University of Liverpool

Department of Computer Science

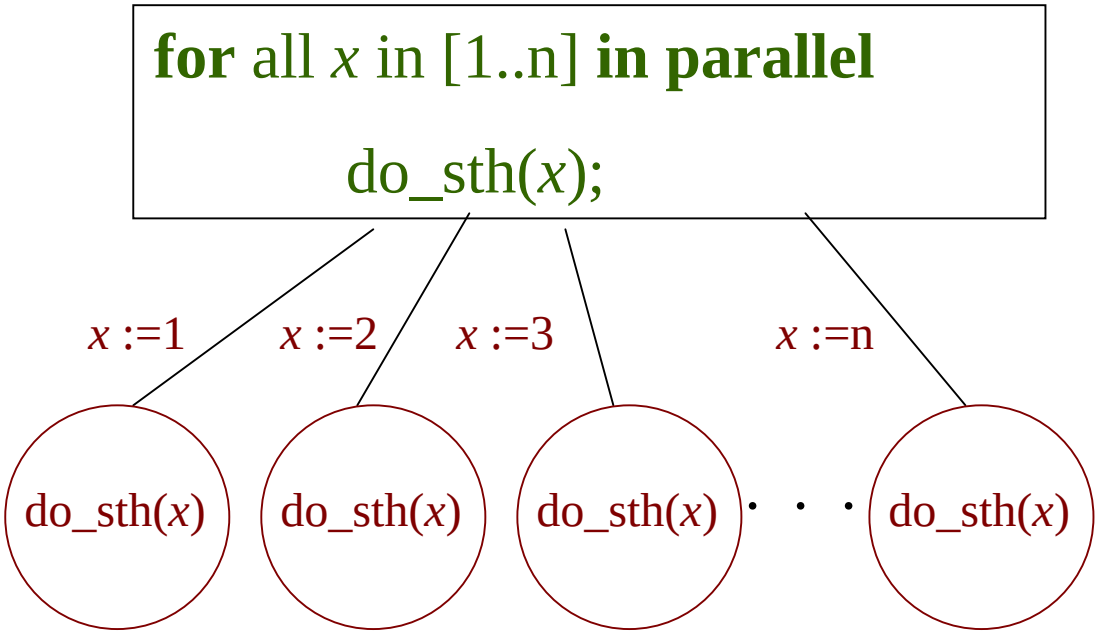
Parallel/Distributed Computing using Parallel Virtual Machine (PVM)

Prof. Wojciech Rytter

Dr. Aris Pagourtzis

Parallel Computation

breaking a large task to smaller independent tasks



What is PVM?

Message passing system that enables us to create a virtual parallel computer using a network of workstations.

Basic book:

PVM: Parallel Virtual Machine

A Users' Guide and Tutorial for Networked Parallel Computing

A.Geist, A. Benguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderman

Available in postscript:
http://www.netlib.org/pvm3/book/pvm-book.ps
and html:
http://www.netlib.org/pvm3/book/pvm-book.html

Advantages of PVM

- Ability to establish a parallel machine using ordinary (low cost) workstations.
- Ability to connect heterogeneous machines.
- Use of simple C/Fortran functions.
- Takes care of data conversion and low-level communication issues.
- Easily installable and configurable.

Disadvantages of PVM

- Programmer has to explicitly implement all parallelization details.
- Difficult debugging.

How PVM works

The programmer writes a master program and one (or more) slave program(s) in C, C++ or Fortran 77 making use of PVM library calls.

master program <----> master task
slave program <-----> group of slave tasks

The master task is started directly by the user in the ordinary way.

Slaves start concurrently by the master task by appropriate PVM call.

Slaves can also be masters of new slaves.

