UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

"ESTUDIO COMPARATIVO SOBRE REDES MYRINET"

P R O Y E C T O

QUE PARA OBTENER EL TITULO DE

LICENCIADO EN CIENCIAS DE LA C O M P U T A C I O N

P R E S E N T A :

EDGAR ARMANDO LEON BORJA

DIRECTOR DE TESIS: M. en C. JOSE DE JESUS GALAVIZ CASAS

2001

MAT. MARGARITA CHAVEZ CANO
Jefa de la División de Estudios Profesionales
P r e s e n t a.

Comunicamos a usted que hemos revisado el reporte de proyecto.

"ESTUDIO COMPARATIVO SOBRE REDES MYRINET"

realizado por: Edgar Armando León Borja

Con número de cuenta 9337969-8, pasante de la carrera de Licenciado en Ciencias de la Computación.

Dicho trabajo cuenta con nuestro voto aprobatorio.

A t e n t a m e n t e

Director del Proyecto	M. en C. José de Jesús Galaviz Casas
Propietario	Mat. Víctor Hugo Dorantes González
Propietario	Mat. Salvador López Mendoza
Suplente	Dr. Benjamín Macías Pimentel
Suplente	M. en C. Javier García García

Consejo Departamental de Matemáticas

M. en C. Guadalupe E. Ibarguengoitia González

Estudio de desempeño en redes Myrinet

Junio 1999 – Diciembre 1999

Edgar Armando León Borja

Universidad Nacional Autônoma de México

30 de noviembre de 2000

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Antecedentes

Hasta hace algún tiempo las aplicaciones científicas que requerían alto desempeño de cómputo se desarrollaban en supercomputadoras (ver tabla 1.1). Hoy en día la tendencia es utilizar conjuntos de computadoras interconectadas a corta distancia conocidos como *clusters* (ver tabla 1.2), que han adquirido creciente importancia en el procesamiento paralelo y distribuido dado que proveen *alto desempeño* (*high-performance*), *escalabilidad y expandibilidad*, y *alta disponibilidad* a un *costo relativamente bajo*. En general cuando se habla de *clusters* se habla también de *nodos*, un nodo puede ser un sistema sencillo o de multiprocesador (PCs, estaciones de trabajo, o SMPs) con memoria, recursos de E/S, y un sistema operativo. Así, un *cluster* se refiere a dos o más nodos conectados entre sí trabajando como uno solo.

nombre	# procs.	procesador	tasa de datos
Cray Research T3E	2048	Alpha 21164	1200 MB/s
HP/Convex	64	PA-8000	980 MB/s
Sequent NUMA-Q	32	Pentium Pro	1024 MB/s
SGI Origin2000	128	MIPS R10000	800 MB/s
Sun Enterprise 10000	64	UltraSPARC 1	1600 MB/s

Tabla 1.1: Parámetros de distintas supercomputadoras disponibles comercialmente en 1997. *# procs.* se refiere al máximo número de procesadores; *tasa de datos* se refiere al ancho de banda de comunicación / enlace. Los datos presentados en esta tabla están basados en [16].

La parte más crítica de un *cluster* es la *red* [3], su capacidad y desempeño afecta directamente la aplicabilidad de todo el sistema. En la tabla 1.3 se muestran algunas de las redes usadas frecuentemente en la interconexión de *clusters*. Redes de alto desempeño tales como *Myrinet*, *Gigabit Ethernet*, *Infiniband*, etc., se han vuelto muy importantes debido a su velocidad de transmisión y confiabilidad [6, 10, 3] por lo que ha surgido la necesidad de

nombre	# procs.	procesador	tasa de datos
HP 9000 EPS21	64	PA-8000	532 MB/s
IBM RS/6000 HACMP R40	16	PowerPC 604	12 MB/s
IBM RS/6000 SP2	512	Power2 SC	150 MB/s
Sun Enterprise Cluster 6000 HA	60	UltraSPARC	100 MB/s
Tandem NonStop Himalaya S70000	4096	MIPS R10000	40 MB/s

Tabla 1.2: Parámetros de distintos clusters disponibles comercialmente en 1997. *# procs.* se refiere al máximo número de procesadores; *tasa de datos* se refiere al ancho de banda de comunicación / enlace. Los datos presentados en esta tabla están basados en [16].

protocolos que nos permitan hacer uso de este tipo de redes. En particular aquellos protocolos "ligeros" de paso de mensajes (aquellos que minimizan la carga del sistema para efectuar la comunicación), cuyo diseño ha sido orientado principalmente a lograr alto ancho de banda y baja latencia. Sistemas de paso de mensajes como *Portals*, *FM*, *GM*, *VIA*, *ST* y *BIP* entre otros, son ejemplos de protocolos ligeros de paso de mensajes.

interconexión	tasa de datos	conmutación	encaminamiento
Fast Ethernet	100 Mb/s	de paquetes	destino
Gigabit Ethernet	1 Gb/s	de paquetes	destino
Myrinet	1.28 Gb/s	wormhole	fuente
ServerNet II	125 MB/s	wormhole	destino
Memory Channel	100 MB/s	de paquetes	destino
Synfinity	1.6 GB/s	wormhole	fuente
SCI	400 MB/s	de paquetes	destino
ATM(OC-12)	155(622) Mb/s	de paquetes	destino
HiPPI	800 Mb/s	de paquetes	destino

Tabla 1.3: Parámetros de distintas interconexiones. Los datos presentados en esta tabla están basados en [3].

Protocolos ligeros de paso de mensajes como los mencionados, "omiten al sistema operativo" (*bypass the operating system*). Los protocolos de comunicación tradicionales generan una interrupción al CPU cuando un mensaje llega a la tarjeta de red. El manejador de la interrupción controla la transferencia del mensaje de la tarjeta de red al espacio de la aplicación. Las tarjetas de red modernas como las de Myrinet son programables y así podemos controlar la transferencia de los mensajes que llegan *sin necesidad de*

interrumpir al CPU. Esta estrategia no requiere del sistema operativo y por ello se dice que lo omite [10].

Afortunada o desafortunadamente, no mucha gente escribe sus aplicaciones paralelas usando sistemas ligeros de mensajes. Afortunadamente, porque sus aplicaciones no serían portables, y, es aquí donde las interfaces estandarizadas juegan un papel importante (como *MPI*, la interfaz de paso de mensajes utilizada en este estudio). Desafortunadamente, porque implementaciones de capas estándar de paso de mensajes como MPI, añaden cierto tiempo de procesamiento para el envío y recepción de mensajes lo que degrada el desempeño. El objetivo es minimizar este tiempo de procesamiento.

Este estudio reporta mediciones acerca del desempeño de ciertas implementaciones de MPI sobre diferentes sistemas ligeros de paso de mensajes, a través de mediciones de ancho de banda y latencia. El desempeño de estos protocolos es medido también.

1.2 Descripción del estudio

El presente estudio mide el desempeño de diferentes protocolos de paso de mensajes a diferentes niveles de abstracción. En el nivel más bajo se usaron *los protocolos GM y FM* que se describirán en detalle más adelante. En el nivel más alto, se usaron diferentes implementaciones de MPI, *MPICH* (Implementación portable de MPI), *MPICH/GM* y *MPI-FM*. Todas las mediciones fueron obtenidas usando la red *Myrinet*, a excepción de MPICH el cual utiliza *Fast Ethernet* en este caso particular. MPICH fue usado tan solo como punto de referencia con *TCP/IP*. Ver figura 1.1. El sistema operativo utilizado para todo el estudio fue Linux [17].

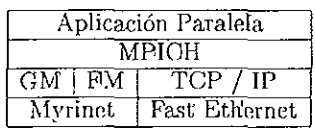

Figura 1.1: Representación en capas para los diferentes protocolos.

El protocolo ligero de comunicación de paso de mensajes ideal tendría un ancho de banda punto-a-punto del 100% de la capacidad "bruta" de la red. Obviamente esto es difícil de lograr debido a que todos los protocolos de comunicación añaden un tiempo de procesamiento para mandar o recibir *mensajes*.

Las mediciones de latencia y ancho de banda[1] son fundamentales en el desarrollo de este estudio. Todas las mediciones de desempeño que aparecen en este reporte son entre procesos de usuario, representativos del desempeño

[1] Refiérase al Glosario para ver las definiciones de ancho de banda y latencia.

ideal de aplicación. Fueron usadas *micro-benchmarks* denominadas "ping-pong" para el cálculo de los parámetros de latencia y ancho de banda (calculados como la mitad del tiempo promedio del tiempo de viaje-redondo). Serán mencionados también algunos aspectos de la instalación y las mediciones de los protocolos en relación con la red y el sistema operativo

En general, dos computadoras son suficientes para realizar las mediciones, computadora 1 y computadora 2, como lo muestra la figura 1.2. Una vez que el protocolo correspondiente ha sido inicializado y se tiene un proceso usuario de ese protocolo en ambas computadoras, llámese "ping" en la computadora 1 y "pong" en la computadora 2, el proceso ping envía un mensaje al proceso pong y lo recibe de regreso de pong. Ping también mido el tiempo que se necesitó para ejectuar el par envío/recepción; esto es, el tiempo de viaje-redondo. El proceso pong hace lo inverso. Las acciones se repiten un número de veces para colectar estadísticas, posiblemente descartando las primeras mediciones para excluír efectos de "calentamiento" (los que tienen lugar cuando el sistema se encuentra aún en proceso de estabilización luego de arrancar). Estas estadísticas son utilizadas para el cálculo de la latencia y el ancho de banda

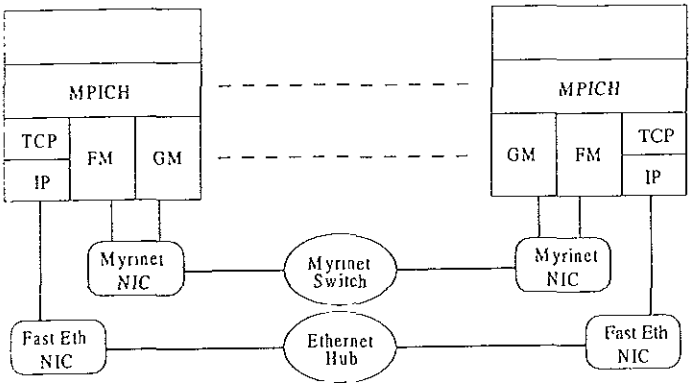

Figura 1.2: Nivel lógico y físico bajo el cual fueron realizadas las mediciones de latencia y ancho de banda.

Capítulo 2

Fundamentos

2.1 Myrinet

Myrinet es una red de alto-desempeño usada para la interconexión de grupos de estaciones de trabajo, PCs, etc. Myrinet ofrece las siguientes características a un costo relativamente bajo [13]:

- **Alto desempeño**, mediante la distribución del cómputo en un conjunto de computadoras con una ventajosa relación costo-beneficio.
- **Alta disponibilidad**, al permitir que el cómputo prosiga con un subconjunto de las computadoras. La interconexión debe ser capaz de detectar y aislar fallas, y de usar caminos de comunicación alternativos.

