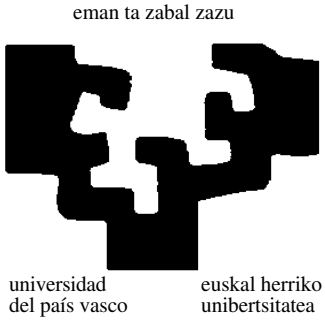
Euskal Herriko Unibertsitatea
Universidad del País Vasco

Konputagailuen Arkitektura eta Teknologia Saila
Dep. de Arquitectura y Tecnología de Computadores



INFORMATIKA FAKULTATEA
FACULTAD DE INFORMATICA

Una evaluación empírica de técnicas para simulación paralela de redes de paso de mensajes

Resumen de la memoria titulada "**An empirical evaluation of techniques for parallel simulation of message passing networks**" que para optar al grado de Doctor en Informática presenta

José Miguel Alonso

Dirigida por Ramón Beivide Palacio

San Sebastián, Octubre de 1995

Este trabajo ha sido parcialmente subvencionado por la Comisión Interministerial de Ciencia y Tecnología, bajo contrato TIC95-0378

Contenidos

Prefacio 1
1 Introducción 1
2 Simulación paralela de sistemas de sucesos discretos 3
3 Entornos de computación paralela 6
4 Una evaluación de técnicas de simulación 8
5 Simulación de redes de paso de mensajes 11
6 Prestaciones de los simuladores 14
7 Conclusiones y líneas abiertas 16
A Sistemas de computación paralela usados en este trabajo 19

Prefacio

Este documento es un resumen, en castellano, de la memoria de tesis doctoral "**An empirical evaluation of techniques for parallel simulation of message passing networks**", escrita originalmente en inglés. La estructura del mismo es paralela a la del documento original, de tal forma que cada sección de este resumen corresponde a un capítulo de la tesis. No se incluyen tablas ni referencias bibliográficas, y apenas aparecen gráficos. Tampoco se ofrecen datos cuantitativos. El resumen de cada capítulo hace más énfasis en las conclusiones del mismo que en el proceso que ha llevado a esas conclusiones. Con todo esto, el autor quiere clarificar que este documento no es autosuficiente, y que no aporta toda la información incluida en el original. Esto no debería de resultar sorprendente: un trabajo del calibre de una tesis doctoral tampoco es totalmente autosuficiente, sino que depende para su total comprensión del uso de las referencias bibliográficas en él incluidas. Aún con estas limitaciones, está en el ánimo del autor recoger aquí la mayor parte de la información de relevancia.

1 Introducción

La simulación por computador es una herramienta básica en muchos campos de las ciencias y las ingenierías, para complementar (y, a veces, sustituir) a otras técnicas como el estudio analítico y el desarrollo de prototipos. Desafortunadamente, la simulación de sistemas grandes y/o complejos es una actividad lenta, incluso usando los procesadores más avanzados existentes en la actualidad. Nuestro grupo de trabajo tiene experiencia de primera mano en este área, porque usa la simulación con frecuencia para la evaluación de propuestas arquitectónicas para el diseño de nuevos computadores paralelos,

especialmente en lo que se refiere a las redes de paso de mensajes que forman la infraestructura de comunicación de los mismos.

El uso de computadores paralelos se ha visto, desde hace bastante tiempo, como la forma más realista de acelerar la ejecución de programas. Esta aceleración ha resultado eficaz en el caso de aplicaciones numéricas, en las que se emplean estructuras de datos grandes y regulares. Sin embargo, explotar el paralelismo con otro tipo de aplicaciones no es, en absoluto, trivial. Los simuladores pueden considerarse en este segundo grupo.

El objetivo de este trabajo es buscar técnicas eficaces para la simulación paralela de redes de paso de mensajes. El interés de este trabajo radica en estos puntos:

- La aceleración de las simulaciones nos permitirá reducir el tiempo empleado en caracterizar el comportamiento de nuestras propuestas de diseño de redes de paso de mensajes. Muchos de los resultados serán, además, extrapolables a la simulación de otro tipo de sistemas.
- Se demostrará que los sistemas de computación paralela no son solamente útiles para aplicaciones numéricas, ampliando así el espectro de utilización de estos, en general costosos, sistemas.

Para desarrollar este trabajo se ha realizado un estudio de los algoritmos de simulación (secuenciales y paralelos) más habituales. Los algoritmos más representativos se han implementado en tres sistemas multicomputador comerciales, lo cual ha requerido un estudio exhaustivo de las herramientas de programación paralela disponibles en cada uno de ellos. Una vez realizadas estas implementaciones, se han tomado múltiples medidas para comprobar cómo afectan a las prestaciones de los simuladores aspectos tales como:

- El algoritmo de simulación empleado.
- Las características del computador utilizado.
- Las características del modelo que se simula.

