Memória Compartilhada Distribuída para Redes UDP/IP: Implementação e Avaliação

Rafael M. da Silva, Lauro Whately, Marcelo Lobosco, Claudio L. Amorim
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ
{rmendes, whately, lobosco, amorim} @cos.ufrj.br

Resumo

Neste trabalho investigamos o uso dos protocolos UDP/IP como suporte à comunicação no sistema software DSM HLRC (Home-Based Lazy Release Consistency), em substituição ao protocolo VIA empregado na versão original do sistema. Divergindo da expectativa inicial, a versão UDP conseguiu desempenho superior em 4 dos 5 benchmarks empregados. A avaliação de desempenho aponta que decisões de implementação levam a uma degradação do desempenho de HLRC/VIA.

1. Introdução

O uso de cluster de computadores pessoais para a computação de alto desempenho estabeleceu-se na última década como um alternativa eficaz. Clusters construídos com hardware e software facilmente encontrados no mercado permitem o processamento de alto desempenho com custo bem mais baixo e acessível que o oferecido pelos supercomputadores vetoriais e massivamente paralelos [3, 10].

A rede usada para a comunicação é, depois dos computadores conectados por ela, o componente mais importante para alcançar bom desempenho a baixo custo. Os *clusters* freqüentemente (mas nem sempre) usam redes Ethernet 100/1000 Mbps e protocolos TCP/IP desenvolvidos e usados em redes locais. Menor latência e maior largura de banda na comunicação entre os processadores podem ser obtidos com redes Myrinet [8], SCI [6], Infiniband [7], entre outras. Essas redes apresentam a desvantagem de necessitarem de protocolos de comunicação particulares para explorar todo o desempenho oferecido e conseqüentemente tornam necessária a modificação das aplicações desenvolvidas para TCP/IP.

Prover um espaço de endereçamento único e compartilhado para as aplicações em *clusters* é um tópico de grande importância, visto que a presença de memória compartilhada pode diminuir consideravelmente a complexidade do desenvolvimento de aplicações paralelas. Diversas técnicas foram desenvolvidas com a finalidade de prover esse espaço compartilhado de endereçamento. Nesse trabalho, vamos nos concentrar em memória compartilhada distribuída baseada em páginas, também conhecida como software DSM [12]. Software DSM implementa memória compartilhada utilizando um ambiente de troca de mensagens, tornando-se assim bastante dependente do desempenho da rede de comunicação [19, 16].

Neste trabalho, apresentamos a implementação de um software DSM, o HLRC (*Home-based Lazy Release Consistency*) [18], para os protocolos de comunicação UDP/IP. HLRC inicialmente foi desenvolvido utilizando o protocolo VIA [11]. Pelo fato do VIA não utilizar o sistema operacional, evitar cópias e acessar diretamente a memória remota, ele oferece melhor desempenho para a comunicação entre processadores, porém, ainda não é amplamente utilizado como os protocolos UDP/IP [1]. De fato, a comunicação utilizando os protocolos UDP/IP possui menor desempenho que VIA, no entanto UDP/IP é a melhor (ou única) alternativa em muitos casos.

As principais contribuições desse trabalho são: a apresentação de uma implementação de HLRC utilizando os protocolos UDP/IP e a análise comparativa de desempenho das implementações VIA e UDP em um cluster de 8 processadores conectados por uma rede fast-ethernet.

O restante desse trabalho é organizado da seguinte forma: a Seção 2 descreve suscintamente o protocolo VIA e o sistema HLRC; a Seção 3 descreve as modificaões feitas no HLRC com o intuito de utilizar UDP/IP; na Seção 4 é feita a avaliação do sistema; na Seção 5 são apresentados os trabalhos relacionados e a Seção 6 é reservada às conclusões.

[1] Apesar de VIA representar a tendência para os futuros protocolos de comunicação em *clusters*, uma nova geração de arquitetura de comunicação e entrada/saída conhecida como Infiniband a substituiu.