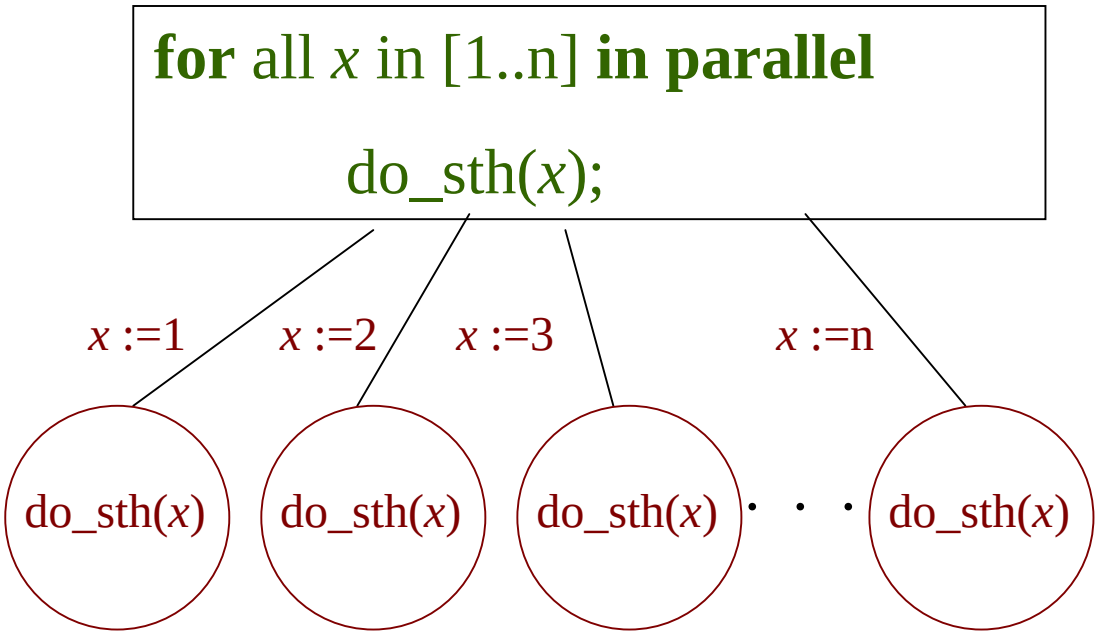
Basic operations

Each task can:

