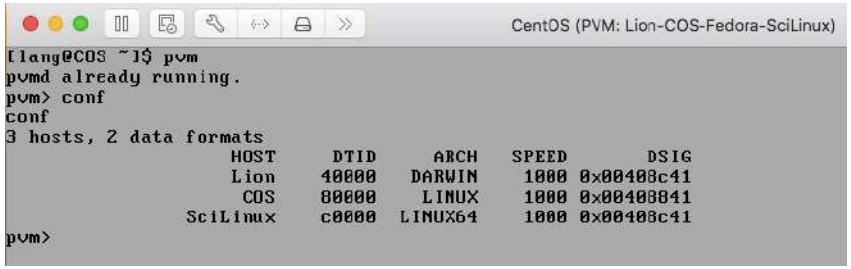
Sử dụng PVM

- Trước hết phải kích hoạt để PVM làm việc

```
$ pvm
pvm>
```

```
Lion:pvm lang$ pvm
pvm> conf
conf
1 host, 1 data format
                     HOST     DTID     ARCH   SPEED       DSIG
                     Lion    40000   DARWIN    1000 0x08408c41
pvm> add COS
add COS
lang@cos's password:
1 successful
                     HOST     DTID
                      COS    80000
pvm> add SciLinux
add SciLinux
1 successful
                     HOST     DTID
                  SciLinux    c0000
pvm> conf
conf
3 hosts, 2 data formats
                     HOST     DTID     ARCH   SPEED       DSIG
                     Lion    40000   DARWIN    1000 0x08408c41
                      COS    80000    LINUX    1000 0x08408841
                  SciLinux    c0000  LINUX64    1000 0x08408c41
pvm>
```

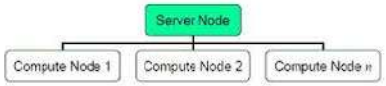




- Tại dấu nhắc **pvm**, có thể thực hiện các lệnh như

```
pvm> add
pvm> delete
pvm> conf
```

- Để thêm, xóa, coi cấu hình của hệ thống máy ảo.



- Có thể quay về dấu nhắc UNIX để làm việc bằng lệnh **quit**

```
pvm>quit
$
```

- Hoặc để thoát ra khỏi, dùng lệnh

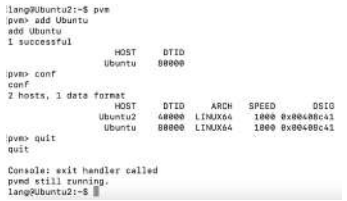
```
pvm>halt
$
```

Lưu ý

- PVM daemon hoạt động theo cơ chế remote shell, vì vậy cần phải có hostname của các máy PVM trong tập tin **/etc/hosts.equiv** (hoặc trong các tập tin **$HOME/.rhosts**) khi dùng rsh.
- Với ssh: không cần
- Các chương trình thi hành bằng lệnh **pvm_spawn()** phải được chỉ đường dẫn tuyệt đối, hoặc được lưu trữ trong thư mục **$PVM_ROOT/bin/$PVM_ARCH**.

Ví dụ trong PVM

- Môi trường với 2 máy Ubuntu như hình
- Thi hành chương trình hello.c trong **$PVM_ROOT/examples**.
- Thực hiện các lệnh
  - **cd $HOME/folder**
  - **aimk hello hello_other**
  - **hello**



- Cũng có thể dùng lệnh make của hệ điều hành để biên dịch chương trình.
- Chẳng hạn

  make hello hello_other

- Tuy nhiên, tập tin makefile hiệu chỉnh lại đôi chút như sau:

Để thực hiện, có makefile như sau:

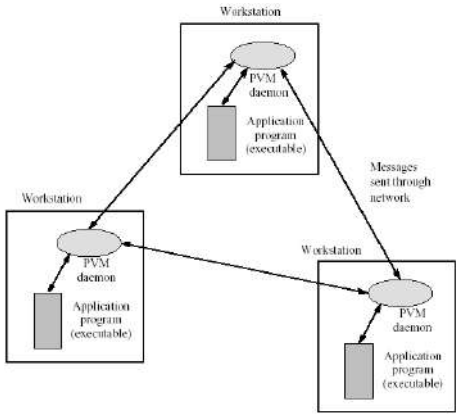
```
SDIR =  .
XDIR =  $(PVM_ROOT)/bin/$(PVM_ARCH)
INC  =  -I$(PVM_ROOT)/include
LIB  =  -L$(PVM_ROOT)/lib/$(PVM_ARCH) -lpvm3
CC   =  gcc
hello: $(SDIR)/hello.c
  $(CC) -o $@ $(SDIR)/$@.c $(INC) $(LIB)
  mv $@ $(XDIR)
hello_other: $(SDIR)/hello_other.c
  $(CC) -o $@ $(SDIR)/$@.c $(INC) $(LIB)
  mv $@ $(XDIR)
```

