Simulación de Arquitecturas de Memoria en Multicomputadores

Benjamín Sahelices Fernández, Diego R. Llanos Ferraris, Agustín de Dios Hernández.

Resumen— **En este artículo se explican las dificultades que los autores encontraron en el desarrollo de un entorno de simulación multicomputador para arquitecturas de memoria compartida distribuida en multicomputadores. Se tratan algunos de los principales problemas que se plantean en este tipo de simulaciones, como son la dificultad de procesar cargas de trabajo representativas por la gran cantidad de recursos que requieren, el tratamiento de las operaciones de sincronización entre procesadores o el entrelazado de las solicitudes de acceso a memoria que realizan los diferentes procesadores. Se discuten algunas de las soluciones empleadas con sus ventajas e inconvenientes y se muestran algunos resultados obtenidos en simulaciones.**

Palabras clave— **Simulación basada en ejecución, simulación basada en traza, memoria compartida distribuida, COMA, multicomputadores.**

I. Introducción

La diferencia entre el tiempo de ciclo de procesador y el tiempo de acceso a la memoria principal es una variable que ha estado incrementándose de forma contínua en los últimos años y que se prevé continuará haciéndolo en los próximos. Esto ha motivado el desarrollo de memorias intermedias rápidas que eviten en lo posible el acceso a memoria principal. La organización de estas memorias caché, la política de gestión y su optimización constituyen un campo de estudio muy activo actualmente por lo que ello implica en el incremento de la velocidad de procesamiento de los computadores.

Los sistemas de memoria son muy sensibles al cambio de cualquiera de sus parámetros, por ejemplo, un pequeño incremento o decremento de la tasa de faltas de acceso a caché se traduce en grandes variaciones en las prestaciones de acceso a memoria por el elevado número de accesos que se realizan por segundo. Consecuentemente diferentes técnicas de simulación han sido utilizadas para la evaluación de nuevos diseños o simplemente modificaciones de los existentes.

Cuando el computador está formado por múltiples procesadores, cada uno de ellos con su propia memoria local (además de las correspondientes memorias caché) y se define un espacio compartido de direcciones que engloba a todos los procesadores el problema mencionado anteriormente se agudiza. La razón es que en estos sistemas de memoria compartida distribuida (DSM, [1], [2], [3]) una falta de acceso a memoria caché puede involucrar en el mejor de los casos un acceso a memoria principal local (o a su propia caché [4]), pero puede también provocar el acceso a una memoria remota a través de una red de comunicaciones gestionada por un cierto protocolo de coherencia ([5], [6], [7], [8]). Como consecuencia, si lo sistemas no están perfectamente ajustados, se corre el peligro de que las ventajas de disponer de múltiples procesadores sean anuladas por un elevado tiempo de acceso a memoria que provoque que dichos procesadores estén parados durante un porcentaje de tiempo excesivo.

La evaluación de rendimiento de sistemas de memoria compartida distribuida tiene un gran interés como pieza básica para estimar rendimientos en sistemas multiprocesadores y multicomputadores. En este artículo se realiza un estudio del método que se ha utilizado para averiguar el rendimiento de los sistemas propuestos. Se pretende discutir sus características, los problemas encontrados y las soluciones y resultados obtenidos. En el siguiente apartado se describe la técnica de simulación basada en ejecución, a continuación se trata la obtención de la traza para los sitemas de procesamiento de traza que son descritos en el último apartado.

II. Simulación Basada en Ejecución

El objetivo que se plantea es la evaluación del comportamiento de un sistema de memoria en una red de estaciones de trabajo con memoria distribuida sobre la que se define un espacio compartido de direcciones a través de un protocolo de coherencia de tipo COMA ([5],[9]). Cada procesador dispone de un espacio de direcciones estrictamente local en el que puede almacenar todo tipo de variables no compartidas y un espacio de direcciones compartido que sirve para la comunicación y sincronización con el resto de los procesadores.

El sistema de simulación inicialmente desarrollado se basa en ejecución. El principal criterio utilizado para elegir este método de simulación fue la exactitud. Para conseguirla la carga de trabajo debe ser representativa, lo cuál se consigue mediante la ejecución de programas paralelos completos y mediante la utilización de datos reales. Este método permite implementar un elevado nivel de detalle en los módulos de simulación de la memoria y del protocolo de coherencia.

