COMPUTACIÓN DE ALTA PERFORMANCE

Curso 2020

Sergio Nesmachnow (sergion@fing.edu.uy)
Nestor Rocchetti (nrocchetti@fing.edu.uy)

Centro de Cálculo

TEMA 7: PVM
(PARALLEL VIRTUAL MACHINE)

CONTENIDO

7.1: INTRODUCCIÓN A PVM

¿ QUÉ ES PVM ?

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Una BIBLIOTECA para el desarrollo de aplicaciones paralelas y distribuidas.
- Cualidades:
  - Potencia.
  - Simplicidad.
  - Portabilidad (Unix, Linux, Windows).
  - Usabilidad:
    - Entorno integrado de desarrollo y control.

¿ QUÉ NO ES PVM ?

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- PVM NO ES un lenguaje de programación.
  - Los programas se desarrollan en lenguaje C ó FORTRAN.

- PVM NO ES un estándar de la industria.
  - Surge como proyecto de laboratorio universitario.
  - Fue adoptado como estándar “de facto”.
  - El estándar diseñado para serlo es MPI.

EVOLUCIÓN HISTÓRICA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- 1989: Proyecto del Oak Ridge National Laboratory
  - La primera versión no fue distribuida, solo se utilizó internamente.
- 1991: Versión 2, Universidad de Tennessee
  - Reescrita completamente.
  - Se comienza a utilizar en aplicaciones científicas.
- 1993: Versión 3
  - Rediseñada completamente.
- A partir de 1995 se popularizó mundialmente.
- En la década del 2000 declinó su uso
  - Como consecuencia del desarrollo de MPI.
- Aún utilizado, especialmente en enseñanza.

CARACTERÍSTICAS DE PVM

- Permite la administración dinámica de un conjunto de equipos que conforman la máquina paralela virtual.
- Provee mecanismos de creación e identificación de procesos.
- El modelo de comunicación entre procesos es basado en el pasaje de mensajes.
- Brinda soporte para implementar mecanismos de sincronización de procesos.
- Brinda soporte para manejar heterogeneidad.
- Permite aprovechar arquitecturas multiprocesador.

ARQUITECTURA DE PVM

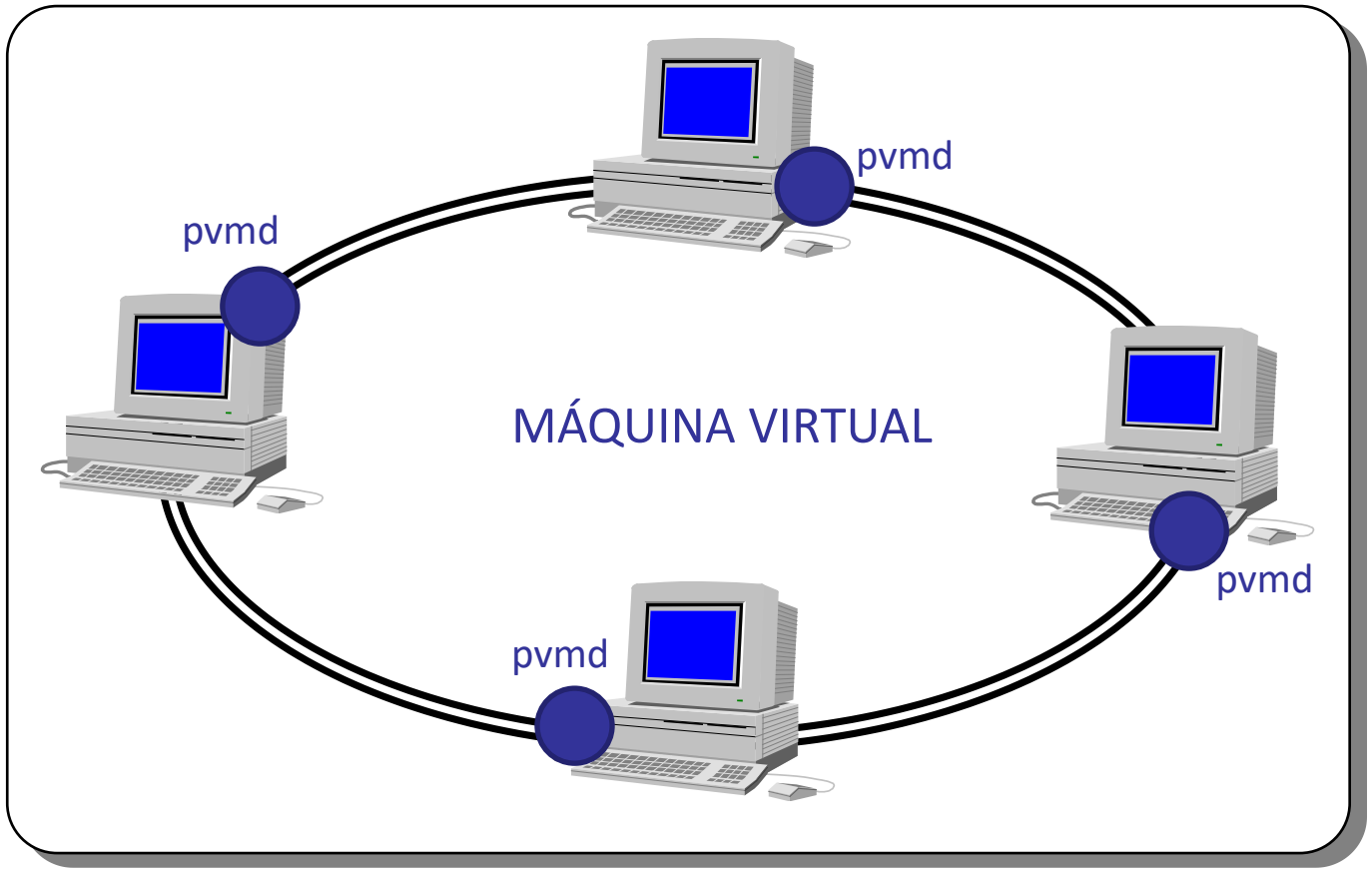
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Dos componentes:
  - **`pvmd`** (demonio ejecutado desde el sistema operativo).
  - Biblioteca (brinda API).

