Una implementación multithread de MPI para Linux

Alejandro Calderón, Félix García, Jesús Carretero

Octubre de 1999

Resumen

En este artículo se presenta una implementación multithread de la interfaz estándar de paso de mensajes MPI, que se está desarrollando en el Departamento de Arquitectura y Tecnología de Sistemas Informáticos (DATSI) de la Universidad Politécnica de Madrid, denominada MiMPI. MiMPI ofrece una semántica multithread y utiliza operaciones multithread para incrementar el rendimiento de ciertas operaciones, como por ejemplo las colectivas. MiMPI ejecuta en estaciones de trabajo UNIX, Linux y en multicomputadores SP2. El artículo describe los principales objetivos de diseño, la actual implementación de MiMPI y los resultados de su evaluación realizada sobre una red de estaciones de trabaoj con Linux.

1 Introducción

MPI [7] es una interfaz estándar de paso de mensajes cuyos objetivos básicos son: la funcionalidad, la eficiencia y la portabilidad. Aunque la semántica que especifica MPI para el paso de mensajes permite su utilización en aplicaciones y entornos multithread, en la actualidad no existe ninguna implementación de este estándar que verdaderamente pueda ser utilizada para desarrollar un programa MPI multithread.

El uso de threads en una aplicación que emplea paso de mensajes y la implementación de un sistema de paso de mensajes utilizando threads puede ser muy útil por varias razones:

- Los threads permiten una implementación natural de operaciones no bloqueantes, basta con crear un thread que procese la operación no bloqueante.
- El uso de threads puede incrementar la eficiencia de la implementación de operaciones colectivas.
- Los threads ofrecen un modelo natural para implementar ciertas operaciones que requieren memoria compartida. En general todo proceso que emplea paso de mensajes para comunicarse con otros, lleva a cabo dos tareas bien diferenciadas, una de cómputo y otra de intercambio de mensajes. Estas tareas pueden implementarse de forma eficiente utilizando threads dentro del mismo proceso.
- Los threads se están convirtiendo en el mecanismo de programación paralela de los multiprocesadores de memoria compartida, ya que permiten explotar el paralelismo de dichas arquitecturas de una forma eficiente.

Las ventajas anteriormente citadas junto con la falta de una implementación de MPI con soporte para la ejecución de aplicaciones multithread en redes de estaciones de trabajo con Linux, motivó a los autores el desarrollo de MiMPI. MiMPI [5] es una implementación de MPI que está desarrollándose en el Departamento de Arquitectura y Tecnología de Sistemas Informáticos (DATSI) de la Universidad Politécnica de Madrid. MiMPI surgió con dos objetivos básicos: ofrecer una implementación de MPI que pueda ser utilizada en programas multithread (*MT-safe*), y utilizar threads para su implementación con el objetivo de mejorar el rendimiento de ciertas operaciones.

En la actualidad MiMPI implementa un subconjunto de todas las primitivas de MPI y ha sido utilizado con éxito en *ParFiSys* [4] [9] un sistema de ficheros paralelo desarrollado en el DATSI.

El resto del trabajo está organizado como sigue: en la sección II se describe brevemente MPI. En la sección III se presentan algunas de las implementaciones existentes de MPI. En la sección IV se realiza una descripción de MiMPI, sus objetivos de diseño e implementaciones actuales. La evaluación de MiMPI se presenta en la sección V y los resultados obtenidos de dicha evaluación se muestran en la sección VI. Estos resultados comparan el rendimiento de MiMPI con el de MPICH (implementación de libre distribución) en una red de estaciones e trabajo con Linux. El trabajo finaliza incluyendo algunas conclusiones y las líneas de trabajo futuras.

2 Breve descripción de MPI

MPI [7] [11] (*message-passing interface*) es la especificación de un interfaz estándar de paso de mensajes, cuyos principales objetivos son: la funcionalidad, la eficiencia y la portabilidad. MPI incluye paso de mensajes punto a punto y operaciones colectivas, todas ellas dentro del ámbito de un grupo de procesos especificado por el usuario. En MPI todos los procesos pertenecen a grupos. Si un grupo contiene n procesos, éstos se identifican mediante enteros dentro del rango 0, n-1.