2 Simulación paralela de sistemas de sucesos discretos

En este capítulo se presentan varios algoritmos de simulación paralela, y se seleccionan tres de ellos para su posterior evaluación.

Hay varias maneras de explotar el paralelismo para realizar trabajos de simulación:

- *Réplica*. Es la forma más obvia de explotar el paralelismo. Normalmente, un estudio de simulación requiere de múltiples ejecuciones del simulador, con diferentes parámetros de entrada. Se puede usar un computador paralelo para ejecutar, simultáneamente, una simulación en cada procesador. El uso que se hace de los recursos, puede resultar excelente, aunque esta técnica no siempre es posible o conveniente.
    - Puede darse el caso de que los procesadores no tengan suficiente memoria como para ejecutar, cada uno de ellos, una simulación completa.
    - Puede que no sea posible ejecutar simultáneamente varias simulaciones, porque los parámetros de una requieran de los resultados de la anterior. Esta situación es habitual si el objetivo del estudio es encontrar los valores óptimos para un conjunto de parámetros.
    - También puede ocurrir que se necesiten N simulaciones y se disponga de M procesadores, con M >> N. En este caso no se aprovecha todo el paralelismo disponible.

  Por lo tanto, resulta evidente que a veces es necesario acelerar *una* simulación.

- *Descomposición funcional*. Se trata de utilizar diferentes unidades funcionales para realizar simultáneamente las tareas necesarias en un simulador: generación de números aleatorios, gestión del calendario de sucesos, entrada/salida, ejecución de los sucesos, representación gráfica,

etc. El grado de paralelismo que se explota no es muy elevado, dado el escaso número de tareas identificables, y las necesidades de coordinación entre ellas.

- *Descomposición del modelo*. Se trata de descomponer el modelo a simular en un conjunto de partes o sub-modelos, asignando la simulación de cada una de esas partes a un *proceso lógico* (*PL*). Esos procesos lógicos se ejecutan en los procesadores disponibles. La descripción del modelo se distribuye entre los diferentes procesadores, por lo que el consumo de memoria en cada uno de ellos es razonablemente reducido. Además, si el modelo es suficientemente grande, se puede distribuir de tal forma que utilice todos los procesadores disponibles.

La última técnica mencionada parece la más prometedora, ya que es la única que hace posible la simulación de modelos de gran tamaño en computadores masivamente paralelos. Por este motivo, en este estudio sólo consideramos los métodos de simulación paralela basados en descomposición del modelo.

Para que una simulación paralela basada en descomposición del modelo sea realmente operativa, no es suficiente que cada sub-modelo se simule de forma concurrente con los demás. En un sistema real, las diferentes componentes del mismo interaccionan de tal forma que hay fuertes relaciones causa-efecto entre los sucesos que afectan a cada componente. El simulador paralelo debe recoger esta característica. Dicho de otra forma, los PLs que componen el simulador deben interaccionar para reflejar no sólo el progreso del sub-modelo que simulan, sino también la evolución del modelo *global*. Esto requiere que los PLs se coordinen entre ellos, de tal manera que las relaciones causales entre los sucesos que afectan a cada sub-modelo se respeten.

Esta coordinación se puede realizar de diferentes maneras. La más restrictiva es hacer que los PLs trabajen de forma síncrona, es decir, compartiendo la misma visión del transcurso del tiempo (simulado). A efectos prácticos, esto significa que sólo se simulan en paralelo aquellos sucesos que en el sistema real ocurrirían simultáneamente. Esto lleva al algoritmo SPED (*Synchronous Parallel Event-Driven*). Esta técnica puede dar buenos resultados

si la densidad de sucesos es elevada. Esta densidad se define como el número medio de sucesos con la misma marca de tiempo. La marca de tiempo de un suceso indica el momento en el que ese suceso debería de ocurrir en el sistema real.

Las limitaciones de SPED sugieren que se puede explotar mejor el paralelismo en la simulación si se simulan a la vez sucesos con diferentes marcas de tiempo. Esto es posible si se determinan las relaciones causales entre sucesos, que son, en general, menos restrictivas que las relaciones temporales: dos sucesos con diferentes marcas de tiempo pueden simularse simultáneamente si ninguno de ellos influencia, de forma directa o indirecta, al otro. Los métodos de coordinación *asíncronos* siguen esta aproximación. Cada PL tiene su propio reloj, y decide de forma autónoma qué hacer con los sucesos que afectan a su parte del sistema, aunque debe hacerlo sin introducir errores causales. Existen dos grandes familias de algoritmos asíncronos:

- Conservadores (CMB-DA). Estos simuladores evitan la aparición de errores causales. Un PL que sigue esta estrategia solamente avanza en la simulación cuando está seguro de que no supone ningún riesgo. Mientras tanto, permanece bloqueado. La sincronización basada en bloqueos puede llevar a situaciones en las que la simulación no avanza, porque se forman bucles de PLs interbloqueados. Se necesita, por tanto, un mecanismo para evitar los interbloqueos. Uno de los más habituales es el uso de *mensajes nulos*, que no llevan información sobre la simulación pero que transportan información sobre el transcurso del tiempo en los PLs, ayudando a estos a avanzar sus relojes. El procesamiento de mensajes nulos supone una sobrecarga de trabajo importante, que en ocasiones puede superar al empleado para el procesamiento de los mensajes (sucesos) que forman realmente parte de la simulación. Un buen uso de una propiedad que pueden tener los sistemas a simular, denominada *lookahead*, puede redundar en una drástica reducción en el número de mensajes nulos necesarios. Esta propiedad es una medida de lo capaz que es un PL de predecir el comportamiento futuro del sistema que está simulando. Desafortunadamente, el *lookahead* está estrechamente relacionado con las

características concretas del modelo simulado, por lo que no existen técnicas generales para su extracción.

- Optimistas (TW). Estos simuladores *permiten* la aparición de errores causales. Sin embargo, estos errores siempre son detectados, de tal manera que un PL puede realizar un proceso de *vuelta atrás* y llevar la simulación de nuevo a una situación libre de errores. Para realizar este mecanismo de vuelta atrás es necesario que cada PL guarde información histórica sobre su estado en diferentes puntos del pasado. Este es uno de los problemas de esta técnica: el almacenamiento de la historia de un PL puede necesitar de cantidades enormes de memoria. Por otra parte, se necesita un mecanismo adicional para determinar el estado global de la simulación, así como para recuperar memoria eliminando copias pasadas del estado del PL que ya no van a ser necesarias porque nunca se va a volver atrás hasta ellas. El tiempo empleado en la gestión de la información histórica, más el control global de la simulación, suponen una sobrecarga de trabajo importante.

3 Entornos de computación paralela

Uno de los problemas que plantean los sistemas de procesamiento paralelo disponibles en la actualidad es la enorme diversidad que existe tanto desde un punto de vista arquitectónico como en lo que se refiere a modelos de programación paralela. En este capítulo se hace referencia a estas diferencias.

Desde el punto de vista de la programación de aplicaciones paralelas, el programador tiene que prestar atención a estos aspectos, entre otros:

- Modelo de paralelismo. Una aplicación paralela puede consistir en múltiples procesos que ejecutan concurrentemente diferentes programas, cada uno operando sobre su propio conjunto de datos (*MIMD, Multiple Instruction Multiple Data*). Alternativamente, puede consistir en un único

programa que se ejecuta, instrucción a instrucción, en múltiples procesadores, aunque en cada caso las operaciones afectan a datos diferentes (*SIMD, Single Instruction Multiple Data*). Un caso particular de MIMD es SPMD (*Single Program Multiple Data*), en el que todos los procesos ejecutan el mismo programa, aunque no necesariamente a la vez.

- Paradigma de comunicación. Un conjunto de procesos concurrentes necesita de algún mecanismo para comunicación y sincronización. Los más habituales son memoria compartida y paso de mensajes. En el primer caso todos los procesos ven el mismo espacio de memoria y se comunican usando variables compartidas; para la sincronización, se necesitan mecanismos tales como los semáforos. En el segundo caso, cada proceso tiene su propio espacio de memoria, y la comunicación-sincronización se hace enviando y recibiendo mensajes. En este trabajo nos hemos centrado en este segundo paradigma de comunicación.
- Semántica de las operaciones de envío y recepción de mensajes. La comunicación por medio de mensajes puede llevar asociada cierta sincronización: éste es el caso cuando un proceso se bloquean hasta que una operación de comunicación termina—comunicación bloqueante. Alternativamente, una operación de comunicación puede ser no bloqueante, debiendo existir en este caso un mecanismo aparte para determinar cuándo ha finalizado.
- Identificación del interlocutor. Las alternativas son: canales entre pares de procesos, donde la identificación es implícita, o uso de direcciones, donde la identificación debe ser explícita.

En lo que se refiere al hardware, los sistemas paralelos suelen clasificarse en dos grupos:

- Multiprocesadores: sistemas paralelos con memoria compartida. Un conjunto de procesadores se comunican, a través de una red de interconexión, con un conjunto de módulos de memoria.

- Multicomputadores: sistemas paralelos con memoria distribuida. Múltiples nodos <procesador, memoria> se comunican entre sí, gracias también a una red de interconexión.

En ambos casos, hay que prestar atención tanto al diseño de los nodos como al diseño de las redes de interconexión, aunque los requerimientos son diferentes en cada caso. En este estudio nos centramos en los multicomputadores.