- PVM considera:
  - Una aplicación es un conjunto de tareas.
  - Una tarea es un proceso identificado por un task ID (TID).

MÁQUINA VIRTUAL DE PVM

- Máquina virtual de PVM: Conjunto de hosts (equipos) que ejecutan el demonio **`pvmd`**.



MÁQUINA VIRTUAL DE PVM

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Los demonios **pvmd** brindan soporte para el diseño de aplicaciones distribuidas, y para la comunicación y sincronización de sus componentes.
- Actúan como representantes de los procesos remotos para las comunicaciones (los **pvmd** se comunican entre sí).
- Los demonios pueden ser lanzados tanto desde la consola de PVM como desde las aplicaciones.
  - Configuración estática de la máquina virtual
  - Configuración dinámica de la máquina virtual

SOPORTE PARA HETEROGENEIDAD

- Permite definir máquinas virtuales que combinen diferentes arquitecturas (mecanismo especial de compilación – aimk).
- Maneja las diferentes representaciones de datos de modo transparente.
- Admite diferentes configuraciones y propiedades de los equipos de la máquina virtual.
- Maneja inteligentemente la carga de los equipos de la máquina virtual.

CONSOLA DE PVM

```
$ pvm
pvm> ?
help   - Print helpful information about a command
Syntax:  help [ command ]
Commands are:
  add    - Add hosts to virtual machine
  alias  - Define/list command aliases
  conf   - List virtual machine configuration
  delete - Delete hosts from virtual machine
  echo   - Echo arguments
  export - Add environment variables to spawn export list
  halt   - Stop pvmds
  help   - Print helpful information about a command
  id     - Print console task id
  ps     - List tasks
```

CONSOLA DE PVM

```
pvm> conf
1 host, 1 data format
            HOST      DTID   ARCH          SPEED
            pcunix122 40000  LINUX86_64    1000
pvm> add pcunix120 pcunix121
2 successful
            HOST      DTID
            pcunix120 80000
            pcunix121 c0000
pvm> conf
3 hosts, 1 data format
            HOST      DTID   ARCH          SPEED
            pcunix122 40000  LINUX86_64    1000
            pcunix120 80000  LINUX86_64    1000
            pcunix121 c0000  LINUX86_64    1000
```

DEFINICIÓN DE MÁQUINA VIRTUAL

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La creación o definición de una máquina virtual PVM puede realizarse a través de un archivo de configuración en forma estática:
  - $ pvm hostfile.
  - Útil para realizarlo en aplicaciones fuera de línea (procesamiento batch).

- Los equipos pueden agregarse dinámicamente a la máquina virtual (función pvm_addhost).