El segundo criterio utilizado fue la minimización del retardo de simulación, definido como el tiempo necesario para realizar la simulación comparado con el tiempo que tarda el programa paralelo en ejecutarse. Para conseguirlo se implementó un simulador capaz de ejecutarse en múltiples computadores simultáneamente que cooperan en la ejecución de una única simulación. El efecto real conseguido no fue el esperado, ya que en realidad lo que se consigue re-

Departamento de Informática, Universidad de Valladolid, Campus Miguel Delibes, 47010 Valladolid (España). Correo electrónico: {benja,diego,agustin}@infor.uva.es.

partir son los elevados requisitos de almacenamiento primario extra que necesita el simulador más que los ciclos de CPU. Es decir, la utilización de múltiples computadores permite ejecutar simulaciones de más procesadores pero no acorta significativamente los tiempos de ejecución.

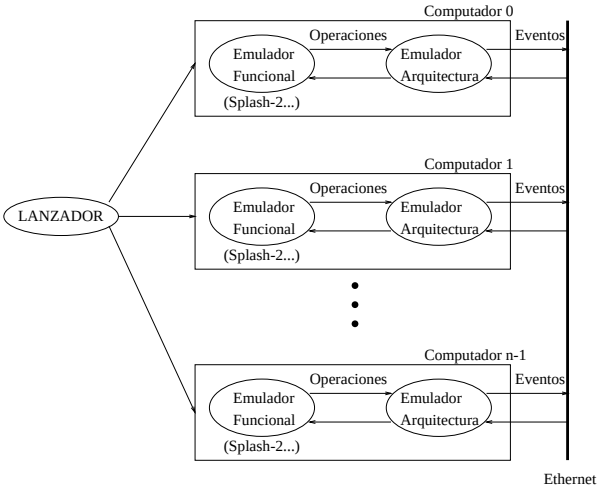

Fig. 1. Estructura de Procesos del Simulador Basado en Ejecución

El sistema desarrollado se muestra en la figura 1 ([5]). Cada procesador se modela mediante dos procesos UNIX, uno representa la aplicación paralela (EmuladorFuncional) y el otro el sistema de memoria (EmuladorArquitectura). Se consigue así reproducir el funcionamiento del sistema real teniendo un módulo funcional independiente que representa el procesador y otro que representa el controlador de coherencia, encargado de gestionar los accesos remotos. Este segundo módulo puede añadir el nivel de detalle que se precise, modelando por ejemplo una caché del espacio compartido de direcciones existente en cada procesador (independiente de las caché estándar utilizadas por el procesador para el acceso al espacio local de direcciones).

La razón por la que se escogen procesos UNIX para modelar las diferentes unidades funcionales es permitir la máxima flexibilidad en la ejecución en una red de computadores. Se utiliza un sistema de paso de mensajes (PVM [10]) para la comunicación evitándose así el uso de memoria compartida para que los procesos no tengan por qué convivir en un único computador. Esto impide la utilización de threads para el modelado de las unidades funcionales.

La solución propuesta plantea algunas cuestiones. Por ejemplo cómo afecta la planificación de los diferentes procesos al resultado final de la simulación. Es decir, se están utilizando procesos que comparten una única CPU para simular el comportamiento de procesadores y para simular módulos que actúan de forma simultánea a los procesadores. Todos los procesos comparten un conjunto de recursos entre los que se encuentra la CPU ya que en el caso extremo todos se ejecutan en un único computador bajo un único sistema operativo. La planificación es tarea del núcleo del sistema operativo y es no determinista ya que no depende únicamente de los procesos que conforman la simulación sino también del resto de procesos que se estén ejecutando.

Teniendo en cuenta las anteriores consideraciones se puede deducir que este tipo de simulación lleva implícita un error debido a la planificación. Es difícil cuantificar dicho error porque sería necesario compararlo con un sistema real, que en este caso no existe. La parte principal del problema surge en las barreras y semáforos, cuando todos los procesos menos uno han llegado a la primitiva de entrada y se produce un cierto retardo en la planificación del último proceso. Esto genera un mayor número de operaciones de acceso a memoria de tipo lectura para la sincronización, al ser estas las operaciones típicas de espera. Normalmente todas ellas producen éxitos de acceso a la caché minimizando consecuentemente su impacto.