Entre las principales características de Myrinet están:

- Enlaces *full-duplex* 1.28+1.28 Gb/s, puertos de conmutación y puertos de tarjeta de red (ver sección 2.1.2).
- Flujo de control. Myrinet inserta bytes de STOP y GO en el canal contrario de un enlace para detener o reiniciar transmisiones de datos en el lado del transmisor.
- Control de errores. En cada etapa se calcula un CRC (código de redundancia cíclico) de 8 bits y reemplaza al de la etapa inmediata anterior. Un paquete con un CRC distinto de cero que entra en la tarjeta de red indica errores durante la transmisión. Además, Myrinet presenta una tasa de error muy baja, *menos de un bit de error* por día en una red de cientos de computadoras, y es altamente robusta con respecto a fallas en la computadora, el conmutador y el cable [13]. Myrinet continuamente se mapea a sí misma y usa rutas alternas de haber disponibles, en caso de desconexiones y falta de corriente.
- Monitoreo continuo en cada enlace.

- Baja-latencia, conmutadores *cut-through* (los datos son pasados inmediatamente a la siguiente etapa tan pronto como la dirección del encabezado es decodificada[1]) de barras cruzadas, con monitoreo para aplicaciones de alta-disponibilidad

- Cualquier topología de red es permitida. Redes Myrinet pueden escalar a miles de computadoras con tasas de datos de bisección de red en Terabits por segundo [13]. Myrinet puede también proveer caminos alternos de comunicación entre computadoras.

- Tarjetas de red que ejecutan un programa de control para interactuar directamente con procesos en la computadora (omitiendo al sistema operativo), y directamente con la red, para el envío, recepción y almacenamiento de paquetes.

- Los paquetes de datos pueden ser de cualquier longitud, incluyendo paquetes IP, sin una capa de adaptación. Los paquetes son enviados usando conmutación *wormhole* [3].

- Soporta varias interfaces de software. Cada paquete es identificado por tipo, por lo que una red Myrinet, como una Ethernet, pueden llevar paquetes de muchos tipos o protocolos al mismo tiempo.

Implementaciones eficientes de *middleware* de interfaces de software como MPI [1] y VIA [6] sobre Myrinet están disponibles en Myricom [13] y terceros.

2.1.1 Tarjetas de red

Una tarjeta de red Myrinet consiste de dos componentes principales: el chip *LANai* y su memoria asociada SRAM. El LANai se encarga de controlar la transferencia de datos entre la computadora y la red. Estos componentes son extremadamente confiables [13].

Un chip LANai consiste del corazón de LANai, con un *procesador RISC interpretador de instrucciones* y una *interfaz de paquetes*, la *interfaz de enlace-Myrinet*, la *interfaz de memoria*, y un *dispositivo de DMA/suma de comprobación*. Su componente principal es un micro-controlador programable el cual controla los dispositivos de DMA responsables de la transferencia de datos en las direcciones: memoria de la computadora $\Longleftrightarrow$ *memoria en* tarjeta $\Longleftrightarrow$ red. Por lo que los datos del mensaje deben ser escritos primero en la memoria SRAM de la tarjeta de red antes de que puedan ser inyectados a la red [12]. La memoria SRAM también almacena el MCP y ciertas colas de trabajos

El procesador LANai es una máquina RISC contexto-dual, con 24 registros de propósito general. Uno de los contextos, el contexto de 'usuario', es

[1] Refiérase al Glosario para ver la definición de *conmutación cut-through*

interruptible. Una interrupción que se lleve a cabo durante la ejecución en el contexto de usuario causa que el procesador cambie al contexto de 'sistema'. El contexto de sistema no es interruptible.

La tarjeta de red Myrinet utiliza la interfaz PCI de 32/64 bits de la computadora. El LANai corre en frecuencias que van desde 33 MHz hasta 66 MHz. Además de controlar la transferencia de datos, el LANai también es responsable del mapeo automático de redes y monitoreo del estado de la red [3]. El tamaño de la memoria en tarjeta SRAM varía de 512 KB a 4 MB. El LANai se comunica con los *manejadores* de dispositivo en la computadora o bibliotecas a nivel de usuario a través de colas de trabajo almacenadas en la SRAM.

2.1.2 Capa de enlaces

Myrinet es especificada al nivel del enlace de datos y en los niveles físicos SAN (red de corto alcance, alto-desempeño y baja latencia en un sistema distribuido de computadoras) y LAN [13].

Myrinet usa enlaces *full-duplex* con canales paralelos de 9 bits (1 bit de control y los 8 restantes de datos) en una dirección a 80 MHz. La red ofrece un ancho de banda físico de 1.28 Gb/s (160 MB/s) sobre un canal, ya que ambos extremos del reloj son usados para la transmisión de datos. Los canales de entrada y salida pueden estar activos al mismo tiempo, resultando en un ancho de banda de 1.28+1.28 Gb/s.

Los puertos "nativos" de la tarjeta de red y los chips de los conmutadores de barras cruzadas son puertos SAN. Los cables SAN pueden ser hasta de 3 metros de longitud, y son para aplicaciones en-cabina. Los enlaces LAN operan a la misma tasa de transferencia de datos como los enlaces SAN, en cables multiconductores de hasta 10.7 metros, y a la mitad de la tasa de transferencia de datos hasta 18.3 metros [13].

2.1.3 Capa de paquetes

Los paquetes de datos consisten de un encabezado de ruta, un campo de tipo, los datos y un CRC. Myrinet usa encaminamiento fuente (*source path routing*). Entrando a un conmutador, el primer byte del encabezado codifica el puerto de salida. El conmutador remueve el byte líder y remite la parte restante del paquete al puerto apropiado de salida. Cuando el paquete alcanza la tarjeta de red destino, el encabezado de ruta fue completamente removido y el campo de tipo dirige el mensaje.

2.2 GM

GM (Glenn's Messages) es un sistema ligero de comunicación de paso de mensajes para Myrinet (ver sección 2.1) diseñado por Myricom [13]. Las principales características de GM son las siguientes:

- Baja latencia y alto ancho de banda.
- Entrega ordenada y confiable entre puntos finales de comunicación
- Escalable: *clusters* de más de 10,000 computadoras [11].
- Mensajes de longitud de hasta $2^{31} - 1$ bytes. Este número es limitado por la cantidad de memoria de acceso directo que el sistema operativo puede asignar [11].
- Mapea automáticamente redes Myrinet.

GM también tiene ciertas limitaciones [11]

- No soporta operaciones *gather*[2] o *scatter* directamente.
- No es posible mandar o recibir mensajes de memoria que no puede ser accedida directamente.

El punto de vista de un "cliente" de GM es muy simple, tan sólo consiste de una biblioteca y un archivo de encabezado: `libgm.a` y `gm.h` respectivamente. GM divide los mensajes en paquetes para limitar el tiempo que cualquier paquete puede monopolizar la red. Las longitudes de paquetes son hasta de `GM_MTU` (4096) bytes.

2.2.1 Modelo de programación

GM provee un modelo de comunicación *no orientado a conexión*, en el sentido que el software cliente no necesita establecer una conexión con un puerto remoto para comunicarse con él. Sin embargo, GM mantiene *conexiones confiables entre* cada par de computadoras en la red, multiplexando el tráfico entre puertos sobre estas conexiones. Esta es la manera en la cual GM provee "*confiabilidad no orientada a conexión*".

GM provee dos niveles de prioridad: `GM_LOW_PRIORITY` y `GM_HIGH_PRIORITY`. Los mensajes con prioridades distintas nunca se bloquean entre sí [11]. A menudo se utiliza alta prioridad para transmitir mensajes de control (como acks cliente-a-cliente).

Para inicializar GM, el cliente debe mandar llamar la función `gm_init()`. La función `gm_finalize()` debe ser llamada después de todas las llamadas a función de GM. Para poder mandar o recibir cualquier mensaje, un *puerto*[3] debe ser inicializado vía la función `gm_open()`.

GM provee las siguientes funciones para asignar y liberar memoria de acceso directo: `gm_dma_malloc()`, `gm_dma_calloc()` y `gm_dma_free()`; y para

[2] Refiérase al Glosario para ver las definiciones de las operaciones *gather* y *scatter*

[3] Por default, GM 1.1.1 solo soporta 8 puertos y solo 6 de estos (puertos 2,3,4,5,6,7) están disponibles a software de nivel de usuario no privilegiado. El puerto 3 es usado por el *manejador* (*driver*) de IP-sobre-Myrinet

grandes requerimientos de memoria: `gm_register_memory()` y `gm_deregister_memory()` para registrar y liberar memoria en sistemas operativos que permiten el registro de memoria.

Las funciones de envío y recepción de *mensajes en GM son*, `gm_send_with_callback()` y `gm_receive()` respectivamente. Estas funciones son reguladas por un simple mecanismo de paso de estafetas (*tokens*) para prevenir el desbordamiento de las colas internas de tamaño limitado de GM. Un cliente necesita poseer una *estafeta de recepción* para poder recibir un mensaje. Analogamente, el cliente debe de poseer una *estafeta de envío* para poder mandar un mensaje. Una vez inicializado un puerto, el cliente posee `gm_num_send_tokens()` estafetas de envío y `gm_num_receive_tokens()` estafetas de recepción. El cliente pierde una estafeta de envío o recepción cada vez que realiza un envío o recepción respectivamente. La estafeta está disponible nuevamente una vez que la *operación ha completado*.

GM tiene ciertas colas internas para controlar el flujo de las estafetas de usuario: una cola de envíos, una cola de recepción y una cola de eventos. Las primeras dos *residen en la memoria SRAM* de la tarjeta de red (memoria LANai) y la última en la *memoria virtual del usuario*. Cuando el software cliente manda llamar una operación de envío o recepción, un descriptor de la operación es registrado en la cola apropiada (cola de envíos o cola de recepción) en memoria LANai. Cuando la operación ha sido completada, un nuevo descriptor es escrito a la cola de eventos. Un cliente puede accesar esta última a través de la función `gm_receive()`, la cual regresa el primer evento recibido que está pendiente o `GM_NO_RECV_EVENT` si no se ha recibido ningún evento. La función `gm_unknown()` deberá ser llamada cuando el cliente recibe un evento no-reconocido.

*A*l continuación se *describen los procesos de envío y recepción*. El cliente realiza un envío a través de la función:

```
gm_send_with_callback(*port, *message, size[4], length, priority,
    target_node_id, target_port_id, callback, *context)
```

El cliente pasa la función `callback` y el apuntador `context` a la función de envío. Cuando el envío ha *sido completado*, GM *manda* llamar la función `callback` a través de la función `gm_unknown()`, pasando un apuntador al puerto de GM, el apuntador aportado por el cliente `context`, y el código de *estado indicando* si el envío se completó con éxito o con error. Cuando GM manda llamar la función `callback` del cliente, la estafeta de envío es pasada implícitamente al cliente nuevamente.

El cliente necesita proveer un *buffer* antes de que pueda recibir algún mensaje, que corresponda al tamaño y prioridad del mensaje por llegar, esto se realiza a través de la función:

```
gm_provide_receive_buffer (*port, *buffer, size, priority)
```

[4] *size* tiene un significado especial en GM. *size*=$\{x \in Z \mid x \geq log_2(\text{longitud} + 8)\}$

Después de proveer buffers para la recepción de mensajes, el cliente puede sondear por recepciones pendientes usando `gm_receive()`. La recepción de eventos de tipo `GM_RECV_EVENT` y `GM_HIGH_RECV_EVENT` describen paquetes recibidos de baja y alta prioridad respectivamente. Cualquier otro evento deberá ser pasado a la función `gm_unknown()`.

