DECK-GM: Uma Implementação do Ambiente DECK para Myrinet

Clarissa Cassales Marquezan
Instituto de Informática - UFRGS
clarissa@inf.ufrgs.br

Rafael Bohrer Ávila
Instituto de Informática - UFRGS
avila@inf.ufrgs.br

Philippe O. A. Navaux
Instituto de Informática - UFRGS
navaux@inf.ufrgs.br

Resumo

O uso da tecnologia Myrinet como um padrão em agregados torna necessário o desenvolvimento de bibliotecas de comunicação que explorem suas características. Esse artigo apresenta a modelagem e a implementação do ambiente DECK para Myrinet, utilizando a API do sistema GM. Serão apresentadas as características do DECK, do GM e será detalhada a implementação mostrando os mecanismos utilizados para que se pudesse alcançar o zero-copy.

1 Introdução

O aumento do poder de processamento dos nodos dos agregados, proveniente do emprego de tecnologias mais rápidas, faz com que sejam necessárias redes de interconexão que acompanhem essa melhora de desempenho. Atualmente as principais tecnologias de interconexão de agregados são: SCI [15], Gigabit [14] e Myrinet [5]. Para cada uma dessas tecnologias são necessárias camadas de software capazes de fazer com que a arquitetura e o hardware utilizados sejam abstraídos no processo de desenvolvimento das aplicações. Nesse sentido diversas bibliotecas de comunicação surgiram ao longo do tempo, como por exemplo: PVM [10], Fast Messages [13], MPI [16], Athapascan [11], etc.

O DECK, desenvolvido no Instituto de Informática da UFRGS, é uma biblioteca de comunicação paralela e distribuída, tendo suporte para *multithreading* e troca de mensagens. Sua estrutura modular permite que o tipo de rede de comunicação seja completamente abstraída. O objetivo desse trabalho é a apresentação da implementação do ambiente DECK para Myrinet, visto que essa tecnologia é amplamente utilizada em agregados, constituindo-se em um padrão *de facto* (cerca de 35% das máquinas que integram o TOP500 são inter conectadas com Myrinet, 178 máquinas) [21]. Para essa implementação foi utilizada como suporte nas interações com a interface Myrinet a API disponibilizada pelo sistema GM [12].

Esse artigo é estruturado da seguinte forma: na Seção 2 o ambiente DECK é descrito e suas estruturas de comunicação apresentadas; na Seção 3 o sistema GM é detalhado, nela as principais características que tiveram de ser observadas para o sucesso dessa implementação são descritas; na Seção 4 a implementação é discutida detalhadamente, são apresentadas as principais estruturas e mecanismos desenvolvidos; na Seção 5 é realizada uma análise do desempenho obtido; na Seção 6 alguns trabalhos relacionados são apresentados e finalizando na Seção 7 são apresentadas as conclusões obtidas a partir dessa implementação.

2 Ambiente DECK

DECK - *Distributed Execution and Communication Kernel*, é um ambiente que visa o suporte ao desenvolvimento de aplicações paralelas e distribuídas e é também um ambiente de execução para o modelo MultiCluster [3]. A primeira descrição desse ambiente pode ser analisada em [4]. Inicialmente o DECK foi desenvolvido para servir como um ambiente de suporte para o DPC++, uma extensão da linguagem C++ para programação distribuída [20]. Desde então mudanças em algumas abstrações foram inseridas ao longo de seu desenvolvimento. O DECK assim como MPI permite a troca de mensagens, entretanto ele permite também o uso de *multithreading* de forma integrada.

Uma característica importante desse ambiente é o comportamento adotado na troca de mensagens. A primitiva de recebimento será sempre bloqueante, enquanto a primitiva de envio dependendo do tamanho da mensagem pode ser bloqueante (mensagens pequenas) ou não bloqueantes (mensagens grandes).

A arquitetura do DECK é modular, composta pelo módulo do μDECK e pelo módulo de serviços, assim como

apresentado na Figura 1. O μDECK possui as abstrações essenciais para a troca de mensagens (*Mail Boxes*) e para *multithreading* (*Threads*). O módulo de serviços do DECK possui funcionalidades específicas, construídas a partir do μDECK. Essa arquitetura permite que as aplicações DECK sejam executadas sem a necessidade de mudanças devido a aspectos da rede utilizada. Atualmente o DECK possui suporte às seguintes tecnologias: SCI [7, 6, 8], Myrinet [1] utilizando a biblioteca BIP [18] e Ethernet [2].

