MPI Broadcast com Compressão de Números de Ponto Flutuante

Jose J. Camata[1] ,Renato N. Elias[2] , Alvaro L.G. A. Coutinho[3]
Núcleo de Atendimento em Computação de Alto Desempenho,
COPPE/Universidade Federal do Rio de Janeiro
Rio de Janeiro/RJ
{camata[1], renato[2], alvaro[3]}@nacad.ufrj.br

Resumo

Aplicações cientificas desenvolvidas para sistemas paralelos de memória distribuída consomem parte do seu tempo total de execução trocando dados entre processos. Portanto, aprimorar o desempenho das rotinas responsáveis pela comunicação vem ganhando cada vez mais importância. Neste contexto, este trabalho investiga a utilização de um algoritmo de compressão de ponto-flutuante na transmissão de mensagens longas. Este algoritmo foi implementado na primitiva broadcast do MPI e foram efetuadas medições de desempenho para diferentes tipos de mensagens em até 512 núcleos de processamento. Os resultados obtidos demonstram que a compressão pode acelerar significativamente o broadcast padrão do MPI.

1. Introdução

Nos últimos anos, sistemas paralelos de memória distribuída vêm ganhando grande importância em aplicações científicas. Segundo a lista TOP500 [1], mais de 80% de todos os supercomputadores do mundo utilizam esse tipo de arquitetura. A comunicação entre processos por trocas de mensagens (MPI) [2] emergiu como o principal paradigma para o desenvolvimento de aplicações de alto desempenho para tais sistemas. Para que estas aplicações possam extrair a melhor escalabilidade paralela, é crucial que as rotinas de comunicação sejam realizadas eficientemente. Estudos indicam que a comunicação entre processos demandam boa parte do custo total de processamento [12]. Aprimorar as rotinas de comunicação é desejável, mas também um desafio. Esses aprimoramentos devem considerar as heterogeneidades das arquiteturas disponíveis bem como as diferentes topologias de rede.

Muitos trabalhos foram realizados com o intuito de melhorar o desempenho das implementações do MPI. A biblioteca STAR-MPI [11] implementa um conjunto de rotinas de comunicação coletiva que é capaz de se adaptar à arquitetura dos sistemas e a carga de trabalho demandada pela aplicação. Outros trabalhos implementaram novos algoritmos específicos para determinadas arquiteturas [15]. Otimizações nas operações de redução são apresentadas em Robenseifner [14]. Em [5] é apresentado um novo mecanismo de *broadcast* que realiza a operação em tempo constante, independente do tamanho do número de processadores participantes.

Freqüentemente, o emprego de algoritmos de compressão de ponto-flutuante em aplicações científicas tem como princípio diminuir os requisitos de armazenamento dos dados produzidos durante a simulação. Por outro lado, surgiram alguns trabalhos que utilizam esquemas de envio de mensagens comprimidas de forma a reduzir a latência inerente à comunicação. Nesse sentido, Ke *et al.* [8] apresentam a biblioteca cMPI que utiliza um algoritmo de compressão baseado em predição [9]. Seus resultados indicam ganhos no desempenho das rotinas de comunicação ponto-à-ponto e coletiva. Kumar *et al.* [6] comparam o desempenho de diversas técnicas de compressão na otimização das rotinas *send-receive* e *AlltoAll* em uma aplicação de modelagem atmosférica.

O presente trabalho tem como objetivo adaptar o algoritmo de compressão de ponto-flutuante desenvolvido por Burtscher e Ratanaworabhan [3] e investigar o efeito da compressão de mensagens longas na redução do tempo de comunicação. O interesse inicial é aplicar a técnica de compressão sobre longas cadeias de números de ponto-flutuante que serão enviadas aos demais processadores que compartilham o mesmo comunicador via *broadcast*.

O trabalho foi organizado nas seguintes seções. A Seção 2 apresenta as motivações para uso da compressão de dados na trocas de mensagens. O algoritmo de compressão de trocas de mensagens é apresentado na seção seguinte. Na seção 4, detalhes de

implementação do broadcast com compressão são discutidos. Resultados são mostrados na seção 5 e as conclusões finalizam o artigo.