A continuación se describen algunas de las principales características que ofrece MPI:

- *Operaciones colectivas.* Una operación colectiva es una operación ejecutada por todos los procesos que intervienen en un cálculo o comunicación. En MPI existen dos tipos de operaciones colectivas: *operaciones de movimiento de datos* y *operaciones de cálculo colectivo*. Las primeras se utilizan para intercambiar y reordenar datos entre un conjunto de procesos. Un ejemplo típico de operación colectiva es la difusión (*broadcast*) de un mensaje entre varios procesos. Las segundas permiten realizar cálculos colectivos como mínimo, máximo, suma, OR lógico, etc, así como operaciones definidas por el usuario.

- *Topologías virtuales,* ofrecen un mecanismo de alto nivel para manejar grupos de procesos sin tratar con ellos directamente. MPI soporta grafos y redes de procesos.

- *Modos de comunicación.* MPI soporta operaciones bloqueantes, no bloqueantes o asíncronas y síncronas. Una operación de envío bloqueante bloquea al proceso que la ejecuta sólo hasta que el buffer pueda ser reutilizado de nuevo. Una operación no bloqueante permite solapar el cálculo con las comunicaciones. En MPI es posible esperar por la finalización de varias operaciones no bloqueantes. Un envío síncrono bloquea la operación hasta que la correspondiente recepción tiene lugar. En este sentido, una operación bloqueante no tiene por que ser síncrona.

- *Soporte para redes heterogéneas.* Los programas MPI están pensados para poder ejecutarse sobre redes de máquinas heterogéneas con formatos y tamaños de los tipos de datos elementales totalmente diferentes.

- *Tipos de datos.* MPI ofrece un amplio conjunto de tipos de datos predefinidos (caracteres, enteros, números en coma flotante, etc.) y ofrece la posibilidad de definir tipos de datos derivados. Un tipo de datos derivado es un objeto que se construye a partir de tipos de datos ya existentes. En general, un tipo de datos derivado se especifica como una secuencia de tipos de datos ya existentes y desplazamientos (en bytes) de cada uno de estos tipos de datos. Estos desplazamientos son relativos al buffer que describe el tipo de datos derivado.

- *Modos de programación.* Con MPI se pueden desarrollar aplicaciones paralelas que sigan el modelo SPMD o MPMP. Además el interfaz MPI tiene una semántica multithread (*MT-safe*), que le hace adecuado para ser utilizado en programas MPI y entornos multithread.

En 1997 se añadieron algunas extensiones al estándar inicial, apareciendo MPI-2 [8]. Este nuevo estándar incluye, entre otras características, funcionalidad para la gestión y creación de procesos, nuevas operaciones colectivas y operaciones de E/S (MPI-IO).

3 Implementaciones existentes

A continuación se describen algunas de las implementaciones de MPI existentes en la actualidad:

- LAM [3] es una implementación disponible en el centro de supercomputación de Ohio, que ejecuta sobre estaciones de trabajo heterogéneas Sun, DEC, SGI, IBM y HP.

- CHIMP-MPI [1] es una implementación desarrollada en el Centro de Computación Paralela de Edimburgo basada en CHIMP [6]. Esta implementación ejecuta sobre estaciones de trabajo Sun, SGI, DEC, IBM y HP, y máquinas Meiko y Fujitsu AP-1000.

- MPICH [10] es una implementación desarrollada de forma conjunta por el Laboratorio Argonne y la Universidad de Mississippi. Esta implementación ejecuta sobre diferentes plataformas, redes de estaciones de trabajo Sun, SGI, RS6000, HP, DEC, Alpha, y multicomputadores como el IBM SP2, Meiko CS-2 y Ncube.

- Unify [14], disponible en la Universidad del Estado de Mississippi, es una implementación que permite el uso de llamadas MPI y PVM de forma simultanea dentro del mismo programa.

- MPI-LITE [2] es una implementación multithread de MPI, que ofrece un núcleo para la creación, terminación y planificación de threads de usuario. Esta implementación es la única conocida por los autores que ofrece un cierto soporte para la ejecución de programas multithread. Sin embargo, el modelo que ofrece MPI-LITE es muy rígido, puesto que los threads son de usuario, cada thread ejecuta una copia de un programa MPI, y además, el número de threads en el programa es fijo y se especifica como un argumento de entrada a un programa MPI.

