

## **Parallel Virtual Machine**

- PVM man pages:

- <http://www.csm.ornl.gov/pvm/man/manpages.html>
- man pvm

- PVM, A Users' Guide and Tutorial for Networked Parallel Computing:

- <http://netlib2.cs.utk.edu/pvm3/book/>

### **PVM definition**

- Provide single interface and environs to exploit resources on different computer interconnected network for execute tasks with help message system.
- Interface: library with function for programmer use with (C, FORTRAN).
- Environs: execution units (processors), memories and network....etc.
- Supports heterogeneous environs.

The Parallel Virtual Machine (PVM) was originally developed at Oak Ridge National Laboratory and the University of Tennessee. It makes it possible to develop applications on a set of heterogeneous computers connected by a network that appears logically to the users as a single parallel computer. The PVM offers a powerful set of process control and dynamic resource management functions. It provides programmers with a library of routines for the initiation and termination of tasks, synchronization, and the alteration of the virtual machine configuration. It also facilitates message passing via a number of simple constructs. Interoperability among different heterogeneous computers is a major advantage in PVM. Programs written for some architecture can be copied to another architecture, compiled and executed without modification. Additionally, these PVM executables can still communicate with each other.

A PVM application is made from a number of tasks that cooperate to jointly provide a solution to a single problem. A task may alternate between computation and communication with other tasks. The programming model is a network of communicating sequential tasks in which each task has its own locus of control, and sequential tasks communicate by exchanging messages.

## **PVM Model Properties**

- Virtual parallel machine, for “cluster ” components:
  1. Node is can add or delete.
  2. Support heterogonous setting, on different nodes.
  3. Compute is divided to tasks on different nodes.
  4. Tasks can (but mustn't) use specification node.
  5. Task communicate explicitly sending message.
- In each node run PVM daemon.
- Application use call library.
- support Language : C, C++, Fortran.

## **PVM consist of:**

1. Task:
  - a- Process, which exploit function from library PVM .
2. Demon (pvmd3)
  - a- Run on each machine which is component of PVM.
  - b- Plans task on each machine.
  - c- Safeguards cover data.
  - d- Safeguards send message among nodes.
3. Console, or Gide User Interface (GUI):
  - a- Make it possible to configure PVM hosts and some control for execute.

The computing environment in PVM is the virtual machine, which is a dynamic set of heterogeneous computer systems connected via a network and managed as a single parallel computer. The computer nodes in the network are called hosts, which could be uniprocessor, multiprocessor systems, or clusters running the PVM software. PVM has two components: a library of PVM routines, and a daemon that

should reside on all the hosts in the virtual machine. Before running a PVM application, a user should start up PVM and configure a virtual machine.

The PVM console allows the user to interactively start and then alter the virtual machine at any time during system operation. The details of how to set up the PVM software, how to configure a virtual machine, and how to compile and run PVM programs can be found at <http://www.epm.ornl.gov/pvm> and in Geist et al. (1994).

The PVM application is composed of a number of sequential programs, each of which will correspond to one or more processes in a parallel program. These programs are compiled individually for each host in the virtual machine. The object files are placed in locations accessible from other hosts. One of these sequential programs, which is called the initiating task, has to be started manually on one of the hosts. The tasks on the other hosts are activated automatically by the initiating task. The tasks comprising a PVM application can all be identical but work on different ranges of data. This model of parallel programming is called SPMD, which stands for Single Program Multiple Data. Although SPMD is common in most PVM applications, it is still possible to have the tasks perform different functions.

A pipeline of parallel tasks that perform input, processing, and output is an example of parallel tasks that are performing different functions. Parallel virtual machine parallel applications can take different structures. One of the most common structures is the star graph in which the middle node in the star is called the supervisor and the rest of the nodes are workers. The star structure is often referred to as a supervisor–workers or a master–slaves model. In this model, the supervisor is the initiating task that activates all the workers. A tree structure is another form of a PVM application.