2. Motivação

Para entender o impacto da comunicação de grandes mensagens em uma aplicação, foi realizado a medição da latência das rotinas MPI através do *benchmark* PerfTest em um cluster SGI Altix ICE 8200[1] em 64 processadores. PerfTest avalia o desempenho das rotinas de comunicação ponto-à-ponto e coletivas realizando sucessivas execuções de forma a minimizar o erro no tempo de execução. Os testes consideram diferentes tamanhos de mensagens. Estas variam de 0 a 128 Kbytes.

As Figuras 1 e 2 mostram a latência obtida para a comunicação ponto-à-ponto e *broadcast*, respectivamente. É possível observar que a latência da comunicação cresce rapidamente para mensagens superiores a 16K em ambos os tipos de comunicação. Note que a redução da mensagem de 128K para 64K corresponde a uma redução de 46.9% na latência na comunicação ponto-à-ponto e uma redução de 45.3% no *broadcast*.

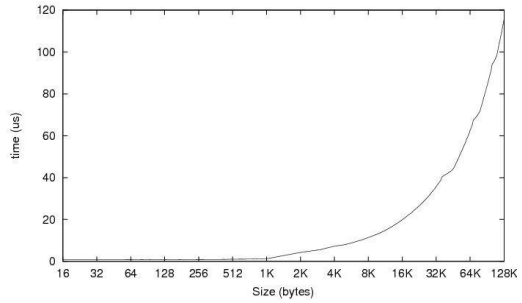


Figura 1: Latência para a comunicação ponto-à-ponto

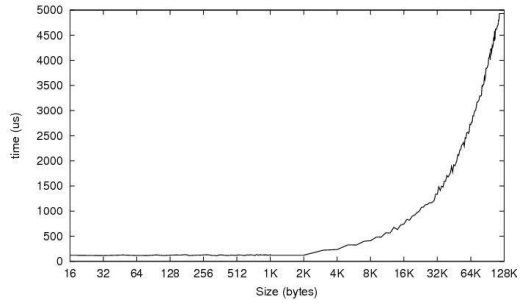


Figura 2: Latência para o *broadcast*.

[1] SGI Altix ICE 8200: Configurado com 32 nós com 2 processadores Intel Xeon Quad-Core 2.3 GHz / 4MB Cache L2. Rede *InfinBand* e OpenMPI versão 1.2.8.

Tais resultados sugerem que a redução do tamanho das mensagens através de algum esquema eficiente de compressão pode melhorar o desempenho de aplicações que utilizam mensagens longas na comunicação entre processos. Na próxima seção é detalhado o esquema de compressão.

3. Compressão de Ponto Flutuante

Neste trabalho é utilizado o algoritmo desenvolvido por Burtscher e Ratanaworabhan [3] para compressão de números de ponto-flutuante (FPC). Este algoritmo comprime sequências lineares de números de ponto flutuante com precisão dupla baseado em um mecanismo de dupla predição. A Figura 3 mostra o fluxograma do algoritmo de compressão.

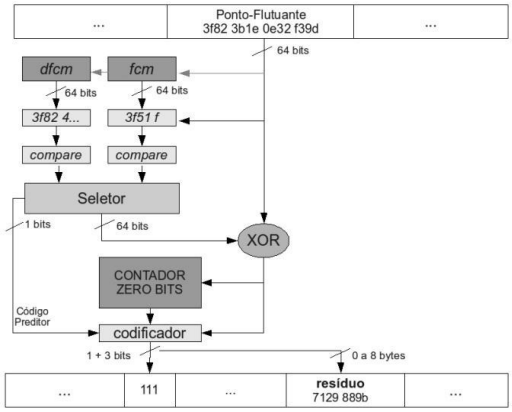


Figura 3: Fluxograma - FPC

Basicamente, o algoritmo de compressão emprega um preditor baseado nos valores previamente utilizados. Para comprimir um número de ponto-flutuante, o algoritmo prediz o próximo valor da sequência de números usando os preditores *fcm*[9] e *dfcm*[10]. A predição mais acurada, isto é, aquela que compartilha mais *bits* significativos, é escolhida para a operação de ou-exclusivo (*XOR*) junto com o valor real. A operação de ou-exclusivo transforma bits idênticos em zeros. Dessa forma, se o valor predito e o valor verdadeiro são próximos, o resultado dessa operação retornará zeros nos bits significativos. O algoritmo, então, obtém a quantidade de zeros, codifica esse número em 3 bits, concatenando-o com um bit auxiliar cuja função é indicar o mecanismo de predição utilizado. Os quatro bits juntamente com o resíduo do ou-exclusivo são escritos no arquivo.