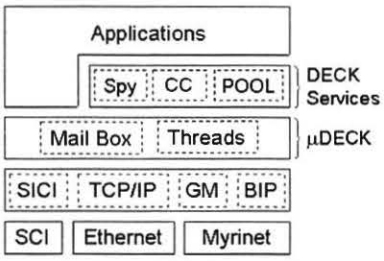

Figura 1. Arquitetura DECK.

As principais funções de comunicação do DECK estão relacionadas com as estruturas de *mail boxes* e de mensagens. Toda informação é empacotada em uma mensagem e enviada a uma *mail box*. Para se ter acesso a informação recebida é necessário que ela seja, primeiramente, retirada da *mail box* e armazenada em uma mensagem, no passo seguinte o usuário desempacota essas informações em suas estruturas de dados. A Tabela 1 apresenta as principais primitivas que permitem a troca de mensagens no DECK.

Tabela 1. Principais funções para troca de mensagens no DECK

Primitiva	Descrição
deck_mbox_create()	Cria uma *mail box*
deck_mbox_clone()	Recupera informações de *mail boxes* de outros processos/threads
deck_mbox_post()	Envia mensagem para *mail box*
deck_mbox_retrv()	Retira informações recebidas em uma *mail box*
deck_msg_pack()	Empacota mensagem
deck_msg_unpack()	Desempacota mensagem
deck_msg_create()	Cria a abstração mensagem

3 Descrição do Sistema GM

GM *(Glenn' Message)* é um sistema eficiente para agregados baseados em Myrinet, possuindo um *driver*, uma interface de programação para interação com o processador LANai, bibliotecas de inclusão e uma API. Suas principais características são: baixa latência, alta banda passante, entrega ordenada e confiável de pacotes e modelo de programação baseado em eventos, *tokens* e DMA. A transmissão de mensagens pode ser alcançada através do uso de métodos comuns, como por exemplo, *send/receive* ou então utilizando RDMA. As principais características do modelo de comunicação do GM serão apresentadas a seguir.

3.1 Conexões Entre os Nodos

O sistema de comunicação no GM é baseado em *end-points* - portas. O GM possui 8 portas, mas somente 5 estão disponíveis para os usuários. Como mencionado acima, o GM garante uma entrega confiável e ordenada para pacotes cujas portas de envio e recebimento coincidam. Entretanto não é necessário que as aplicações clientes estabeleçam conexões reais entre essas portas, basta que elas estejam abertas e o GM garante essas conexões. Esse processo é chamado de *connectionless reliability*.

3.2 Alocação de Tokens

A comunicação no GM é regulada por *tokens*, isto é, para enviar uma mensagem é necessário que o programa cliente possua *tokens* de envio e para receber é necessário que se tenha *tokens* de recebimento. Os programas clientes, no início de seu processamento, possuem *tokens* implícitos de envio e recebimento. Eles são passados para as primitivas de comunicação do GM quando elas são chamadas, e dependendo do tipo dessa primitiva o *token* é devolvido ao cliente (depois do *gm_*send()*, por exemplo) ou o programa é responsável por adquirir novamente esse *token* (no caso de querer reutilizar um buffer de recebimento).

3.3 Prioridades do GM

Dois níveis de prioridade são disponíveis no GM: prioridade baixa e alta. O GM provê dois canais de comunicação, um para mensagens de prioridade baixa e outro para mensagens de prioridade alta. É importante ressaltar, que mensagens com prioridades diferentes nunca se bloqueiam. Essas prioridades são associadas aos *tokens* e buffers do GM. Dessa forma, para enviar uma mensagem com baixa prioridade o programa cliente deve possuir um *token* de baixa prioridade, a mesma correspondência é necessária no envio de mensagens de alta prioridade.

3.4 Fornecendo Buffers de Comunicação

Tanto no envio quanto no recebimento de mensagens no modelo GM um buffer deve ser alocado com DMA.

Essas áreas de memória são associadas a uma porta de comunicação. Sendo assim, o programa cliente pode utilizar buffers com as mesmas características mas associados a diferentes portas.

O requisito para a criação de um buffer de envio é a sua alocação em uma porta de comunicação e a reserva de um *token* de envio com a prioridade da mensagem. Entretanto, prover um buffer de recebimento requer alguns detalhes a mais. Os procedimentos de reserva de uma região de memória DMA, reserva de *token* com a mesma prioridade da mensagem também devem ser observados na criação de um buffer de recebimento. Mas a esse *token* também deve-se associar o *size* da mensagem a ser recebida. Isto é, se o programa cliente deve receber uma mensagem com prioridade alta e tamanho de 10 bytes isso significa que um buffer de recebimento deve possuir prioridade alta e um *size* igual a 5 (tamanho da mensagem é igual a $2^{size} - 8$).