2.3 FM

FM (Fast Messages) es una capa de paso de mensajes de bajo nivel que omite al sistema operativo con semántica de AM (ver sección 2.3.1). A continuación se mencionan algunas de las características más importantes de este protocolo [7, 3]:

- Provee acceso eficiente a la capa de transporte subyacente.
- FM es usado para la implementación de capas de más alto nivel y para *dar a las aplicaciones un mayor control sobre la comunicación.*
- FM es el corazón de la capa de comunicación del proyecto HPVM (ver sección 2.4). Las principales características de HPVM son las siguientes [8]:
  - Entrega confiable
  - Preserva el orden de transmisión
  - Superposición de comunicación y cómputo
  - Libre de bloqueo (deadlock) de comunicación
  - Manejo del flujo de control y memoria intermedia
  - Operaciones eficientes de *gather/scatter*
  - Procesos múltiples por computadora (nodo)
  - APIs (interfaz de aplicación) múltiples por programa
- Si no es usado en el contexto de HPVM, FM puede ser usado para:
  - *Implementar capas de paso de mensajes de alto nivel y alto desempeño.*
  - Escribir programas dinámicos (programas en los cuales los procesos *vienen y se van durante el curso del cómputo*).
  - Escribir programas que usan estructuras de comunicación de propósito especial, los cuales serían difícil de implementar usando otra capa de paso de mensajes [7].

2.3.1 AM

AM (Active Messages) es un *paradigma de comunicación unilateral*; esto es, cuando el proceso transmisor envía un mensaje, el intercambio de información ocurre independientemente de la actividad actual del proceso receptor. Como un efecto secundario, no existe la necesidad de una operación de recepción.

El objetivo de AM es reducir el impacto en el procesamiento de la comunicación en el desempeño de la aplicación [3]. La semántica de AM puede eliminar la necesidad de mucho almacenamiento temporal para mensajes a lo largo del camino de la comunicación, disminuyendo notablemente el tiempo de respuesta [3].

La semántica de AM difiere del modelo tradicional *send/receive* como sigue: tan pronto como cada mensaje es entregado, éste dispara una función programada por el usuario del proceso destino, llamada *manejador receptor*. El manejador receptor actúa como un hilo (thread) el cual "consume" puntualmente el mensaje, separando el manejo del mensaje, de la actividad actual del hilo principal del proceso destino. Cuando se menciona la palabra "consumir" se refiere a integrar la información del mensaje en el curso del cómputo del hilo principal, notificando la llegada del mensaje al mismo, y posiblemente configurando algunas estructuras de datos para consumir puntualmente el siguiente mensaje tan pronto como llegue.

FM es un buen ejemplo de un sistema con semántica de AM. Con la mayoría de los protocolos orientados a mensajes activos, el manejador receptor está *basado en el transmisor* [7, 3]. Cada mensaje está compuesto de dos partes denominadas el cuerpo del mensaje y, un apuntador explícito al manejador receptor del destino, el cual extrae el mensaje y lo almacona en una estructura de datos en el espacio de direcciones del receptor. Tal modelo requiere que el proceso destino comparta su propia dirección de código con el proceso transmisor, una condición fácilmente satisfecha por el paradigma *SPMD (Single Program Multiple Data)*.

2.4 HPVM

HPVM[5] (High Performance Virtual Machines) es un conjunto de herramientas de software diseñadas para permitir cómputo de alto-desempeño en recursos computacionales distribuidos. Fue diseñado para trabajar en *clusters* de microprocesadores de bajo costo como una alternativa a procesadores vectoriales que son muy caros [9]. HPVM se enfoca principalmente en los siguientes puntos [8]:

- Entregar alto-desempeño en la comunicación con APIs estándar de alto-nivel.

[5] diseñado por el Dr. Andrew A. Chien [5] y sus asociados en el grupo CSAG (Concurrent Systems Architecture Group).

- Coordinar la planificación y manejo de recursos.
- Manejar la heterogeneidad.

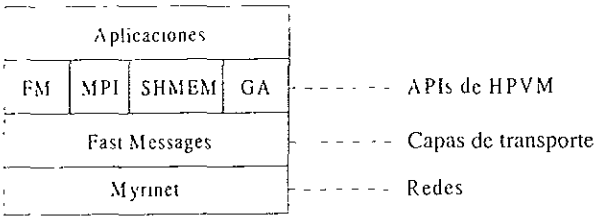

Figura 2.1 Sistema en capas de HPVM.

HPVM se puede considerar como un sistema en capas, como lo muestra la figura 2.1. Existen cuatro capas principales (de bajo a alto nivel): *Red*, *Transporte*, *API* y *Aplicación*. Las aplicaciones pueden ser de bajo o alto nivel dependiendo del API de la capa inferior. Los APIs son implementados usando FM (ver sección 2.3). Actualmente, HPVM solo soporta *Myranet* [13]. Está planeado que en un futuro sean soportadas *Servernet*, *Giganet* y "*Winsock*[6]" [9]. HPVM soporta las siguientes APIs:

FM (***Fast Messages***) Capa de bajo nivel de paso de mensajes que omite al sistema operativo con semántica de AM, usada principalmente para implementar capas de paso de mensajes de un nivel más alto.

MPI (***Message Passing Interface***) Capa estándar de alto nivel de paso de mensajes con semántica *send/receive* Contiene un gran número de funciones para las operaciones de comunicación *colectiva* y *punto-a-punto* MPI es usado para construir aplicaciones paralelas, portables y de propósito-general.

GA (***Global Arrays***) Capa de alto nivel de paso de mensajes con semántica *put/get*[7] y un modelo de programación del espacio de direccionamiento compartido, pero de acceso no-uniforme (NUMA) usado por programas intensivos de álgebra lineal.

SHMEM (***Shared Memory***) Capa de bajo nivel de paso de mensajes con semántica *put/get* y un modelo de programación del espacio de direccionamiento compartido. Puede ser usado para exportar aplicaciones de la Cray T3D/T3E a un *cluster* con HPVM.

[6] Wmsock (Windows Socket) no es una red, es un API para el desarrollo de programas en Windows que pueden comunicarse con otras computadoras vía el protocolo TCP/IP.

[7] *Put* y *Get* son operaciones para la comunicación entre procesos, ejecutando operaciones remotas de lectura y escritura. A diferencia de las operaciones bilaterales *send/receive* éstas son unilaterales ya que no necesitan la cooperación del proceso(s) que posee los datos.

> Se está trabajando en el soporte de nuevos APIs para incluir memoria compartida y distribuida basada en paginación, tales como el DCOM (Distributed Component Object Model), sockets, y HPF (High-performance FORTRAN). La idea es permitir que los usuarios puedan poner su código directamente en un *cluster* NT. Si su código se ejecuta bien en una Paragon, SP2, o CM-5, entonces se ejecutará bien bajo HPVM."[9]

Entre otras de las partes principales de HPVM está el DCS[8] (*Dynamic Co-Scheduler*). El DCS es un mecanismo que coordina la planificación de operaciones interdependientes, distribuidas por una aplicación paralela, para correr concurrentemente en procesadores separados. El DCS reconoce cuando operaciones distribuidas requieren interactuar con otras y trabaja conjuntamente con el sistema operativo para sincronizar a los procesadores ejecutando esas operaciones [8]. En otras palabras, el DCS está a cargo de la coordinación de los recursos del *cluster* (planificando procesadores, memoria y entrada/salida) para mejorar su desempeño.

Actualmente HPVM soporta solo Windows NT, versiones anteriores[9] soportaban Linux 2.x también, pero no más. Finalmente, el desempeño de HPVM supuestamente debe competir con sistemas MPP tales como IBM SP2 y Cray T3D:

> "Alcanza un desempeño en la comuninación de bajo nivel impresionante a través del *cluster*: latencias $\lhd$ 11usec y, anchos de banda > 100MBytes/sec—aún para paquetes pequeños ($\lhd$256 bytes]." [8]

2.5 MPI

El objetivo de MPI (*Message Passing Interface*) es desarrollar un estándar usado ampliamente para escribir programas de paso de mensajes. Como tal, la interfaz procura establecer un estándar *práctico, portable, eficiente* y *flexible* para el *paso de mensajes* [2].

En el diseño de MPI, el MPIF intentó hacer uso de las características más atractivas de los sistemas de paso de mensajes existentes en lugar de tomar uno de ellos y adoptarlo como el estándar. Así, MPI ha sido influenciado fuertemente por IBM T. J. Watson Research Center, Intel's NX/2, Express, nCUBE's Vertex, p4 y PARMACS [1]. Otras contribuciones importantes han venido de Zipcode, Chimp, PVM, Chameleon y PICL.

Las ventajas principales del establecimiento de un estándar de paso de mensajes son *portabilidad* y *facilidad-de-uso* [1]. Los beneficios de la estandarización son particularmente aparentes en un ambiente de comuni-

[8] Basado en el trabajo realizado en "Massachusetts Institute of Technology" por Patrick Sobalvarro.

[9] La versión más reciente de HPVM es: HPVM 1.9 para Windows NT 4.0 para x86

cación de memoria distribuida en el cual las rutinas de alto nivel y/o abstracciones son construidas sobre rutinas de paso de mensajes de un nivel más bajo. Más aún, la definición de un estándar de paso de mensajes provee a vendedores, de un conjunto de rutinas básicas claramente definidas que pueden implementar eficientemente, o en ciertos casos proveer el hardware necesario para soportarlas, incrementando así la escalabilidad [2].

A continuación se presentan ciertos conceptos clave en MPI: *grupo*, *comunicador* y *contexto*. Entre las características de éstos, aparece la noción de **objetos opacos**. MPI maneja la memoria del sistema usada para el almacenamiento de mensajes en memoria temporal, y para el almacenamiento de representaciones internas de varios objetos de MPI tales como grupos, comunicadores, tipos de datos, etc. Esta memoria no está accesible directamente al usuario, y los objetos almacenados ahí son *opacos*: el usuario no puede "ver" su tamaño ni estructura. Los objetos opacos son accedidos a través de manejadores, existentes en el espacio del usuario. A los procedimientos de MPI que operan en objetos opacos, se les pasan manejadores para el acceso a estos objetos.

Un **grupo** es un conjunto ordenado de procesos. Cada proceso en un grupo es asociado con un entero, llamado *rango*. Los rangos son contiguos y empiezan desde cero. Los grupos son representados por grupos de objetos *opacos*, por lo que no pueden ser transferidos directamente de un proceso a otro. Un grupo es usado dentro de un *comunicador* para describir los participantes en un "universo" de comunicaciones y para clasificar tales participantes (dándoles así nombres únicos dentro de ese universo de comunicación). Hay un grupo especial pre-definido: `MPI_GROUP_EMPTY`, el cual no tiene miembros. La constante predefinida `MPI_GROUP_NULL` es el valor usado para manejadores de grupo inválidos.

Los **comunicadores** conjuntan los conceptos de grupo y contexto. Las operaciones de comunicación de MPI hacen referencia a comunicadores para determinar el alcance y el "universo de comunicación" en el cual una operación punto-a-punto o colectiva operará. Cada comunicador contiene un grupo de participantes válidos; este grupo siempre incluye el proceso local. La fuente y el destino de un mensaje son identificados por el rango del proceso dentro de ese grupo.