2. Conhecimentos Básicos

Nesta seção apresentamos alguns conhecimentos básicos necessários para o entendimento deste trabalho. Descreveremos brevemente o conceito de DSM, os protocolos VIA e UDP/IP e o sistema HLRC.

2.1 Software DSM

Software DSM [15] é um sistema que provê para a camada de aplicação a abstração de uma memória compartilhada distribuída, que é implementada em um ambiente baseado em troca de mensagens. A idéia básica é utilizar o mecanismo de proteção de páginas fornecido pelo sistema de memória virtual para implementar um protocolo de consistência de memória. No entanto, esta abordagem apresenta dois problemas principais: o alto custo gerado pelo tratamento de falhas de páginas no sistema operacional, e o falso compartilhamento, que ocorre em função do grande tamanho da unidade de coerência, gerando assim um tráfego desnecessário de mensagens. O falso compartilhamento é minimizado com a adoção de protocolos de múltiplos escritores.

Ainda com o intuito de diminuir o tráfego desnecessário de mensagens foram desenvolvidos protocolos de consistência relaxada [13, 4], onde a consistência é garantida apenas em pontos de sincronização.

2.2 HLRC

HLRC é um sistema Software DSM baseado em residência de páginas que utiliza um modelo de consistência relaxada para manter a coerência da memória compartilhada. Neste modelo, a consistência de memória é garantida em pontos de sincronização utilizando-se de *locks* e *barreiras*. Estes pontos de sincronização são delimitados por *acquires* e *releases*, que correspondem a aquisição de *locks* e liberação dos mesmos, assim como a chegada a barreiras. O protocolo de consistência é baseado em invalidações, que são propagadas através de *write-notices* no instante em que o nó faz um *acquire*. A barreira é uma operação de sincronização global, onde os nós enviam todas as suas invalidações uns aos outros. No caso do *lock*, as invalidações são enviadas do nó que possuía o *lock* para o que fez o *acquire*.

HLRC implementa um protocolo baseado em residência onde cada página possui um nó que atua como gerente desta página. Quando um processo precisa acessar uma página, ela é buscada do seu nó residência, que é o responsável por manter sempre a cópia mais atualizada da mesma. Antes de ser feita qualquer alteração na página, um nó não residência deve fazer uma cópia dela, chamada *twin*. Alterações feitas na página são computadas e enviadas para a residência utilizando-se *diffs*, que são gerados comparando-se a página modificada com a sua *twin*. Ao chegar à residência, os *diffs* são aplicados à respectiva página.

2.3 Protocolos de comunicação

2.3.1 UDP/IP

O Protocolo UDP/IP é extensivamente utilizado em redes locais e na Internet. Trata-se, na verdade, de uma pilha de protocolos, gerenciada pelo sistema operacional. As transferências de dados na rede são efetuadas através de chamadas ao sistema operacional. O UDP implementa um canal de comunicação não confiável e não orientado a conexões.

2.3.2 VIA

Um dos principais problemas com relação ao desempenho de protocolos como o TCP e o UDP é o *overhead* de software necessário na transmissão de dados. Este *overhead* é composto por diversos fatores, como interrupções, chaveamentos de contexto e de modo de execução, além da necessidade de cópia dos dados a cada nível da pilha de protocolos. Por estes motivos, protocolos como o TCP e o UDP podem limitar o desempenho que se pode obter em *clusters*, que tipicamente demandam uma rede de alta largura de banda e baixa latência.

Para minimizar os problemas mencionados acima, foram desenvolvidos protocolos de rede em nível de aplicação, ou seja, a aplicação possui acesso direto à interface de rede, como é o caso do VIA. Em VIA, a aplicação *a priori* registra a(s) area(s) de memória cujos dados serão transmitidos. Assim, a aplicação pode, no momento da transmissão acessar diretamente a interface de rede. Por isso o VIA é considerado um protocolo *zero-copy*, ou seja, não são necessárias cópias de dados para *buffers* intermediários. No caso de UDP/IP a transmissão é feita pelo sistema operacional, o que implica em fazer a cópia dos dados entre a área da aplicação e a do sistema operacional.