No sistema GM, para o recebimento de uma mensagem, é estritamente necessário que o buffer de recebimento possua as mesmas características da mensagem que se deseja receber, entendendo-se por características a mesma prioridade e o mesmo *size*. Caso essa condição não seja atendida a mensagem não poderá ser enviada e um erro será gerado.

3.5 Eventos no GM

Como descrito anteriormente, o modelo de programação do GM é baseado em eventos. Na Figura 2 é possível perceber os principais procedimentos na captura dos eventos de envio e de recebimento. Essa figura apresenta a comunicação entre 2 nodos, no Nodo#0 envia uma mensagem e o Nodo#1 a recebe. Nesse exemplo, não foi utilizado o modelo de comunicação baseado em RDMA.

Na função *gm_*_receive_*()* são retornados todos os eventos de envio e recebimento gerados na porta passada como parâmetro para essa função. O tipo *gm_recv_event_t* contém as informações referentes ao evento recebido, sendo que uma dessas informações descreve o tipo de evento. Caso um evento não seja reconhecido ele é passado para a função *gm_unknown()* que se responsabilizará em tratá-lo.

Para enviar uma mensagem é necessário que se use a primitiva *gm_*_send_*_with_callback()*. Quando o procedimento de envio termina a função de *CALLBACK*, passada como parâmetro, possuirá o estado do processo de envio. Essa função é chamada pela primitiva *gm_unknown()*, sendo assim o procedimento *gm_*_receive_*()* deve ser chamado para tratar o estado de um envio de dados.

Existem diferentes tipos de recebimento, essas diferenças estão relacionadas com:

- tamanho da mensagem: as mensagens no GM são divididas em pequenas, que não necessitam de alocação de memória DMA (menores do que 120 bytes), e mensagens grandes, que necessitam a alocação de memória DMA (maiores que 121 bytes). O evento do tipo *FAST* indica o recebimento de mensagens pequenas;
- prioridade da mensagem: eventos do tipo *HIGH* indicam mensagens de prioridade alta;
- portas: o tipo de evento *PEER* indica que a porta de origem e a porta de destino são iguais.

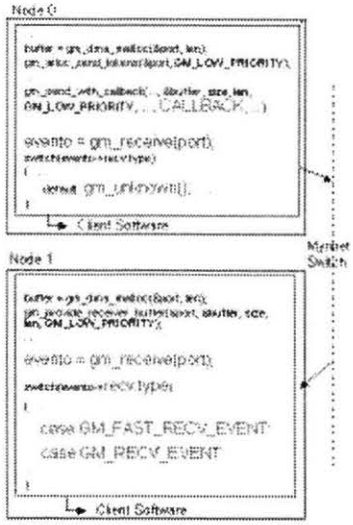

Figura 2. Captura de Eventos em uma Comunicação no GM.

Quando uma mensagem é recebida o *token* associado ao buffer de recebimento é retornado para a fila de *tokens* de recebimento do GM. Por este motivo outro *token* deve ser providenciado com a mesma prioridade e o mesmo *size* se uma nova mensagem com essas características tiver de ser recebida.

4 Detalhamento da Implementação

Essa implementação apresenta uma solução *zero-copy* para a comunicação em DECK utilizando o sistema GM, desenvolvido para Myrinet. O *zero-copy* é um mecanismo no qual não são necessárias cópias intermediárias de memória entre a interface de comunicação e o espaço de recepção do usuário. Esse mecanismo permite que se possa alcançar melhores resultados na comunicação, diminuindo a latência final das mensagens (característica importante no desenvolvimento de uma biblioteca de comunicação).

Essa versão não utiliza o modelo de comunicação baseado em RDMA. Foram utilizadas as primitivas convencionais (*send/receive*) do GM. Nessa seção, serão explicadas

as considerações feitas para o desenvolvimento, assim como serão analisadas as características do DECK-GM.

4.1 Paradigmas Diferentes

Como descrito em seções anteriores, a comunicação no GM é baseada em eventos, enquanto no DECK é baseada no paradigma procedural. Um desafio no desenvolvimento do ambiente foi a adaptação desses dois conceitos diferentes mantendo a semântica do DECK. Na chamada de cada primitiva de comunicação do GM, um evento é gerado na porta que se está trabalhando. Vários fluxos de execução na mesma porta possibilitam que um deles capture algum evento que não pertence ao seu escopo, o que implica em um redirecionamento desse evento.