Para realizar a descompressão, o FPC especifica, no inicio do arquivo comprimido, os números de ponto-flutuante que foram codificados. A descompressão inicia com a leitura dos quatro bits e do resíduo. Os

três bits que indicam o número de zeros são decodificados e, juntamente com o resíduo, chega-se a um número de 64-bits. Aplicando-se a operação de *ou-exclusivo* sobre este número e a predição do *fcm* ou *dfcm*, recria-se o ponto-flutuante original. Essa reconstrução é sem perda já que a operação *ou-exclusivo* é reversível. Outro ponto relevante é que os mecanismos de predição devem retornar a mesma sequência de valores nos processos de compressão e descompressão. Para tanto, eles são inicializados com zero e atualizados com o valor real após cada predição. Um estudo completo da complexidade computacional dos algoritmos de compressão e descompressão pode ser encontrado em [3].

4. MPI *Broadcast* e FPC

A primitiva *broadcast* do MPI é responsável pela distribuição de dados de um processo, chamado raiz, para todos os processadores que compartilham o mesmo comunicador. Estudos de Rabenseifner [14] identificaram que tal primitiva é uma das operações coletivas de maior custo. Uma vez que somente a semântica e a sintaxe dessa operação foram padronizados, códigos mais adequados à topologia de rede [4] e aos tipos/tamanhos de mensagens trocadas [11] foram sendo desenvolvidos.

A idéia aqui foi incluir e adaptar o algoritmo de compressão, apresentado na seção anterior, dentro do mecanismo de troca de mensagens do *broadcast*. O algoritmo FPC original acessa blocos de dados diretamente de arquivos. Para adequá-lo aos nossos objetivos, o algoritmo foi modificado para que os dados fossem lidos diretamente da memória de forma a aplicá-lo, em tempo de execução, à mensagem transmitida. A seguir é apresentado o pseudocódigo do *broadcast* com compressão de ponto-flutuante. A partir deste ponto, ele será identificado no texto por Bcast/FPC.

```
Algoritmo: Bcast/FPC
 Entrada:
  Buf  : mensg. com ponto-flutuante
  nfp  : número de ponto-flutuante
  Raiz : processo raiz
  Com  : comunicador MPI
 Saída:
  Buf  : mensg. com ponto-flutuante

  Obter ID do processador
 Se ID = raiz Então
   FPC_Comp(buf, size, zbuf,  zsize)
   BroadCast(zbuf,zsize,Raiz, Com)
 Caso Contrário
   BroadCast(zbuf,zsize, Raiz,Com)
   FPC_Decomp(zbuf, zsize, buf)
 Fim Se
```

Figura 3: Pseudocódigo algoritmo FPC

O algoritmo tem como parâmetros de entrada uma mensagem (*buf*) com *nfp* números de ponto-flutuante. O processador raiz, ou seja, aquele responsável pelo envio aos demais processadores, compacta a mensagem e envia aos demais processadores pertencentes. Por outro lado, os demais processadores, ao receberem a mensagem comprimida, realizam a descompressão da mesma para obter os dados originais.

5. Estudo de Desempenho

5.1. Sistema Computacional e esquema de medição

Para análise de desempenho do algoritmo desenvolvido neste trabalho, foi utilizado o cluster SUN denominado Ranger. Tal máquina foi disponibilizada pelo TACC (*Texas Advanced Computer Center*) e conta com 62976 núcleos de processamento. O processador é o AMD Barcelona 2.3GHz. Cada núcleo tem acesso a 2 GB de memória RAM. A rede é *infiniBand* e o MPI utilizado foi o MVAPICH2 versão 1.0. O sistema operacional é o CentOS 5.0 e o compilador é o Intel com as chaves de otimização padrão (-O).

Para as tomadas de tempo usou-se a rotina *MPI_Wtime*(). Para evitar medições errôneas do tempo de execução do *broadcast*, usou-se o tempo médio de cinco execuções do algoritmo.

5.2. Configuração dos Dados

A fim de testar o efeito da compressão de ponto-flutuante no broadcast, três configurações de dados foram consideradas. A primeira representa uma função linear com comportamento suave do tipo $f(x) = ax$. Essa configuração será identificada ao longo do texto por linear. Essa configuração representa a melhor situação para o algoritmo de compressão de dados uma vez que favorece o mecanismo de predição. Três tamanhos de vetores de pontos-flutuantes foram considerados: 32K, 64K e 128K. Para os três tamanhos de vetores, a taxa de compressão alcançada foi igual a 15.0. Isso significa que a mensagem comprimida é 15 vezes menor que a mensagem original.