VIA dispõe de um mecanismo chamado *RDMA* (*Remote Direct Memory Access*), que permite que o conteúdo de determinada posição de memória de um nó seja transferido diretamente para a memória de outro nó. As principais desvantagens de VIA são: a) a complexidade de programação, fato que contribui para sua pouca difusão; e b) o protocolo é suportado apenas por algumas interfaces de rede.

3. Alterações no HLRC

A implementação corrente de HLRC foi desenvolvida utilizando VIA. O objetivo desse trabalho é avaliar a viabilidade de uma versão que utilize UDP/IP, visto que este é suportado pela maioria das interfaces de rede.

HLRC define originalmente buffers para o envio e recebimento de requisições, assim como para transmitir *diffs* e *write-notices*. Todas estas estruturas, assim como as páginas de memória, passam por um processo de registro durante a inicialização, que as impedem de serem paginadas pelo sistema operacional. Este processo é imprescindível para que possam ser transmitidas via *RDMA*. Na versão do HLRC com UDP/IP, este processo de registro não é necessário.

Na implementação do HLRC com VIA, o tratamento de requisições entre os nós é feito da seguinte forma: é disparada uma *thread*, chamada Servidor, em cada nó, que é utilizada para atender requisições de outros nós. O Servidor bloqueia-se na fila de recepção de mensagens a espera de uma requisição. A implementação com UDP/IP utiliza o mesmo esquema para tratar requisições: o Servidor mantêm-se bloqueado na ausência de requisições, sendo notificado pelo sistema operacional na chegada de uma nova requisição.

Finalmente, foi feita uma modificação na forma como um nó espera a resposta à uma requisição. Na implementação com VIA, como a resposta é transmitida por *RDMA*, o nó que fez a requisição verifica constantemente a posição de memória para onde é transmitida a resposta. Esta abordagem faz com que o nó gaste ciclos da CPU no processo de espera por uma resposta. Na implementação com UDP/IP, tendo em vista que a transferência dos dados é feita pelo Servidor do nó que fez a requisição, a *thread* incumbida de fazer o processamento da aplicação bloqueia em um semáforo à espera da resposta. Quando a transmissão da resposta termina, o Servidor notifica a *thread* da aplicação sinalizando o semáforo, assim não há consumo de ciclos da CPU durante a espera.

4. Avaliação

4.1 Ambiente de testes

Todos os experimentos foram realizados em um cluster de oito nós. Cada nó contém um processador Pentium III de 650MHz, cache L2 de 256 KBytes, 512 MBytes de memória principal e página virtual de 4 KBytes. O sistema operacional utilizado é o Linux na versão 2.2.14. Cada nó se comunica por uma interface de rede Fast Ethernet *Intel EthernetPro 100*, sendo os nós interligados por um *switch Micronet EtherFast*. Para executar VIA na Fast Ethernet, usamos M-VIA 1.1 [9], uma implementação VIA em software para Linux. Existe uma implemantação do VIA com suporte de hardware que possui maior desempenho, porém, o custo desta implementação é bem maior. Tendo em vista que este trabalho visa apresentar um software DSM que, possa ser utilizado em redes comuns e de baixo custo, omitimos os resultado obtidos utilizando a implementação do VIA em hardware.

4.2 Aplicações

Foram avaliadas cinco aplicações científicas paralelas. Três aplicações fazem parte do pacote SPLASH2 [20], da Stanford University: FFT, LU, Radix. SOR e IS foram desenvolvidos na University of Rochester.

FFT realiza a transformada rápida de Fourier em N pontos. A comunicação existente é encontrada na transposição de uma matriz raiz de N por raiz de N. FFT distribui a matriz entre P processadores, de forma que cada processador receba um conjunto contínuo de N/P linhas. Por causa da transposição, cada processador lê dados de outro processador, logo, a granularidade de escrita é larga, ao passo que a granularidade de leitura depende de N e P. Em problemas de tamanhos típicos, normalmente são encontrados fragmentação e falso compartilhamento. Assim, o desempenho do FFT é afetado pela taxa de comunicação/computação e pela comunicação induzida pela fragmentação e falso compartilhamento.