Para evitar que as primitivas do DECK tivessem que redirecionar eventos, uma *thread* que centraliza todos os eventos de um nodo é criada na inicialização do ambiente - ela foi denominada *ReceiverThread*. A implementação que está sendo descrita utiliza apenas uma porta por nodo, todas as comunicações do nodo são realizadas através dessa porta. A *ReceiverThread* monitora todos os eventos que chegam na porta de comunicação do nodo e ela é capaz de determinar a origem dos eventos de envio e o destino dos eventos de recebimento, independente do número de *threads* do nodo.

4.2 Alcançando Zero-Copy

Como apresentado na descrição do ambiente DECK, originalmente, os buffers de recebimento eram associados às estruturas mensagens do DECK - *deck_msg_t*. Para se alcançar *zero-copy* nessa implementação, os buffers de recebimento foram deslocados para a estrutura *mail box* do DECK - *deck_mbox_t*. Mesmo com a mudança do local dos buffers de recebimento a semântica do DECK não foi alterada.

Na versão DECK-TCP a estrutura mensagem possui somente um buffer. Ele é utilizado para empacotar os dados a serem enviados e no caso do recebimento de dados ele é o buffer passado para o socket TCP que irá receber a mensagem. Entretanto, o fato de mover o buffer de recebimento para dentro da estrutura *mail box* leva a uma outra mudança estrutural. O único buffer, dentro da estrutura mensagem na versão DECK-TCP, teve de ser dividido em dois na versão DECK-GM. Essa modificação foi necessária devido ao fato de que se um único buffer fosse alocado, um *token* de recebimento também teria de ser alocado. Sob a ótica da operação de envio não existe problema algum, no entanto, considerando a operação de recebimento de mensagens não existe garantia de que os dados contidos nesse buffer pertençam realmente a essa estrutura mensagem. Quando um buffer de recebimento é fornecido ao sistema GM ele pode receber qualquer mensagem cujo *size* e a prioridade sejam iguais às estipuladas para esse buffer.

Os principais campos da estrutura *deck_msg_t* no DECK-GM são: um buffer de envio (*send_buf*), alocado com DMA no momento da criação da estrutura, e um ponteiro para um buffer de recebimento (*recv_buf_ptr*). O *send_buf* permite que o GM acesse essa área de memória e transmita os dados sem a inserção de cópias de memória. Da mesma forma, os principais campos da estrutura *deck_mbox_t* são: uma lista de buffers de recebimento (*recv_buf_t*), a qual é formada por vários buffers de diferentes *sizes*, e uma fila de mensagens recebidas (*recv_msgs*). Quando uma *mail box* é criada os buffers dentro do *recv_buf_t* são alocados com DMA e *tokens* de recebimento são associados para cada um desses buffers. Isso permite que o sistema GM receba mensagens com as mesmas características desses buffers. No momento que a *ReceiverThread* capta um evento de recebimento ela irá inserir um nodo na fila *recv_msgs*, esse nodo vai conter um ponteiro para o buffer onde a mensagem foi recebida. Esse processo evita que seja feita a cópia da mensagem.

No momento que a primitiva *deck_mbox_retrv()* é executada o ponteiro *recv_buf_ptr* da estrutura mensagem passada como parâmetro, irá apontar para o mesmo buffer do primeiro nodo da fila *recv_msgs* da estrutura *mail box*. Essa operação é feita sem cópias de memória. É importante ressaltar que quando uma mensagem é recebida, pela *ReceiverThread*, o *token* de recebimento associado a ela deixa de existir, isto é, para o GM o buffer que contém a mensagem não pode ser usado para o recebimento de mensagens. Esses buffers de recebimento voltarão a estar disponíveis ao sistema GM quando a primitiva *deck_mbox_retrv()* for chamada tendo como parâmetro uma estrutura mensagem já utilizada, ou quando a primitiva *deck_msg_clear()* for executada. Esse processo garante a reutilização dos buffers de recebimento criados dentro da estrutura *mail box*, e permite também a redução da quantidade de memória DMA usada pela aplicação.

4.3 Protocolo de Comunicação

Os protocolos de comunicação no DECK-GM estão bastante ligados ao tamanho das mensagens. Como o GM requer alocação de memória DMA para o envio e o recebimento de mensagens e o controlador de DMA limita a quantidade de memória que pode ser acessada diretamente, nem todos os buffers de recebimento com seus respectivos *tokens* podem ser disponibilizados na inicialização da *mail box*. Sem essa possibilidade dois protocolos de comunicação foram desenvolvidos: um para mensagens padrão e outro para mensagens grandes.