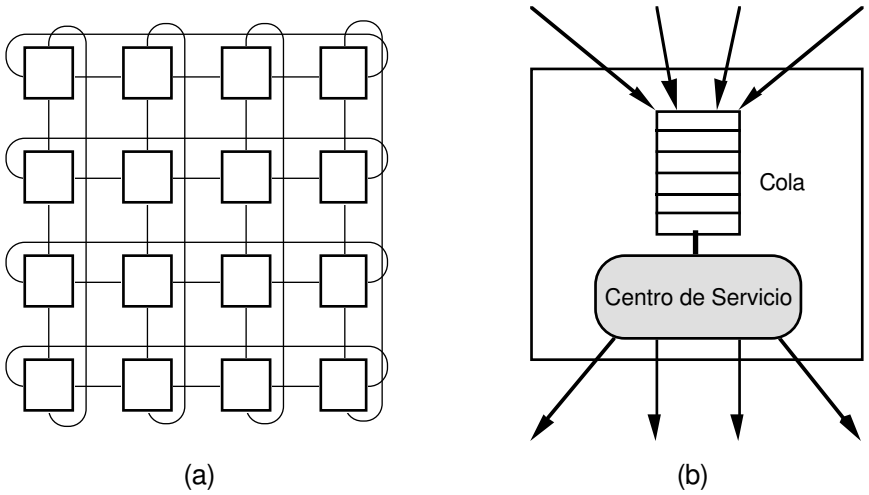
Cada nodo de un multicomputador debe tener: un procesador (a veces, más de uno), memoria y alguna forma de comunicación (intercambio de mensajes) con el resto de los nodos. En los multicomputadores de primera generación las tareas de gestión de mensajes las realizan los mismos procesadores encargados de realizar cómputo. En los sistemas actuales cada nodo tiene un *encaminador*, elemento hardware especializado en la gestión de las comunicaciones entre nodos: el procesador entrega mensajes al encaminador y éste, cooperando con el resto de encaminadores, se encarga de que esos mensajes lleguen a sus destinos. Un buen diseño del encaminador es, por lo tanto, fundamental para obtener un alto rendimiento.

La red de interconexión, formada por el conjunto de encaminadores y los enlaces de comunicaciones que los unen, debe intentar minimizar la *latencia* de los mensajes y, a la vez, maximizar el *throughput* (número de mensajes gestionados por unidad de tiempo). Para conseguir satisfacer estos requisitos, el diseño debe considerar cuidadosamente características tales como topología, técnicas de conmutación, mecanismos de control del flujo de mensajes, estrategias de encaminamiento y métodos de gestión de interbloqueos en las comunicaciones.

Para este trabajo se han utilizado tres multicomputadores distintos: un Supernode, un Paragon y una red de estaciones de trabajo. Las características hardware/software de estos sistemas se describen en el apéndice A, al final de este documento.

4 Una evaluación de técnicas de simulación

En este capítulo se evalúa el comportamiento de varios algoritmos de simulación, secuenciales y paralelos, usando implementaciones de estos algoritmos en uno de los multicomputadores disponibles, el Supernode. Como modelo a simular se ha elegido una simple red toroidal de servidores FCFS (ver figura).



(a) Red toroidal de 4x4 nodos. Los canales de comunicación que unen dos nodos son bidireccionales. (b) Detalle de cada nodo.

En este modelo se pueden variar parámetros tales como: distribución del tiempo de servicio, número total de tareas en el toro, tamaño del toro, función de encaminamiento de las tareas y distribución del cómputo en cada nodo. Este último parámetro representa una carga sintética que se añade al simulador, carga que intenta reflejar el esfuerzo de cómputo que supondría la simulación de cada una de las tareas en un simulador "real", no en uno construido únicamente para efectuar una evaluación de los métodos de simulación paralela.

Se eligen este tipo de modelos a la hora de hacer los experimentos porque suponen una prueba de estrés de los métodos de simulación paralela: aparece un número considerable de PLs, con un alto nivel de comunicación entre ellos.

Si un método da buenos resultados con un modelo de esta naturaleza, premeditadamente "malo", se puede inferir que dará buenos resultados en general.

Con este modelo sencillo, el simulador pasa más tiempo realizando funciones de sincronización entre PLs que realmente simulando el movimiento de tareas por la red. Un parámetro adicional del simulador, denominado *carga sintética*, permite evaluar situaciones en las que la simulación de las tareas supone un coste computacional más alto.

Uno de los parámetros del modelo que se considera más interesante es la función de distribución del tiempo de servicio de los centros de servicio, dada su estrecha relación con el lookahead. En particular, en este modelo el lookahead es igual al valor mínimo de la distribución del tiempo de servicio.