Para la comunicación colectiva, el comunicador especifica el conjunto de procesos que participan en la operación colectiva. Así, el comunicador restringe el alcance "espacial" de la comunicación, y provee un direccionamiento de procesos independiente de la computadora a través de los rangos. Los comunicadores son representados por objetos de comunicador opacos. Un comunicador inicial `MPI_COMM_WORLD` de todos los procesos con los que el proceso local puede comunicarse después de la inicialización (incluido él mismo), es definido una vez que `MPI_INIT` ha sido invocado. Además, se provee el comunicador `MPI_COMM_SELF`, el cual incluye solo el proceso mismo. La constante predefinida `MPI_COMM_NULL` es el valor usado por manejadores de comunicador inválidos.

Hay dos tipos de comunicadores: *intra-comunicadores* para operaciones dentro de un grupo de procesos, e *inter-comunicadores*, para la comunicación punto-a-punto entre dos grupos de procesos que no se traslapan. Los intra-comunicadores contienen una instancia de un grupo, *contextos* de comunicación para la comunicación punto-a-punto y colectiva, y otros atributos.

Un **contexto** es una *propiedad* de los comunicadores que permite dividir el espacio de comunicación en particiones. Los contextos proveen la habilidad de tener un universo separado de paso de mensajes entre dos grupos. Una operación de envío en el grupo local siempre es una operación de recepción en el grupo remoto, y viceversa. Un mensaje enviado en un contexto no puede ser recibido en otro contexto. No hay comunicación colectiva de propósito general en los inter-comunicadores, por lo que los contextos son utilizados para aislar la comunicación punto-a-punto. Los contextos no son objetos de MPI explícitos; aparecen tan solo como parte de la realización de los comunicadores.

MPI utiliza un modelo de procesos de *conexión-total*, donde a cada proceso le es permitido realizar envíos a cualquier proceso en la aplicación, una vez que la biblioteca de MPI es inicializada. Dado que una implementación de MPI no puede saber los patrones de comunicación de la aplicación *a priori*, conexiones punto-a-punto deben ser completamente establecidas durante la inicialización de la biblioteca, o cuando se necesiten. La mayoría de las implementaciones de MPI establecen todas las conexiones durante inicialización, para evitar complejidad y proveer un desempeño determinístico. El establecimiento de conexiones con base en el uso requiere que un "oyente" esté siempre listo para establecer una conexión, incrementando así la complejidad de la implementación. Operaciones con latencia baja, requieren que las conexiones sean establecidas antes de la ejecución de las operaciones de comunicación

MPI soporta el concepto de *mensajes inesperados*. Mientras que el estándar de MPI puede soportar implementaciones sin *buffers*, el tiempo de procesamiento del protocolo incurrido por tal acercamiento, es usualmente significativo [1]. La latencia baja para mensajes pequeños es usualmente alcanzada a través de proveer memoria temporal del lado del receptor. La cantidad de memoria temporal requerida en el receptor es dependiente de diversos factores, tales como *latencia y ancho de banda de la red, ancho de banda del copiado de memoria* y el *patrón de comunicación* [1].

Hay dos versiones del documento de *MPI-1*: versión 1.0 y versión 1.1. Hay nuevas secciones de funcionalidad tratadas en el estándar *MPI-2*, entre otras:

- Manejo de procesos dinámicos
- Comunicación unilateral
- Operaciones colectivas extendidas

- Interfaces externas
- E/S paralela

Hay diversas implementaciones de MPI. Algunas de ellas creadas por vendedores, otras son libres y portables a diferentes máquinas. Las implementaciones de MPI utilizadas en este estudio son *MPI-FM* (University of Illinois, CSAG), *MPICH* (ANL) y *MPICH/GM* (Myricom).

Capítulo 3

Estudio

3.1 Equipo

- Hardware

  Sistemas Dell Dimension XPS T500 Microprocesador Intel Pentium III a 500 MHz. Caché de nivel 1 (L1) que consiste de 32 KB de SRAM residente en el corazón del procesador. Caché de nivel 2 (L2) que consiste de 512 KB de SRAM burst en un chip separado en la tarjeta del procesador. 256-MB de SDRAM.

  Tarjetas de red Myrinet Interfaces Myrinet-LAN/PCI. Interfaz de canal PCI de 32-bits, 33-MHz. Procesador de interfaz LANai-4 RISC operando a la tasa del reloj del PCI. Memoria SRAM de 512KB (128KBx4B). La memoria opera al doble de la tasa del reloj del PCI. La tasa de datos de los puertos Myrinet-LAN es de 1.28+1.28 Gb/s.

  Conmutador Myrinet–LAN de 8 puertos Conmutador de 8 entradas, 8 salidas, implementación de barras cruzadas (full-crossbar) con cada canal operando a 1.28 Gb/s para una tasa de datos de biseccion total do 10.24 Gb/s. 8 puertos Myrinet-LAN, tasas de datos de 1.28+1.28 Gb/s. Latencia *cut-through*: 300*ns*.

  Cables Myrinet-LAN Estos cables transportan paquetes a una tasa de datos de 1.28+1.28 Gb/s. Son de aproximadamente 0.34 pulgadas de diámetro, y son suficientemente flexibles para doblarse con un radio de 3 pulgadas.

  3Com OfficeConnect Hub TP800 Permite la conexión de hasta 8 usuarios a alta velocidad sobre una Fast Ethernet LAN. Provee 8 puertos 100BASE-TX RJ-45.

- Software

  – FM 1.0

- GM 1.04
- MPI-FM 1.0
- MPICH 1.1.2
- MPICH/GM 1.1.2..10
- Red Hat Linux 6.0 (Kernel 2.2.5-15)

3.2 GM

Este primer conjunto de mediciones fueron tomadas usando GM (ver sección 2.2) versión 1.04. Las pruebas fueron realizadas utilizando dos computadoras, una de ellas enviando y la otra recibiendo mensajes de cada longitud graficada repetidamente por un intervalo dado (1 segundo, en este caso).

Como se muestra en la figura 3.1, el ancho de banda alcanza hasta 76.4 MB/s, que constituye cerca del 47.7% del ancho de banda de Myrinet. Esto significa que el tiempo de procesamiento para el envío y recepción de mensajes es muy alto. El ancho de banda máximo es alcanzado aproximadamente a los 4 KB. Es importante mencionar que GM fragmenta mensajes en paquetes de 4 KB, esto se ve reflejado en las discontinuidades de la gráfica cada 4 KB.

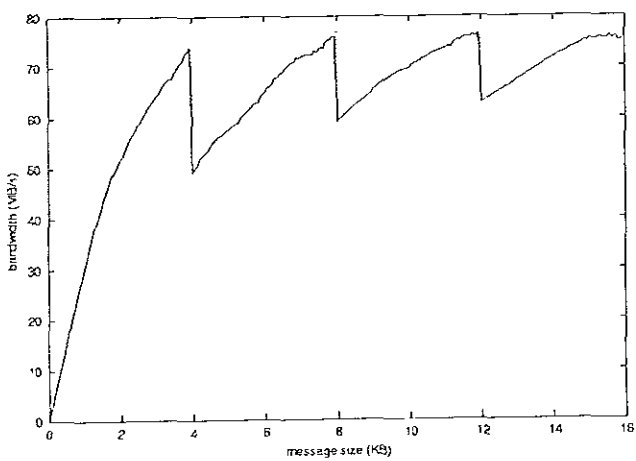

Figura 3.1: Ancho de banda de GM para mensajes de hasta 16 KB.

Como se muestra en la figura 3.2, GM presenta una caída significativa (de 74.6 a 60.1 MB/s) cerca de los 36 KB. Después de ese punto presenta un comportamiento inestable fluctuando entre los 50 y 60 MB/s como se muestra en la figura 3.3.

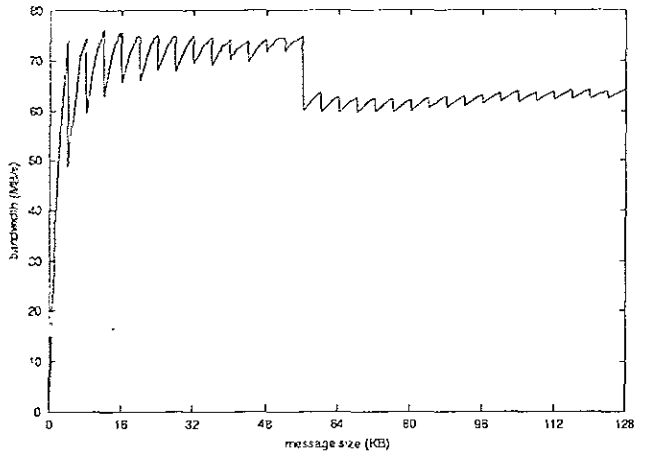

Figura 3.2: Ancho de banda de GM para mensajes de hasta 128 KB.

La latencia de GM para mensajes pequeños es de 30.7μs (ver figura 3.4) e incrementa *linealmente* a una tasa de 0.017 como se muestra en la figura 3.5.

Myricom ha reportado muy buen desempeño usando la nueva versión de GM (GM 1.1.1) y las interfaces Myrinet/PCI de 64 bits, 66 MHz: ancho de banda de 140 MB/s para mensajes grandes y una latencia de mensajes pequeños de 18μs. Todavía existen muchas incompatibilidades de hardware para la nueva versión de GM y las interfaces Myrinet/PCI de 32 bits, 33 MHz. Esta es la razón principal por la cual se ha usado la versión anterior de GM en lugar de la nueva para este estudio.

3.3 FM

Originalmente se pretendían incluir en este estudio mediciones del desempeño de HPVM (ver sección 2.4) usando los APIs FM y MPI. Sin embargo, HPVM ya no está disponible para plataformas Linux, la versión más reciente de HPVM[1] es para Windows NT solamente. Por lo que esta sección y la 3.6

[1] HPVM 1.9 para Windows NT 4.0 en x86.

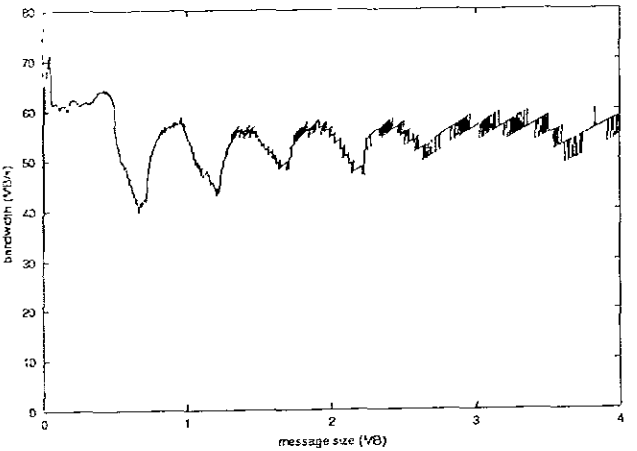

Figura 3.3: Ancho de banda de GM para mensajes de hasta 4 MB.

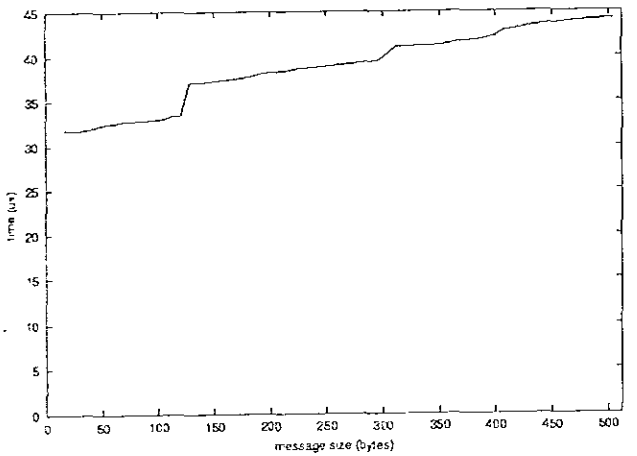

Figura 3.4: Latencia de GM para mensajes de hasta 512 bytes.