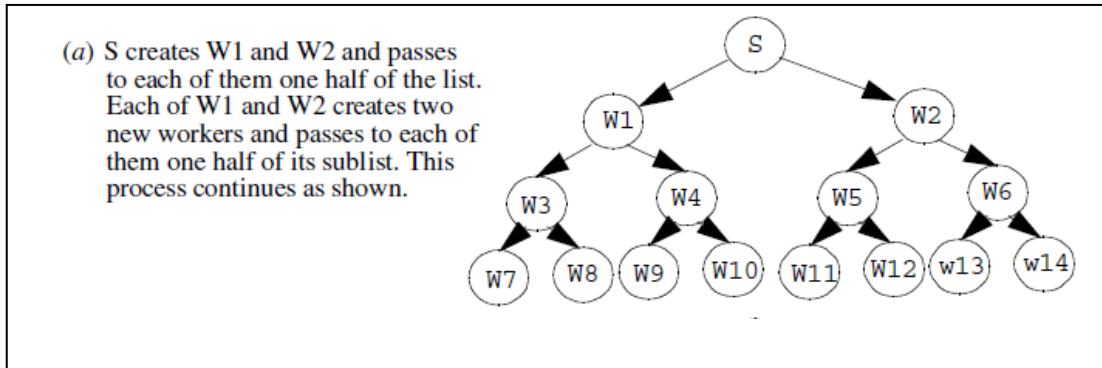
The root of the tree is the top supervisor and underneath there are several levels in the hierarchy. We will use the terms supervisor–workers and hierarchy to refer to the star and the tree structures, respectively.

## **PVM Task**

- Parallel compute is divided into sequence tasks, which can execute parallel.
- Tasks is can start on separate nodes, where execute is not migration.
- Each task has a one identification TID, which is create by PVM daemon.
- Message addressing by help TID.
- Tasks can range to groups.
- Task is implementing as OS process.

## Task Identifier Retrieval

Parallel virtual machine provides a number of functions to retrieve TID values so that a particular task can identify itself, its parent, and other tasks in the system. Task's TID A running task can retrieve its own TID by calling the PVM function `pvm_myid()`.



PVM allows running tasks to belong to named groups, which can change at any time during computation. Groups are useful in cases when a collective operation is performed on only a subset of the tasks. For example, a broadcast operation, which sends a message to all tasks in a system, can use a named group to send a message to only the members of this group. A task may join or leave a group at any time without informing other tasks in the group. A task may also belong to multiple groups.

PVM provides several functions for tasks to join and leave a group, and retrieve information about other groups.

Communication among PVM tasks is performed using the message passing approach, which is achieved using a library of routines and a daemon. During program execution, the user program communicates with the PVM daemon through the library routines.

## **PVMd – daemon execute**

1. Master: usually start from control command.
2. Execute command from control command.
3. Create socket to communicate with tasks and pvmd.
4. Read hostfile.
5. Start slave pvmd- on remote node.
6. Slave: receive parameters from master through arguments and configuration message.
7. In loop verify incoming message from local task and from network.
8. Certificate receive message.
9. Return results to master.
10. Master: wait all tasks to end then find final results.
  
11. Terminate daemon:
  - a) Logout from pvm (delete): break join with master pvmd, error signal from OS.
  - b) Terminate (kill) all his tasks : Send message to all pvmd in his table nodes to end.
  
12. Configuration :
  - c) Master pvmd keep table nodes, and each have copy.
  - d) Content table are: name, address, state communication for each node.

