Mapaná: una plataforma paralela para red de microcomputadores

UNIVERSIDAD DE LOS ANDES

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Fabio Alexander Corzo. Ing. Sistemas y Computación Universidad de Los Andes. Profesor de Cátedra Universidad de Los Andes. Asistente de investigación Universidad de Los Andes. Estudiante de Magister en Ingeniería de Sistemas y Computación Universidad de Los Andes.
f-corzo@uniandes.edu.co

Rafael Gómez. Ing. Sistemas y Computación Universidad de los Andes D.E.A Informática Grenoble. Francia. Profesor Asociado Universidad de Los Andes.
rgomez@uniandes.edu.co

Resumen

Se describe una plataforma paralela construida usando Win32 y TCP/IP, lo cual le permite ejecutar sobre Windows NT y Windows 95. La plataforma crea un nivel de abstracción que permite ver los nodos de una red como procesadores de una máquina paralela, definiendo así una máquina paralela virtual.

La plataforma ofrece recursos que facilitan la programación paralela sobre redes de micros y, en particular, simplifica la construcción de compiladores de lenguajes paralelos (v.g. OCCAM o extensiones de C).

1. Introducción

El uso de las máquinas paralelas en problemas numéricamente pesados se ha mostrado como una alternativa importante, y ha hecho viable solucionar problemas que antes no era posible atacar por falta de poder computacional.

Sin embargo, no es usual disponer de tecnología paralela. Se puede facilitar el acceso a la programación paralela mediante el uso de plataformas sobre redes de computadores. Estas plataformas se caracterizan por ser librerías o procesos demonios que proveen servicios específicos para la construcción de aplicaciones paralelas [1][4].

Sin embargo, en opinión de los autores, este tipo de programación basada en librerías induce al desarrollo intrínsecamente no estructurado de software paralelo. En efecto, un lenguaje tradicional, con la adición de una librería, no posee las estructuras de control y el poder de expresión necesarios para permitir un nivel de abstracción adecuado para el desarrollo de software paralelo. Hablando gráficamente, esto se constituye en el "GOTO" de la programación paralela [2], puesto que conduce a una programación anárquica y desordenada.

Las consecuencias de esta situación son: software difícil de diseñar, verificar, desarrollar y mantener; software poco portable y, por último, mayor complejidad en la enseñanza de técnicas paralelas. Todo esto, eventualmente, puede conducir a desaprovechar los recursos disponibles.

Por otro lado, existen lenguajes estructurados para la programación paralela, sean diseñados explícitamente con este fin -como OCCAM[6]- o sean extensiones de lenguajes tradicionales como C-; pero o no son populares (como OCCAM) o no es habitual disponer de ellos sobre plataformas tradicionales (extensiones de C) o son simulaciones de paralelismo (mono-procesador).

Nuestro propósito último consiste en desarrollar un conjunto de herramientas para programación paralela que permitan eliminar, o al menos atenuar, los problemas antes mencionados. Las restricciones que se deben cumplir son:

- Las herramientas deben correr sobre redes de microcomputadores; esto con el fin de obviar el problema de disponibilidad, y para ofrecer un paralelismo real.
- Las redes deben ser homogéneas; el uso de múltiples plataformas hardware quita transparencia al desarrollo de software -es necesario dividir un programa, de forma no natural,

entre las plataformas- y lo puede hacer dependiente de características específicas de las máquinas sobre las que corre.

- Las herramientas deben respetar al máximo el sistema operacional nativo; de otra manera no serían muy portables entre versiones del sistema. Esto implica que el sistema operativo debe ser de un cierto nivel; de lo contrario, la interferencia con el sistema sería inevitable.
- El lenguaje debe ser fácil de aprender y estructurado para paralelismo. Por lo cual debe ser cercano a un lenguaje tradicional y no debe usar librerías para manejar el paralelismo.

Basados en lo anterior, escogimos como soporte hardware redes de micros con sistema operativo Windows NT o Windows 95 y como lenguaje C extendido con los construcciones paralelas de OCCAM. Las etapas del proyecto son:

1. Desarrollo de la plataforma de base (Mapaná): ofrecerá un API con acceso a las funciones necesarias para implantar los constructores de OCCAM que se le agregarán a C.
2. Desarrollo del compilador de C extendido para la plataforma.
3. Difusión de las herramientas anteriores (vía Internet y otros).
4. Desarrollo de herramientas para depuración, monitoreo y balanceo de carga.