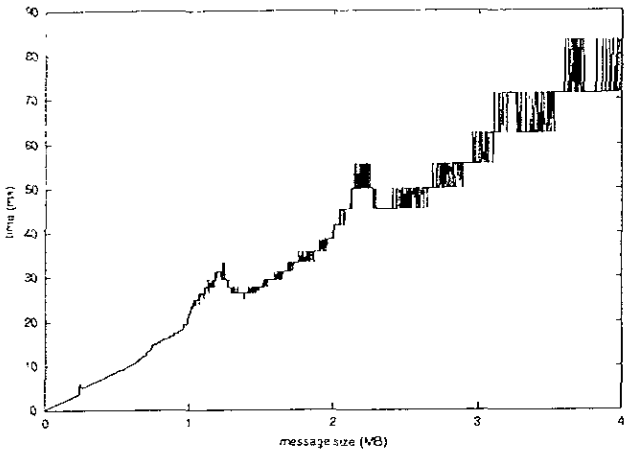

Figura 3.5: Latencia de GM para mensajes de hasta 4 MB.

está dedicada tan solo a mencionar algunas de las mediciones de desempeño realizadas por [8] de MPI-FM y FM.

FM (ver sección 2.3) constituye la capa de transporte en el sistema de HPVM. FM puede mejorar el desempeño de la comunicación en dos maneras importantes: la latencia es reducida eliminando el exceso de copiado de datos y la capacidad de ejecución es incrementada permitiendo que el receptor comience a procesar el mensaje aún antes de que el emisor haya terminado de mandarlo.

La latencia de FM para mensajes pequeños comienza a los 9.63μs como se muestra en la figura 3.6.

Desafortunadamente, el ancho de banda no es muy bueno, ya que FM tan solo obtiene 63.6% del ancho de banda bruto de Myrinet. El máximo ancho de banda alcanzado es cerca de los 101.8 MB/s aproximadamente a los 2 KB (ver figura 3.7).

3.4 MPICH

En este caso particular, *la red subyacente es Fast Ehternet.* El ancho de banda bruta para este tipo de redes es de 100 Mb/s i e., 12.5 MB/s. Encima de Ethernet está *TCP/IP* y finalmente en la capa de aplicación, *MPICH 1.1.2.* Las pruebas fueron realizadas utilizando dos computadoras, una de ellas

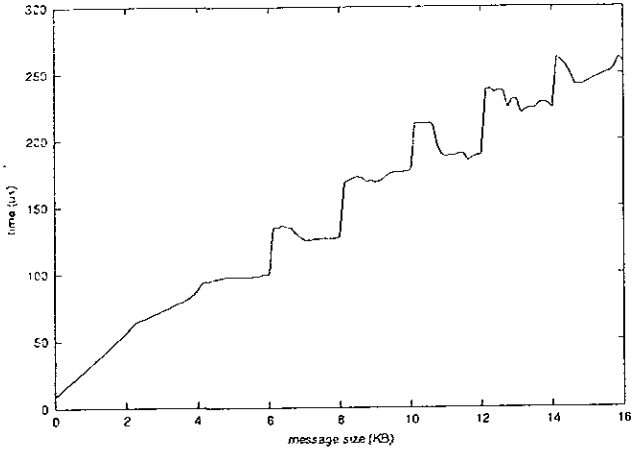

Figura 3.6: Latencia de FM para mensajes de hasta 16 KB.

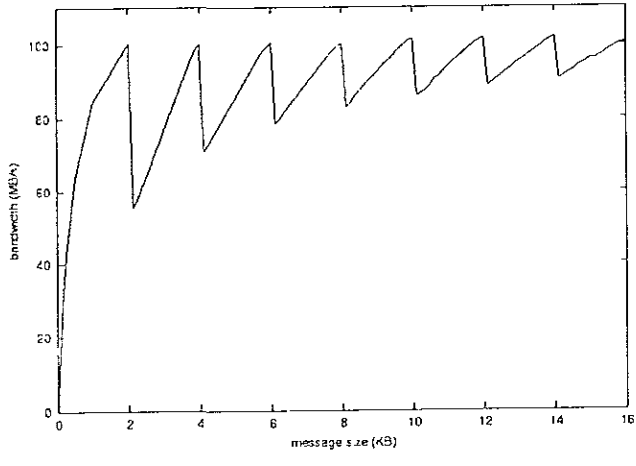

Figura 3.7: Ancho de banda de FM para mensajes de hasta 16 KB.

enviando y la otra recibiendo mensajes usando *non-blocking sends/receives* de cada longitud graficada repetidamente por un intervalo dado, el cual depende de la longitud del mensaje a enviar (varía de 10^{-1} a 10^{-6}).

Como se muestra en la figura 3.8, hay discontinuidades cada 1460 bytes. Este es el resultado del tamaño de la MTU inherente a la red Fast Ethernet, 1500 bytes. La diferencia entre estos dos números se debe al tamaño de los encabezados de IP y TCP (normalmente 20 bytes cada uno). El ancho de banda alcanza hasta 10.2 MB/s (ver figura 3.9), que constituye cerca del 81.8% del ancho de banda bruto. Este es un buen número considerando que MPICH usa el protocolo TCP/IP. El ancho de banda máximo es alcanzado aproximadamente a los 16 KB. El comportamiento de MPICH es bastante estable aún para mensajes grandes, fluctuando cerca de los 9 MB/s como muestra la figura 3.10.

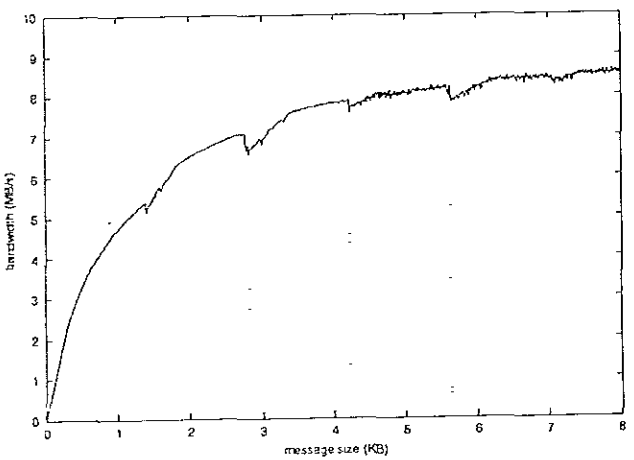

Figura 3.8: Ancho de banda de MPICH para mensajes de hasta 8 KB.

La latencia para mensajes pequeños de MPICH es de 115.2μs (ver figura 3.11) e incrementa linealmente a una tasa de 0.110 como se muestra en la figura 3.12. La latencia es alta debido a las copias de memoria necesarias en TCP/IP.

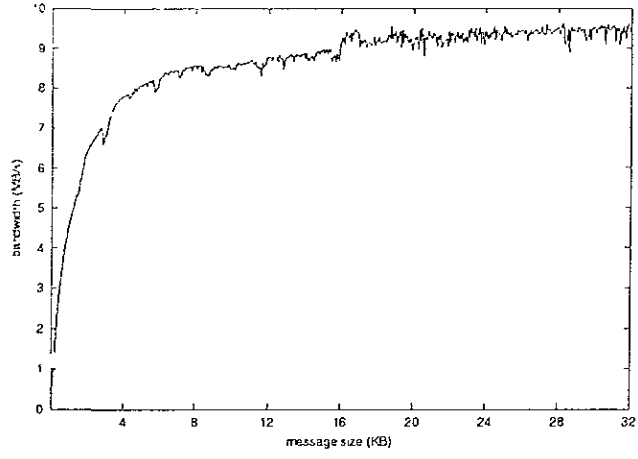

Figura 3.9: Ancho de banda de MPICH para mensajes de hasta 32 KB.

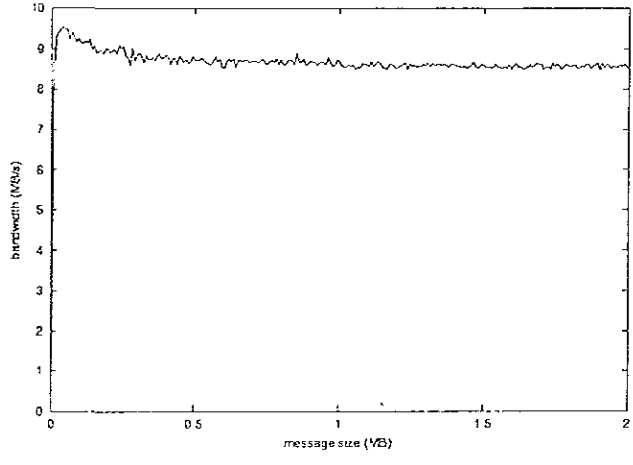

Figura 3.10: Ancho de banda de MPICH para mensajes de hasta 2 MB.

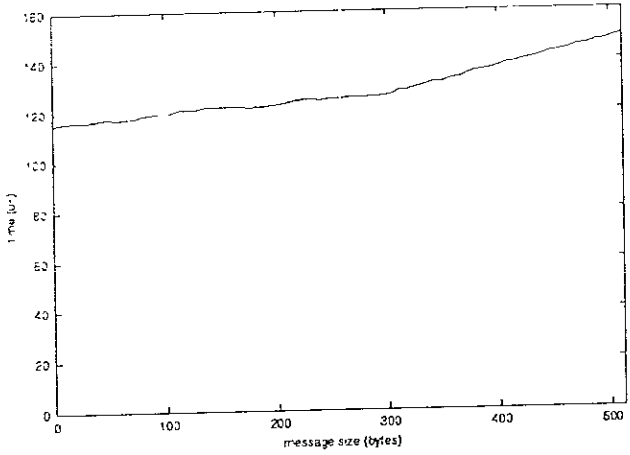

Figura 3.11: Latencia de MPICH para mensajes de hasta 512 bytes.

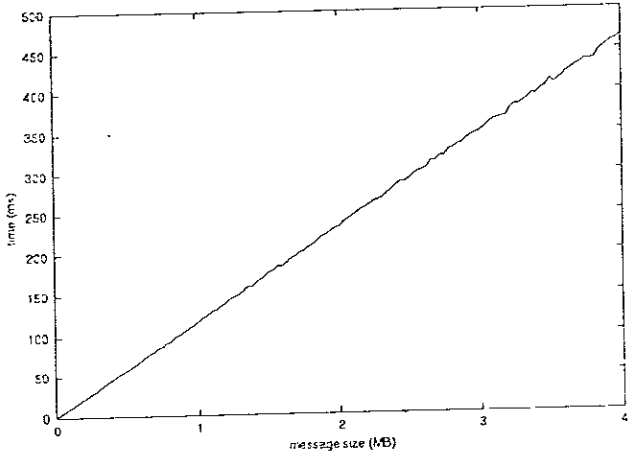

Figura 3.12: Latencia de MPICH para mensajes de hasta 4 MB.