Se han realizado múltiples simulaciones de este modelo usando CMB-DA, TW y un simulador secuencial. Los tiempos de ejecución del simulador secuencial sirven como punto de referencia para calcular las prestaciones, en términos de aceleración o *speedup*:

speedup = (tiempo de ejecución paralela / tiempo de ejecución secuencial)

Para cada simulación paralela, los experimentos se han repetido variando el número de procesadores. Así se puede comprobar cuán bien escala el tiempo de ejecución con el volumen de recursos empleado.

El estudio experimental nos permite concluir que CMB-DA es un algoritmo muy prometedor. La ejecución paralela es, casi siempre, más rápida que la secuencial, y escala con el número de procesadores. El grado de aceleración obtenido depende, en buena medida, de las características del modelo simulado:

- A mayor tamaño y número de tareas (factores que influyen en la densidad de sucesos), mejor rendimiento. El motivo: el simulador permanece sincronizado mientras intercambia mensajes "útiles" y, por tanto, el uso de mensajes nulos (sobrecarga) apenas es necesario.

- A mayor carga sintética, mejor rendimiento. La proporción cómputo/comunicación se equilibra. La comunicación es siempre el cuello de botella de un multicomputador.
- A mayor lookahead, mejor rendimiento. El número de mensajes nulos se reduce cuando el lookahead aumenta.

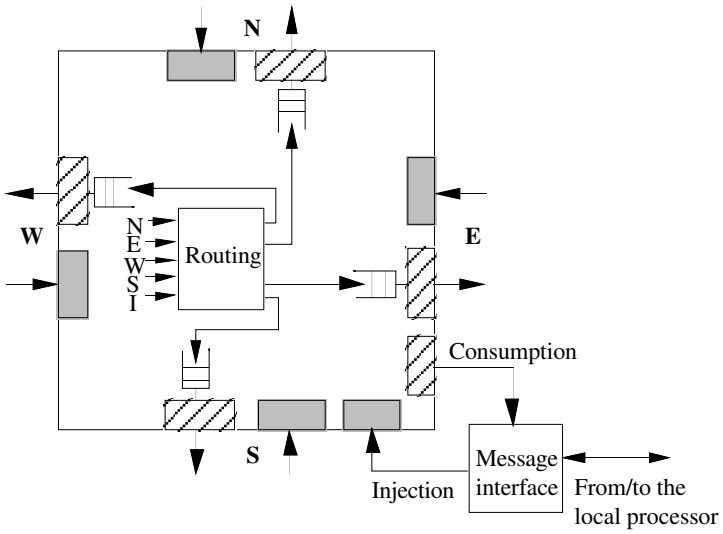
Las conclusiones sobre TW no son tan halagüeñas. El rendimiento es siempre peor que el de CMB-DA, excepto cuando los parámetros del modelo son los peores posibles para CMB-DA: número pequeño de tareas, bajo lookahead. TW es mucho menos sensible a estos dos parámetros. En cuanto a la carga sintética, los resultados no son concluyentes: un cierto grado de carga sintética ayuda, pero si ésta es excesiva el rendimiento puede caer, puesto que sucesos simulados de forma especulativa (porque sus efectos son anulados durante una vuelta atrás) suponen una importante pérdida de tiempo, mayor cuanto más grande es la carga sintética. El principal motivo del bajo rendimiento de TW es la complejidad requerida para la gestión de las estructuras de datos (memoria) empleadas por este algoritmo.

5 Simulación de redes de paso de mensajes

Tras la experiencia ganada con la implementación de CMB-DA y TW en el Supernode, usando el modelo sencillo descrito en el capítulo anterior, estamos en condiciones de simular un modelo más complejo (y realista) usando nuevas versiones, más elaboradas, de CMB-DA y TW, no sólo en el Supernode, sino también en el Paragon y en la red de estaciones de trabajo. Se incorpora a este estudio, además, el algoritmo SPED.

El modelo simulado en el capítulo anterior es una versión simplificada de la clase de modelos que a nosotros nos interesa. Nuestro objetivo es simular, con el mayor grado de detalle posible, redes de interconexión para multicomputadores, donde en cada nodo tenemos no una simple cola FCFS, sino un complejo encaminador de mensajes implementado en hardware. Lo

primero es, entonces, describir detalladamente el modelo exacto que queremos simular: una red toroidal de encaminadores hardware, cada uno de ellos con cuatro enlaces de comunicaciones (el toro es bidimensional), con control de flujo *cut-through* y encaminamiento estático. El funcionamiento de la red es síncrono: los mensajes avanzan (dentro y fuera de los encaminadores) *flit* a *flit*, un *flit* por ciclo. La siguiente figura datalla cómo son los encaminadores simulados:



Detalle de un encaminador de mensajes. Cada uno de ellos se conecta a cuatro vecinos. En conjunto, forman una red con topología toroidal.