El principal problema de la simulación basada en ejecución es la cantidad de recursos que necesita. Para el caso de un sistema formado por 16 procesadores con un espacio compartido de direcciones de 16-MBytes se generan un total de 33 procesos con unos requisitos totales de memoria de 512-MBytes sin tener en cuenta la memoria necesaria para los procesos. En el simulador desarrollado, los 33 procesos se pueden ejecutar en máquinas diferentes conectadas mediante una red de área local. La ventaja que supone el reparto de los ciclos de CPU entre los diferentes computadores se ve eclipsada por los retardos de comunicación implícitos en la utilización de una red de área local de propósito general. Los tiempos de simulación que se obtienen son muy elevados y surge así la necesidad de hacer evolucionar el esquema de simulación basada en ejecución conservando su principal virtud, la exactitud.

III. Simulación Basada en Traza

Partiendo de la simulación basada en ejecución se implementó un sistema de generación y procesado de traza, obteniéndose ésta directamente de los accesos realizados durante la ejecución. Las ventajas que proporciona la traza son las siguientes:

- Reducción de los ciclos de CPU necesarios. Una vez generada la traza únicamente se realiza su procesado con los módulos que simulan el comportamiento de la arquitectura de memoria simulada. Por lo tanto no se ejecutan una y otra vez las diferentes aplicaciones mientras se simula el funcionamiento de la memoria.
- Reducción de la memoria necesaria. Al no ejecutar las aplicaciones que forman el conjunto de trabajo, ya no es necesario gestionar los valores del espacio de memoria a simular, ahorrándose esa memoria en el módulo que representa a cada procesador.
- Independencia de plataforma. Es esta la principal ventaja de la simulación basada en traza. El módulo de procesamiento de traza únicamente gestiona las referencias a memoria almacenadas en los ficheros de traza. Esta tarea se desliga

completamente de la ejecución de la aplicación y la captura, mediante algún método dependiente de arquitectura, de las referencias a memoria. Esta independencia de plataforma permite simular en estaciones de trabajo desocupadas y lanzar múltiples experimentos simultáneamente.
- Capacidad de repetición y de utilización de subconjuntos de traza. Al haber sido generada la traza por la ejecución de una cierta aplicación con una cierta carga de trabajo, los experimentos son reproducibles. Además se han insertado puntos de chequeo de forma que se pueden utilizar partes de la traza para experimentación preliminar con pequeño tiempo de espera.

Los simuladores basados en traza se componen de tres módulos, generador, reductor y procesador de traza [11]. El sistema desarrollado no tiene en estos momentos un módulo reductor de traza. A pesar que la traza obtenida no es completa, lo cuál puede ser considerado como una forma de reducir la traza, no se descarta en el futuro insertar algún método adicional de reducción [11].

A. Generación de Traza

El método de generación de traza usado es el de anotación de código estático ([11]). Se basa en interrumpir el proceso de generación de programas en el nivel de código fuente en ensamblador y expandir dicho código (ver figura 2) de forma que se capturan todas las instrucciones. El sistema desarrollado analiza el código ensamblador para la arquitectura SPARC-v9 insertando las instrucciones necesarias de invocación a una función para cada instrucción en ensamblador. Esta técnica es fácil de implementar aunque la obtención de traza es incompleta ya que no se obtienen las referencias realizadas por el sistema operativo ni por las bibliotecas, tanto estáticas como dinámicas. Al ser el objetivo la simulación de un sistema multicomputador en la ejecución de aplicaciones paralelas se puede considerar secundario el hecho de no capturar al sistema operativo, mientras que se debe evitar la utilización de bibliotecas, insertando todo el código directamente en el segmento de texto.