3.5 MPICH/GM

MPICH/GM es una implementación portable de MPI diseñada para correr sobre GM (protocolo que omite al sistema operativo). Las mediciones fueron realizadas utilizando dos computadoras, una de ellas enviando y la otra recibiendo mensajes usando *non-blocking sends/receives* de cada longitud graficada repetidamente por un intervalo dado, que depende de la longitud del mensaje a enviar (varía de 10^{-1} a 10^{-6}).

Como se ha mencionado antes, la fragmentación de paquetes tiene un impacto significativo en el ancho de banda y latencia de un protocolo de comunicación. En este caso, la fragmentación de GM se ve reflejada en el desempeño de MPICH como se muestra en la figura 3.13. El ancho de banda máximo alcanzado es de 50.7 MB/s (31.7% del ancho de banda bruto), alcanzado cerca de los 128 KB. Nuevamente, la utilización no es muy buena dado que el protocolo de la capa inferior (GM) tiene un tiempo de procesamiento muy alto. Hay una caída grande (de cerca de 8 MB/s) a los 128 KB como muestra la figura 3.14. Después de ese punto, la capacidad de ejecución es bastante estable, cerca de los 39.4 MB/s (ver figura 3.15).

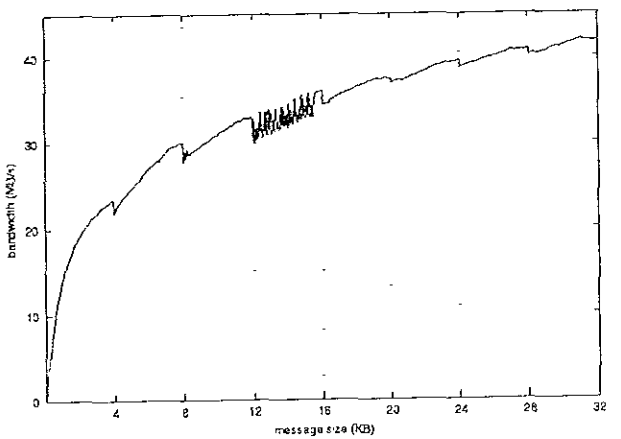

Figura 3.13: Ancho de banda de MPICH/GM para mensajes de hasta 32 KB.

La latencia de MPICH/GM para mensajes pequeños es de 31.18μs (ver figura 3.16) e incrementa linealmente a una tasa de 0.025 como se muestra en la figura 3.17.

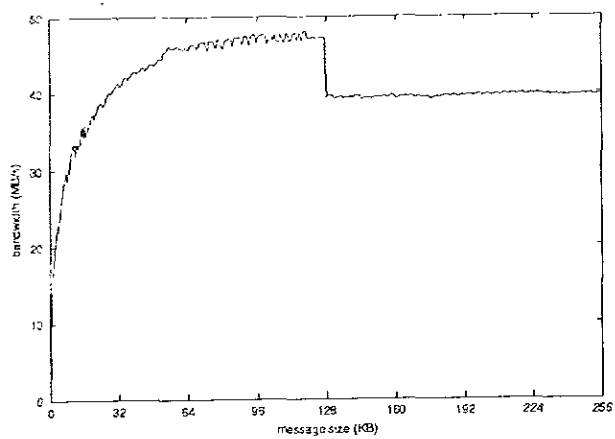

Figura 3.14: Ancho de banda de MPICH/GM para mensajes de hasta 256 KB

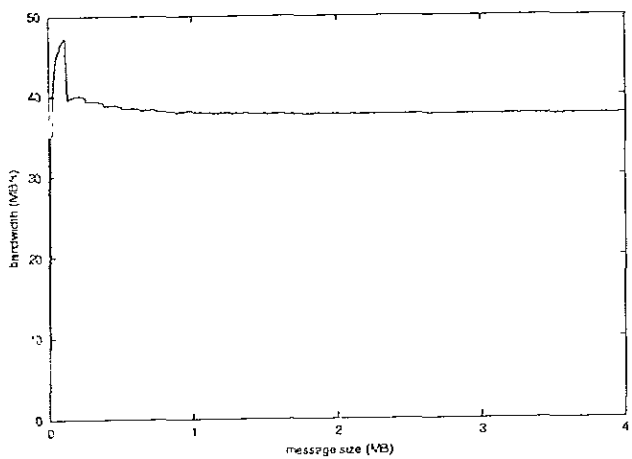

Figura 3.15: Ancho de banda de MPICH/GM para mensajes de hasta 4 MB.

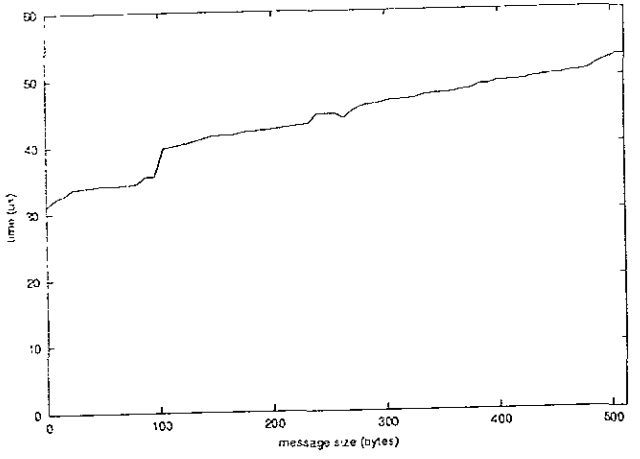

Figura 3.16: Latencia de MPICH/GM para mensajes de hasta 512 bytes.

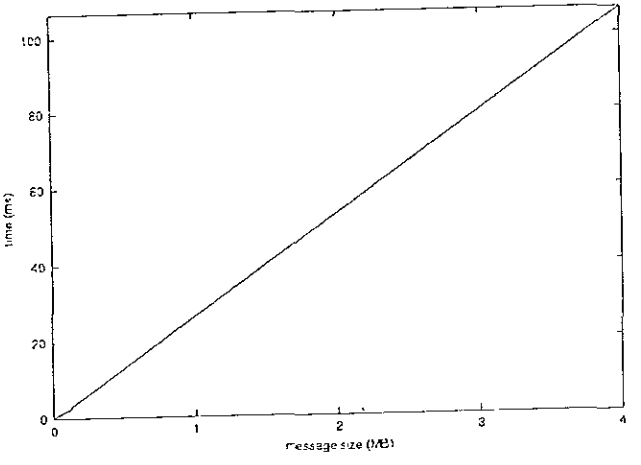

Figura 3.17: Latencia de MPICH/GM para mensajes de hasta 4 MB.

3.6 MPICH/FM

MPI-FM es una implementación de MPI basada en MPICH y portada a FM La latencia de MPI-FM para mensajes pequeños comienza a los 12.43μs (ver figura 3.18).

Como se muestra en la figura 3.19, el máximo ancho de banda alcanzado es de 85 7 MB/s cerca de los 14 KB, que constituye cerca del 53.6% del ancho de banda bruta. Este número no es muy bueno debido a la capa inferior (FM), más que de la implementación de MPI-FM por si sola Esta última obtiene un 84.2% del ancho de banda alcanzado por FM.

3.7 Poniendo todo junto

En las secciones anteriores de este capítulo se ha hecho énfasis en diversos aspectos que cada protocolo por separado ha presentado en términos de desempeño, como la fragmentación de mensajes, caídas significativas en el ancho de banda, etc. Esta sección está dedicada a resumir y comparar algunos de los resultados más importantes discutidos en estas secciones. En general y como referencia a la información tratada más adelante, ver tablas 3.1 y 3.2.

	GM	FM	MPICH	MPICH/GM	MPI-FM
MA	76.4	101.8	10.2	50.7	85.7
BL	50–60	¿?	9	39.4	¿?
ML	4	2	16	128	14
PN	47.7	63.6	81.8	31.7	53.6
PL	NA	NA	¿?	66.5	84.2
FL	4096	2048	1460	4096	¿?
LS	30.74	9.63	115.28	31.18	12.43
LL	71428.6	¿?	465589.5	106256.4	¿?
LR	0.01703	¿?	0.11097	0.02532	¿?

Tabla 3.1: Valores de desempeño para los diferentes protocolos.

No es justo comparar todos los protocolos discutidos previamente entre ellos mismos, dado que trabajan a diferentes niveles: "red/transporte" y aplicación. En el nivel de red/transporte se tienen dos sistemas de paso de mensajes: FM y GM. En el nivel de aplicación: MPICH, MPICH/GM y MPI-FM.

Como se muestra en la figura 3.20, FM es claramente mejor que GM en todos los aspectos, logrando un ancho de banda mas alto y a una longitud de mensaje más pequeña. Sin embargo, FM no es muy bueno en general, dado que obtiene tan solo el 63.6% del ancho de banda bruto de Myrinet (un "buen" protocolo obtendría arriba del 90%). Con respecto a la latencia,

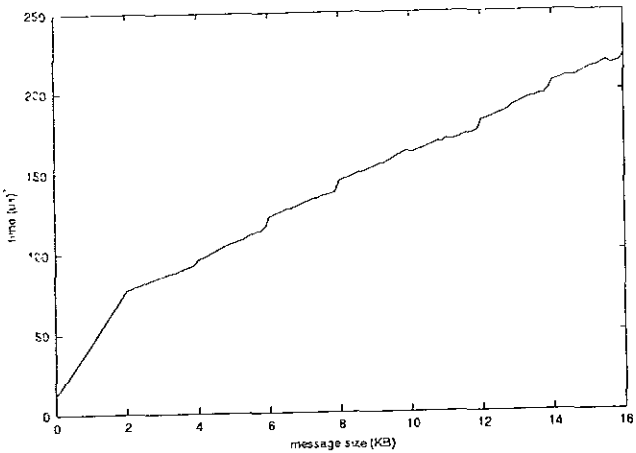

Figura 3.18. Latencia de MPI-FM para mensajes de hasta 16 KB.

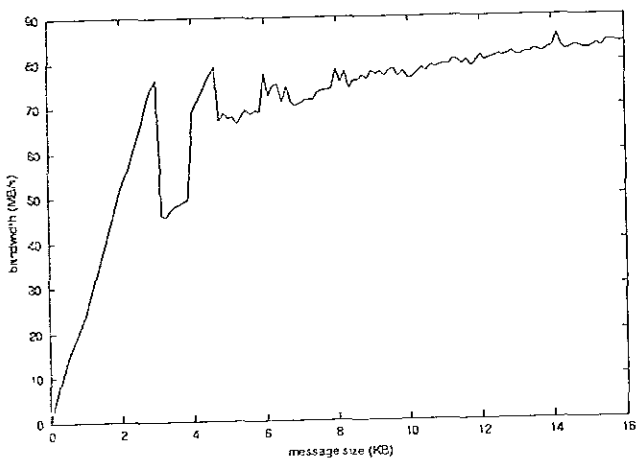

Figura 3.19: Ancho de banda de MPI-FM para mensajes de hasta 16 KB.

MA	Máximo ancho de banda alcanzado (MB/s)
BL	Ancho de banda para mensajes grandes (4 MB) en MB/s
ML	Longitud del mensaje para MA (KB)
PN	% de utilización del ancho de banda bruto
PL	% de utilización del ancho de banda de la capa inferior
FL	Longitud de la fragmentación (bytes)
LS	Punto de comienzo de latencia (μs)
LL	Latencia para mensajes grandes (4 MB) en μs
LR	Tasa de crecimiento de la latencia
¿?	Datos no recolectados :-(

Tabla 3.2: Abreviaciones usadas en tabla 3.1.

podemos observar en la figura 3.21, que FM es mejor nuevamente, logrando una latencia menor para mensajes pequeños y grandes.