Los parámetros más importantes del modelo son:

- *tamaño* de la red,
- *longitud* de los mensajes, en *flits*, y
- *carga* o actividad, en términos de generación de mensajes, de los nodos de la red.

Con estos parámetros, la densidad de sucesos gestionados por el simulador es directamente proporcional al tamaño y a la carga, e inversamente proporcional a la longitud.

Una vez definido el modelo, se describen seis simuladores capaces de trabajar con él:

- Un simulador secuencial, que se puede ejecutar en un nodo de cualquiera de los computadores empleados en este trabajo: un transputer del Supernode, un i860 del Paragon o una estación de trabajo SUN.
- Una implementación de CMB-DA para el Supernode.
- Una implementación de CMB-DA para el Paragon.
- Una implementación de CMB-DA usando la biblioteca MPI, que puede ejecutarse en una red de estaciones de trabajo.
- Una implementación de TW para el Paragon.
- Una implementación de SPED para el Paragon.

Los simuladores paralelos aceptan, como parámetro de entrada, el número de procesadores a utilizar.

La descripción más exhaustiva de CMB-DA se refiere a la implementación para el Supernode. En ella se hace especial énfasis en estos conceptos:

- Tamaño de grano de los PLs: cómo se organizan los PLs dentro de cada procesador. Se consideran diferentes alternativas: tamaño de grano máximo (un PL por procesador, que simula un grupo numeroso de encaminadores), mínimo (múltiples PLs por procesador, cada uno de ellos simulando un único encaminador) e intermedio (unos pocos PLs por procesador, cada uno de ellos simulando un grupo no muy grande de encaminadores).
- *Lookahead*: cuando el tamaño de grano es mínimo, un PL (encargado de simular un único encaminador) puede obtener un cierto grado de *lookahead*, mayor cuanto más largos son los mensajes y mayor es la carga del modelo. En el caso de tamaños de grano intermedio o máximo, extraer

lookahead no es sencillo, aunque siempre se puede contar con un lookahead de un ciclo.

- Organización de los PLs. Dado que el transputer sólo ofrece comunicación bloqueante, el riesgo de interbloqueos en la comunicación es muy elevado. Para evitarlo, se estructuran los PLs en tres sub-procesos, que se comunican pasándose mensajes: un *proceso input*, encargado de recibir mensajes y gestionar los calendarios de sucesos, un *proceso simulador*, que se encarga de ejecutar los sucesos que le entrega el proceso input, y un *proceso output*, que se encarga de hacer llegar a un proceso input vecino los sucesos generados por su proceso simulador.
- Gestión de los mensajes nulos. Se han implementado varias estrategias para reducir el número de estos mensajes, y el esfuerzo de cómputo asociado a la gestión de los mismos. En concreto, sólo se envían mensajes nulos cuando son realmente útiles; además, el único efecto asociado a la recepción de un mensaje nulo es el incremento de un reloj—es decir, no hace falta insertarlos en ningún calendario.

La implementación de CMB-DA para Paragon es bastante diferente a la del Supernode, por distintos motivos:

- Debido a la política de planificación de procesos usada por el sistema operativo del Paragon, no resulta eficiente la compartición de un procesador entre varios procesos.
- El modelo de comunicación del Paragon es no bloqueante, luego el riesgo de interbloqueos es mucho menor.

El resultado es que la implementación de CMB-DA para el Paragon sólo utiliza tamaño de grano máximo (un efecto secundario es que no se puede explotar el lookahead del modelo), y que cada PL es monolítico, no se descompone en sub-procesos. Estos comentarios también son válidos para las implementaciones de TW y SPED en el Paragon.

La implementación de CMB-DA usando MPI es prácticamente idéntica a la del Paragon, puesto que los dos entornos de programación son muy similares.

La implementación de TW en el Paragon sigue el algoritmo básico descrito en el capítulo 2 de la memoria, con dos optimizaciones: salvaguarda incremental del estado, y ventanas de tiempo conservadoras. Esta implementación es la que más tiempo de desarrollo y depuración ha requerido, debido a la dificultad que supone la gestión de las estructuras de datos asociadas a TW.

La implementación de SPED en el Paragon sigue las ideas básicas dadas en el capítulo 2. Su implementación ha sido la más sencilla de todas.

6 Prestaciones de los simuladores

En este capítulo se describe una colección exhaustiva de experimentos realizados para evaluar cada uno de los 5 simuladores paralelos descritos en el capítulo anterior. Se han tenido en cuenta en los experimentos tanto los parámetros del modelo (tamaño, longitud de los mensajes, carga) como los del simulador (número de procesadores, tamaño de grano, uso de lookahead). Resumimos las conclusiones de cada estudio, simulador a simulador.