Radix ordena uma série de chaves inteiras em ordem crescente. A fase dominante é a permutação das chaves. Em Radix, um processador lê seu conjunto de chaves locais de um vetor fonte e escreve-os em um vetor destino de uma maneira bastante regular e espalhada. Este padrão de escrita induz substancialmente o falso compartilhamento no acesso às páginas.

SOR resolve equações diferenciais parciais usando uma estratégia do tipo "vermelho-preto" para realizar relaxações sucessivas. Cada iteração é composta por duas fases separadas por barreiras. Os valores da matriz vermelha são calculados na primeira fase e os da matriz preta na segunda. Cada elemento da matriz é calculado como sendo a média de seus vizinhos na matriz original. Como a matriz é dividida em fatias entre os processadores, o compartilhamento de dados ocorre nas bordas de cada fatia.

LU fatora uma matriz densa em um produto de duas metades triangulares da mesma matriz. Barreiras são usadas para assegurar a sincronização entre o processador que está produzindo a linha pivô e os processadores que irão consumir esta linha. Este *kernel* exibe acessos de granularidade larga e a computação é inerentemente balanceada.

IS (Integer Sort) ordena um vetor de N inteiros usando chaves no intervalo [0, Bmax] através da técnica *Bucket Sort*. As chaves são divididas pelos processadores e cada iteração consiste de três fases separadas por barreiras. Na primeira fase o processador zero inicializa o vetor global de *buckets*. Na segunda fase, cada processador conta suas próprias chaves armazenando resultados parciais em um vetor local para, imediatamente antes de chegar à barreira, adicionar os resultados globais ao vetor global de *buckets*. Na última fase, todos os processadores leêm o vetor global para

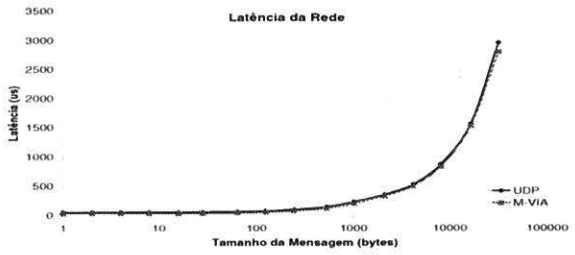

Figura 1. Latência UDP x M-VIA

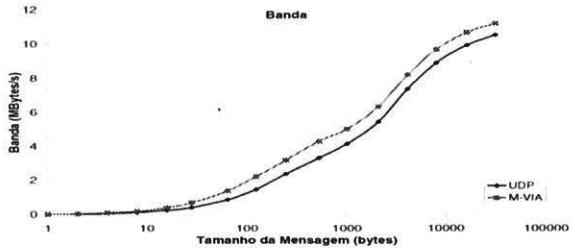

Figura 2. Largura de Banda UDP x M-VIA

classificar suas chaves locais, não havendo escritas a dados compartilhados.

4.3 Resultados

Todas as aplicações foram executadas nas versões UDP e VIA, visto que: a) TCP possui um desempenho inferior a UDP, o que provocou *slowdown* (de uma ordem de grandeza) em todas as aplicações, e b) redes locais usadas em *clusters* são bastante confiáveis.

Nas figuras 1 e 2 observamos que a latência de VIA é aproximadamente 70% da latência de UDP, enquanto que a banda é aproximadamente 7% superior.