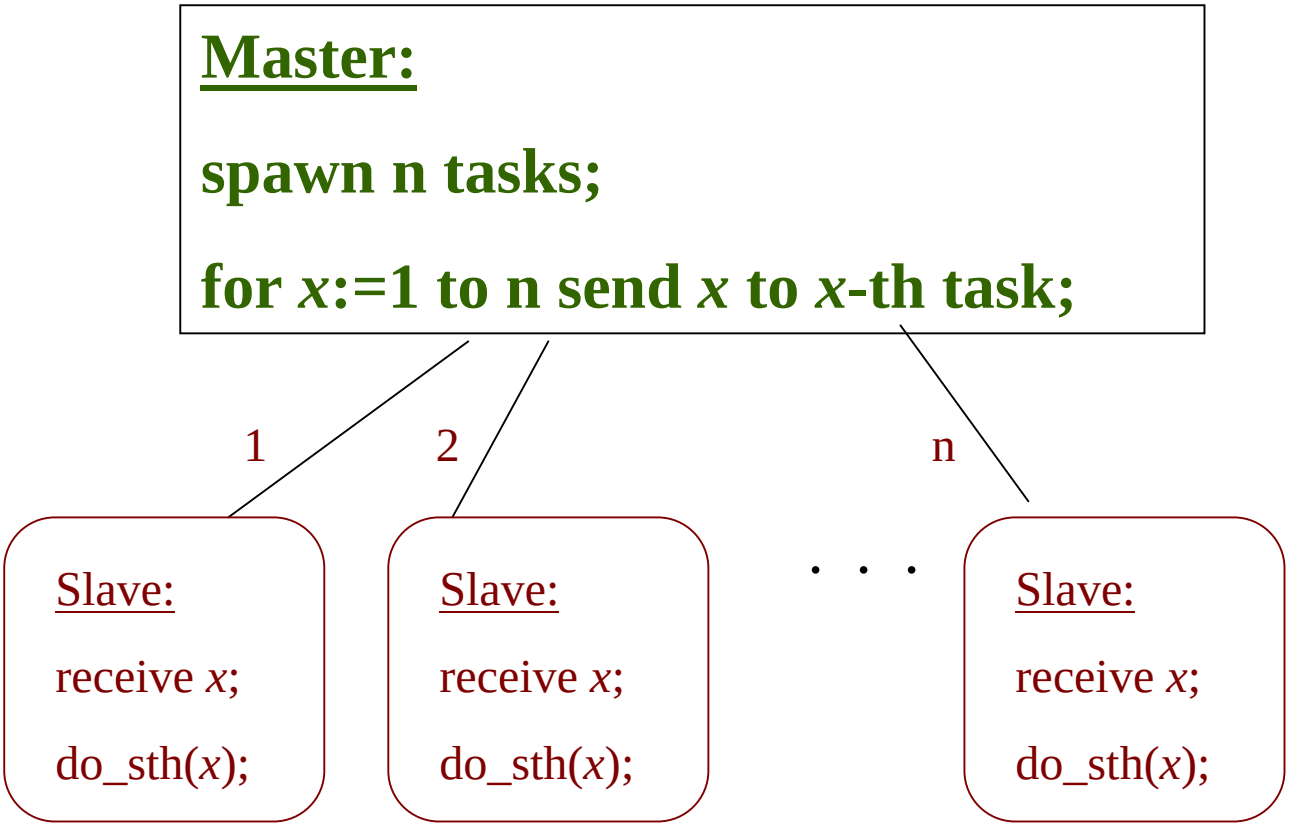
- Spawn tasks and associate them with some executable program.
- Send data to another task.
- Receive data from another task.

Each task gets a unique ID (tid) known to itself and to its parent (master).

Each task knows the ID of its parent.


for all x in [1..n] in parallel
do_sth(x);
x :=1
x :=2
x :=3
x :=n
do_sth(x)
do_sth(x)
do_sth(x)
do_sth(x)

Using PVM:


Master:
spawn n tasks;
for x:=1 to n send x to x-th task;
1
2
n
Slave:
receive x;
do_sth(x);
Slave:
receive x;
do_sth(x);
Slave:
receive x;
do_sth(x);

PVM: a simple example

Computing the sum of cubes of n numbers

Master-Slave Design:

Master

spawn n tasks;

for i:=1 to n send i to i-th task;

for i:=1 to n {receive x from i-th task;

sum:=sum+x;}

Slave

receive i from parent (master);

x:=i*i*i;

send x to parent;

```
/************* master.c **************/
/*********   SUM OF CUBES *********/
/*************************************/

#include <pvm3.h>
#define MASTERTAG 1
#define SLAVETAG 2
#define N 10
#define SLAVE_EXE
            "/users/ra/aris/Pvm/HPPA/slave_sum"

main(){

  int nt, i, x, sum, tid[N];

  nt = pvm_spawn(SLAVE_EXE,
                      (char**)0, 0, "", N, tid);
```

```
  if (nt = = N){
     for (i=0; i<N; i++){
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&i,1,1);
        pvm_send(tid[i], MASTERTAG);
     }

     sum=0;
     for (i=0; i<N; i++){
        pvm_recv(tid[i], SLAVETAG);
        pvm_upkint(&x,1,1);
        sum=sum+x;
     }
        printf("The sum of cubes is: %d \n", sum);
  }
  else
     printf("can't start all tasks\n");
  /* end if-else */
  pvm_exit();
  return 0;
}
```

```
/************** slave.c **************/
/*********   SUM OF CUBES *********/
/*************************************/

#include <pvm3.h>
#define MASTERTAG 1
#define SLAVETAG 2

main(){
  int ptid, i, x;

  ptid = pvm_parent();
  pvm_recv(ptid, MASTERTAG);
  pvm_upkint(&i,1,1);
  x=i*i*i;

  pvm_initsend(PvmDataDefault);
  pvm_pkint(&x,1,1);
  pvm_send(ptid, SLAVETAG);

  pvm_exit();
  return 0;
}
```

Spawning operation

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

Parameters

task: executable file name

argv: arguments to the executable.
If the executable needs no arguments, NULL.

flag:

`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

where:

where to start the PVM process.