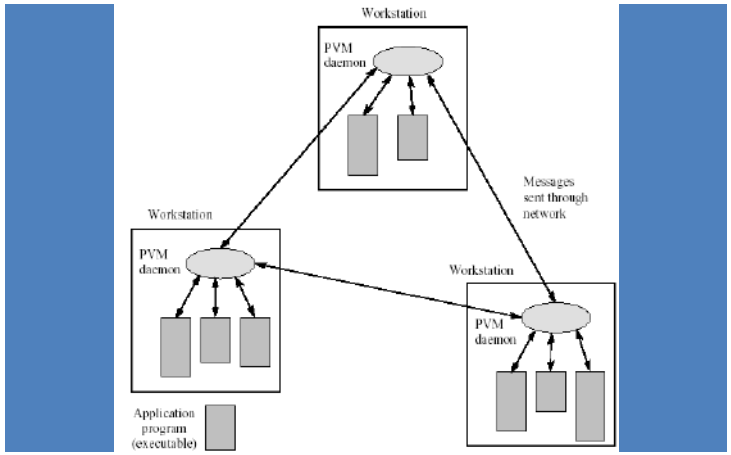


```
pvm> quit
quit

Console: exit handler called
pvmd still running.
Lion:pvm lang$ hello
I'm t40002
From t80001:---> hello, world from COS
Lion:pvm lang$ hello
I'm t40003
From tc0001:---> hello, world from SciLinux
Lion:pvm lang$ hello
I'm t40004
From t40005:---> hello, world from Lion
Lion:pvm lang$ hello
I'm t40006
From t80002:---> hello, world from COS
Lion:pvm lang$
```

Lập trình với PVM

- Thực chất của việc Parallel Programming trên distributed system là truyền thông điệp.
- Để chương trình thi hành trên virtual machine (VM), cần có pvmd (PVM daemon) hoạt động trên các node (các workstation) của VM này.
- Các application program (Executable) được nạp vào các workstation



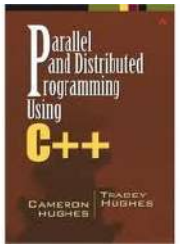


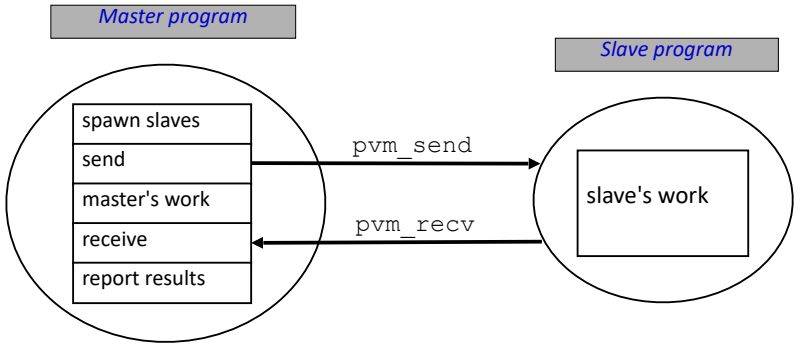
Mô hình lập trình

- Trong PVM có 2 mô hình lập trình thông dụng:
  - Mô hình master-slave
  - Mô hình task-to-task

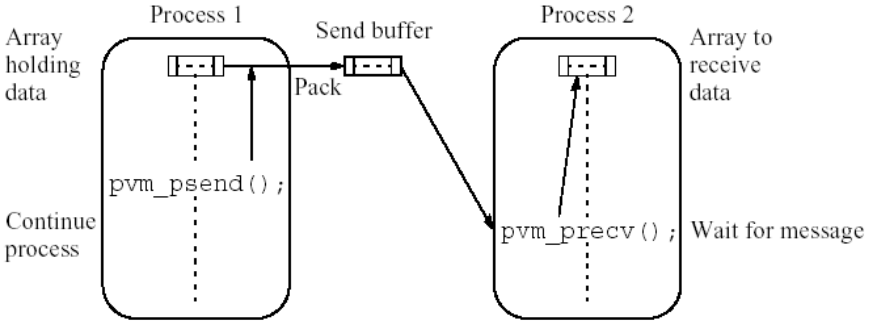
a) Mô hình master- slave

- Trong mô hình thiết kế *master-slave*, một máy chủ điều khiển sự hoạt động của các máy còn lại như là các *slave* thông qua các task ID (*identify*)





Hàm gửi và nhận cơ bản



- Trong mô hình master-slave, chương trình master phát sinh và điều khiển một vài chương trình slave để thực hiện các tính toán.
- PVM không có một hạn chế gì trên mô hình nầy, bất kỳ một task PVM nào cũng có thể đóng vai trò của một master.