Este artículo se centra en la primera etapa: el desarrollo de la plataforma paralela.

Como se dijo anteriormente, se ha tomado OCCAM como referencia para determinar los requerimientos de la plataforma paralela. Esto determina los siguientes requerimientos:

- En comunicaciones: Se permite la comunicación sincrónica entre procesos locales o de diferentes procesadores. Esta comunicación se realiza a través de canales, en los cuales la lectura y escritura son bloqueantes. Además, la diferencia de utilización de un canal remoto a uno local es casi nula.

- En procesos: Se pueden lanzar procesos (lo más livianos posibles) locales y remotos.

Existen 2 tipos de procesos: alta y baja prioridad. Los procesos de alta prioridad no son interrumpidos durante su ejecución. Los procesos de baja prioridad son interrumpidos ya sea para darle el tiempo de CPU a un proceso de mayor prioridad o a otro proceso de baja prioridad.

- En sistemas de espera de eventos: Se pueden realizar esperas pasivas para realizar una comunicación sincrónica o para realizar esperas de tiempo o *timeout* sobre el tiempo del reloj del sistema.

A partir de las anteriores características se plantean como requerimientos fundamentales de la plataforma:

- Transparencia del carácter remoto de los diversos recursos y en especial de los procesadores de cada una de las máquinas de la plataforma virtual paralela.
- El sistema de procesos no solo debe tener características de *preemptive multitasking* sino que debe ofrecer mecanismos para manejar los procesadores de la red como si fuera una máquina multiprocesador.
- Debe existir un sistema de sincronización de procesos, tanto locales como remotos.
- Debe existir un sistema de intercambio de información entre procesos locales y procesos remotos.
- Se debe ofrecer un sistema de espera pasiva que interactúe con los mecanismos de comunicaciones y de manejo de relojes.
- Se deben unificar los diferentes sistemas administradores de memoria de tal manera que se puedan definir bloques de memoria globales.

2. Arquitectura

La plataforma se compone de un módulo de comunicación, que debe residir en cada uno de los nodos que hacen parte de la máquina virtual, y una librería que se encadena con los ejecutables y

que controla la creación remota de canales y procesos. En la fig. 1 se puede apreciar un diagrama de conexión entre dos nodos de la plataforma. Este esquema permite que varios usuarios puedan aprovechar la red con máquinas virtuales independientes.

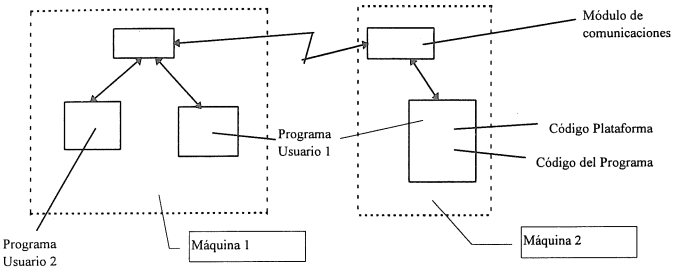

Fig. 1. Diagrama conexiones con los módulos de comunicación y ubicación de códigos ejecutables.

El módulo de comunicación se encarga de conectarse con los otros módulos. Una vez conectados los módulos, se puede cargar y ejecutar cualquier programa. Cada programa, crea un proceso Win32 en cada nodo donde se requieran recursos. Cada usuario tiene un proceso en cada máquina donde requiera servicios. Estos procesos aislan los recursos de cada programa y facilitan la comunicación entre *threads* de una aplicación. El primer módulo (identificador 1) iniciado corresponde a la máquina deniminada BOOT. Esta se encarga de sincronizar las demás conexiones

El lugar de ejecución de cada *thread* se asigna en compilación; se crean tantos ejecutables como nodos vaya a usar el programa. Cada ejecutable lleva asociado el código de los servicios de la plataforma. Este código es ejecutado por un *thread*, TPVP (*Thread* Plataforma Virtual Paralela), encargado de la interacción con el módulo de comunicación.

El proceso manejador de comunicaciones se encarga de: realizar la conexión inicial con los otros procesos de comunicaciones, creación de procesos en el momento de cargar un programa, traducir los números de nodo a direcciones IP, distribuir los mensajes que llegan por la red y realizar transferencias de ejecutables.