4.3.1 Protocolo Padrão

As mensagens padrão possuem *sizes* entre 3 a 17, isto é, mensagens de 0 a 131064 bytes. Para o recebimento des-

sas mensagens padrão os buffers de recebimento e *tokens* são alocados na inicialização da *mail box*. Mas, como visto anteriormente, propiciar apenas um buffer de recebimento não é o suficiente. Por este motivo foi definida uma constante para disponibilizar mais buffers de recebimento para cada *size* e prevenir situações de *deadlock*. Atualmente, são alocados 10 buffers de recebimento para cada *size*, em cada *mail box* criada. Os tamanhos apresentados aqui são baseados em experimentos.

O protocolo padrão de comunicação desenvolvido utiliza mensagens com baixa prioridade, e seu cabeçalho é apresentado na Figura 3. Os campos que formam o cabeçalho do pacote são: a identificação do nodo destino, a identificação da thread que possui a *mail box* destino, o índice da *mail box* que receberá a mensagem e o tamanho da mensagem a ser recebida. No momento que uma mensagem chega na *ReceiverThread* o cabeçalho da mensagem é verificado no intuito de prevenir erros e determinar a qual *mail box* pertence a mensagem recebida.

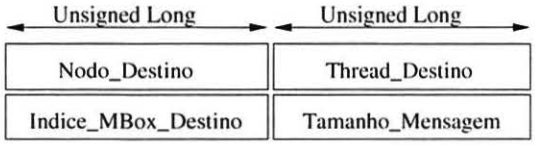

Figura 3. Cabeçalho do Protocolo Padrão de Comunicação.

4.3.2 Protocolo Para Mensagem Grandes

Esse protocolo foi desenvolvido para permitir que mensagens maiores de 131064 bytes fossem transmitidas no DECK-GM. Como não existem buffers e *tokens* de recebimento disponibilizados na inicialização da *mail box* para esse tipo de mensagem, é preciso uma negociação entre o processo que deseja enviar e o que irá receber a mensagem para que sejam estabelecidas as condições para a troca de mensagens através do GM. Todas as mensagens nesse protocolo utilizam a prioridade alta, sendo que foi assim desenvolvido para que houvesse uma diferença entre as mensagens dos dois protocolos na *ReceiverThread*.

Um *handshake* de duas mensagens foi implementado. A Figura 4 mostra a seqüência de mensagens. A mensagem CTR1_MSG é enviada no intuito de sinalizar que um buffer e um *token* de recebimento de mensagem grande devem ser disponibilizados. Em seguida a CTR2_MSG é enviada sinalizando que as condições necessárias já foram providenciadas. Logo após essas mensagens de controle a mensagem grande é enviada.

Cada mensagem de controle possui um cabeçalho diferente. A Figura 5 apresenta os campos de cada mensagem. A CTR1_MSG possui os seguintes campos: a identificação do tipo da mensagem, o índice da *mail box* que deverá receber a mensagem grande, o índice da *mail box* para onde deve ser enviada a resposta da CTR1_MSG, a identificação do nodo para onde a resposta deve ser enviada e o tamanho da mensagem grande que deve ser recebida. A mensagem CTR2_MSG possui bem menos campos que a primeira, sendo que um campo indica qual o tipo da mensagem de controle e o outro o índice da *mail box* que enviará a mensagem grande.

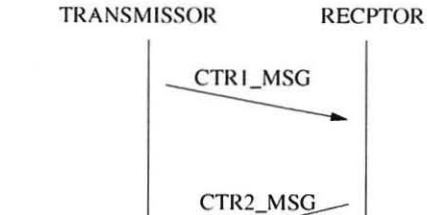

Figura 4. Handshake do Protocolo de Mensagem Grandes.

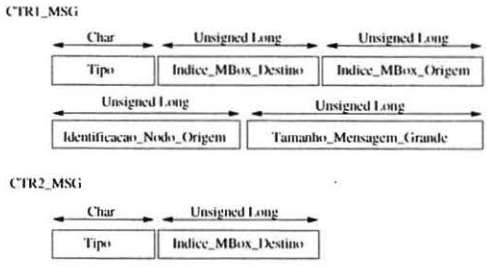

Figura 5. Cabeçalho das Mensagens de Controle do Protocolo de Mensagens Grandes.

5 Resultados e Análises

A avaliação de desempenho do DECK-GM foi realizada baseando-se no algoritmo tradicional de ping-pong, sendo que os tempos apresentados são o resultado da média de

1000 execuções, possuindo a metade do tempo de um ciclo (ida e volta). A primitiva de aquisição do tempo utilizada foi a **gettimeofday** da linguagem C. Esse mesmo algoritmo foi implementado em MPI, para que se pudesse fazer a comparação.