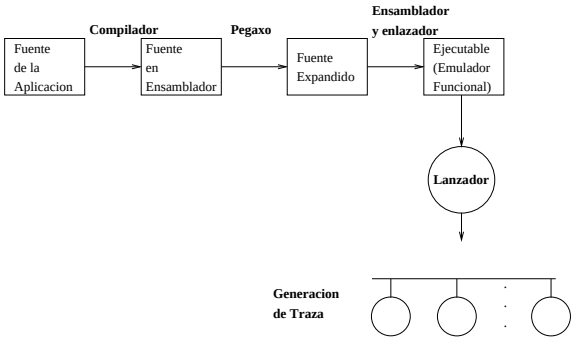

Fig. 2. Anotación de Código Estático

El código de cada procesador, anotado según el método descrito, al ser ejecutado provoca la invocación a una cierta función para la ejecución de todas y cada una de sus instrucciones. Dicha función determina el tipo de instrucción y, en caso de que sea de acceso a memoria, determina si la dirección entra dentro del rango de direcciones compartidas. Si se da este último caso se escribe una línea de traza en un fichero. Existe un fichero por cada procesador para almacenar la traza. Cada línea del fichero tiene un primer campo conteniendo el tipo de operación de memoria. En estos momentos hay cuatro tipos de operaciones, la lectura (**L**), la lectura de sincronización (**LS**), la escritura (**E**) y la escritura de sincronización (**ES**). En un futuro está previsto insertar nuevas operaciones explícitas de sincronización, aunque esto depende del protocolo de coherencia utilizado. El segundo campo de cada línea contiene la dirección del espacio compartido de direcciones al que se accede. El tercer campo es el tamaño de acceso. El cuarto campo es una marca temporal que se explicará posteriormente y el quinto es el número de instrucciones de todo tipo ejecutadas entre el acceso actual y el siguiente acceso al espacio compartido.

Existen algunas características específicas de la anotación de código estático en multiprocesadores que conviene analizar en relación al sistema desarrollado:

- Discontinuidades. Un sistema generador de traza tiene discontinuidades si no es capaz de capturar todas las referencias en la ejecución completa de la aplicación por limitaciones del método o de la implementación. No se da en el sistema realizado ya que se expanden todas las instrucciones y la traza se almacena en ficheros.
- Dilatación del tiempo. El hecho de expandir el código supone la ejecución de más instrucciones que en el programa paralelo original. Estos programas llegan a ir de 10 a 30 veces más lentos en sistemas monoprocesador (más lentos aún en sistemas multiprocesador). La dilatación del tiempo hace que dispositivos lentos aparezcan como muy rápidos, por ejemplo dispositivos de E/S, que llegan a interrumpir un número relativo de veces mucho mayor. Este problema no se da en el sistema implementado ya que está centrado en la ejecución de un único programa paralelo.
- Dilatación de la memoria. La expansión del código hace que el sistema de memoria virtual se comporte de forma diferente, por ejemplo el TLB tendrá valores diferentes. Hay que tener en cuenta que las direcciones obtenidas en la traza son virtuales de forma que lo que se altera es el patrón de accesos a memoria física. Este tampoco es un problema cuando lo que se pretende, como es el caso, es simular únicamente el patrón de accesos a una porción del espacio virtual considerada como memoria compartida y con función de mapa directa sobre una memoria física.

Hay dos aspectos especialmente relevantes en la simulación multiprocesador basada en traza, que son

el entrelazado de operaciones entre los procesadores y las operaciones de sincronización.

A.1 El Problema de las Marcas Temporales

Una de las principales diferencias entre la generación de traza en monoprocesadores y multiprocesadores es la necesidad de entrelazar las solicitudes de acceso a memoria. Mientras que en un sistema monoprocesador es suficiente con reproducir dichas operaciones de acceso según se han ido produciendo, en los multiprocesadores es necesario generar dichas operaciones de forma intercalada entre todos los procesadores, tal y como fueron realizadas en realidad. De no hacerlo de esta forma la traza no reproducirá el comportamiento de la aplicación paralela, es decir, el patrón temporal de acceso a los datos o la competencia en la compartición de una cierta variable, entre otras cosas.

Para conseguir un correcto entrelazado de operaciones, se ha insertado una marca temporal en cada línea de los ficheros de traza generados. Esta marca debe indicar el instante en el que se produjo dicha referencia de forma relativa al resto de referencias realizadas por el mismo procesador y por el resto de procesadores. La obtención de una marca temporal exacta en sistemas multicomputador es complicado al no existir un reloj único para todos ellos. Por ello la generación de traza, en este momento, debe ser realizada en un único computador y se utilizan los servicios de su sistema operativo para conseguirla.

Este método no está libre de errores ya que la obtención de la marca temporal supone invocar una primitiva de servicio del sistema operativo. Esta invocación está formada por una interrupción software, un cambio de contexto, el cambio a modo privilegiado del procesador, la ejecución de la rutina correspondiente, un nuevo cambio de modo del procesador y un nuevo cambio de contexto. Todo ello contribuye a distorsionar el instante obtenido. Además se incrementa la probabilidad de que el proceso sea desalojado y se planifique un nuevo proceso correspondiente a un nuevo procesador incrementándose aún más la distorsión de la marca temporal obtenida.