ARCHIVO HOSTFILE

- Define la configuración inicial de la máquina virtual.
- Puede incluir información de hosts a ser agregados posteriormente (estática o dinámicamente).
- Forma más simple: lista de nombres de hosts.
  - Líneas en blanco se ignoran.
  - Líneas que comienzan con # son comentarios.
- Opciones (separadas por espacio)
  - **lo = userid** (userid alternativo para el login).
  - **so = pw** (password alternativa para el login).
  - **dx = ubicación del pvmd** (permite copias personalizadas del pvmd).
  - **ep = paths a los ejecutables de usuario** (permite organización personal de ejecutables por defecto en `$HOME/pvm3/bin/PVM_ARCH`).

ARCHIVO HOSTFILE

- Opciones:
  - **sp = valor** (velocidad relativa del host respecto a otros en la configuración, en el rango de 1 a 1000000, por defecto vale 1000).
  - **bx = ubicación del debugger** (script para debug, por defecto es PVM_DEBUGGER).
  - **wd = directorio de trabajo** (para tareas lanzadas, por defecto es $HOME).
  - **ip = hostname** (nombre alternativo para resolver la dirección IP).
  - **so = ms** (inicio manual para pvmd esclavo. Útil cuando servicios de red ssh/rsh están deshabilitados).

EJEMPLO DE ARCHIVO HOSTFILE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
# línea de comentario
lennon.fing.edu.uy
marley.fing.edu.uy
joplin.fing.edu.uy
# diferente usuario y password
harrison.fing.edu.uy lo=user_pvm so=pw
# diferente path al pvmd
morrison.fing.edu.uy dx=/bin/progs/pvm/pvmd
# diferente directorio con ejecutables
mercury.fing.edu.uy ep=$HOME/xxx/pvm3/bin/PVM_ARCH
# opción por default
* lo=std_user
vaughan.fing.edu.uy
cobain.fing.edu.uy
* lo=
# no en la configuración inicial
&berry.fing.edu.uy
&richard.fing.edu.uy
&mayall.fing.edu.uy
```

COMPILACIÓN

- AIMK: Architecture Independent MaKe.
- aimk: utilitario de PVM que permite tener un mismo ejecutable compilado para varias arquitecturas y sistemas operativos
  - En defintiva es un wrapper de Make
- Mantiene los objetos y los programas en estructuras de directorios diferentes para cada arquitectura
  - Utilizando la variable de entorno $PVM_ARCH
- Por defecto ejecuta:

```
(cd $PVM_ARCH ; \
make -f $PVM_ROOT/conf/$PVM_ARCH.def \
   -f ../Makefile.aimk PVM_ARCH=$PVM_ARCH <aimk args>)
```

7.2: INTERFAZ PARA APLICACIONES (API)

INTERFAZ PARA APLICACIONES

- PVM propone cinco tipos de funciones:
  1. Funciones de control de procesos.
  2. Funciones para pasaje de mensajes.
     - Funciones de empaquetamiento de datos.
     - Funciones de envío y recepción de mensajes
  3. Funciones para manejo de grupos de procesos.
     - Operaciones colectivas.
  4. Operaciones sobre el contexto.
  5. Señales y notificaciones.

1. CONTROL DE PROCESOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- pvm_spawn (task, arg, flag, where, ntask, tids)
  - Creación de procesos idénticos.
  - task: nombre de la tarea (se busca de acuerdo a la arquitectura).
  - arg: argumentos para la(s) tareas(s) a crear.
  - flag: opciones de ejecución de la(s) tarea(s) creadas.
    - taskDefault, taskHost, taskArch, taskDebug, taskTrace, etc.
  - where: host o arquitectura particular.
  - ntask: número de tareas a crear.
  - tids: retorna la lista de identificadores de las tareas creadas.
  - Retorna el número de tareas creadas.

1. CONTROL DE PROCESOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- int pvm_mytid(void)
  - Retorna el identificador de tarea del proceso invocante.
- int pvm_parent(void)
  - Retorna el identificador de tarea del proceso padre.
- int pvm_exit(void)
  - Indica al pvmd local que el proceso deja el entorno pvm.
  - El proceso no finaliza, puede realizar actividades seriales.
- int pvm_kill(int tid)
  - Envía una señal de finalización (SIGTERM) a la tarea tid.

2. PASAJE DE MENSAJES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Buffers de mensajes:
  - Definen el formato en que se enviarán y recibirán los datos.

- El mecanismo de envío tiene tres fases:
  - Inicializar el buffer (pvm_initsend).
  - Empaquetar datos (pvm_pk*).
  - Enviar (pvm_send).

2. PASAJE DE MENSAJES

- Inicialización del buffer:

  int bufid = pvm_initsend (int encoding)

  – encoding: tipo de codificación utilizado.
    - dataDefault (XDR): por defecto, para entorno heterogéneo.
    - dataRaw: no codifica (en la misma arquitectura).
    - dataInPlace: buffer contiene punteros y tamaños de los elementos a enviar, los datos se envían directamente desde memoria. Evita copias de mensajes, pero exige que no se modifiquen los datos entre el pack y el send.

2. PASAJE DE MENSAJES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Rutinas de empaquetado de datos
  - Definen el orden en que se van a enviar un conjunto de datos dentro del mismo mensaje.

    int info = pvm_pk*(tipo *p, int nitem, int stride)
  - Cada rutina pk* empaqueta un array del tipo de dato especificado al buffer activo (pkdouble, pkint, pkbyte, etc.).
  - p: puntero al primer elemento a empaquetar
  - nitem: número de elementos a empaquetar.
  - stride: paso (valor 1 para empaquetar un vector contiguo, 2 para empaquetar dato por medio, etc.).
  - En caso exitoso, retorna info=0 sino info<0.

2. PASAJE DE MENSAJES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Rutinas de desempaquetado de datos:
  - Por cada rutina de empaquetado, existe una de desempaquetado: pvm_upkbyte(), pvm_upkdouble(), pvm_upkstr(), etc.

    int info = pvm_upk*(tipo *p, int nitem, int stride)
  - Cada rutina upk* desempaqueta un arreglo del tipo de dato especificado del buffer activo.
  - Los mensajes se deben desempaquetar en el mismo orden en que fueron empaquetados.
  - En caso exitoso, info = 0 sino info < 0.

2. PASAJE DE MENSAJES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Envío y recepción de datos:
  - Soporte para tráfico de datos, en el orden en que fueron empaquetados y con la codificación de su buffer.

    int info = pvm_send(int tid, int msgtag)

    int info = pvm_mcast(int *tid, int ntasks, int msgtag)

    int bufid = pvm_recv(int tid, int msgtag)

    int info = pvm_bufinfo(int bufid, int *bytes, int *tag, int *tid)

  - La recepción de datos es BLOQUEANTE.

2. PASAJE DE MENSAJES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Pasaje de mensajes NO BLOQUEANTES:

  int bufid = pvm_nrecv(int tid, int msgtag)

  int bufid = pvm_probe(int tid, int msgtag)

  int bufid = pvm_trecv(int *tid, int msgtag, struct timeval *timeout)

- Permiten que los procesos continúen en forma asincrónica mientras se reciben mensajes.
- El modelo de mensajes de PVM garantiza que si la tarea 1 envía un mensaje A y luego un mensaje B a la tarea 2, el mensaje A llegará primero que el B.

3. GRUPOS DE PROCESOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Grupos dinámicos de procesos:
  - Permite definir órdenes o jerarquías de tareas.
  - Construidos sobre las rutinas básicas de pvm.
  - Requiere linkeditar con la biblioteca gpvm3.
- Diseñados para ser genéricos, aunque se pierda algo de eficiencia.
- Cualquier tarea puede unirse o abandonar un grupo en cualquier momento (sin informar a otras tareas del grupo).
- Unirse a un grupo (o creación de grupo):

  int inum = pvm_joingroup (char * group);
- Abandonar un grupo:

  int info = pvm_lvgroup (char * group);
- Obtener información del grupo

  int tid = pvm_gettid (char *group, int inum);

  int size = pvm_gsize(char *group);

3. GRUPOS DE PROCESOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Sincronización por barrera:

  int info = pvm_barrier (char * group, int count)

  – El proceso invocante debe pertenecer al grupo.
  – Los procesos se bloquean hasta que count miembros del grupo hagan la invocación.
  – Da error si los procesos invocan una barrera para grupo al que no pertenecen o si count > #procesos en el grupo.

3. GRUPOS DE PROCESOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Broadcast:

  int info = pvm_bcast (char * group, int msgtag)

  – No es necesario que el proceso invocante pertenezca al grupo.
  – Marca los mensajes con un identificador msgtag y los envía a todos los integrantes del grupo, excepto a si mismo.
  – Es asíncrono.
  – Esta operación primero recolecta los tids de los integrantes del grupo seleccionado. Si el grupo cambia luego de recolectada la lista de tids, estos cambios no se verán reflejados en el broadcast.
    - Si una tarea se une al grupo durante un broadcast, no se enviará el mensaje.
    - Si una tarea abandona un grupo luego de invocar un broadcast, una copia del mensaje será entregada.

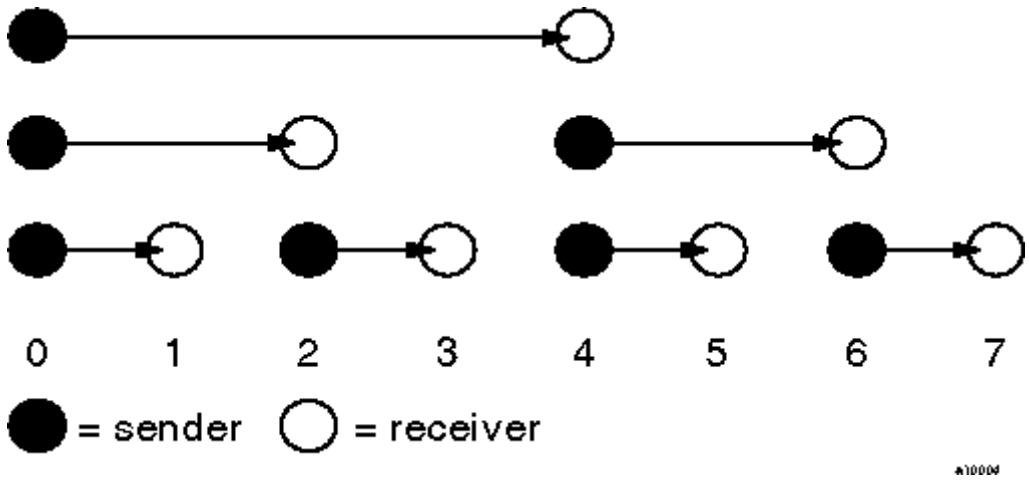
3. GRUPOS DE PROCESOS

pvm_reduce(void (*func)(), void *data, int nitem, int datatype, int msgtag, char *group, int root)

- Se realiza una operación aritmética global en un grupo.
- El resultado de la operación llega a la tarea con TID root.
- Funciones predefinidas: PvmMax, PvmMin, PvmSum, PvmProduct ó definidas por el usuario (func).
- Ejemplo en $PVM_ROOT/examples/gexamples.
- pvm_reduce() no es bloqueante.
- Si una tarea la invoca y luego abandona el grupo antes que el root invoque a pvm_reduce, se produce un error.

BCAST o MCAST

- Dos métodos para el envío de mensajes a múltiples receptores.
- Multicast: int pvm_mcast(int *tid, int ntasks, int msgtag)
- Broadcast: int info = pvm_bcast (char * group, int msgtag)
- La operación pvm_mcast es más flexible, pero pvm_bcast es más eficiente bajo ciertas condiciones.
- pvm_bcast hace uso de pvmd intermedios para reenviar el mensaje.



4. OPER. SOBRE EL CONTEXTO

- PVM brinda operaciones que permiten obtener y modificar la configuración de información del contexto (escenario de ejecución).

  pvm_config(int *nhost, int *narch, struct pvmhostinfo **hostp)

  - Ofrece información de la configuración de la máquina virtual.
  - PVM struct pvmhostinfo (definida en pvm3.h):