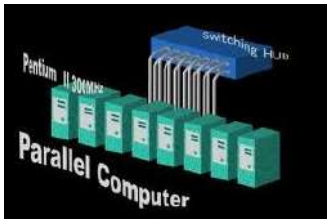
- Mô hình master-slave là một mô hình thuận tiện để minh họa thuật giải.
- Chương trình master gọi **pvm_spawn()** để thi hành một số các chương trình slave trên các máy khác trong hệ thống PVM.


```
int numt = pvm_spawn(char *task, char **argv, int
flag, char *where, int ntask, int *tids)
```

- task: tên tập tin thi hành có trên host mà nó thi hành. Tên này có thể là tập tin trong vùng tìm kiếm của PVM, hoặc chỉ các absolute path.
  - By default, PVM looks for executable in **$PVM_ROOT/bin/$PVM_ARCH/**


```
int numt = pvm_spawn(char *task, char **argv, int
flag, char *where, int ntask, int *tids)
```

- argv: các argument của tập tin thi hành. Nếu tập tin thi hành không có argument, tham số này có giá trị là **NULL**.




```
int numt = pvm_spawn(char *task, char **argv,
int flag, char *where, int ntask, int *tids)
```

- flag: có thể là tổng của các giá trị:
  - PvmTaskDefault 0: PVM can choose any machine to start task
  - PvmTaskHost 1: “where” specifies a particular host
  - PvmTaskArch 2: “where” specifies a type of architecture
  - PvmTaskDebug 4: Start up processes under debugger
  - PvmTaskTrace 8: Processes will generate PVM trace data
  - PvmMppFront 16: Start process on MPP front-end
  - PvmHostCompl 32: Use complement host set

```
int numt = pvm_spawn(char *task, char **argv, int
flag, char *where, int ntask, int *tids)
```

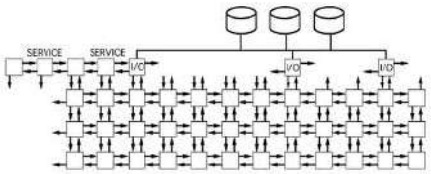
- where: depending on value of flag, can specify a hostname or an architecture.
  - This argument can also be used to spcify a custom working directory for each given spawn command. For example,

"mar.lang.ac.vn:/home/lang/project"


```
int numt = pvm_spawn(char *task, char **argv, int
flag, char *where, int ntask, int *tids)
```


- ntask: number of copies of executable to start up
- tids: integer array of length at least ntask, storing the TID of PVM processes started by this pvm_spawn call

- numt: storing the actual number of task started.
  - Values less than zero indicates system error
  - A positive value less than ntask indicates a partial failure (In this case the user should check the tids for the error codes)



Ví dụ

- Các slave gửi host name ở đó slave được kích hoạt về cho master, đồng thời gửi cả task ID.
- Sau đó master xuất ra màn hình những gì nhận được tứ các slave này.

- Để làm điều này, viết 2 chương trình bằng ngôn ngữ C:
  - Chương trình master làm nhiệm vụ gọi hàm pvm_spawn() và nhận kết quả tứ các slave
  - Chương trình slave gửi các giá trị về cho master

Master program (lmaster.c)

```
/* spawn P lslave program */
pvm_spawn("lslave",(char**)0,0,"",P,tid);

/* receive buf from slaves */
for ( i = 0; i <P; i++ ){
  pvm_recv(-1,-1);
  pvm_upkstr(buf);
  printf( "%x %s\n", tid[i], buf );
}
```


Slave program (lslave.c)

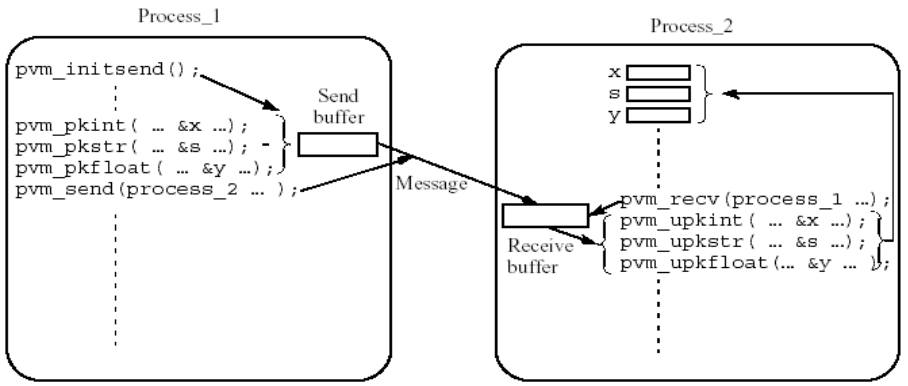
```
/* get hostname of slave */
gethostname( buf, 10 );

/* send buf to master */
pvm_initsend( 0 );
pvm_pkstr( buf );
pvm_send( pvm_parent(), 111 );
```


- Thay vì dùng make, có thể dịch với các lệnh:

gcc -o lmaster lmaster.c -I$PVM_ROOT/include -L$PVM_ROOT / lib/$PVM_ARCH -lpvm3

gcc -o lslave lslave.c -I$PVM_ROOT/include -L$PVM_ROOT /lib/ $PVM_ARCH -lpvm3

mv lslave $PVM_ROOT/bin/$PVM_ARCH



Chuẩn bị máy

- See pvm_hello.c, pvm_hello_other.c, makefile
- Minh họa trên hệ thống cluster gồm 3 máy:
  - Mac OS X: Lion (làm master)
  - CentOS Linux: CentOS
  - Scientific Linux: SciLinux