Esses algoritmos foram executados em 2 máquinas duais Pentium III 1GHz pertencentes ao agregado denominado Corisco, no Instituto de Informática da UFRGS. A versão do GM utilizada foi a 1.6.4 e a versão MPICH-GM foi a 1.2.5..10.

A primeira análise feita sobre os resultados do DECK-GM foi com relação a questão do limiar entre mensagens grandes, necessário para o protocolo dessas mensagens. Para definição desse limiar a principal preocupação foi minimizar o uso de buffers de recepção para os possíveis *sizes* das mensagens. Isso significa que o problema está em encontrar a configuração mais adequada para possuir buffers de recebimento alocados para os tamanhos de mensagens mais freqüentes nas aplicações. Evitando assim a necessidade do uso do protocolo de comunicação para mensagens grandes, que aumenta a latência em cerca de 40 μs, referentes a transmissão das 2 mensagens de controle (CTR1_MSG e CTR2_MSG).

A Tabela 2 mostra alguns possíveis valores para limiares. É importante lembrar que para cada *size* existem DECK_MBOX_GM_MAX_BUF_PER_SIZE buffers alocados (atualmente essa constante é igual a 10). Essa tabela mostra o valor total de memória DMA alocada por *mail box* de acordo com o limiar escolhido.

Tabela 2. Tamanho das mail boxes de acordo com o limiar do protocolo de comunicação para mensagens grandes.

Size Limite	Tamanho (KB)
10	19,29
11	39,21
12	79,14
13	159,06
14	318,98
15	638,90
16	178,82
17	2558,75
18	5118,67
19	10238,59
20	112625,97

Baseado nos valores apresentados na Tabela 2 optou-se pelo *size* 17 como sendo o limiar, ou seja, mensagens com tamanho menor ou equivalente a ele serão transmitidas através do protocolo padrão, a partir dele o protocolo de mensagens grandes entra em vigor. Com o estabelecimento do limite entre a ação dos protocolos foi possível iniciar os testes entre a implementação descrita e a biblioteca MPICH.

Os primeiros testes realizados comparando as implementações foram com mensagens pequenas, de acordo com a definição do GM. Ou seja, mensagens que não precisam de alocação de DMA para serem recebidas pelo sistema GM. O gráfico na Figura 6 apresenta os resultados obtidos. A implementação MPICH possui latência em torno de 9 μs enquanto a latência do DECK-GM se mantém na casa de 23 μs para mensagens até 64 bytes. É importante lembrar que na implementação do DECK foi preciso a alocação de DMA para essas mensagens para que se pudesse atingir o *zero-copy*.

Entretanto na análise da latência das mensagens médias (mensagens que pelo sistema GM precisam de alocação de DMA e pela implementação do DECK-GM são alocados buffers de recebimento na inicialização de uma *mail box*), percebe-se uma fronteira a partir da qual a implementação DECK-GM passa a apresentar melhor resultado em comparação com a implementação MPICH-GM. Essa melhora no desempenho do DECK-GM ocorre a partir de mensagens maiores que aproximadamente 6KB, também apresentado no gráfico da Figura 6.

Essa vantagem do ambiente DECK se mantém mesmo nos testes com as mensagens grandes, onde na implementação DECK-GM existe o acréscimo de latência gerado pelas mensagens de controle, que corresponde em média a 40 μs. O gráfico da Figura 7.

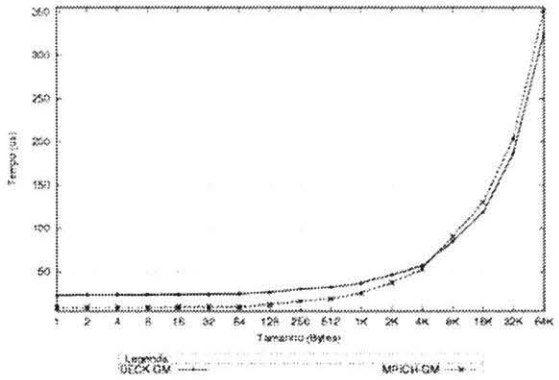

Figura 6. Latência do protocolo padrão.

O DECK-GM também apresenta uma banda passante melhor que o MPICH-GM. O gráfico da Figura 8 apresenta os resultados. Com o DECK a maior banda passante foi de 228,50 MB/s enquanto o melhor resultado obtido com o MPICH foi 217,37 MB/s.