CMB-DA en el Supernode. Los resultados son muy satisfactorios. El algoritmo escala bien con el número de procesadores, especialmente si el modelo a simular es de gran tamaño. Las prestaciones mejoran con la densidad de sucesos (tamaño grande, longitud de mensajes corta, carga elevada). La mejor opción de tamaño de grano es la intermedia: con tamaño de grano mínimo, el esfuerzo de sincronización es demasiado grande. Con tamaño de grano máximo, los transputers pierden tiempo mientras los PLs están bloqueados. El tamaño de grano intermedio supone un compromiso muy efectivo. Esto supone renunciar al uso del *lookahead* del modelo. Si se utilizase tamaño de grano mínimo, si resultaría conveniente el uso del lookahead, porque mejora notablemente las prestaciones—aunque no lo suficiente como para usar la opción

<tamaño de grano mínimo, *lookahead*> en vez de <tamaño de grano intermedio, no *lookahead*>.

CMB-DA en el Paragon. Las conclusiones obtenidas con el Supernode son también aplicables, excepto en lo que se refiere al tamaño de grano y al uso del lookahead (sólo se ha podido usar tamaño de grano máximo, sin *lookahed*). El rendimiento obtenido es muy bueno, mejor incluso que en el Supernode, debido a un diseño más simplificado de los PLs y a que el Paragon cuenta con un soporte hardware muy efectivo para las operaciones de paso de mensajes. También se ha verificado que las prestaciones escalan muy bien con el número de procesadores. En el Paragon se han usado hasta 100 procesadores (en el Supernode el máximo ha sido 25), comprobándose que el aumento de velocidad es lineal con el número de procesadores, siempre que la densidad de sucesos sea lo suficientemente elevada como para mantener a los procesadores ocupados (haciendo computaciones útiles) la mayor parte del tiempo.

TW en el Paragon. Resultados muy negativos. Las simulaciones usando TW son *mas lentas* que las secuenciales. El modelo a simular no reúne las características necesarias para aprovechar la política de ejecución especulativa de sucesos realizada por TW.

SPED en el Paragon. Resultados muy satisfactorios, especialmente si se tiene en cuenta la simplicidad del algoritmo empleado. Los resultados obtenidos son comparables a los de CMB-DA. El mayor problema de SPED es que escala peor que CMB-DA: hace falta un problema de gran tamaño para sacar el máximo partido a este algoritmo. Dicho de otra forma, el rendimiento obtenido no es bueno si la densidad de sucesos es baja, porque entonces los procesadores pasan demasiado tiempo sincronizándose entre ellos, en vez de ejecutando sucesos.

CMB-DA con MPI en una red de estaciones de trabajo. Los resultados no son buenos—no se obtienen aceleraciones. El problema radica en que, en el

caso de una red de estaciones de trabajo, la red de comunicaciones (Ethernet, en este caso) es un cuello de botella: la proporción cálculo / comunicación está demasiado desequilibrada. CMB-DA exige un intercambio frecuente de mensajes de tamaño corto, mientras que Ethernet (las redes locales, en general) ofrece el máximo rendimiento en entornos en los que se necesita de un intercambio esporádico de mensajes de gran tamaño.

7 Conclusiones y líneas abiertas

Las conclusiones y aportaciones de este trabajo se pueden resumir en los siguientes 11 puntos:

1 Hemos hecho una descripción de alternativas de simulación paralela y hemos evaluado esas alternativas usando un modelo sencillo.

2 Hemos descrito tres entornos diferentes para la realización de aplicaciones paralelas, identificando qué características de estos entornos influyen en la forma en la que se implementan los algoritmos de simulación paralela.

3 Hemos modelado un encaminador de mensajes, de tal forma que este modelo es apto para ser simulado usando diferentes técnicas, secuenciales y paralelas.

4 Hemos implementado cinco simuladores paralelos, usando tres técnicas de sincronización y tres multicomputadores:

CMB-DA en el Supernode
CMB-DA en el Paragon
CMB-DA en una red de estaciones de trabajo, usando MPI
TW en el Paragon
SPED en el Paragon

También se ha implementado un simulador secuencial, que puede ser ejecutado en cualquiera de los tres computadores enumerados. Este simulador se usa como punto de referencia para calcular las aceleraciones conseguidas con los simuladores paralelos.

5 Hemos diseñado una optimización para CMB-DA que permite una reducción en el número de mensajes nulos que se necesitan para mantener la sincronización: un mensaje nulo se envía sólo si tiene algún efecto positivo en el receptor. Además, los mensajes nulos no se almacenan, sino que se limitan a incrementar los relojes de canal de los receptores. De esta forma, el esfuerzo de sincronización necesario en CMB-DA se reduce notablemente.

