
PVM and Parallel Programming

What is Parallel? (Hardware)

Flynn's Taxonomy:

- SISD - Single Instruction stream over a Single Data stream: IBM PC, Mac, Sun, etc. Called **scalar machines**.
 - MISD - Multiple Instruction stream over a Single Data stream.
 - SIMD - Single Instruction Multiple Data stream: Cray, MP-2, CM-2. Called **vector machines**.
 - MIMD - Multiple Instruction Multiple Data stream: Intel Paragon, SP-2, networked Suns.
-
- SIMD (Cray) is good at synchronization, but poor at branching.
 - MIMD (SP-2) is good at branching, but poor at synchronization.

Fit the tool to the problem.

What is Parallel? (Software)

- Perfect Parallelism
 - Application can be divided into a set of processes that require little or no communication (also called “embarrassingly parallel”).
 - Example: Monte Carlo techniques; pick a random starting point, find solution, save result, repeat until “good-enough” solution is found.
- Data Parallelism
 - Different parts of the data are mapped to different processors (also called domain decomposition)
 - Example: matrix distributed over processors
- Functional Parallelism
 - Look at the problem as a set of operations, and assign different processors for each operation (also called control decomposition)
 - Example: simulating a power plant; each processor controls a separate system

Automatic Parallelizing Compilers

- Exploit loop level parallelism by assigning independent iterations of a loop to different processors
- Though this requires minimal effort for the programmer, it does not exploit functional parallelism

Message Passing

- Programming for multiple processors requires message passing: communication between the processors.
- Data distribution and communication must be managed explicitly by the programmer using SEND and RECEIVE messages
- Concepts that come in:
 - blocking/non-blocking sends
 - blocking/non-blocking receives
 - packing/unpacking buffers
 - broadcast/multicast

PVM: Parallel Virtual Machine

- Message passing library which is highly portable: IBM SP-2, Intel Paragon, Multi-Processor Suns, networked workstations
- Creates a virtual multi-processor machine, and manages message passing between nodes (processors) of that machine
- Processes are managed by the pvm daemon (**pvmd3**) running on each machine used in parallel
- Communication is done through rsh, so a .rhosts file needs to be set for all machines taking part in the processing
- Disadvantage to PVM: does not take into account special features on parallel machine, like the high performance switch on the SP-2, which would tend to slow down performance slightly
- Advantage to PVM: highly portable. allows for parallel programming, and debugging, on a non-parallel architecture.

Compiling and Running PVM

- Set up the PVM environment: paths, hostfile, .rhosts
- Design and write the program.
- Compile with proper makefile: include the pvm and socket libraries
- Start the PVM daemon, giving the hostfile

```
/home/pedja-> pvmd3 ~/hostfile &
```

- If you want, check on the PVM daemons

```
/home/pedja-> pvm
pvmd already running.
pvm> conf
3 hosts, 1 data format
          HOST      DTID      ARCH      SPEED
          cns3      40000    SUNMP     1000
          128.148.54.27 80000    SUN4SOL2  1000
          128.148.54.28 c0000    SUN4SOL2  1000
pvm> quit
pvmd still running.
/home/pedja->
```

Compiling and Running PVM

- Run your program

```
/home/pedja-> cd ~/pvm3/bin/SUNMP
/home/pedja-> hello
my id: 40002 my parent's id: fffffffe9
I'm the master!!
Spawning task 0 (id=80001)...done
Spawning task 1 (id=c0001)...done
Spawning task 2 (id=40003)...done
Spawning task 3 (id=80002)...done
Master Receiving... from process id 40003: I am child 40003
Master Receiving... from process id c0001: I am child c0001
Master Receiving... from process id 80001: I am child 80001
Master Receiving... from process id 80002: I am child 80002
```

- Stop the daemons

```
/home/pedja-> pvm
pvmd already running.
pvm> halt
[1] Done pvmd3 ~/hostfile
/home/pedja->
```

- Look at log files

```
/home/pedja-> id pedja
uid=129(pedja) gid=129
/home/pedja-> ls /tmp/pvm*.129
```

```

#define NTASKS 4
#include <stdio.h>
#include "pvm3.h"

main()
{
    int cc, tid[NTASKS],i,parent_tid,my_tid,bufid,recv_tid;
    int nbytes,msgid;
    char buf[100];

    my_tid=pvm_mytid();
    parent_tid=pvm_parent();

    printf("my id: %x  my parent's id: %x\n", my_tid,parent_tid);

    if (parent_tid==PvmNoParent) { /* master process */
        printf("I'm the master!!\n");

        for (i=0; i<NTASKS; i++) {
            printf("Spawning task %d ",i);

            cc = pvm_spawn("hello", NULL , PvmTaskDefault, "", 1, &tid[i]);
            printf("(id=%x)...done\n",tid[i]);
        }

        for (i=0; i<NTASKS; i++) {
            printf("Master Receiving... ");

            bufid = pvm_recv(-1, -1);
            pvm_bufinfo(bufid, (int *) 0 , (int *)0, &recv_tid);
            pvm_upkstr(buf);

            printf("from process id %x: %s\n", recv_tid, buf);
        }
    } else {
        cc=pvm_initsend(PvmDataDefault);
        sprintf(buf,"I am child %x",my_tid);
        cc=pvm_pkstr(buf);
        cc=pvm_send(parent_tid,1);

        printf("I (%x) am done\n",my_tid);
    }
    pvm_exit();
    exit(0);
}

```