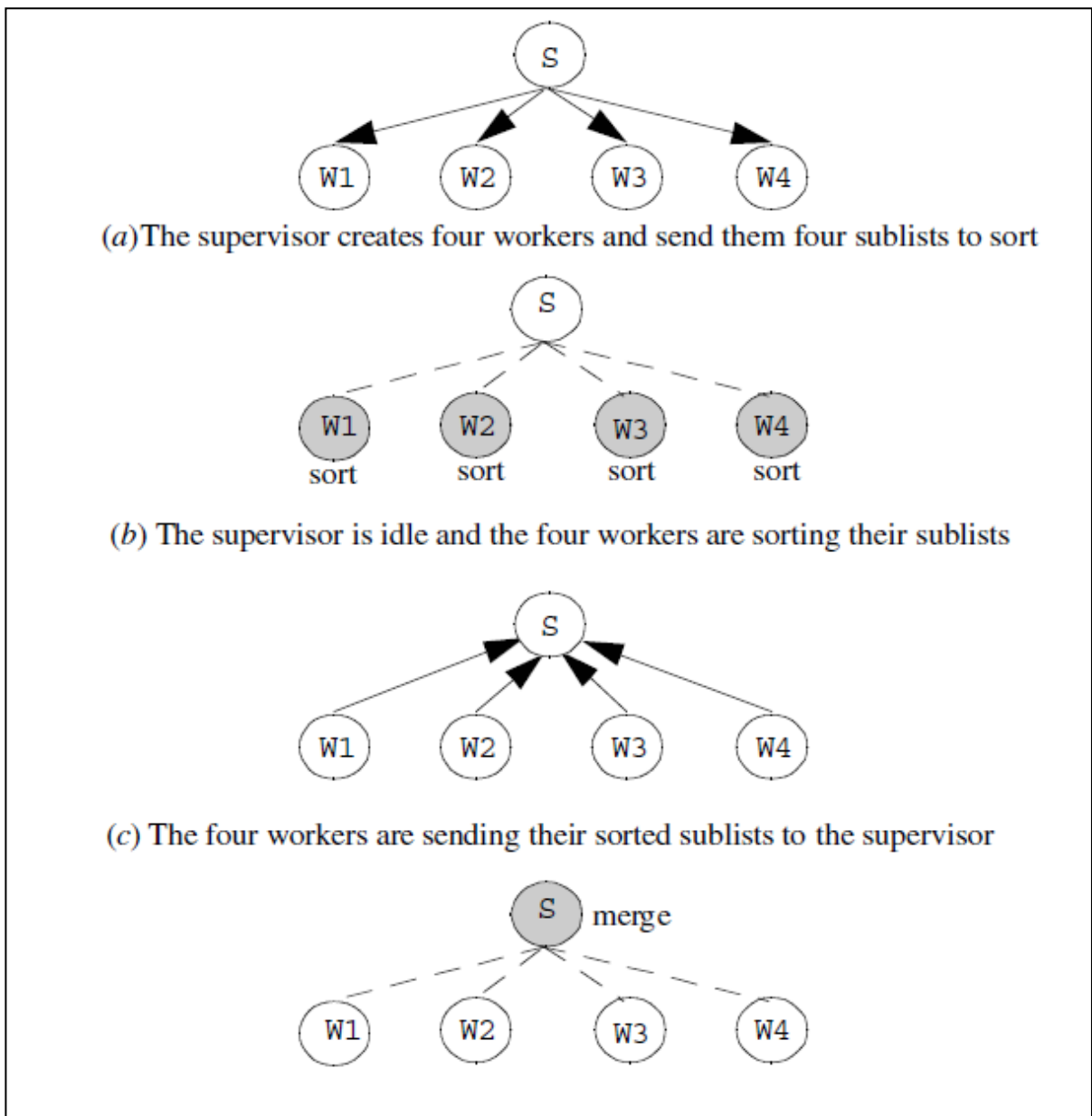
There is only one level of hierarchy in this structure: one supervisor and many workers.

The supervisor serves as the initiating task that is activated manually on one of the hosts. The supervisor, which is also called the master, has a number of special responsibilities.