6 Hemos introducido el concepto de *tamaño de grano* de un proceso lógico, identificando cómo este parámetro afecta a las prestaciones de CMB-DA. Usar un tamaño de grano máximo (un proceso por procesador) minimiza el esfuerzo de sincronización, pero no aprovecha bien el procesador si el proceso se bloquea—situación muy común en CMB-DA. Si el tamaño de grano es mínimo, múltiples procesos usan la CPU, luego el uso de este recurso se optimiza; a cambio, el esfuerzo de sincronización aumenta considerablemente. La mejor alternativa es usar un tamaño de grano intermedio, siempre que el computador lo permita.

7 Se ha caracterizado la *densidad de sucesos* como el parámetro que más influencia tiene en las prestaciones de CMB-DA y SPED: a mayor densidad de sucesos, mejores prestaciones.

8 Se ha descrito cómo obtener información sobre el *lookahead* del modelo de encaminador de mensajes usado como banco de pruebas. Esta información permite mejorar el rendimiento de CMB-DA, pero sólo en el caso de que se use tamaño de grano mínimo. Resulta, sin embargo, más interesante renunciar al uso del *lookahead* y usar tamaños de grano intermedio.

9 Se ha comprobado que CMB-DA y SPED escalan muy bien con el número de procesadores, especialmente si la densidad de sucesos es elevada, y si el multicomputador utilizado da un buen soporte para el paso de mensajes.

10 Se ha llegado a la conclusión de que TW no resulta apropiado para el tipo de simulaciones que nos interesan.

11 Por último, se concluye que tanto CMB-DA como SPED resultan muy útiles para realizar simulaciones de redes de paso de mensajes, explotando adecuadamente el potencial de los multicomputadores comerciales actuales—y, previsiblemente, también de los futuros.

En cuanto a las líneas abiertas de trabajo, se considera interesante realizar un estudio analítico de los mecanismos de simulación paralela evaluados (empíricamente) en este trabajo. También sería interesante utilizar las conclusiones de este trabajo para realizar una herramienta genérica de simulación de redes de interconexión, que resultase útil para que un arquitecto de computadores paralelos evaluase diferentes alternativas de diseño.

A Sistemas de computación paralela usados en este trabajo

La realización del trabajo presentado en esta tesis ha requerido del uso de tres computadores paralelos:

- Supernode: un sistema multicomputador basado en transputers, diseñado y comercializado por Parsys. Este multicomputador está disponible en la Facultad de Informática de la UPV/EHU.
- Paragon: un sistema multicomputador construido por Intel. El autor ha usado el Paragon de Purdue University (Indiana, USA).

- Una red de estaciones de trabajo SUN, que forman parte del *Parallel Processing Laboratory* de la *School of Electrical and Computer Engineering*, también en Purdue.

En el Apéndice A de la memoria se describen, con bastante detalle, las características más importantes de cada uno de estos tres sistemas. Estas descripciones incluyen los siguientes puntos:

- Hardware de cada máquina:
  - Número de procesadores, características de los mismos.
  - Red de interconexión utilizada, características de la misma.
- Entorno de programación paralela ofrecido:
  - Lenguajes de programación.
  - Bibliotecas de funciones. Modelos de comunicación ofrecidos.
  - Herramientas para el desarrollo de programas.

La información más relevante de cada sistema se resume en estas dos tablas:

	Supernode	**Paragon**	**Red E.T.**
Procesador	Transputer T800	Intel i860 (two)	Sun Sparcstation 5
Número de procesadores	34	140	4
Implementación de funciones de paso de mensajes	Software	Hardware	-
Topología de la red	Configurable por el usuario	Red 2D	Bus (Ethernet)
Técnica de conmutación	Conmutación de paquetes	Conmutación de paquetes	Conmutación de paquetes
Control del flujo de mensajes	*Store-and-forward*	*Wormhole*	-
Encaminamiento	Configurable por el usuario	Estático	-
Gestión de interbloqueos	Configurable por el usuario	-	-

Resumen de las características hardware de los sistemas usados en este trabajo.

	Supernode	**Paragon**	**Red E.T.**
Herramienta de programación	ANSI C *toolset*, con bibliotecas de funciones para progrmación paralela especialmente adaptadas al transputer.	ANSI C con la biblioteca de funciones NX de Intel.	ANSI C con la biblioteca de funciones MPI (*Message Passing Interface*).
Modelo de paralelismo	MIMD.	MIMD (se recomienda el uso de SPMD).	MIMD (se recomienda el uso de SPMD).
Paradigma de comunicación	Paso de mensajes. Memoria compartida (sólo dentro del mismo nodo).	Paso de mensajes. Memoria compartida (sólo dentro del mismo nodo).	Paso de mensajes.
Modelo de comunicación	Bloqueante.	Bloqueante, no bloqueante.	Bloqueante, no bloqueante.
Identificación del interlocutor	Implícita (canales).	Explícita (direcciones).	Explícita (direcciones).

Resumen de las características software de los sistemas usados en este trabajo.