Ninguna de estas implementaciones, a excepción de MPI-LITE, puede ser utilizada para ejecutar programas MPI multithread en estaciones de trabajo con Linx, y como se comentó anteriormente MPI-LITE ofrece un modelo muy poco flexible que no puede emplearse en programas donde se creen y destruyan threads de forma dinámica.

4 Descripción de MiMPI

MiMPI es una nueva implementación de MPI cuyos objetivos de diseño son:

- Ser una implementación de MPI adecuada para la ejecución de aplicaciones multithread, es decir, una implementación *MT-safe*.
- Utilizar threads para su implementación con el objetivo de mejorar el rendimiento de ciertas operaciones, como por ejemplo, las no bloqueantes y colectivas (ver figura 1).
- Analizar la adecuación del uso de threads en diferentes estrategias a utilizar en la implementación de ciertas operaciones de paso de mensajes.
- Asegurar la portabilidad entre diferentes arquitecturas.
- Ofrecer un entorno integrado de herramientas gráficas para el arranque de procesos, visualización y depuración de la ejecución paralela de programas que utilicen MiMPI.

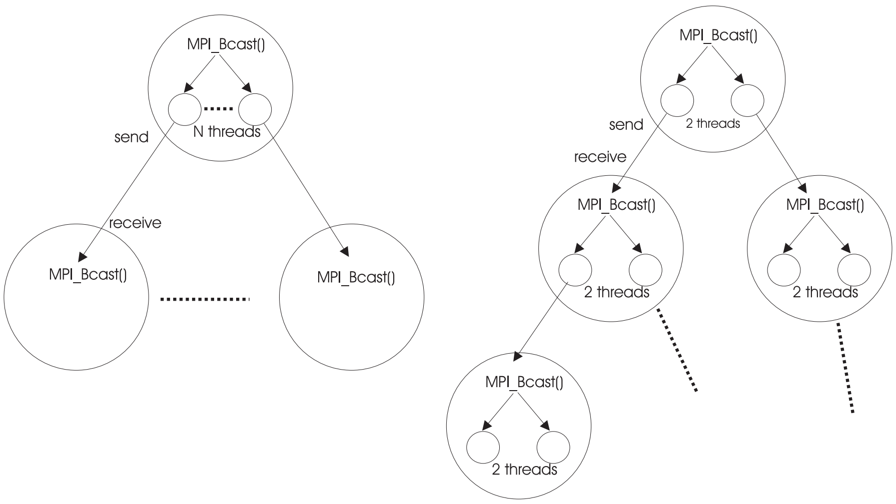

Figura 1: Implementación de operaciones colectivas en MiMPI

MiMPI se ha estructurado en tres niveles (ver figura 2):

- Capa de servicios básicos independiente de la plataforma.
- Micro-núcleo de comunicaciones, denominado XMP.
- Capa de servicios que implementa la interfaz MPI.

La capa de servicios básicos pretende independizar de la plataforma utilizada. En la implementación actual empleamos los servicios y threads ofrecidos por POSIX y la comunicación basada en TCP/IP mediante el uso de los sockets de UNIX.

El micro-núcleo de comunicaciones XMP pretende ser utilizado en el futuro para desarrollar otros modelos de paso de mensajes, como por ejemplo PVM. Este micro-núcleo ofrece un interfaz compuesto exclusivamente por tres funciones:

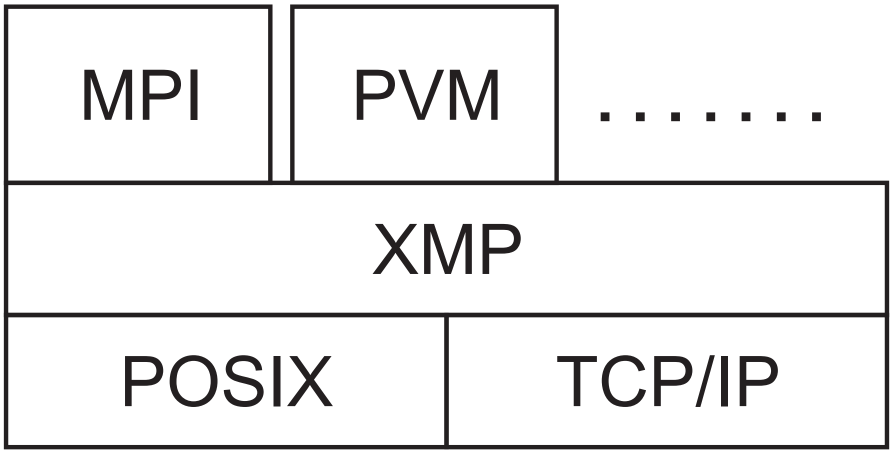

Figura 2: Arquitectura de MiMPI

- `XMP_Init()`
- `XMP_Finalize()`
- `XMP_Request(struct request *)`

Las funciones `XMP_Init()` y `XMP_Finalize()` se utilizan para iniciar el sistema y para liberar los recursos respectivamente. La función `XMP_Request()` se emplea para solicitar las acciones de intercambio de mensajes, envío (*send*) y recepción (*receive*).

El último nivel lo constituye MiMPI, que es la implementación de la interfaz estándar MPI.

MiMPI pretende incluir además, un conjunto integrado de herramientas gráficas, que se irá ampliando en el futuro, que facilitan la utilización del entorno de MiMPI y que servirán para la ejecución, visualización y depuración de programas MPI.

En la actualidad MiMPI ejecuta sobre tres plataformas: estaciones de trabajo y multiprocesadores Sun, estaciones de trabajo con Linux y multicomputadores IBM SP2.

5 Evaluación

En esta sección se describen los *benchmarks* empleados para evaluar el rendimiento de MiMPI. Estos *benchmarks*, similares a los descritos en [13], se centran en la medida del ancho de banda de las comunicaciones punto a punto y colectivas. A continuación se describen estos *benchmarks*.

5.1 Comunicación punto a punto

Este *benchmark* mide el ancho de banda de la comunicación punto a punto entre dos procesos que se intercambian un par de mensajes entre ellos. Un proceso mide el tiempo necesario para realizar una operación MPI_Send seguida de una operación MPI_Recv (el otro proceso realiza la operación simétrica). De esta forma se consigue medir el tiempo necesario para intercambiar dos mensajes (uno en cada dirección) entre los dos procesos. La latencia se mide como la mitad de este tiempo, y el ancho de banda se calcula considerando el tamaño del mensaje y la latencia obtenida. Este proceso se realiza 1000 veces.

Al igual que en [13] los resultados muestran los valores medios ya que consideramos que estos son los más representativos del rendimiento que puede obtener un usuario normal.

Este mismo *benchmark* se ha ejecutado con una pequeña variación. El objetivo era evaluar el ancho de banda en una comunicación punto a punto entre varios threads de dos procesos (comunicación punto a punto multithread). El ancho de banda en este caso se calcula considerando el tamaño del mensaje, la latencia calculada y el número de threads que intervienen en la comunicación.

5.2 Operaciones colectivas

Las operaciones colectivas que se ha evaluado en este trabajo han sido MPI_Bcast (*broadcast*), MPI_Gather (*gather*) y MPI_Scatter (*scatter*).

Este *benchmark* se ha realizado empleando 4 procesos que ejecutan en cuatro estaciones de trabajo con Linux, ejecutando todos ellos las operaciones colectivas. El proceso 0 (el proceso raíz de la operación) realiza 1000 iteraciones, y en cada iteración ejecuta una operación colectiva seguido por una barrera de sincronización (MPI_Barrier). El resto de procesos realizan las mismas operaciones. Para calcular el tiempo consumido en realizar la operación colectiva, el proceso 0 mide el tiempo empleado en realizar las operaciones colectivas y MPI_Barrier, y a continuación le resta el tiempo gastado en la ejecución de la barrera de sincronización previamente calculado. Las figuras muestran los valores medios del ancho de banda agregado, que se ha calculado considerando la cantidad de información que se transfiere a todos los procesos.

6 Resultados de la evaluación

En esta sección presentamos los resultados obtenidos en la evaluación de los *benchmarks* descritos en la sección anterior. Los resultados se han obtenido sobre cuatro biprocesadores Pentium II con Linux (RedHat 6.0) conectadas por una FastEthernet. Se ha evaluado y comparado el rendimiento de MiMPI con el de MPICH, una implementación de libre distribución que ejecuta sobre máquinas con Linux.