- Trên các máy Linux có thể share folder /home/lang/pvm3 của máy CentOS. Để làm việc này cần:
  - tạo file /etc/exports trên CentOS với nội dung

    /home/lang/pvm3 SciLinux (rw,sync) OtherLinux (rw,sync)
- Trên SciLinux và các máy Linux khác cần:
  - mount folder của CentOS lên folder có trên máy của nó bằng lệnh:

    mount –t nfs CentOS:/home/lang/pvm3 /home/lang/pvm3

Thực hiện

- Dùng Lion làm máy master (dùng Hệ điều hành Mac OS X)
  - dịch pvm_hello.c và pvm_hello_other.c ra ngôn ngữ máy bằng lệnh
    - make pvm_hello pvm_hello_other
  - move tập tin ngôn ngữ máy pvm_hello_other đến folder $HOME/pvm3/bin/DARWIN
    - mv pvm_hello_other $HOME/pvm3/bin/DARWIN

- copy tập tin pvm_hello_other.c vào folder /home/lang/work của máy CentOS một trong những lệnh sau:
  - tar cf – pvm_hello_other.c | rsh CentOS "cd /home/lang/work; tar xvf –"
  - tar cf – pvm_hello_other.c | ssh CentOS "cd /home/lang/work; tar xvf –"
  - scp pvm_hello_other.c CentOS :/home/lang/work
- Dịch pvm_hello_other.c ra ngôn ngữ máy và move sang folder /home/lang/pvm3/bin/LINUX64
  - dùng lệnh make với makefile có sẵn

Hoặc đơn giản hơn

- Giả sử có 2 máy Ubuntu và Ubuntu2 dùng Linux., máy ảo PVM được tạo ra từ 2 máy này

- Trên cả 2 máy cùng share một folder có tên là **/media/psf/Example/PVM**.

- Khi đó mọi sự edit tập tin trên máy ở tại folder nêu trên đều dùng được cho cả 2 máy

- Để minh hoạ, với make file như sau

```
XDIR    =       $(PVM_ROOT)/bin/$(PVM_ARCH)
INC     =       -I$(PVM_ROOT)/include
LIB     =       -L$(PVM_ROOT)/lib/$(PVM_ARCH) -lpvm3

default:        pvm_hello pvm_hello_other

pvm_hello:      ./pvm_hello.c
        gcc -o $@ ./$@.c $(INC) $(LIB)
        mv $@ $(XDIR)

pvm_hello_other:        ./pvm_hello_other.c
        gcc -o $@ ./$@.c $(INC) $(LIB)
        mv $@ $(XDIR)
```

- Biên dịch 2 file pvm_hello.c và pvm_hello_other.c bằng lệnh **make**

  $.> make

- Trong đó, pvm_hello.c

```c
//  pvm_hello.c
//  Created by Tran Van Lang

#include <stdio.h>
#include <pvm3.h>
#include <stdlib.h>

int main( int argc, char** argv ){
  int cc, tid[100], i, P;
  char buf[100];

  if ( argc < 2 ){
    printf( "\nSynstax: pvm_hello ntasks\n" );
    return 0;
  }

  printf("I'm t%x\n", pvm_mytid());

  P = atoi( argv[1] );
  cc = pvm_spawn( (char*)"pvm_hello_other", (char**)0, 0, (char*)"", P, tid );

  if (cc == P) {
    for ( i = 0; i < P; i++ ){
      pvm_recv( -1, -1 );
      pvm_upkstr( buf );
      printf("Task ID t%x: %s\n", tid[i], buf );
    }
  }
  else
    printf("can't start pvm_hello_other\n");

  pvm_exit();
  return 1;
}
```

- Và pvm_hello_other.c

```c
// pvm_heello_other.c
// by Tran Van Lang

#include <pvm3.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

int main(){
    int ptid, tid;
    char buf[200], t[30];

    ptid = pvm_parent();
    tid = pvm_mytid();
    strcpy(buf, "Hello, world from ");
    gethostname( buf+strlen(buf), 64 );
    sprintf( t, ", Task ID: t%x", tid );
    strcat( buf, t );

    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, 1);

    pvm_exit();
    return 1;
}
```

Chứa trong **/Users/lang/Documents/Works/Training/Parallel Computing/Example/PVM**

b) Mô hình task-to-task

- Bên cạnh mô hình *master-slave*, còn có mô hình SPMD - *single program multiple data*,
- Mô hình nầy chỉ có một chương trình, không có chương trình master điều khiển các tính toán.

- Trong chương trình này, có hai phần:
  - Phần để spawn các task khác
  - Phần cho các task thực hiện
- Chương trình nầy trước tiên phải khởi **pvm_parent()** để kiểm tra, từ đó biết được bản thứ nhất khởi động chưa.