La mayoría de los servicios de la plataforma se incorporan en el código ejecutable durante el encadenamiento. Así, un mismo proceso controla todos los *threads* locales, de un usuario, que sean creados.

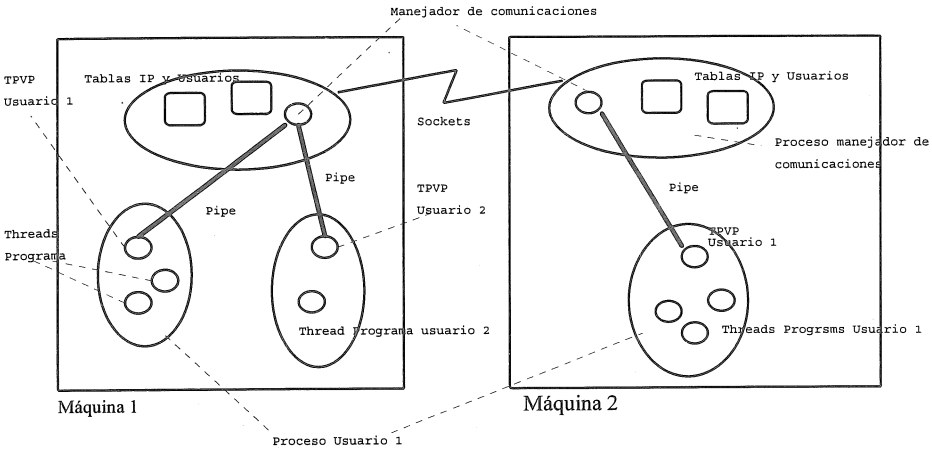

Fig. 2. Procesos, *threads* y conexiones de Mapaná

Un *thread,* de un proceso usuario, ve las conexiones con los demás *threads* como canales OCCAM. La implantación de los canales usa una zona de memoria como identificador del canal Las direcciones de memoria se componen de la dirección física y un identificador de nodo. Si el canal es remoto, el sistema se da cuenta por el identificador de nodo y realiza los requerimientos de entrada o salida a través del módulo de comunicación.

La comunicación entre módulos de comunicación se realiza mediante *stream sockets.* Hay un *socket* por cada par de nodos de la red. El enrutamiento lo realizan los módulos de comunicación; pero el manejo de los canales en sí (área de memoria asignada) lo realiza el TPVP.

Para cada canal se reserva una zona de memoria y se crea un apuntador extendido (i.e dirección física + número nodo). Estos valores pueden ser manejados localmente y pasados como argumentos a procedimientos y funciones.

El módulo de comunicación atiende las solicitudes de ejecución remota. Él se encarga de comunicarse con el módulo homólogo del nodo donde se va a realizar la ejecución. Allí se crea un proceso para el programa y dentro de este se crean 2 *threads:* el TPVP y el *thread* que ejecutará el código del usuario.

Durante la iniciación del sistema no se carga ningún programa, pero hay un servicio LoadProgram para cargar un programa en los nodos de la plataforma. La iniciación solo interconecta y sincroniza los módulos de comunicación..

3. Comunicaciones

El sistema de comunicaciones tiene dos partes: la primera, en el servidor de comunicaciones, se encarga de realizar el arranque, de transfererir mensajes y archivos de inicio y crear procesos -en el momento de carga de programas-. Este servidor trabaja en exclusión mutua, para evitar problemas de reentrancia e inconsistencias de los datos. La segunda parte se encuentra pegada a los programas que van a ejecutar sobre la plataforma. A cada programa se le encadenan ciertas funciones, que controlan la mayor parte de los servicios de comunicaciones. El código encadenado crea el TPVP, que implementa los mecanismos de comunicación, manejo de memoria y creación remota de *threads.*

El sistema de comunicaciones se basa en un identificador lógico que se le asigna a cada uno de los módulos de comunicación de la máquina virtual. El identificador es un número entre 1 y 32767. La asignación de estos números se realiza durante el arranque.

Los módulos de comunicación tienen una tabla para traducir las direcciones lógicas de la plataforma a las físicas de la red. La tabla se crea en el arranque y, aunque la dirección lógica puede estar en el rango 1..32767, la verdadera cantidad de identificadores depende del tamaño de la tabla. También se dispone de una tabla de usuarios que define las aplicaciones que han sido cargadas en la plataforma. Cada aplicación tiene un número único sobre la plataforma y, en cada nodo, se le asigna un medio de comunicación y sincronización con el TPVP.