```
struct pvmhostinfo {
    int hi_tid;
    char *hi_name;
    char *hi_arch;
    int hi_speed;
};
```

4. OPER. SOBRE EL CONTEXTO

pvm_tasks(int which, int *ntask, struct pvmtaskinfo **taskp)

- which = 0 todas las tareas
- which = DTID tareas en un host
- which = TID de una única tarea

– Ofrece información de las tareas en ejecución:

```
struct pvmtaskinfo {
    int ti_tid;
    int ti_ptid;
    int ti_host;
    int ti_flag;
    char *ti_a_out;
}
```

4. OPER. SOBRE EL CONTEXTO

int info = pvm_addhosts(char **hosts, int nhosts, int *infos)

int info = pvm_delhosts(char **hosts, int nhosts, int *infos)

- Agrega/quita los equipos en hosts a la máquina virtual.
  - hosts: arreglo de nombres de equipos (con pvmd instalado y cuenta de usuario disponible).
  - nhost: largo del array de hosts.
  - infos: arreglo de enteros de largo nhost con el status de cada host.
  - Retorna un entero con el número de hosts agregados (si es menor que nhost hay un fallo parcial, si es menor que 1 hay un fallo total).
  - Sintaxis de nombres: la del pvmd hostfile (hostname seguido por parámetros de la forma xx=y).
- Ejemplo:

```
static char *hosts[] = {"sparky","thud.cs.utk.edu",};
info = pvm_addhosts( hosts, 2, infos )
```

5. SEÑALES Y NOTIFICACIONES

int info = pvm_sendsig(int tid, int signum)

- Envia una señal signum a la tarea con tid dado.

int info = pvm_notify(int what, int msgtag, int cnt, int * tids)

- Solicita a los pvmd ser notificado de ciertos eventos.
  - Eventos (what):
  - PvmTaskExit (fin de tarea).
  - PvmHostDelete (host eliminado o falla).
  - PvmHostAdd (host agregado).
  - msgtag: tag de identificación (definido por el usuario).
  - tids: array que especifica a quién monitorear (en el caso de TaskExit y de HostDelete).

7.3: EJEMPLO MAESTRO-ESCLAVO

master.c

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <pvm3.h>
4. int main() {
5.      int myTID; int x = 12; int res;
6.      int children[10];
7.      myTID = pvm_mytid();
8.      printf("Master: TID is %d\n", myTID);
9.      if ((res = pvm_spawn("slave",NULL,PvmTaskDefault,"",1,children))< 1){
10.             printf("Master: pvm_spawn error\n");
11.             pvm_exit();
12.             exit(1);
13.     }
14.     pvm_initsend(PvmDataDefault);
15.     pvm_pkint(&x, 1, 1);
16.     pvm_send(children[0], 1)
17.     pvm_recv(-1, -1);
18.     pvm_upkint(&x, 1, 1);
19.     printf("Master has received x=%d\n", x);
20.     pvm_exit();
21.     exit(0);
22.}
```

Biblioteca PVM

Lanzar tarea distribuida

Inicialización de buffers

Empaquetado de dato y envío de mensaje

Recepción de mensaje y desempaquetado de dato

slave.c

```
1. #include <stdio.h>
2. #include <pvm3.h>
3. int main(){
4.    int myTID, masterTID;
5.    int x = 12;
6.    myTID = pvm_mytid();
7.    printf("Slave: TID is %d\n", myTID);
8.    pvm_recv(-1, -1);
9.    pvm_upkint(&x, 1, 1);
10.   printf("Slave has received x=%d\n", x);
11.   sleep(3);
12.   x = x + 5;
13.   pvm_initsend(PvmDataDefault);
14.   pvm_pkint(&x, 1, 1);
15.   pvm_send(pvm_parent(), 1);
16.   pvm_exit();
17.   return 0;
18.}
```

Recepción de mensaje y desempaquetado de dato

Inicialización de buffers

Empaquetado de dato y envío de mensaje

Makefile.aimk

```
1. LDIR    =  -L$(PVM_ROOT)/lib/$(PVM_ARCH)
2. PVMLIB  =  -lpvm3
3. SDIR    =  ..
4. BDIR    =  $(HOME)/pvm3/bin
5. XDIR    =  $(BDIR)/$(PVM_ARCH)
6. CFLAGS  =  -O3 -I$(PVM_ROOT)/include
7. LIBS    =  $(LDIR) $(PVMLIB) $(ARCHLIB)

8. all: master slave

9. master: $(SDIR)/master.c $(XDIR)
10.     $(CC) $(CFLAGS) -o $@ $(SDIR)/$@.c $(LIBS)
11.     mv $@ $(XDIR)
12.slave: $(SDIR)/slave.c $(XDIR)
13.     $(CC) $(CFLAGS) -o $@ $(SDIR)/$@.c $(LIBS)
14.     mv $@ $(XDIR)
15.$(XDIR):
16.     - mkdir -p $(BDIR) $(XDIR)
```

LINUXX86_64

~/pvm3/bin/LINUXX86_64

EJECUCIÓN

```
$ pvm hostfile
pvm> conf
conf
2 hosts, 1 data format
                     HOST     DTID     ARCH   SPEED       DSIG
   pcunix122.fing.edu.uy    40000 LINUXX86_64    1000 0x00408c41
   pcunix121.fing.edu.uy    c0000 LINUXX86_64    1000 0x00408c41
pvm> spawn -> master
spawn -> master
[1]
1 successful
tc0001
pvm> [1:t40003] Slave: TID is 262147
[1:t40003] Slave has received x=12
[1:t40003] EOF
[1:tc0001] Master: TID is 786433
[1:tc0001] Master has received x=17
[1:tc0001] EOF
[1] finished
```

PVM EN LA FING

- Variables de entorno
  - `PVM_DIR=${HOME}/pvm3`
  - `PVM_ROOT=/usr/share/pvm3`
  - `PVM_DPATH=/usr/share/pvm3/lib/pvmd`
  - `PVM_RSH=/usr/bin/ssh`
  - `PVM_ARCH=LINUXX86_64`