O tempo seqüencial para a execução das aplicações é apresentado na Tabela 1. Cada aplicação foi executada 5 vezes; os tempos apresentados são uma média desses valores. Também apresentamos nesta tabela a entrada para cada aplicação, assim como o tipo de sincronização utilizado. Os *speedups* das aplicações são apresentados nas figuras 3, 4, 5, 6, 7. A Figura 8 divide o tempo de execução das aplicações em seis componentes distintos: computação, que representa o tempo efetivamente gasto na execução da aplicação; busca de página, que representa o tempo gasto, em uma falha de página, esperando pelo pedido e envio desta pelo nó residência; *lock*, que representa o tempo gasto esperando por um *lock* ser adquirido; barreira, que representa o tempo gasto esperando em uma operação de barreira; *overhead*, que representa o tempo gasto em ações do protocolo, como criação de *twins* e geração de *diffs*; e servidor, que representa o tempo gasto pelo tratamento dos pedidos recebidos pelo Servidor. Nesta figura utilizamos, para fins de comparação os tempos de execuções em 8 nós. A exceção é Radix, onde comparamos os tempos para 4 nós, visto que os resultados para 8 nós em M-VIA não estão disponíveis.

Em FFT, observamos que o *speedup* da aplicação rodando em UDP é superior ao obtido quando rodando em VIA. Para dois nós, notamos uma pequena diferença de desempenho entre ambos, sendo UDP aproximadamente 4,5% melhor que VIA. A diferença começa a se acentuar para a execução em 4 nós, tornando-se 31% em 8 nós. VIA apresenta pior desempenho em todas as componentes que formam o tempo de execução, sendo a diferença mais substancial presente no tempo gasto esperando por páginas. Esse comportamento, entretanto, não se repete para as demais aplicações, onde o tempo gasto esperando por páginas é sempre inferior em VIA do que em UDP.

Em IS, observamos que a versão em VIA também tem pior desempenho que a versão UDP. Apesar do tempo gasto esperando por páginas ser menor em VIA e de UDP gastar 60% mais tempo na barreira, o processamento do sincronismo em *lock* em VIA é 140% maior que em UDP, compensando aquelas diferenças. Como resultado, o *speedup* de UDP chega a ser 10% superior.

A sincronização é novamente a responsável pelo pior desempenho de VIA em LU e Radix. O tempo gasto em barreira em LU é aproximadamente 3.25 vezes superior ao tempo de UDP. Já em Radix, o tempo em barreira é 2.17 vezes superior. Com isso, o *speedup* de VIA para LU é 32% menor que o encontrado para a versão UDP. Enquanto que, no caso de Radix em 4 nós, a versão VIA atinge um *speedup* igual a apenas 54% do encontrado em UDP.

A única aplicação em que a versão VIA apresenta desempenho igual a UDP é SOR. Dois componentes são responsáveis pelo desempenho apresentado: o tempo gasto esperando por páginas, e o tempo gasto no servidor.

4.4 Análise

Contrariando a nossa expectativa, o desempenho da versão UDP apresentou-se melhor que a versão VIA. Relacionamos este fato a forma como HLRC é implementado em VIA: por usar *RDMA* para responder aos pedidos, a *thread* que está esperando testa continuamente uma região de memória para saber se a resposta chegou.

Esta abordagem é válida quando o intervalo de espera entre mensagens é pequeno, caso contrário, acarreta uma serialização no processamento da aplicação, já que esta *thread* estará competindo pelo uso do processador com o Servidor, atrasando a resposta de pedidos remotos. Na

Aplicação	Entrada	Tempo Seq.	Sincronização
IS	2^{20} chaves, 5 iterações	0.5 segs	*lock*s, barreiras
Radix	20M chaves	40 segs	*lock*s, barreiras
LU	2048 x 2048	105 segs	barreiras
SOR	2096 elementos, 8 iterações	26 segs	barreiras
FFT	2048 x 2048	90 segs	barreiras

Tabela 1. Características das aplicações

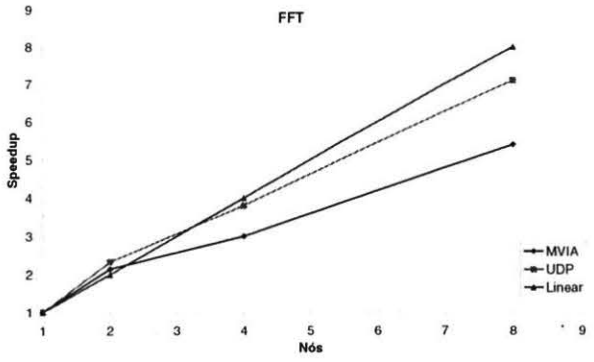

Figura 3. Speedup do FFT.