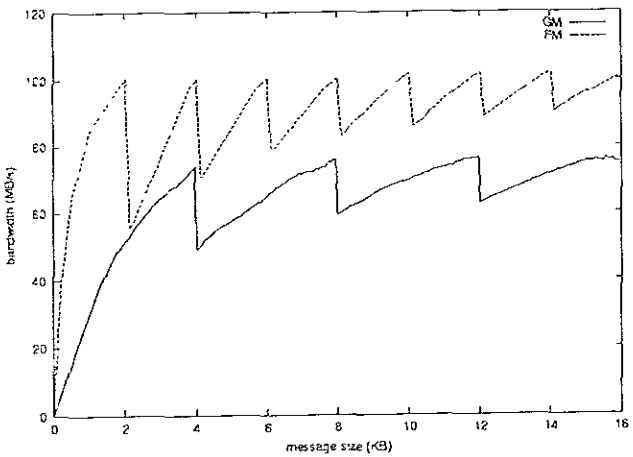

Figura 3.20: Ancho de banda: GM vs. FM.

En el nivel de aplicación, podemos comparar tres implementaciones de MPI, pero no debemos olvidar que MPICH corre sobre Fast Ethernet, mientras que MPICH/GM y FM-MPI corren sobre Myrinet. Esa es la razón por la que las medidas de desempeño favorecen notablemente las últimas dos implementaciones. Myrinet es una red de 160 MB/s mientras que Fast Ethernet es de tan solo 12.5 MB/s. La intención, al medir el desempeño sobre las dos redes es:

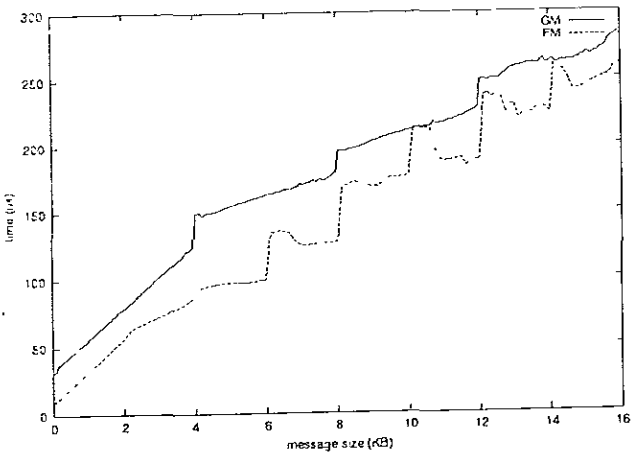

Figura 3.21: Latencia: GM vs. FM.

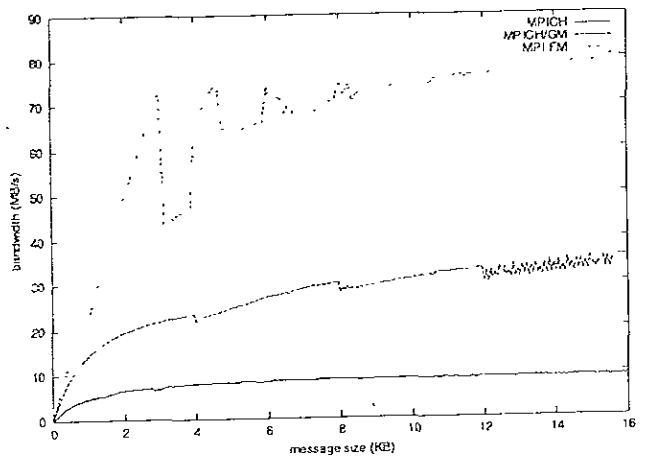

Figura 3.22: Ancho de banda: MPICH, MPICH/GM y MPI-FM.

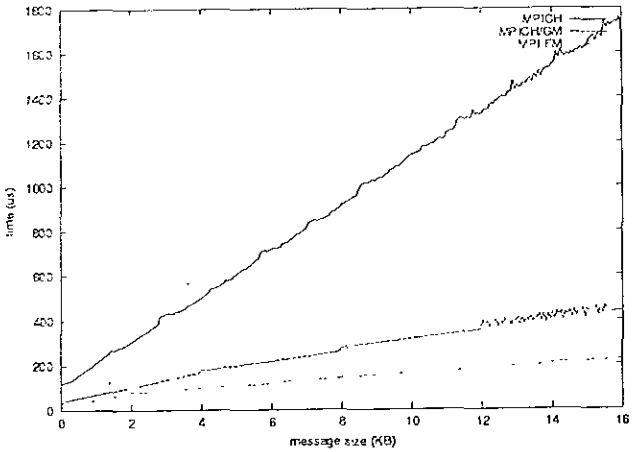

Figura 3.23: Latencia: MPICH, MPICH/GM y MPI-FM.

- observar el impacto resultante en el desempeño de la aplicación al utilizar *Fast Ethernet* en contraste con *Myrinet*.
- comparar la utilización de red de un protocolo como MPICH vs. su equivalente en una red de alta-velocidad.
- enfatizar el mejor desempeño logrado por protocolos que omiten al sistema operativo como FM y GM vs. protocolos como TCP/IP tradicionales y ampliamente usados.

Como se muestra en la figura 3.22, FM-MPI alcanza un ancho de banda mucho mayor que sus equivalentes competidores, como se esperaba. Sin embargo, si nos fijamos en la utilización del ancho de banda bruta, entonces MPICH es el mejor, con 81.8%, seguido de MPI-FM con 53.6%. Aunque la implementación de MPI-FM no es mala (la utilización de la capa inferior, FM, es cerca del 84.2%), FM es el factor limitante en este esquema.

Las mediciones de latencia son mostradas por la figura 3.23. Nuevamente, la latencia de MPI-FM es definitivamente menor que la de MPICH, y cerca de la mitad de la de MPICH/GM.

Capítulo 4

Conclusiones y trabajo futuro

FM—MPI-FM presenta el mejor desempeño en las mediciones realizadas en comparación con GM—MPICH/GM y MPICH. Esto se debe a varias razones:

- FM tiene la propiedad de omitir al sistema operativo.
- Bajo tiempo de procesamiento en la implementación de MPI-FM.
- FM trabaja sobre una red de alta-velocidad (Myrinet)

Aunque también GM es un protocolo que omite al sistema operativo, el tiempo de procesamiento en el envío y recepción de mensajes es muy alto. Afortunadamente, GM corre sobre Myrinet y eso hace que su desempeño en términos de ancho de banda y latencia sea mejor que MPICH. MPICH (para esta implementación en particular y con este hardware y software en particular) tiene fuertes limitaciones:

- Usa TCP/IP (alto tiempo de procesamiento debido a copias de memoria).
- Fue probado sobre Fast Ethernet (red lenta).

Como ya se mencionó, TCP/IP tiene la desventaja de correr sobre una red LAN, mientras que GM y FM corren sobre una red SAN. Si bien la comparación no es muy pareja dado que estos tipos de redes tienen propósitos diferentes, nos da una idea clara de la gran diferencia en desempeño entre éstas. Al respecto, *Duke University* ha estado trabajando en un proyecto muy interesante llamado "Trapeze" [4]. En este proyecto, se ha escrito TCP/IP Myrinet *firmware* para sistemas Intel y Alpha en Digital Unix y FreeBSD. Esto permite utilizar MPICH sobre Myrinet utilizando TCP/IP. Desgraciadamente la implementación para Linux no estaba *disponible* cuando se realizaron las mediciones.

Dados los resultados presentados es de señalar la importancia de la técnica de omitir al sistema operativo en sistemas ligeros de paso de mensajes y el uso de redes de alta-velocidad como Myrinet [11, 10]. Sin embargo, cabe señalar que el ancho de banda de la red fue limitado por los protocolos ligeros de comunicación más que por las implementaciones particulares de MPICH.

Todavía hay mucho trabajo por hacer. En la versión original de este estudio, estaba contemplado programar la tarjeta de red Myrinet, usando el trabajo de James Otto [14] para proveer primitivas de envío y recepción para uso de MPICH, y entonces comparar el desempeño de este protocolo "pequeño-ligero" con las otras implementaciones de MPICH [10]. La idea es simple: cada vez que un mensaje llega de la red, sería copiado a la memoria de la tarjeta de red usando el LANai, y, de ahí controlar la transferencia al espacio de la aplicación. Para el envío de mensajes, el LANai controlaría la transferencia del espacio de la aplicación, a la memoria de la tarjeta de red, y de ahí a la red. Aunque no es precisamente un protocolo de "cero-copia" (*zero-copy*), si omite al sistema operativo.

Sin embargo, la dificultad para programar el LANai (debido a incompatibilidades entre Red Hat Linux 6.0 y el trabajo de Otto) hizo imposible alcanzar el objetivo descrito.

Uno de los objetivos más importantes que se pretendía alcanzar con la programación de la tarjeta de red era el de poseer una clara idea experimental de todos los aspectos que un protocolo ligero de comunicación tiene que afrontar [7, 6, 2, 10]. Esto ayudaría a analizar más profundamente los protocolos existentes y ver sus ventajas y desventajas principalmente en términos de desempeño.

Además, es necesario realizar el mismo conjunto de mediciones que se hicieron para GM 1.0 pero ahora usando la versión más reciente, GM 1.1, ya que supuestamente es "mucho mejor" que la versión anterior [13]. Como se mencionó anteriormente, hay incompatibilidades de hardware entre las tarjetas de red Myrinet de 32 bits/33 MHz y esta nueva versión de GM.

Apéndice A

Glosario

A.1 Siglas y abreviaturas

ANL Argonne National Laboratory.

API *A*pplication *P*rogramming *I*nterface. Colección de llamadas a función exportadas por bibliotecas y/o servicios.

CRC Cyclic Redundancy Check. Número derivado, almacenado y transmitido con un bloque de datos para detectar corrupción en éstos. El receptor puede detectar algunos tipos de errores de transmisión recalculando el CRC y comparándolo con el valor original transmitido [15].

CSAG Concurrent Systems Architecture Group. La investigación realizada por este grupo se enfoca principalmente en aspectos de arquitectura de hardware y software en sistemas de computadoras paralelos y distribuidos (e.g. MPP's y *clusters*) [8].

DMA Direct Memory Access.

DSM Distributed Shared-Memory. Multiprocesadores de propósito general con memoria distribuida físicamente. Las memorias físicamente separadas pueden ser referidas como un espacio de direcciones compartido lógicamente; es decir, que una referencia a memoria puede ser realizada por cualquier procesador a cualquier localidad de memoria, asumiendo que tiene los permisos de acceso correctos. El término *memoria compartida* se refiere al hecho de que el espacio de direcciones es compartido: esto es, la misma dirección física en dos procesadores se refiere a la misma localidad en memoria [15]. Las máquinas DSM también son llamadas NUMA's, ya que el tiempo de acceso depende de la localidad de una palabra de datos en memoria.

[GM][Bb] [\<Giga\>\<Mega\>] [\<bytes\>\<bits\>][1].

[1] Siguiendo expresiones regulares en UNIX

LAN Local Area Network.

LANai LANai es un chip VLSI CMOS integrado en las tarjetas de red Myrinet encargado de controlar la transferencia de datos entre la computadora y la red.