La figura 3 compara el ancho de banda obtenido en la comunicación punto a punto. Como se puede observar el rendimiento es muy similar en ambas implementaciones, aun teniendo en cuenta el esfuerzo que debe llevar a cabo MiMPI para asegurar una semántica multithread.

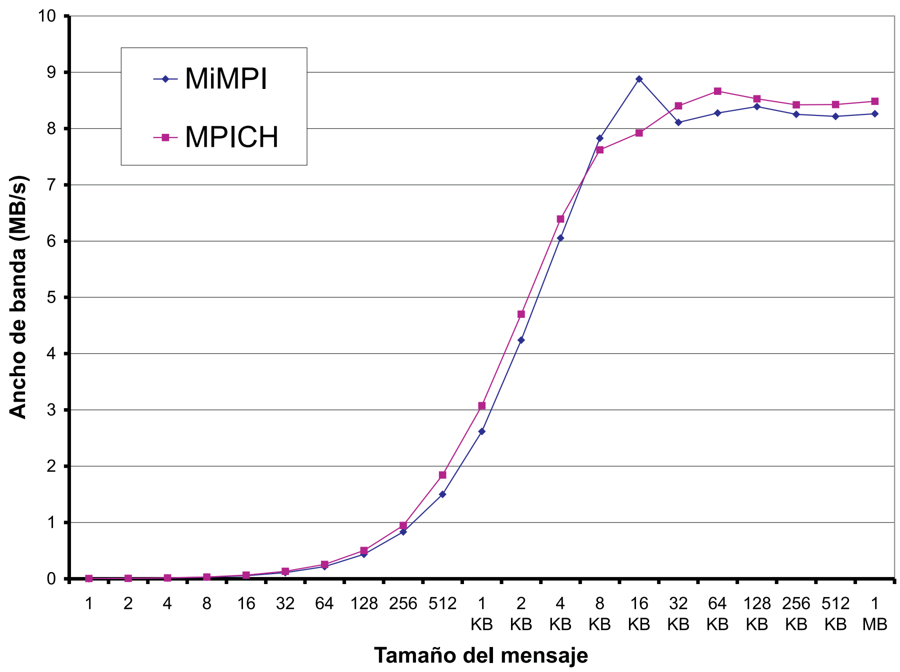

Figura 3: Rendimiento de la comunicación punto a punto

Las figuras 4, 5 y 6 muestra los resultados obtenidos para las operaciones colectivas. Como puede observarse MiMPI obtiene mejores resultados con tamaños grandes, gracias a la utilización de operaciones multithread en la ejecución de las operaciones colectivas. Los resultados son aun mejores en otro tipo de arquitecturas como la SP2 [5].

7 Conclusiones

En este trabajo se ha hecho una breve descripción de MiMPI, una nueva implementación de MPI cuya característica fundamental es que permite la ejecución de programas MPI multithread sobre estaciones de trabajo con Linux. MiMPI se está implementando utilizando threads para mejorar el rendimiento de ciertas operaciones, como por ejemplo, las colectivas y no bloqueantes. MiMPI además ofrece un conjunto integrado de herramientas que se irá ampliando en el futuro, que facilitan la gestión de entorno de programas MPI: ejecución, visualización y depuración de programas. La evaluación que hemos realizado de MiMPI nos ha permitido observar que ofrece un rendimiento muy comparable e incluso superior, en algunas operaciones, con MPICH, una implementación muy utilizada en estaciones de trabajo con Linux, presentando la ventaja de poder ejecutar programas que multithread que utilizan MPI.