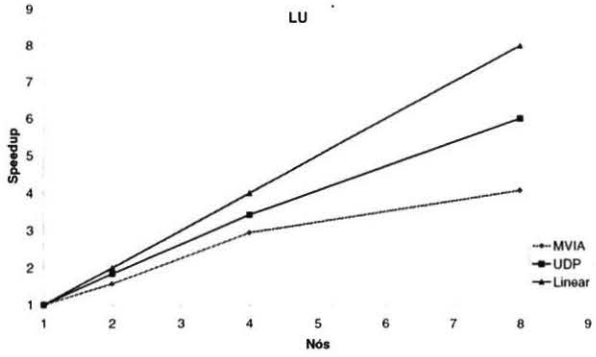

Figura 5. Speedup do LU.

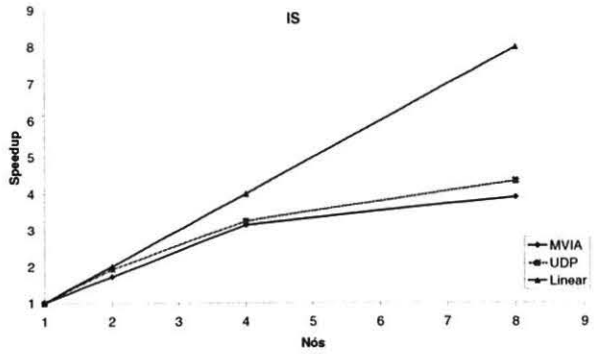

Figura 4. Speedup do IS.

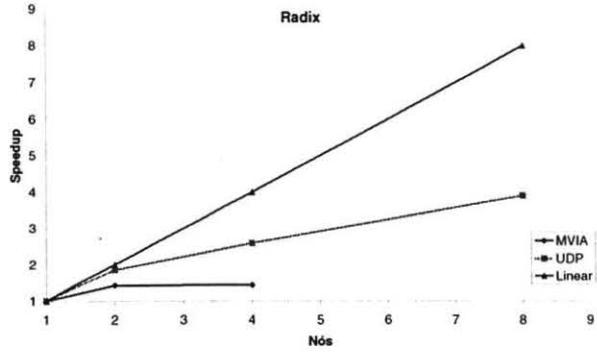

Figura 6. Speedup do RADIX.

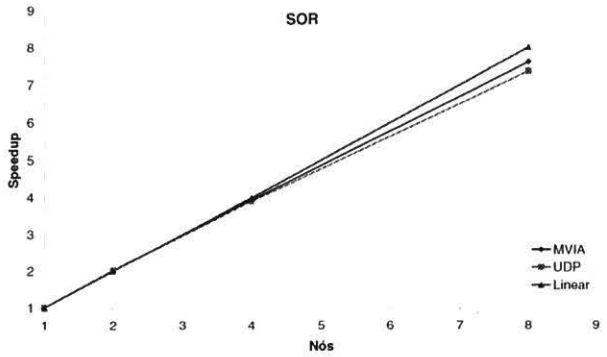

Figura 7. Speedup do SOR.

implementação UDP, utilizamos um semáforo de tal forma que a *thread* requerente espera bloqueada pela resposta. A *thread* Servidor recebe os dados pedidos, escreve os dados na posicão esperada e desperta a *thread* requerente, evitando assim a concorrência entre as *threads* durante a espera da resposta.