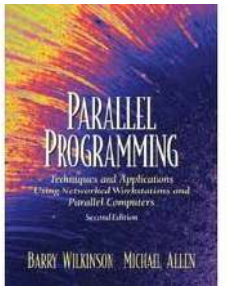
- Sau đó sinh ra các bản sao khác của nó và chuyển vào một mảng *task ID* (*identify*).
- Vào thời điểm nầy mỗi bản sao hoạt động như nhau trong phần dữ liệu của nó và cùng hợp tác với các tiến trình khác trong hệ thống máy ảo.

Cách thức viết chương trình

- **Tính tổng của dãy số dùng mô hình master - slave**
- **Master thực hiện** các công việc sau:
  - **pvm_spawn()**: kích hoạt các task
  - **pvm_initsend()**, **pvm_pk*()**, **pvm_send()**: gửi dữ liệu là các khối tương ứng đến slave.

- Tính toán phần dư còn lại
- **pvm_recv()**, **pvm_upk*()**: nhận lại các tổng con
- Tổng hợp các tổng con.
- Master program có các phần chính như sau:


```
aP = pvm_spawn( "pvm_sum_slave", (char**)0,
   PvmTaskDefault, "", P-1, tid );

for ( p = 0; p < aP; p++ ){
   pvm_initsend( PvmDataDefault );
   pvm_pkint( &r, 1, 1 );
   pvm_send( tid[p], 11111 );
   pvm_initsend( PvmDataDefault );
   pvm_pkdouble( &a[p*r], r, 1 );
   pvm_send( tid[p], 22222 );
}
for ( i = aP*r; i < N; i++ )
   s += a[i];

for ( p = 0; p < aP; p++ ){
   pvm_recv( tid[p], 99999 );
   pvm_upkdouble( &sp, 1, 1 );
   s += sp;
}
```



```
//**********************//
//*    pvm_sum.cc      //
//* by Tran Van Lang   //
//**********************//
#include <iostream>
#include <iomanip>
#include "pvm3.h"
#include <stdlib.h>

using namespace std;

int main( int argc, char** argv ){
    int P, p, aP, mytid;
    long N, i, r;
    double s = 0.0, sp;
    char hostname[100];

    if( argc < 3 ){
        cout << "Syntax: pvm_sum NumberofElements NumberofTasks\n";
        return 0;
    }

    N = atoi( argv[1] );
    P = atoi( argv[2] );

    r = N/P;

    double *a = new double[N];
    int *tid = new int[P];

    srand( (unsigned)time(NULL) );
    for ( i = 0; i < N; i++ )
        a[i] = rand();

    for ( i = 0; i < N; i++ )
        s += a[i];
    cout << setiosflags(ios::fixed)
    << setprecision(4);
    cout << "Sequential sum: "
    << setw(20) << s << endl;

```



```
    mytid = pvm_mytid();
    s = 0.0;
    aP = pvm_spawn( (char*)"pvm_sum_slave", (char**)0, PvmTaskDefault, (char*)"", P, tid );
    if ( aP == P ){
        for ( p = 0; p < aP; p++ ) {
            pvm_initsend( PvmDataDefault );
            pvm_pklong( &r, 1, 1 );
            pvm_send( tid[p], 11111 );

            pvm_initsend( PvmDataDefault );
            pvm_pkdouble( &a[p*r], r, 1 );
            pvm_send( tid[p], 22222 );
        }
        for ( i = P*r; i < N; i++ )
            s += a[i];
        //cout << "Slave Hosts for computing:\n";
        for ( p = 0; p < aP; p++ ) {
            pvm_recv( tid[p], 99999 );
            pvm_upkdouble( &sp, 1, 1 );
            pvm_upkstr( hostname );
            s += sp;
            cout << "Task No." << p << ": Partial Sum: "
            << sp << " from host: " << hostname << endl;
        }

        cout << "Parallel sum: "
        << setw(23) << s << endl;
    }
    else
        cout << "Error\n"

    pvm_exit();
    return 1;
}
```


- **Slave thực hiện** các công việc sau:
  - **pvm_parent()**: xác định từ đâu gửi đến
  - **pvm_recv()**, **pvm_upk*()**: nhận các phần tương ứng của dãy dữ liệu.
  - Tính tổng của phần được giao
  - **pvm_initsend()**, **pvm_pk*()**, **pvm_send()**: gửi các tổng về cho master


```
pvm_recv( ptid, 11111 );
pvm_upkint( &r, 1, 1 );

data = new double[r];

pvm_recv( ptid, 22222 );
pvm_upkdouble( data, r, 1 );

for ( i = 0; i < r; i++ )
  sp += data[i];

pvm_initsend( PvmDataDefault );
pvm_pkdouble( &sp, 1, 1 );
pvm_send( ptid, 99999 );
```