3.1 Diseño del módulo de comunicaciones

Una de las tareas fundamentales del sistema de comunicaciones es realizar el proceso de arranque, en el cual se sincronizan y conectan las diferentes máquinas que hacen parte de la máquina virtual.

Las máquinas se conectan dinámicamente durante el proceso de arranque o después, y la conexión se realiza estableciendo líneas virtuales (*stream sockets*) entre todas las máquinas. Cada máquina tiene un proceso que supervisa las conexiones con los demás nodos de la máquina virtual. Cada servidor de comunicaciones va a ser un proceso independiente de los procesos y *threads* de los usuarios.

Mediante las conexiones virtuales se realiza un protocolo de intercambio de mensajes con el cual se implementan los servicios de canales, de espera pasiva (ALT) y de memoria global. Estas conexiones también se usan en la operación y monitoreo de la plataforma.

3.1.1 Canales

Un identificador de un canal consiste de un número de nodo y dirección física. La diferenciación entre los identificadores de canal de las aplicaciones se realiza durante el enrutamiento de los paquetes hacia el TPVP. Los paquetes que llegan al módulo de comunicación se identifican solo como paquetes de uno u otro nodo.

El sistema de comunicaciones ofrece como servicio fundamental la creación de canales (sincrónicos) virtuales entre procesos. En la zona de memoria de un canal, se indica el número de nodo y manija del *thread* que realiza la operación complementaria, un apuntador a la zona de memoria donde se va a realizar la transferencia y un indicador del estado de espera pasiva (ALT).

El funcionamiento de los canales en modo ALT se basa en una tabla, que tiene cada aplicación, donde se relacionan los procesos que se encuentran en ALT. Dicha tabla indica si el proceso esta

en espera o si ya fue despertado. Además, se relaciona el canal que despertó el proceso o si fue por *timeout*.

3.1.2 Memoria Global

La máquina virtual ofrece la posibilidad de compartir zonas de memoria entre sus componentes. Para esto se utilizan apuntadores extendidos, estos, como los identificadores de canales, están compuestos por dos partes: el identificador del nodo y el identificador del bloque de memoria local. Los identificadores de memoria local son direcciones virtuales en el espacio de direcciones del proceso que contiene *threads* del usuario. No hay un un administrador de memoria en la plataforma, sino que el módulo de comunicación se encarga de esta labor..

El funcionamiento del sistema de memoria global trabaja con un sistma de "representación"; es decir, el sistema de comunicaciones es el "repreentante" de un proceso remoto, y se encarga de realizar la operación sobre la memoria y transmitir los resultados. Este mecanismo es sincrónico y bloqueante; además el sistema garantiza la exclusión y, por consiguiente, la consistencia durante una lectura o escritura en la memoria global.

El funcionamiento de las e/s en memoria global se apoya en información enviada al nodo donde se desea leer un bloque de memoria. Un requerimiento de lectura o escritura sobre un bloque de memoria global lleva una manija que permite despertar el *thread* que realiza la operación; además, lleva un número de nodo origen y un indentificador de memoria global con los cuales se puede accesar una dirección de memoria de la misma aplicación en otro nodo.

3.1.3 Espera Pasiva (ALT)

El sistema de espera pasiva facilita la implantación de la instrucción ALT de OCCAM. Los eventos que pueden despertar un proceso en espera pasiva se manejan de manera similar a como lo hace el Transputer [9].

El evento más importante que puede esperar un ALT es una sincronización vía un canal con otro proceso. Por ello, los canales tienen un espacio en su palabra de control que indica si el canal se encuentra en espera pasiva o no.

En la tabla de ALT que maneja el sistema de comunicaciones, se encuentra la dirección del identificador del canal que despertó al ALT; el usuario se encarga de borrar la entrada del canal que despertó al ALT. Las esperas pasivas pueden tener *timeouts*; estos se implementan utilizando los servicios de manejo de relojes del sistema.

4. Procesos

La plataforma está diseñada para ser construida sobre sistemas que ofrezcan multitarea y tengan una filosofía *preemptive* para manejo de procesos. Desde este punto de vista, el despachador global del sistema se refiere a la construcción de un protocolo de comunicación que permita ejecutar procesos en otras máquinas de la plataforma.