Podemos verificar que a nossa abordagem aumentou o número de requisições atendidas pelo Servidor. Entretando, para FFT, IS e LU, esse aumento no atendimento de requisições não impactou negativamente os tempos do servidor. O excesso de requisições atendidas pelo servidor pode ser observada em Radix e SOR. Para Radix, o tempo de servidor é 60% maior na versão UDP, enquanto que para SOR, a versão UDP apresenta esse tempo 11 vezes superior.

5. Trabalhos Relacionados

Um trabalho [17] anterior faz a análise de desempenho de três redes popularmente utilizadas para interconectar computadores em um cluster: FastEthernet, Myrinet e Giganet. É feita a avaliação do desempenho do protocolo VIA em relação ao UDP/IP. Para isso é utilizado o conjunto de benchmarks para aplicações paralelas NAS[1], sendo as aplicações implementadas utilizando MPI[5]. Como conclusão é apresentado que a rede FastEthernet possui uma boa relação custo benefício, quando utilizada em conjunto com o protocolo M-VIA. No entanto, a relação de desempenho entre UDP e M-VIA difere dos nossos resultados.

Outro trabalho [2] descreve a implementação do software DSM TreadMarks [14] utilizando o protocolo VIA sobre duas redes: Gigabit Ethernet e Myrinet. O modelo de comunicação do TreadMarks não se encaixa perfeitamente na API do VIA, sendo necessário desenvolver um substrato baseado em VIA para tratar este problema. São apresentados diversos esquemas para implementar este substrato e é feita uma avaliação de desempenho em relação à versão original do TreadMarks, sendo utilizado o próprio conjunto de aplicações do pacote do TreadMarks. Este trabalho conclui que o VIA pode diminuir em até duas vezes o tempo de execução das aplicações avaliadas, utilizando as redes mencionadas anteriormente.

6. Conclusões

Neste trabalho estudamos o impacto de dois protocolos de rede, UDP/IP e VIA, no desempenho do sistema software DSM HLRC. O desempenho de UDP foi superior ao de VIA em 4 dos 5 benchmarks empregados. Atribuímos esse resultado principalmente à forma como a versão de HLRC com VIA foi implementado: a necessidade de testar uma região de memória para determinar o recebimento de mensagens gera uma ocupação desnecessária da CPU e conseqüentemente serializa a execução de todas as *threads* da aplicação paralela. O outro fator que também muito contribui para os resultados encontrados é pequena diferença encontrada no desempenho dos protocolos M-VIA e UDP.

Estamos otimizando o sistema HLRC na versão VIA e estamos realizando os mesmos estudos para uma rede de largura de banda de gigabits por segundo, onde encontramos uma implementação em hardware de VIA. Algumas otimizações ainda podem ser feitas na implementação na versão UDP/IP, como: algumas mensagens de controle podem ser empacotadas juntamente com determinadas mensagens de dados, tendo em vista a latência de envio de mensagens no UDP/IP.

Podemos afirmar que a nossa implementação do sistema HLRC para o protocolo UDP/IP se apresenta como uma boa alternativa para o processamento de alto desempenho em *clusters* de baixo custo.

Referências

[1] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, March 1994.

[2] M. Banikazemi, J. Liu, D. K. Panda, and P. Sadayappan. Implementing Treadmarks over Via on Myrinet and Gigabit Ethernet: Challenges, Design Experience, and Performance Evaluation. In *Int'l Conference on Parallel Processing*, September 2001.

[3] G. Bell and J. Gray. What's Next in High-Performance Computing ? *Communications of the ACM*, 45(2):91–95, February 2002.