pvm_sum.cc, pvm_sum_slave.cc và makefile

```
//*********************//
//* pvm_sum_slave.cc  //
//* by Tran Van Lang  //
//*********************//
#include <pvm3.h>
#include <unistd.h>

int main(){
    int ptid = pvm_parent();
    long r, i;
    double sp = 0.0;
    char hostname[100];
    gethostname( hostname, 64 );

    pvm_recv( ptid, 11111 );
    pvm_upklong( &r, 1, 1 );

    double *data = new double[r];
    pvm_recv( ptid, 22222 );
    pvm_upkdouble( data, r, 1 );

    for ( i = 0; i < r; i++ )
        sp += data[i];

    pvm_initsend( PvmDataDefault );
    pvm_pkdouble( &sp, 1, 1 );
    pvm_pkstr( hostname );
    pvm_send( ptid, 99999 );

    pvm_exit();
    return 1;
}
```

- Make File sử dụng

```
XDIR    =       $(PVM_ROOT)/bin/$(PVM_ARCH)
INC     =       -I$(PVM_ROOT)/include
LIB     =       -L$(PVM_ROOT)/lib/$(PVM_ARCH) -lpvm3

default:        pvm_sum pvm_sum_slave
```

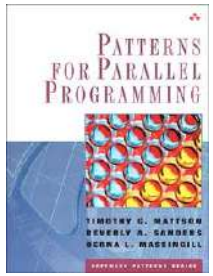
```
CC      =       g++
pvm_sum: ./pvm_sum.cc
        $(CC) -o $@ ./$@.cc $(INC) $(LIB)
        mv $@ $(XDIR)

pvm_sum_slave: ./pvm_sum_slave.cc
        $(CC) -o $@ ./$@.cc $(INC) $(LIB)
        mv $@ $(XDIR)
```


- **Tính tổng của dãy số dùng mô hình task – to – task.**
- Trong trường hợp này, toàn bộ phần master và phần slave được đưa vào trong cùng một tập tin.
- Để phân biệt đâu là công việc của master, đâu là của slave, căn cứ vào hàm **pvm_parent()**.

- Nếu **pvm_parent()** trả về giá trị **PvmNoParent**, điều đó có nghĩa instance đang là master, ngược lại, đây là slave.

```
if (pvm_parent() == PvmNoParent)
   //Master's statements
else
   //Slave's statements
pvm_exit();
```

See pvm_sum_spmd.cc




```
//*********************//
//* pvm_sum_spmd.cc   //
//* by Tran Van Lang  //
//*********************//
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <unistd.h>
#include "pvm3.h"

using namespace std;

int main( int argc, char** argv ){
    int ptid = pvm_parent();

    if ( ptid == PvmNoParent ){ //M
        int P, aP, p;
        long N, i, r;
        double sp, s = 0.0;
        char hostname[100];

        if ( argc <3 ){
            cout << "Syntax: pvm_sum_spmd  NumberofElements NumberofTasks\n";
            return 0;
        }

        N = atoi( argv[1] );
        P = atoi( argv[2] );

        r = N/P;
        double *a = new double[N];
        int *tid = new int[P];

        srand( (unsigned)time(NULL) );
        for ( i = 0; i < N; i++ )
            a[i] = rand();

        for ( i = 0; i < N; i++ )
            s += a[i];
        cout << setiosflags(ios::fixed)
        << setprecision(4);
        cout << "Sequential sum: "
        << setw(20) << s << endl;

        s = 0.0;
        aP = pvm_spawn( (char*)"pvm_sum_spmd", argv,
        PvmTaskDefault, (char*)"", P-1, tid );
```

```
        if ( aP == P-1 ){
            for ( p = 0; p < aP; p++ ){
                pvm_initsend( PvmDataDefault );
                pvm_pklong( &r, 1, 1 );
                pvm_send( tid[p], 11111 );

                pvm_initsend( PvmDataDefault );
                pvm_pkdouble( &a[p*r], r, 1 );
                pvm_send( tid[p], 22222 );
            }
            for ( i = (P-1)*r; i < N; i++ )
                s += a[i];

            //cout << "Slave hosts for computing:
            for ( p = 0; p < aP; p++ ){
                pvm_recv( tid[p], 99999 );
                pvm_upkdouble( &sp, 1, 1 );
                pvm_upkstr( hostname );
                s += sp;
                cout << "Task No." << p << ": Partial Sum: "
                << sp << " from host: " << hostname << endl;
            }

            cout << "Parallel sum: "
            << setw(23) << s << endl;
        }
        else
            cout << "Can not start program\n";
    }
```

```
    else {  //Slave's work
        long r, i;
        double sp = 0.0;
        char hostname[100];
        gethostname( hostname, 64 );

        pvm_recv( ptid, 11111 );
        pvm_upklong( &r, 1, 1 );
        double *data = new double[r];
        pvm_recv( ptid, 22222 );
        pvm_upkdouble( data, r, 1 );

        for ( i = 0; i < r; i++ )
            sp += data[i];

        pvm_initsend( PvmDataDefault );
        pvm_pkdouble( &sp, 1, 1 );
        pvm_pkstr( hostname );
        pvm_send( ptid, 99999 );
    }
    pvm_exit();
    return 1;
}
```

Tính tích ma trận với vector dùng mô hình task-to-task

- Tích AX = Y, với A là ma trận n dòng n cột n, X có n thành phần,
- Sử dụng P task, giả sử r = n/P là số nguyên.
- Phân A thành P ma trận, mỗi ma trận có r dòng n cột.
- Tiến trình thứ nhất phân phối các ma trận con của A và phát tán vector X đến P-1 tiến trình còn lại. Và tính tích của ma trận con với vector X.