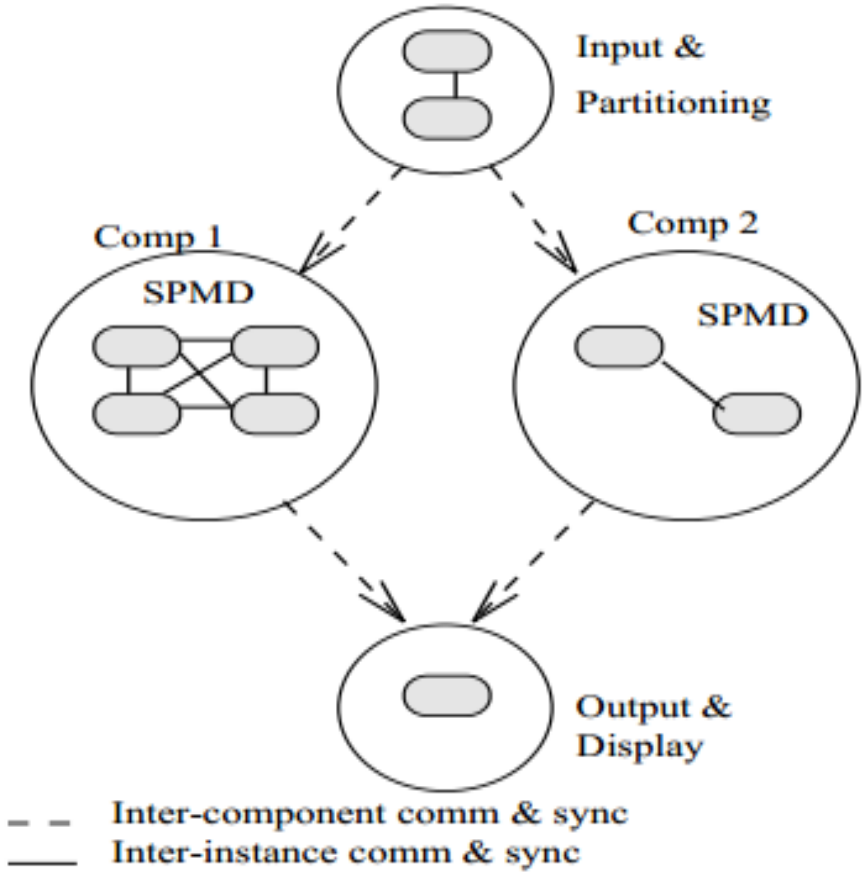
- Los binarios de PVM están en `/usr/share/pvm3/lib`
  - `export PATH=${PATH}:/usr/share/pvm3/lib`

- Es necesario crear y habilitar un par de claves publica/privada
  - `$ ssh-keygen`
  - `$ cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys`

ANEXO: MODELOS DE PROGRAMACIÓN DISTRIBUIDA

MODELOS DE PROGRAMACIÓN

- Modelos de computación:
  - Crowd computing
  - Tree computing



CROWD COMPUTING

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Modelo más comúnmente utilizado
- Procesos estrechamente relacionados (típicamente SPMD)
- División de dominio e intercambio de resultados intermedios
- Dos tipos: master-slave y node-only
- Modelo master-slave (o host-node)
  - Proceso maestro crea esclavos, los inicializa, recolecta y despliega resultados
  - Procesos esclavos realizan cómputo
- Modelo node-only
  - Una instancia realiza las tareas de gestión y tareas de cómputo