La utilización de un modelo de consistencia secuencial junto con la ejecución real de las aplicaciones para la obtención de la traza minimiza el error descrito. Aunque las marcas temporales estén sujetas a variaciones reflejarán el orden en el que se ha accedido a los datos en las aplicaciones ejecutadas y aunque ese orden no sea único sí es correcto ya que se han obtenido los datos en el momento en que eran necesitados y dichos datos eran datos correctos. El error generado en las marcas temporales se referirá únicamente a aquellos accesos a datos realizados simultáneamente y sin sincronización. El hecho de no existir dicha sincronización implica la irrelevancia de mantener un orden en el acceso a los datos y por lo tanto se minimiza el impacto del error de obtención de las marcas. En la siguiente sección se realiza una comparación entre los datos obtenidos en la simulación basada en ejecución y en traza mostrándose lo mínimo de las diferencias existentes para los experimentos realizados.

A.2 El Problema de las Operaciones de Sincronización

Respecto a las operaciones de sincronización, el tratamiento no es evidente. Se han considerado las siguientes posibilidades:

1. Anotar en los ficheros de traza la ocurrencia de cada operación de sincronización junto con su tipo. Cada una de ellas puede ser identificada fácilmente a través de la posición de memoria en la que se localiza. El problema es identificarlas en tiempo de ejecución, es decir, si por ejemplo una barrera está dentro de un bucle la pregunta es cómo se sabe si la ejecución actual de la barrera se corresponde con la ejecución 25 ó 37 (o cualquier otra) de dicho bucle. Para conseguirlo se tendría que realizar una identificación en tiempo de ejecución, cuestión compleja.
   Supongamos que se pudiera realizar esta identificación. En este caso todos los emuladores de arquitectura tendrían que comportarse como indican las diferentes operaciones recibidas de los ficheros de traza, es decir, tendrían que agruparse en barreras, esperar en la entrada de una sección crítica ocupada, etcétera. Sin embargo todas estas tareas no son propias del módulo de memoria, considerando que el comportamiento de éste se debe basar en recibir peticiones y responder.
2. *Desenrrollar* las operaciones de sincronización en las diferentes operaciones sencillas que las componen. Para esta solución los ficheros de traza deberán contener todas las solicitudes realizadas a los módulos de memoria. Estas solicitudes son por lo tanto las que es capaz de resolver el protocolo de coherencia y dependerán en gran medida de éste. Quiere esto decir que si el protocolo no es capaz de implementar ninguna operación atómica de carga y modificación (caso del protocolo de coherencia implementado en el momento actual), entonces todas las operaciones de sincronización se deben realizar a través de algoritmos del tipo del de Peterson, que al final se van a traducir en solicitudes de lectura/escritura al espacio compartido de direcciones.
   Los ficheros de traza contendrán por lo tanto el conjunto de operaciones que cada procesador tuvo que realizar para conseguir sincronizar en la ejecución en la que se obtuvo la traza, junto con su correspondiente marca temporal. Si se vuelve a obtener una nueva traza en una nueva ejecución no habrá coincidencia ya que es posible que un procesador tarde más o menos en diferentes ejecuciones en llegar a una barrera debido a la planificación utilizada. Sin embargo las diferencias no son representativas ya que, como se ha explicado previamente, estas diferencias se traducen en un número mayor o menor de lecturas

de espera a que una cierta variable sea modificada.

Es útil la obtención de la traza *desenrrollada* para las operaciones de sincronización ya que así se consigue que en la simulación posterior todos los procesadores sean independientes tal y como ocurre en una simulación basada en traza típica.

B. Procesamiento de Traza

El objetivo de la simulación basada en traza es conocer el rendimiento de un cierto sistema, en nuestro caso una cierta arquitectura de memoria en un sistema con varios computadores. Para conseguirlo es preciso procesar la traza múltiples veces para diferentes configuraciones de la memoria. Es esta la etapa más larga de todo el proceso de simulación ya que el número de parámetros a evaluar puede llegar a ser elevado.

En esta etapa es donde aparecen los diferentes elementos a simular hasta el nivel de detalle deseado. En el sistema desarrollado el principal objetivo es la evaluación del protocolo de coherencia siendo su principal cuello de botella la red de comunicaciones que conecta los diferentes computadores entre sí. El módulo de procesamiento recibe las operaciones de acceso al espacio compartido de direcciones y las resuelve bien de forma local, bien de forma remota. En este último caso se utiliza la red de comunicaciones y el protocolo de coherencia, los cuáles deben ser también modelados.