- Các tiến trình từ 2 đến P thực hiện tính tích các ma trận con với vector X để được r thành phần.
- Sau đó tiến trình thứ nhất sẽ thu thập tất cả các nhóm r thành phần từ P-1 tiến trình còn lại để được vector Y.

- See **mat.cc**

Tìm kiếm trên tập tin

- Giả sử đã có tập tin data.data trên các host
- Mỗi host có chương trình lần lượt đọc từng phần tử có trong tập tin đó,
- Nếu tìm thấy gửi thông báo về cho host điều khiển chung
- Khi host điều khiển nhận được thông báo tìm thấy được sẽ kết thúc việc tìm kiếm

Tổ chức dạng master - slave

- Slave:
  - Xác định master
    - int parent = pvm_parent();
  - nhận giá trị x để tìm
    - pvm_recv(parent, 99999);
    - pvm_upkfloat(&x, 1, 1);

  - lần lượt đọc dữ liệu và so sánh với x
    - while (!feof(fp)){
    - fscanf(fp, "%f", &data);if(x == data){
    - find = 1;
    - break;
    - }
    - }

  - gửi thông báo host nào tìm thấy về cho master
    - gethostname(buff, 64);
    - pvm_initsend(PvmDataDefault);
    - pvm_pkint(&find, 1, 1);
    - pvm_pkstr(buff);
    - pvm_send(parent, 11111);
  - và kết thúc sự hoạt động
    - pvm_exit(1);

- Master
  - Kích hoạt các slave
    - pvm_spawn("datafromfile", (char**)0, PvmTaskDefault, "", NTASKS, tids)
  - gửi giá trị cần tìm x cho các slave
    - pvm_initsend(PvmDataDefault);
    - pvm_pkfloat(&x, 1, 1);
    - pvm_mcast(tids, NTASKS, 99999);

– nhận thông báo kết quả từ các slave
  - i = 0;
  - do {
  - pvm_recv(tids[i], 11111);
  - pvm_upkint(&find, 1, 1);
  - pvm_upkstr(buff);
  - i++;
  - } while(i < NTASKS && find != 1);
- Xem pvm_fromfile.c, pvm_fromfile_slave.c

Một vài ví dụ khác

- **Đo thời gian di chuyển của dữ liệu**
  – **Master** kích hoạt một host cụ thể
    - pvm_spawn((char*)"pingpong_slave", NULL, PvmTaskHost, argv[1], 1, &tid)
  – Lấy thời gian
    - gettimeofday(&t0, NULL)
  – Gửi dữ liệu cho **Slave**
    - pvm_initsend(PvmDataDefault);
    - pvm_pkint(&data, 1, 1);
    - pvm_send(tid, 5555);

– **Slave** nhận và gửi trả lại
  - pvm_recv(parent, 5555);
  - pvm_upkint(&data, 1, 1);
  - pvm_initsend(PvmDataDefault);
  - pvm_pkint(&worker, 1,1);
  - pvm_send(parent, 7777);
– **Master** nhận và đánh dấu thời gian
  - pvm_recv(tid, 7777);
  - pvm_upkint(&a, 1, 1);
  - gettimeofday(&t1, NULL);
- Master and Slave Program

Một vài ví dụ khác

- **Tính số π** theo công thức (thuật toán Spigot - http://en.wikipedia.org/wiki/Pi)

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right)$$

- **Master** kích hoạt các slave
  - pvm_spawn((char*)"pislave", (char**)0, PvmTaskDefault, (char*)"", P-1, tid)
- Sau đó gửi những dữ liệu cần thiết như r = N/P, thứ tự tiến trình dùng làm chỉ số trong tổng
  - for (p = 0; p < aP; p++){
  - pvm_initsend(PvmDataDefault);
  - pvm_pkint(&r, 1, 1);
  - pvm_pkint(&p, 1, 1);
  - pvm_send(tid[p], 22222);
  - }

- Trong khi Slave nhận, **Master** thực hiện những tính toán của mình:
  - for (i = (P-1)*r; i < N; i++){
  - s1 = 4.0/(8*i + 1);
  - s2 = 2.0/(8*i + 4);
  - s3 = 1.0/(8*i + 5);
  - s4 = 1.0/(8*i + 6);
  - s += (s1-s2-s3-s4)/powl(16.0,i);
  - }