A host name such as ‘‘ibm1.epm.ornl.gov”

or a PVM architecture class such as ‘‘SUN4”.

If **flag** is 0, then **where** is ignored.

ntask:

number of copies of the executable to start.

tids:

Integer array of length ntask

returning the tids of the PVM processes

started by this pvm_spawn call.

numt:

actual number of tasks started.

<0 : system error.

positive value < ntask: partial failure.

Sending operation

1. Initalization (data format specification)

int info = **pvm_initsend**(int encoding)

encoding

next message's encoding scheme

`PvmDataDefault 0:` XDR

`PvmDataRaw 1:` no encoding

`PvmDataInPlace 2:` data left in place

info

< 0 indicates an error

2. Message packing

int info = **pvm_pkint**(int *ip, int nitem, int stride)

ip Array of items. In general, address of (pointer to) the first item to be packed.

nitem The total number of items to be packed (not the number of bytes).

stride The stride to be used when packing the items. E.g., if stride = 2 then every other item will be packed.

info Integer status code returned by the routine. Values less than zero indicate an error.

pvm_pkdouble(double *dp, int nitem, int stride)
pvm_pkfloat(float *fp, int nitem, int stride)
pvm_pkbyte(char *xp, int nitem, int stride)
pvm_pkuint
pvm_pkushort pvm_pkshort
pvm_pkulong pvm_pklong
pvm_pkcplx pvm_pkdcplx

pvm_pkstr(char *sp)

3. Message sending

int info = **pvm_send**(int tid, int msgtag)

tid

task identifier of destination process.

msgtag

message tag (>= 0)

info

< 0 indicates an error

Receiving operation

1. Message receiving

int info = **pvm_recv**(int tid, int msgtag)

tid

task identifier of sending process.

msgtag

message tag (>= 0)

info

< 0 indicates an error

2. Message unpacking

int info = **pvm_upkint**(int *ip, int nitem, int stride)

ip Array of items. In general, address of (pointer to) the first item to be unpacked.

nitem The total number of items to be unpacked (not the number of bytes).

stride The stride to be used when packing the items. E.g., if stride = 2 then every other item will be unpacked.

info Integer status code returned by the routine. Values less than zero indicate an error.

pvm_pkdouble(double *dp, int nitem, int stride)
pvm_pkfloat(float *fp, int nitem, int stride)
pvm_pkbyte(char *xp, int nitem, int stride)
pvm_pkuint
pvm_pkushort pvm_pkshort
pvm_pkulong pvm_pklong
pvm_pkcplx pvm_pkdcplx

pvm_pkstr(char *sp)

PVM Setup

Usually carried out by an administration
However not difficult (see manual)

How to start PVM

From your PVM directory:

$ pvm <*hostfile*>
<*hostfile*>: a file that contains host names

In PVM console:

pvm> conf
shows configuration (active host names)

pvm> quit
quits PVM console but PVM still runs

pvm> halt
terminates PVM

How to run programs

- **Start** PVM.
- **Write** your master.c and slave.c files and **put** them in your PVM directory.
- In your PVM directory give:
  $ **aimk master slave**

  PVM supplies a Makefile.aimk file which works only with program names master.c and slave.c. However, contains instructions on how to compile programs with different file names.

  Executables are placed in a subdirectory called according to the architecture type (for HP-9000 this is HPPA)
- In your PVM directory give:
  $ **./HPPA/master**