TABLA I
COMPARACIÓN DE FALTAS DE ACCESO EN SIMULACIÓN BASADA EN EJECUCIÓN Y EN TRAZA (FFT CON 8 PROCESADORES)

Núm.Proc.	Ejecución	Traza
Proc. 0	7468	7470
Proc. 1	9809	9760
Proc. 2	9884	9828
Proc. 3	9771	9722
Proc. 4	9769	9732
Proc. 5	9835	9778
Proc. 6	9711	9685
Proc. 7	9781	9743

En la tabla I se muestran los resultados preliminares obtenidos en la simulación basada en traza respecto a la simulación basada en ejecución. En particular se muestran las faltas de acceso al espacio compartido de direcciones obtenidas con ambos métodos. Se puede observar cómo los resultados son muy similares.

IV. CONCLUSIONES

La principal dificultad en la simulación de sistemas multiprocesadores es la gran cantidad de recursos que son necesarios para modelar todos los procesadores junto con el resto de sus unidades funcionales. Si además se pretende disponer de una carga de trabajo representativa de aplicaciones paralelas, la cantidad de recursos necesarios se incrementa aún más ya que las aplicaciones paralelas suelen consumir muchos recursos de memoria y CPU. Se ha mostrado en este artículo el camino utilizado por los autores para simular sistemas formados por varios procesadores con modelos de memoria de tipo compartido distribuido.

La simulación basada en ejecución requiere una cantidad de recursos excesiva por lo que el camino elegido consistió en generar traza de referencias a memoria compartida distribuida a partir del sistema de simulación basado en ejecución. La traza obtenida lleva asociadas marcas temporales para la resolución del problema del entrelazado de direcciones. Las operaciones de sincronización se descomponen en sus operaciones básicas de lectura y escritura, que junto a las marcas temporales permiten reproducir el comportamiento de barreras y semáforos en tiempo de ejecución.

Los resultados obtenidos hasta la fecha muestran gran concordancia entre las simulaciones basadas en ejecución y las basadas en traza. Estas últimas tienen la ventaja de consumir menos memoria, menos ciclos de CPU, ser independiente de plataforma y tener capacidad de repetición.

REFERENCIAS

[1] D. Culler, J.P. Singh, and A. Gupta, *Parallel Computer Architecture*, Morgan Kaufmann, 1996.

[2] D. Lenoski and W.D. Weber, *Scalable Shared Memory Multiprocessing*, Morgan-Kaufmann Publishers, 1995.

[3] S. Raina, *Emulation of a Virtual Shared Memory Architecture*, Ph.D. thesis, U. of Bristol, UK, 1993.

[4] C. Schimmel, *UNIX Systems for Modern Architectures. Symmetric Multiprocessing and Caching for Kernel Programmers*, Addison-Wesley Professional Computing Series, Reading, MA, 1994.

[5] B. Sahelices Fernández, *COMA-BC: una Arquitectura de Memoria Sólo Cache en Bus Común no Jerárquica*, Ph.D. thesis, Departamento de Informática. Universidad de Valladolid, May 1998.

[6] D.R. Llanos Ferraris, B. Sahelices Fernández, and A. de Dios Hernández, "Diseño de un protocolo coma en bus común distribuido con reemplazo," in *X Jornadas de Paralelismo. La Manga del Mar Menor, Murcia.*, 1999, pp. 241–246.

[7] S. Cho, J. Kong, and L. Gyungho, "Coherence and replacement protocol of dice - a bus-based coma multiprocessor," Tech. Rep., Dept. of Computer Science and Engineering. University of Minnesota.

[8] J. Truman, *COMA-F: A Non-Hierarchical Cache Only Memory Architecture*, Ph.D. thesis, Department of Electrical Engineering Stanford University, 1995.

[9] B. Sahelices Fernández, D.R. Llanos Ferraris, and A. de Dios Hernández, "Exploiting parallelism in a network of workstations," *ACM Computer Architecture News*.

[10] A. Geist, A. Beguelin, J.J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM - Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press, Cambridge, Massachusetts, 1994.

[11] R.A. Uhlig, "Trace-driven memory simulation: A survey," *ACM Computing Surveys*, vol. 29, no. 2, June 1997.