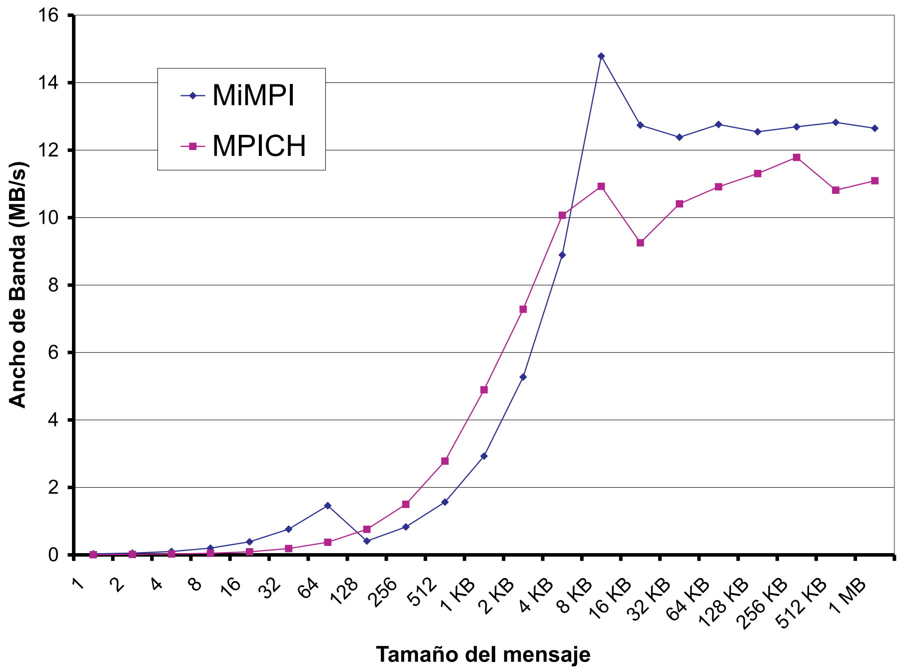

Figura 4: Rendimiento del Broadcast

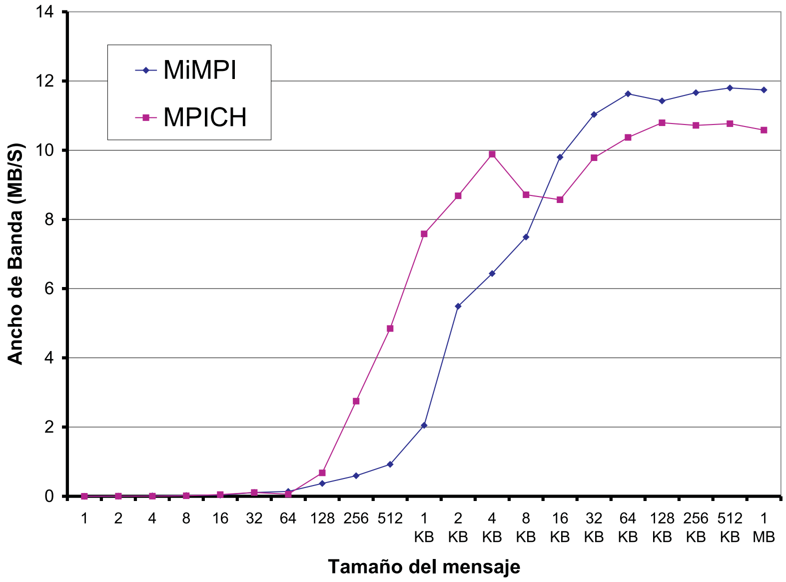

Figura 5: Rendimiento de Gather

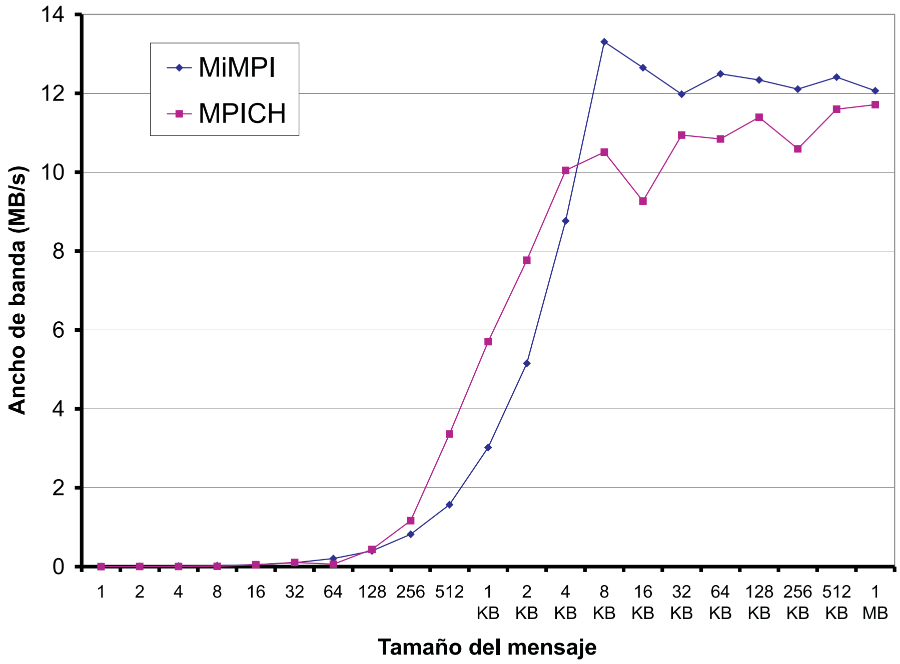

Figura 6: Rendimiento de Scatter

8 Trabajo futuro

Son varias las líneas de trabajo a seguir:

- Finalizar la implementación de todas las primitivas que ofrece MPI.
- Realizar una evaluación más exhaustiva de MiMPI, incluyendo plataformas constituidas por redes de estaciones de trabajo.
- Optimizar MiMPI para reducir la latencia de las operaciones sobre mensajes pequeños. Esta optimización es importante, ya que muchas aplicaciones intercambian muchos mensajes de tamaño pequeño [12].
- Seguir trabajando en las herramientas de visualización y gestión del entorno MiMPI.
- Implementar MPI-IO sobre el sistema de ficheros paralelo *ParFiSys*.

9 Agradecimientos

Quisiéramos expresar nuestro agradecimiento al Centro de Supercomputación de Cataluña (CESCA) por permitirnos el uso de la máquina SP2.

Referencias

[1] R. Alasdair, A. Bruce, J.G. Mills, and Smith A.G. CHIMP-MPI user guide. Technical Report EPCC-KTP-CHIMP-V2-USER 1.2, Edinburgh Parallel Computing Centre, 1994.

[2] P. Bhargava. MPI-LITE User Manual, Release 1.1. Technical report, Parallel Computing Lab, University of California, Los Angeles, CA 90095, April 1997.

[3] G. Burns, R. Daoud, and J. Vaigl. LAM: An open cluster environment for MPI. In *Proceedings of Supercomputing Symposium '94*, pages 379–286. In John W. Ross, editor, 1994.

[4] J. Carretero, F. Pérez, P. De Miguel, F. García, and L. Alonso. ParFiSys: A Parallel File System for MPP. *ACM SIGOPS*, 30(2):74–80, April 1996.