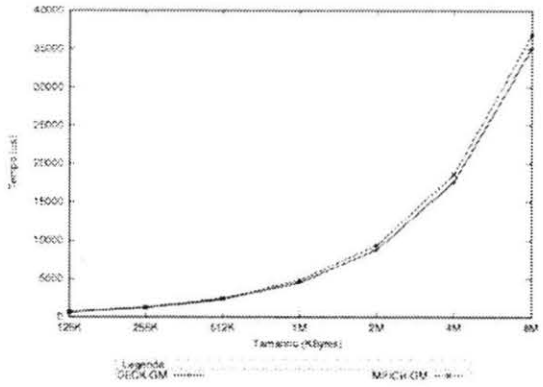

Figura 7. Latência do protocolo para mensagens grandes.

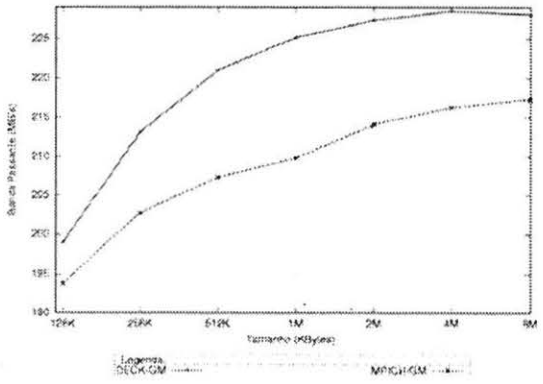

Figura 8. Banda passante alcançada com o protocolo para mensagens grandes

6 Trabalhos Relacionados

Com o objetivo de evitar que os usuários tenham que lidar com os detalhes da troca de mensagens em cada tecnologia de rede, bibliotecas são desenvolvidas para tratar dessas características e permitir que os usuários se concentrem no desenvolvimento de suas aplicações em alto nível.

Sendo assim, várias bibliotecas foram implementadas, tais como Fast Messages[13], MPICH-GM [17] e o VIA-GM [22], sendo as duas últimas implementadas pela Myricom, empresa detentora da tecnologia Myrinet. Existem poucas informações detalhadas a cerca do funcionamento dessas últimas duas bibliotecas.

A implementação da biblioteca FM[13] não foi baseada sobre o sistema GM. Nela foram implementados o controle sobre a máquina e sobre a interface Myrinet (programa de controle para o processador LANai). Nessa versão todas as mensagens são segmentadas em pacotes de 256 KB. O objetivo é utilizar melhor os recursos da Myrinet, tendo em vista a relação latência versus banda passante.

A implementação VIA-GM possui um *daemon* que roda nos nodos do agregado baseado em Myrinet. Através desse processo, denominado *connection manager*, são controladas as conexões entre a interface Myrinet e as aplicações VIA-GM. A implementação permite a utilização de várias portas para comunicação.

A implementação DECK-GM difere da implementação FM por utilizar o sistema GM como meio de comunicação com a interface Myrinet e por não segmentar as mensagens. Já com relação à implementação VIA-GM ela difere no sentido de que o gerenciamento da porta de comunicação é feito por uma *thread* criada no momento da execução de uma aplicação DECK e não por um processo externo.

7 Conclusão

Neste trabalho foi apresentada a implementação do ambiente DECK para tecnologia Myrinet utilizando a biblioteca GM. Inicialmente o ambiente DECK foi apresentado, em seguida as características do GM foram discutidas, e então a modelagem da comunicação do DECK-GM foi descrita.

Foram desenvolvidos dois protocolos de comunicação, um protocolo padrão e outro específico para mensagens grandes. Através dos testes realizados foi possível constatar que o protocolo de comunicação padrão mostrou que precisa ser aperfeiçoado, pois apresenta latência alta para mensagens pequenas, principalmente para mensagens até 1KB. Em compensação, esse mesmo protocolo se mostra eficiente para mensagens a partir de 6KB, se comparado com a implementação MPICH.

Com relação ao protocolo de comunicação para mensagens grandes pode-se concluir que ele é bastante eficiente, alcançando banda passante em torno de 228MB/s. A existência de mensagens de controles, necessárias nesse protocolo, não teve um impacto negativo na latência final.

Sendo assim, através da análise desses resultados conclui-se que o DECK-GM é uma biblioteca de comunicação paralela e distribuída que atinge seu melhor desempenho na transmissão de mensagens acima de 8KB, se comparado com a biblioteca largamente utilizada, o MPICH. Dessa forma aplicações de grande porte, como por exemplo o modelo Hidra [9, 19] desenvolvido no Instituto de Informática, podem se beneficiar ao utilizar o ambiente DECK para suas trocas de mensagens. E mesmo no caso de aplicações que possuem troca de mensagens pequenas, a vantagem de se utilizar o DECK possuindo um desempenho um pouco menor que o MPICH é justificada pela possibilidade de utilização de *multithreading*.