MCP Myrinet Control Program. Programa que controla el procesador de LANai en una tarjeta de red Myrinet. También conocido como *firmware* [3].

MPI Message Passing Interface.

MPIF MPI Forum. Grupo con representantes de diversas organizaciones que define y mantiene el estándar MPI.

MPP Massively Parallel Processor. Esta red de interconexión puede conectar miles de computadoras y la distancia máxima es típicamente menor a 25 metros [15]. Generalmente las computadoras se encuentran en una fila de cabinas adyacentes.

MTU Maximum Transfer Unit. La longitud de marco más grande que puede ser enviada en un medio físico [6].

NIC Network Interface Controller. Tarjeta de red que permite el intercambio de información entre una computadora y una red. La tarjeta de red copia datos de la memoria de la computadora al medio de red (transmisión de datos), y de la red a la memoria (recepción de datos) [3].

NUMA Non-Uniform Memory Access. Ver DSM.

OS Operating System.

RISC Reduced Instruction Set Computer.

SAN System Area Network. Red de corto alcance, alto-desempeño y baja latencia en un sistema distribuido de computadoras.

WAN Wide Area Network.

A.2 Términos relacionados

Ancho de banda El ancho de banda asintótico pretende ser una caracterización de que tan rápida una transferencia de datos puede ocurrir de un emisor a un receptor. Para aislar la transferencia de datos de cualquier otro evento de procesamiento relacionado con la semántica de sincronización, la velocidad de transferencia es medida para cantidades de datos muy grandes (idealmente infinitas, por lo tanto, "asintótico"). Si D es el tiempo necesario para mandar S bytes de datos, medidos desde el instante en que el emisor inicia la operación "envío" y el instante

en que el receptor obtiene el mensaje completo, entonces el ancho de banda asintótico B es calculado como S/D cuando S es muy grande [3].

Ancho de banda de bisección Esta medida de desempeño es calculada dividiendo la interconexión en dos partes iguales, cada una con la mitad de las computadoras Después se suma el ancho de banda de los enlaces que cruzan esa línea divisoria imaginaria. Para interconexiones completamente conectadas el ancho de banda de bisección is $(n/2)^2$, donde n es el número de computadoras [15].

Cluster Un *cluster* es un sistema de procesamiento paralelo o distribuido, que consiste de una colección de ordenadores interconectados entre sí trabajando juntos como una sola fuente integrada de cómputo [3].

Conmutación *cut-through* El significado es el mismo que conmutación *wormhole* pero permite que la cola del mensaje continúe cuando la cabeza es bloqueada, comprimiendo el mensaje en un solo conmutador [15]. Claramente, el encaminamiento *cut-through* requiere de memoria temporal lo suficientemente grande como para contener el paquete más grande.

Conmutación de paquetes (*packet switching*) En este tipo de conmutación, cada paquete se almacena completamente en un conmutador de la red antes de que pueda ser transmitido a la siguiente etapa [3]. Este mecanismo de almacenamiento y transferencia necesita una cota superior para el tamaño del paquete (MTU) y espacio de almacenamiento temporal para uno o varios paquetes. En redes tradicionales LAN y WAN (Fast Ethernet y ATM) se usa este tipo de conmutación.

Conmutación *wormhole* Los datos son pasados inmediatamente a la siguiente etapa tan pronto como la dirección del encabezado es decodificada [3]. La longitud del mensaje puede ser definida como variable, pero mensajes grandes pueden bloquear varias etapas al mismo tiempo en su camino al destino. También, la corrección de errores es difícil, datos corruptos pueden ser pasados antes de que sean reconocidos erróneos.

Conmutador de barras cruzadas (*crossbar switch*) Interconexión — completamente conectada que permite a cualquier computadora comunicarse con cualquier otra en un paso a través de la interconexión. Usa n^2 conmutadores, donde n es el número de procesadores. Puede encaminar simultáneamente cualquier permutación del patrón de tráfico entre los procesadores [15].

Control de la congestión El control de la congestión se refiere a esquemas que reducen el tráfico cuando el tráfico colectivo de todas las computadoras es muy grande para que la red los pueda manejar [15].

El flujo de control, ayuda al control de la congestión, pero no es una solución universal.

Encaminamiento destino (*table-based routing*) En el encaminamiento destino, los conmutadores tienen una tabla de encaminamiento (*routing table*) la cual asocia a cada nodo destino su puerto de salida correspondiente. El conmutador determina el canal de salida de los mensajes que llegan a través de la tabla de encaminamiento. El tamaño de esta tabla es proporcional al número de nodos, lo que puede ser un factor limitante para configuraciones de *clusters* con cientos o miles de nodos [3].

Encaminamiento fuente (*source routing*) En el encaminamiento basado en la fuente, el mensaje especifica el camino al destino [15].

Escalabilidad (*scalability*) Se refiere a la capacidad de las redes de crecer casi lineal con el número de computadoras. La red debe tolerar el incremento en la carga y entregar casi el mismo ancho de banda y latencia para *clusters* pequeños (8-32 computadoras) y para grandes (cientos de computadoras) [3].

Flujo de control La idea es usar retro-alimentación para decirle al emisor cuando está permitido el envío del siguiente paquete para evitar que el receptor se sature y empieze a tirar paquetes [15]. Note que el flujo de control describe tan solo dos computadoras de la interconexión y no toda la red de interconexión entre todos los sistemas finales.

Full-duplex Se refiere a la transmisión simultánea e independiente de datos en ambas direcciones sobre un enlace de comunicaciones.

Gather Función de comunicación colectiva por medio de la cual un proceso recibe datos de todos los demás procesos dentro de su espacio de comunicación.

Latencia La definición más común de latencia es el tiempo necesario para enviar un mensaje de tamaño mínimo de un emisor a un receptor, desde el instante en que el emisor inicia una operación de envío hasta el instante en que el receptor es notificado del arrivo del mensaje [3]. Normalmente, "emisor" y "receptor" son procesos de nivel de aplicación. Una técnica común para el cálculo de la latencia es usando una micro-benchmark ping-pong. La latencia es calculada como la mitad del promedio del tiempo de viaje redondo.

Micro-*benchmark* ping–pong El proceso "ping" envía un mensaje al proceso "pong" y lo recibe de regreso de "pong." Ping también mide el tiempo que se necesitó para ejectuar el par envío/recepción; esto es, el tiempo de viaje-redondo. El proceso "pong" hace lo inverso. Las acciones se repiten un número de veces para colectar estadísticas,

posiblemente descartando las primeras mediciones para excluir efectos de "calentamiento" [3].

Omitir al sistema operativo (*OS bypass*) *Un protocolo típico de cero-copia* hace que la tarjeta de red genere una interrupción para el CPU cuando un mensaje llega de la red. El manejador de la interrupción controla la *transferencia del mensaje* al espacio de direcciones de la aplicación apropiada. La latencia de la interrupción, el tiempo de la iniciación de una interrupción hasta que el manejador de la interrupción se está ejecutando, es significativa. Para evitar este costo, algunas tarjetas de red modernas tienen procesadores que pueden ser programados para implementar parte de un protocolo de paso de mensajes. En un protocolo diseñado adecuadamente, es posible programar la tarjeta de red para controlar la transferencia de mensajes que llegan, sin necesidad de interrumpir al CPU. Dado que esta estrategia no involucra al sistema operativo en cada transferencia de mensaje, es llamada frecuentemente "omitir al sistema operativo" [10]. ST, VIA, FM, GM y Portals son ejemplos de protocolos que omiten al sistema operativo.

Protocolo de "cero-copia" En arquitecturas de sistemas tradicionales, los paquetes de la red llegan a la tarjeta de red, son pasados a través de una o más capas de protocolos en el sistema operativo, y eventualmente son copiados al espacio de direcciones de la aplicación. Tan pronto como el ancho de banda de la red comenzó a alcanzar la tasa de las copias de memoria, la reducción de copias de memoria se ha convertido una preocupación crítica. En un protocolo de paso de mensajes de cero-copia, *las copias del mensaje son eliminadas o traslapadas* para evitar la pérdida de ancho de banda [10].

Puerto Un puerto es un punto-final de comunicación, y sirve como la interfaz entre el software cliente y la red [11].

Scatter Función de comunicación colectiva por medio de la cual un proceso *distribuye datos entre* todos los demás procesos en su espacio de comunicación

Bibliografía

[1] ANL Mathematics and Computer Science. *The Message Passing Interface (MPI) standard.* <http://www-unix.mcs.anl.gov/mpi>, 1999.

[2] Ron Brightwell and Arthur B. Maccabe. Scalability limitations of via in supporting the computational plant runtime system. Technical report, Computational Sciences, Computer Sciences, and Mathematics Center, Sandia National Laboratories and Department of Computer Science, The University of New Mexico, September 1999.

[3] Rajkumar Buyya, editor. *High Performance Cluster Computing*, volume 1: Architectures and Systems. Prentice Hall PTR, 1999.

[4] Jeffrey S. Chase et al. *The Trapeze Project.* Duke University, <http://www.cs.duke.edu/ari/trapeze>, 1999.

[5] Andrew A. Chien. *Home Page.* Department of Computer Science and Engineering at the University of California at San Diego, <http://www-csag.ucsd.edu/individual/achien/achien.html>, 1999.

[6] Compaq Computer Corp., Intel Corp. and Microsoft Corp. *Virtual Interface Architecture Specification (version 1.0)*, December 1997.

[7] Concurrent Systems Architecture Group. *Fast Messages (FM).* University of California at San Diego, <http://www-csag.ucsd.edu/projects/comm/fm.html>, 1999. Last updated August 1999.

[8] Concurrent Systems Architecture Group. *High Performance Virtual Machines Project.* University of California at San Diego, <http://www-csag.ucsd.edu/projects/clusters.html>, 1999. Last updated January 8, 1998

[9] Deborah A. Israel. *COMMODITY CLUSTERS: HIGH PERFORMANCE COMPUTING ON DESKTOP SYSTEMS. ALLIANCE*, <http://access.ncsa.uiuc.edu/Features/HPVM/HPVM.html>, 1999. Last updated June 1, 1999.

[10] Arthur B. Maccabe et al. The portals 3.0 message passing interface revision 1.0. Technical report, Computational Sciences, Computer Sciences,

and Mathematics Center, Sandia National Laboratories and Department of Computer Science, The University of New Mexico, September 1999.

[11] Myricom, Inc. *The GM API.* <http://www.myri.com/GM/doc/-gm_toc.html>, 1999. Document generated on October 15, 1999.

[12] Myricom, Inc. *LANai 4.X Documentation.* <http://www.myri.com/-scs/documentation/mug/development/LANai4.X.txt>, 1999. Document generated on January 17, 1996.

[13] Myricom, Inc. *Myricom: High-Speed Computers and Communications.* <http://www.myri.com>, 1999.

[14] James S. Otto. *The Myrinet Tutorial.* Department of Computer Science, The University of New Mexico, <http://www.cs.unm.edu/-~jotto/myri/myri.html>, 1999.

[15] David A. Patterson and John L. Hennessy. *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, second edition, 1996.

[16] David A. Patterson and John L. Hennessy. *Computer Organization & Design.* Morgan Kaufmann Publishers, second edition, 1998.

[17] Red Hat, Inc. *redhat LINUX.* <http://www.redhat.com>, 1999.