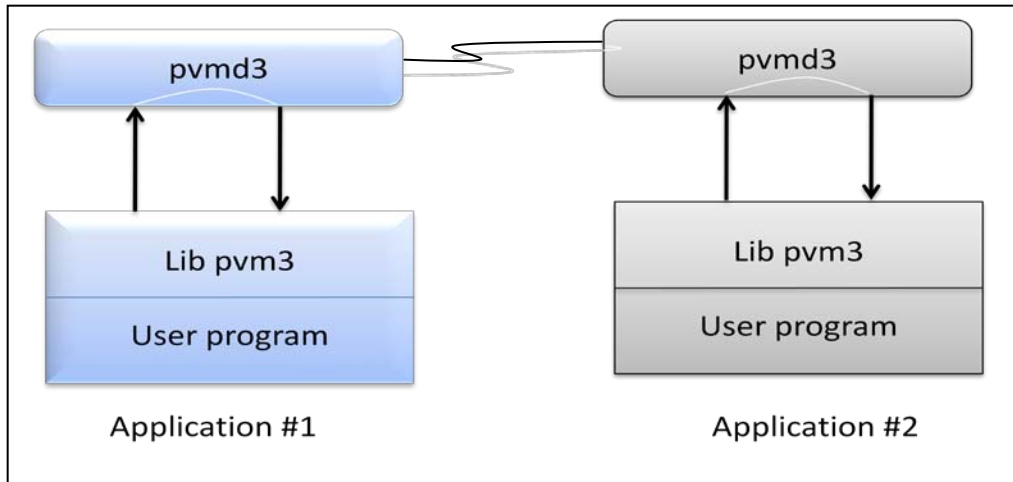
It normally interacts with the user, activates the workers on the virtual machine, assigns work to the workers, and collects results from the workers.

The workers, which are also called slaves, are activated by the supervisor to perform calculations. The workers may or may not be independent. If they are not independent, they may communicate with each other before sending the result of the computation back to the supervisor.

For example, a simple idea to sort an array of numbers using the supervisor–workers structure can be described as follows. The supervisor creates a number of workers and divides the array elements among them such that each worker gets an almost equal number of elements. Each worker independently sorts its share of the array and sends the sorted list back to the supervisor. The supervisor collects the sorted lists from the workers and merges them into one sorted list. Figure shows an example of sorting an array of elements using one supervisor (S) and four workers (W1, W2, W3, and W4). Note that in this example the workers are entirely independent and communicate only with the supervisor that performs the merge procedure on the four sorted sublists while the workers remain idle.



## PVM communication



## Initiation PVM from console

- \$pvm hostfile
  - hostfile : content list(index) nodes, which have be component of PVM (on each row one name).
- Instruction PVM in console:
  1. Add *host name* , delete *host name*.
  2. Conf (extract actual configuration).
  3. Halt (Stand off environs ), quit (end console ).
  4. Spawn (initiation new task).

## ***Starting PVM***

Before we go over the steps to compile and run parallel PVM programs, you should be sure you can start up PVM and configure a virtual machine. On any host on which PVM has been installed you can type:

```
% pvm
```

And you should get back a PVM console prompt signifying that PVM is now running on this host. You can add hosts to your virtual machine by typing at the console prompt:

```
pvm> add hostname
```

And you can delete hosts (except the one you are on) from your virtual machine by Typing:

```
pvm> delete hostname
```

If you get the message "\Can't Start pvmd," then check the common startup problems section and try again.

To see what the present virtual machine looks like, you can type

```
pvm> conf
```

To see what PVM tasks are running on the virtual machine, you type

```
pvm> ps -a
```

Of course you don't have any tasks running yet; that's in the next section. If you type "\quit" at the console prompt, the console will quit but your virtual machine and tasks will continue to run. At any Unix prompt on any host in the virtual machine, you can type:

```
% pvm
```

And you will get the message "\pvm already running" and the console prompt. When you are finished with the virtual machine, you should type

*pvm> halt*

This command kills any PVM tasks, shuts down the virtual machine, and exits the console. This is the recommended method to stop PVM because it makes sure that the virtual machine shuts down cleanly.

## **PVM Cluster**

- Linux, Fedora 9.
- PVM library.
- Join to system through home address.
- Equal configuration.