En esta primera versión de la plataforma paralela, y cumpliendo estrictamente con las necesidades de un lenguaje tipo OCCAM, se implementaron servicios de lanzamiento remoto de procesos estáticos desde el punto de vista de cada programa. Es decir, en compilación, se decide en cuál procesador será corrido cada proceso. Esto para disminuir muchos de los problemas que surgen al tratar de pensar en lanzamiento dinámico de procesos remotos (v.g. problemas de relocalización, de compartición de código entre diferentes áreas de direccionamiento, etc).

El despachador global del sistema es un "representante" en cada máquina de las otras máquinas conectadas a la plataforma. Este representación se comporta como un *thread* que recibe ordenes por la red y de acuerdo con ellas lanza *threads* o corre procedimientos.

4.1 Diseño del cargador

El código objeto de los *threads* que se ejecutarán remotamente, se transmiten en el momento de carga. Así, una vez iniciada la ejecución, el código de los *threads* remotos se carga y ejecuta

fácilmente en los nodos remotos. Cada nodo de la máquina virtual tiene una tabla con los *threads* remotos que está ejecutando y con los *threads* remotos que ha ordenado ejecutar; lo cual permite guardar el padre de cada *thread* remoto para poder enviarle un código de terminación. El código de terminación indica la manera como terminó el *thread* y, eventualmente, serviría para implementar sincronización de barrera. La carga y ejecución locales son manejadas por el sistema operacional.

El manejo de identificadores se realiza utilizando las manijas globales de *threads* de WIN32; dentro de la plataforma se les adiciona un número de nodo dentro de la máquina virtual.

4.1.1 Procesos Remotos

Como se ha mencionado anteriormente, los procesos de la plataforma se identifican por una manija extendida;. la cual se compone de una manija local, provista por la plataforma, e identificador del *thread* en alguna de las máquinas. El número de nodo o de entrada en la lista de máquinas de la plataforma permite al API determinar si debe invocar al sistema de comunicaciones de otra máquina para que este se encargue de lanzar el proceso.

La base del funcionamiento del mecanismo global de manejo de procesos de la plataforma está en las restricciones que se aplican a los procesos remotos. Estas restricciones son:

- El lugar (nodo) de ejecución de cada proceso será determinado en compilación
- El código de los procesos remotos debe estar aislado y compilado en ejecutables independientes.
- La sincronización y compartición de información debe ser realizada utilizando los mecanismos ofrecidos por el sistema de comunicaciones.

5. Ejecutables de la plataforma.

Un programa ejecutable para la plataforma se compone de varios módulos. Estos son distribuidos sobre la plataforma paralela se según un archivo de configuración de programa.

Luego de enviar a cada nodo el código de los *threads* que serán lanzados allí, se inicia la creación de procesos y *threads* (inicialmente dormidos). En la medida que se van creando, se va reportando su creación al nodo donde se inició la carga del programa. Cuando todos los *thread* han sido creados, se envía un mensaje a cada nodo indicándole que despierte el *thread.*

El manejo de la pila se deja, en su mayor parte, al sistema así como el manejo local de *threads.* La extensión de la existencia de un *thread* hacia toda la plataforma implica el encadenar de alguna manera los threads. Para esto se utiliza la tabla de procesos remotos y padres de procesos remotos en cada nodo.

7. Bibliografía

1. Bridges, Patrick. Doss, Natha. Gropp, William. Karrels, Edward. Lusk, Ewing. Skjellum, Anthony. *User's Guide to mpich, a Portable Implementation of MPI.* Argonne National Laboratory. 1995
2. Dijkstra, Edsger W. *Structured Programming.* Academic Press, New York. 1972
3. Davis, Ralph. *Windows Network Programming.* Addison Wesley. 1993
4. Geist, Al. *PVM 3 user's guide and reference manual.* Oak Ridge National Laboratory. Sep./1994
5. Graham, Ian. King, Tim. *The Transputer Handbook Inmos Ltd.* 1990
6. Jones, Geraint. *Programming in OCCAM.* Prentice Hall.1987
7. Microsoft Certified Professional Course 484. *Windows Operating Systems and Services Architecture.* Microsoft Press. 1995
8. Microsoft Corporation. *Development Library.* Microsoft Corporation. Julio 1995.
9. Shepard, David. *The Transputer Compiler Writer's Guide.* Inmos Ltd. Feb/1987.