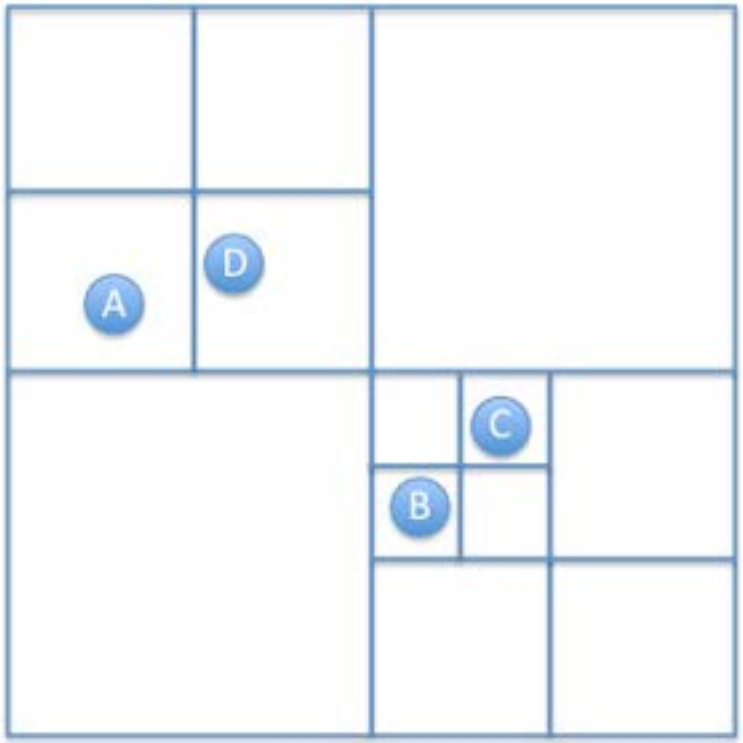
TREE COMPUTING

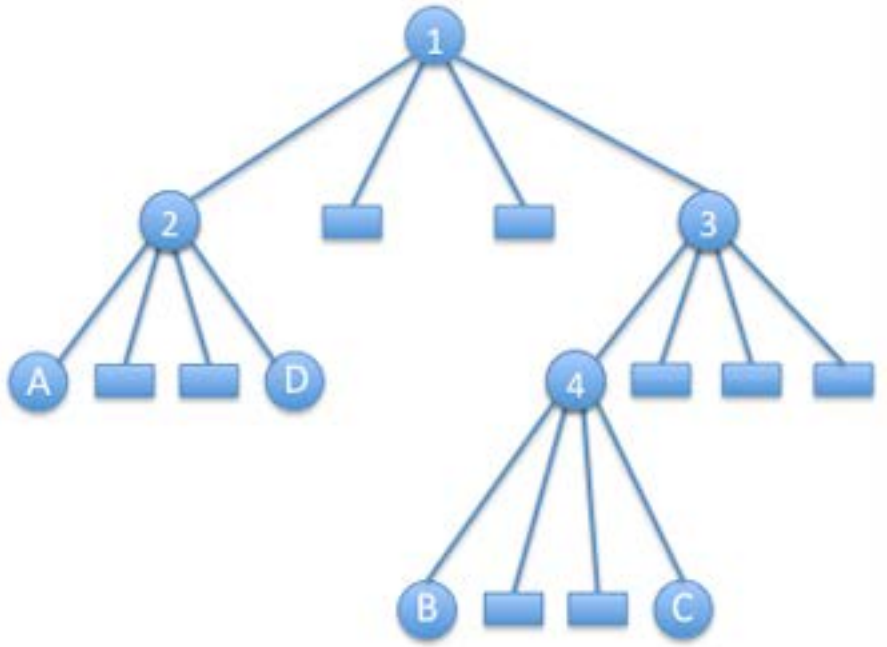
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Estructura de control arborescente. En general, con un patrón de comunicaciones arborescente
- Procesos creados (en general dinámicamente) siguiendo un modelo de árbol, estableciendo relaciones padre-hijo
- Opuesto al modelo crowd computing donde se tienen relaciones en forma de estrella
- Modelo usado principalmente para aplicaciones donde la carga total de trabajo es desconocida
- Branch-and-bound, divide and conquer, etc.

TREE COMPUTING

- Ejemplo estructura quadtree:





ASIGNACIÓN Y BALANCE DE CARGA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Descomposición de datos
  - Divide el dominio de datos de acuerdo a cierta estructura
- Descomposición funcional
  - Divide el trabajo computacional de acuerdo a diferentes funciones
- La descomposición puede hacerse: estática o dinámica
- La descomposición estática en general **NO** es útil
  - Aún en los casos ideales en los que la partición puede hacerse balanceada
  - Los entornos distribuidos son altamente susceptible a influencias externas, ocasionando que algunos procesos completen su trabajo antes que el resto

MANEJO DE ENTRADA/SALIDA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El manejo de entrada y salida resulta muy importante en entornos realistas de ejecución
- En general, se aplican tres enfoques:

1. Cada proceso individual genera sus datos internamente
   - Por ejemplo, usando números aleatorios o registros estadísticos
2. Cada proceso individual lee sus datos desde dispositivos
   - Simple, pero puede comprometer el desempeño si no se cuenta con la capacidad de I/O en paralelo
3. Un proceso de control envía subconjuntos de datos a cada proceso
   - Enfoque más común. El método es también apropiado cuando los subconjuntos de datos se obtienen de cómputos previos