[5] F. García, A . Calderón, J. Carretero MiMPI: A Multithread Implementation of MPI. *6th European PVM/MPI User's Group Meeting*, 207–214, Sep 1999.

[6] J. Clarke, A. Fletcher, M. Trewin, A. Bruce, A. Smith, and R. Chapple. Reuse, Portability and Parallel Libraries. In *Proceedings of IFIP WG10.3 – Programming Environments for Massively Parallel Distributed Systems*, 1994.

[7] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. 1995. www.mpi-forum.orf/docs/mpi-11.ps

[8] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*. 1997. www.mpi-forum.orf/docs/mpi-20.ps

[9] F. García, J. Carretero, F. Pérez, and P. de Miguel. Evaluating the *ParFiSys* Cache Coherence Protocol on an IBM SP2. Technical Report FIM/104.1/datsi/98, Facultad de Inform'atica, UPM, Campus de Montegancedo, 28660 Madrid, Spain, January 1998.

[10] W. Gropp, E. Lusk, N Doss, and A. Skyellum. A High Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, (22):789–828, 1996.

[11] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI, Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1995.

[12] K. Langendoen, R. Bhoedjang, and H. Bal. Models for Asynchronous Message Handling. *IEEE Concurrency*, (2):28–38, 1997.

[13] J. Miguel, R. Beivide, A. Arruabarrena, and J.A. Gregorio. Assessing the Performance of the New IBM SP2 Communication Subsystem. In *VII Jornadas de Paralelismo*, pages 115–130, September 1986.

[14] P. L. Vaughan, A. Skjellum, D. S. Reese, and F. Chen Cheng. Migrating from PVM to MPI, part I: The Unify System. In *Fifth Symposium on the Frontiers of Massively Parallel Computation*. IEEE Computer Society Technical Commitee on Computer Architecture, IEEE Computer Society Press, 1995.