- **Slave** nhận những dữ liệu Master gửi cho
  - pvm_recv(ptid, 22222);
  - pvm_upkint(&r, 1, 1);
  - pvm_upkint(&p, 1, 1);

- Rồi thực hiện các tính toán giống như Master
  - for(i = p*r; i < (p+1)*r; i++){
  - s1 = 4.0/(8*i + 1);
  - s2 = 2.0/(8*i + 4);
  - s3 = 1.0/(8*i + 5);
  - s4 = 1.0/(8*i + 6);
  - sp += (s1-s2-s3-s4)/powl(16.0,i);
  - }

- Sau đó gửi về cho Master
  - pvm_initsend(PvmDataDefault);
  - pvm_pkdouble(&sp, 1, 1);
  - pvm_send(ptid, 99999);
- Master nhận và tính dồn kết quả để xuất ra
  - for (p = 0; p < aP; p++){
  - pvm_recv(tid[p], 99999);
  - pvm_upkdouble(&sp, 1, 1);
  - s += sp;
  - } // Master and Slave program

Kết hợp PVM với OpenMP

- Trên một hệ thống mạng có những host có năng lực mạnh. Khi đó có thể xây dựng hệ thống để tận dụng được năng lực của hệ thống máy bên dưới.
- Khi đó, trên các host sử dụng kiến trúc Shared-Memory bởi OpenMP; giữa các máy trên hệ thống mạng sử dụng kiến trúc Distributed-Memory

- Ví dụ, tính tổng của N số nguyên tự nhiên đầu tiên, trong đó N là một số lớn.
- Sử dụng hệ thống tính toán song song gồm P host
- Khi đó, các host thứ i (i=0,..., P-1) đảm trách tính toán r = N/P số liệu từ ir đến (i+1)r-1
- Mỗi host dùng OpenMP để tận dụng năng lực các processor của để tính tổng theo cách song song.

- Chương trình phía master, có các lệnh xử lý để gửi dữ liệu là vùng tính toán đến các slave như sau:

```
for ( i = 0; i < p; i++ ){
  r0 = i*r;
  r1 = (i+1)*r-1;
  pvm_initsend( PvmDataDefault );
  pvm_pkint( &r0, 1, 1 );
  pvm_pkint( &r1, 1, 1 );
  pvm_send( tid[i], 11111 );
}
```

- Sau khi các slave nhận dữ liệu là vùng tính toán tương ứng, sẽ tạo ra tổng riêng S_p của mình rồi gửi về cho master để tính tổng S.

```
S = 0.0;
for ( i = 0; i < p; i++ ){
  pvm_recv( tid[i], 22222 );
  pvm_upkdouble( &sp, 1, 1 );
  S += sp;
}
S += n;
```


- Phía các slave nhận dữ liệu là điểm đầu r_0 và điểm cuối r_1 cần tính toán từ master (parent).

```
pid = pvm_parent();
pvm_recv( pid, 11111 );
pvm_upkint( &r0, 1, 1 );
pvm_upkint( &r1, 1, 1 );
```


- Sau khi có dữ liệu, dùng OpenMP để tính tổng riêng S_p theo cách song song tận dụng năng lực của host với kiến trúc shared-memory.

```
  sp = 0.0;
#pragma omp parallel for reduction(+:sp)
  for ( i = r0; i <= r1; i++ )
    sp += i;
```


- Sau khi có kết quả, tổng riêng Sp được slave gửi về cho master

```
pvm_initsend( PvmDataDefault );
pvm_pkdouble( &sp, 1, 1 );
pvm_send( pid, 22222 );
```

Một số ví dụ

1. Tìm số điện thoại của một người nào đó trên cơ sở dữ liệu đã lưu trữ
2. Gửi thông báo đồng thời đến nhiều máy tính khác nhau trên mạng
3. Song song hóa thuật toán Smith-Waterman tìm sự tương đồng cục bộ giữa 2 trình tự DNA
4. Song song hóa thuật toán Needleman-Wunsch tìm sự tương đồng toàn cục giữa 2 trình tự DNA
5. Song song hóa thuật toán Dijsktra trong việc tìm đường đi ngắn nhất

6. Song song hóa thuật toán Floyd tìm đường đi ngắn nhất.
7. Song song hóa thuật toán tính tổng dãy số theo thuật toán chia và chế ngự (devide and conquer)
8. Thuật toán song song trong việc tạo dãy tăng từ 2 dãy tăng cho trước (thuật toán merge).
9. Dùng thuật toán merge, sort tuần tự và cây nhị phân để song song hóa thuật toán merge-sort.

Using vi

- yy copy
- p paste
- u undo
- x delete current character
- :e [file] edit file
- :w [file] save file
- set number set line number
- set nonumber turn off set number
- 5co12 copying 5th line to 12nd line
- 4,8co12 copying all lines in the block 4-8 to 12nd line
- 5m12 moving 5th line to 12nd line
- 4,8m12 moving all lines in the block 4-8 to 12nd line
- 1,$s/str1/str2/g replacing str1 by str2 from line 1 to end of file