Referências

[1] M. Barreto, R. Ávila, R. Cassali, A. Carissimi, and P. Navaux. Implementation of the DECK environment with BIP. In *Proc. of the 1st Myrinet User Group Conference*, pages 82–88, Lyon, France, 2000. Lyon, INRIA Rocquencourt.

[2] M. Barreto, R. Ávila, F. de Oliveira, R. Cassali, and P. Navaux. DECK: an enviroment for parallel programming on clusters of multiprocessors. In *Proc. of the 12th Symposium on Computer Architecture and High-Performance Computing*, pages 321–329, São Pedro, SP, 2000. São Carlos, UFSCAR.

[3] M. Barreto, R. Ávila, and P. Navaux. The MultiCluster model to the integrated use of multiple workstation clusters. In J. Rolim et al., editors, *Proc. of the 3rd Workshop on Personal Computer based Networks of Workstations*, volume 1800 of *Lecture Notes in Computer Science*, pages 71–80, Cancun, 2000. Berlin, Springer-Verlag.

[4] M. E. Barreto, P. O. A. Navaux, and M. P. Rivière. DECK: a new model for a distributed executive kernel integrating communication and multithreading for support of distributed object oriented application with fault tolerance support. In *Trabajos Seleccionados del 4o. Congreso Argentino de Ciencias de la Computación*, volume 2, pages 623–637, Neuquén, Argentina, 1998. Neuquén, Universidad Nacional del Comahue.

[5] N. Boden et al. Myrinet: a gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.

[6] F. A. D. de Oliveira. Uma biblioteca para programação paralela por troca de mensagens de clusters baseados na tecnologia SCI. Master's thesis, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.

[7] F. A. D. de Oliveira, R. B. Ávila, M. E. Barreto, and P. O. A. Navaux. Low-latency and zero-copy message passing protocols for SCI-based clusters. In A. C. M. A. Melo, M. A. R. Dantas, and J. Panetta, editors, *Proc. of the 13th Symposium on Computer Architecture and High-Performance Computing*, pages 148–155, Pirenópolis, GO, 2001. Brasília, Departamento de Ciência da Computação da UNB.

[8] F. A. D. de Oliveira, R. B. Ávila, M. E. Barreto, P. O. A. Navaux, and C. De Rose. DECK-SCI: High-performance communication and multithreading for SCI clusters. In D. S. Katz, T. Sterling, M. Baker, L. Bergman, M. Paprzycki, and R. Buyya, editors, *Proc. of the 3rd IEEE International Conference on Cluster Computing*, pages 372–379, Newport Beach, CA, 2001. Los Alamitos, CA, IEEE Computer Society.

[9] R. V. Dorneles. *Particionamento de Domínio e balanceamento de Carga*. PhD thesis, Instituto de Informática - UFRGS, Porto Alegre, RS, 2002.

[10] A. Geist et al. *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, 1994.

[11] I. Ginzburg. *Athapascan0b: Intégration Efficace et Portable de Multiprogrammation Légère et de Communication*. PhD thesis, Institut National Polytechnique de Grenoble (INPG), Grenoble, 1997.

[12] GM. Available at http://www.myri.com/scs/, July 2003.

[13] G. Iannello, M. Lauria, and S. Mercolino. Cross-platform analysis of fast messages for myrinet. *Lecture Notes in Computer Science*, 1362:217–??, 1998.

[14] IEEE. Gigabit ethernet. IEEE P802.3z, 1997.

[15] INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. IEEE standard for scalable coherent interface (SCI). IEEE 1596-1992, 1992.

[16] MPI Forum. The MPI message passing interface standard. Technical report, University of Tennessee, Knoxville, Apr. 1994.

[17] MPICH-GM. Available at http://www.myri.com/scs/, July 2003.

[18] L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance networking on Myrinet. In J. Rolim, editor, *Parallel and Distributed Processing — Workshop on Personal Computer Based Networks of Workstations*, volume 1388 of *Lecture Notes in Computer Science*, pages 472–485, Berlin, Springer-Verlag, 1998.

[19] R. L. Rizzi. *Modelo Computacional Paralelo para a Hidrodinâmica e para o Transporte de Massa Bidimenssional e Tridimenssional*. PhD thesis, Instituto de Informática - UFRGS, Porto Alegre, RS, 2002.

[20] A. Silveira, R. Ávila, M. Barreto, and P. Navaux. DPC++: Object-oriented programming applied to cluster computing. In H. R. Arabnia, editor, *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 5, pages 2515–2521, Las Vegas, 2000. Las Vegas, CSREA Press.

[21] TOP500. Available at http://www.top500.org, 2003.

[22] VIA-GM. Available at http://www.myri.com/scs/, July 2003.