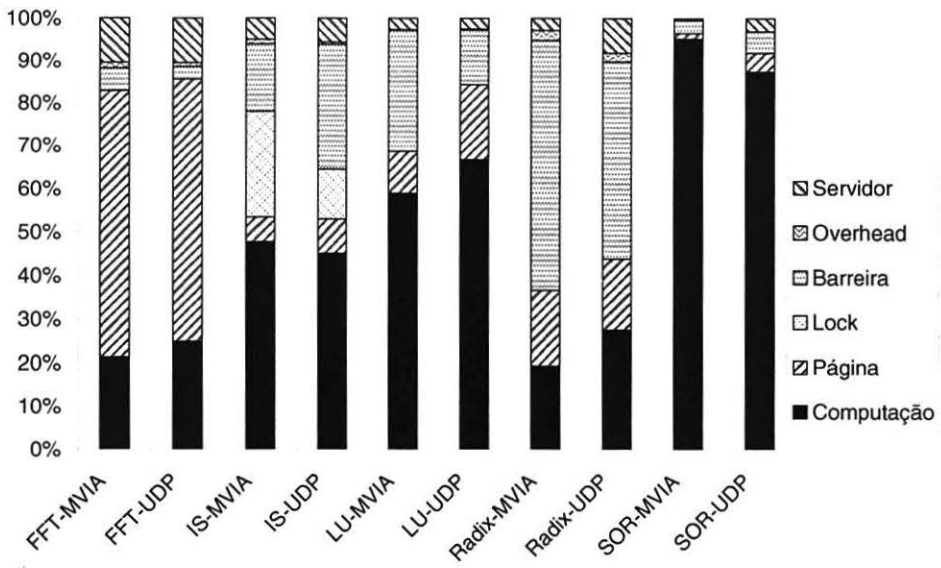

Figura 8. Componentes da Execução.

[4] S. Dwarkadas, P. Keleher, A. L. Cox, and W. Zwaenepoel. Evaluation of Release Consistent Software Distributed Shared Memory on Emerging Network Technology. In *Proc. of the 20th An. Int'l Symp. on Computer Architecture (ISCA'93)*, May 1993.

[5] Message Passing Interface Forum. MPI: A Message Passing Interface. In *Proceedings of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, 1993.

[6] http://www.dolphinics.com/. *Dolphin SCI.*

[7] http://www.infinibandta.org. *Infiniband Trade Association.*

[8] http://www.myri.com. *Myricom.*

[9] http://www.nersc.gov/research/FTG/via/. *M-VIA: A High Performance Modular VIA for Linux.*

[10] http://www.top500.org. *TOP500 Supercomputer Sites.*

[11] http://www.vidf.org. *Virtual Interface Architecture Specification, Version 1.0.* Compaq Corporation, Intel Corporation, and Microsoft Corporation, 1997.

[12] L. Iftode and J.P.Singh. Shared Virtual Memory: Progress and Challenges. *Proc. of the IEEE, Special Issue on distributed Shared Memory*, 87(3):498–507, 1999.

[13] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. In *Proc. of the 19th An. Int'l Symp. on Computer Architecture (ISCA'92)*, pages 13–21, May 1992.

[14] P. Keleher, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. Treadmarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proc. of the 1994 Winter Usenix Conference*, January 1994.

[15] K. Li and P. Hudak. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems*, pages 321–359, November 1989.

[16] L.I.Kontothanassis and M.L.Scott. Using Memory-Mapped Network Interfaces to Improve the Performance of Distributed Shared Memory. In *Proc. of the 2nd IEEE Symp. on High-Performance Computer Architecture (HPCA-2)*, 1996.

[17] M. Lobosco, V. S. Costa, and C. L. Amorim. Performance Evaluation of Fast Ethernet, Giganet, and Myrinet on a Cluster. In *International Conference on Computational Science*, pages 296–297, 2002.

[18] M. Rangarajan and L. Iftode. Software Distributed Shared Memory over Virtual Interface Architecture:

Implementation and Performance. In *Proc. of the 3rd Extreme Linux Workshop*, October 2000.

[19] W. E. Speight and J. K. Bennett. Using multicast and Multithreading to Reduce Communication in Software DSM Systems. In *Proc. of the Fourth IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-4)*, pages 312 – 322, Febrary 1998.

[20] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH2 Programs: Characterization and Methodological Considerations. In *Proc. of the 22nd An. Int'l Symp. on Computer Architecture (ISCA'95)*, pages 24–36, May 1995.