A Figura 4 representa a segunda configuração de números de ponto-flutuante, identificada por exponencial. Tal configuração foi obtida pela função $f(x,y) = exp(-x^2)\ exp(-y^2)$ no domínio [-2,2] x [-2,2].

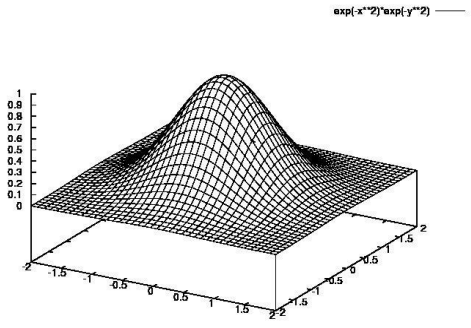


Figura 4: Configuração exponencial

A Tabela 1 reporta as taxas de compressão para três tamanhos de mensagens. Cada uma delas representa discretizações do domínio em 256 × 256, 512 × 512 e 1024 × 1024 células. Note que, estamos considerando duas formas de ordenamentos dos dados. A primeira é a ordenação natural onde os valores são obtidos de acordo com o esquema abaixo:

```
nf = 0
Para j= 0 até ny faça
     y =  y_inicial + j*dy
     Para i = 0 até nx faça
          x =  x_inicial + i*dx
          Buffer[nf] = f(x,y)
          nf = nf + 1
      Fim Para
Fim Para
```

Já a segunda configuração considera os valores da mensagem ordenados em ordem crescente. Vale ressaltar que o tempo gasto na ordenação dos dados é insignificante no tempo total de execução do algoritmo Bcast/FPC. Entretanto, em aplicações práticas a ordenação previa dos dados degrada o desempenho do esquema proposto uma vez que há o custo extra para a recuperação da ordem inicial dos dados. Diante disso, a configuração ordenada dos dados serve apenas para efeito de comparação.

Tabela 1: Taxa de Compressão – Função Exponencial

Ponto Flutuante	*Taxa de Compressão*	
	Normal	*Ordenados*
256 × 256	1.190	6.419
512 × 512	1.204	6.137
1024 × 1024	1.249	6.910

Finalmente, a terceira configuração representa a função trigonométrica *f(x,y) = sin(x)cos(y)* aplicada no domínio [-6,6] x [6,6], veja Figura 5.

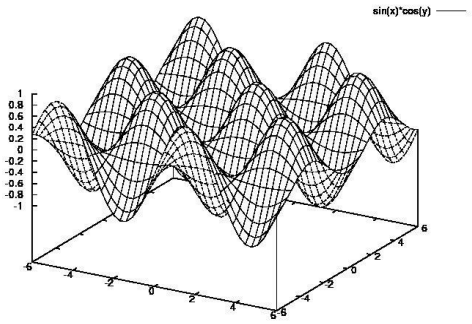


Figura 5: Configuração Trigonométrica

Como no caso anterior, a Tabela 2 lista as taxas de compressão para três tamanhos de *grids*, considerando também, os dois tipos de ordenação. Tal configuração apresenta os dados com maior variação espacial entre os já vistos até aqui. Consequentemente, os mecanismos de predição do FPC tendem fornecer valores não tão próximos dos valores reais. Diante disso, as taxas de compressão não são significativas mesmo considerando os dados ordenados crescentemente.

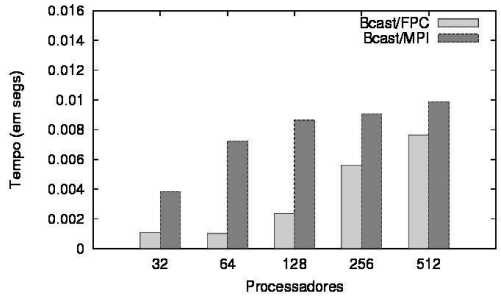
Tabela 2: Taxa de Compressão – Função Trigonométrica

Ponto Flutuante	*Taxa de Compressão*	
	Normal	*Ordenados*
256 × 256	1.127	2.296
512 × 512	1.154	2.306
1024 × 1024	1.255	2.480

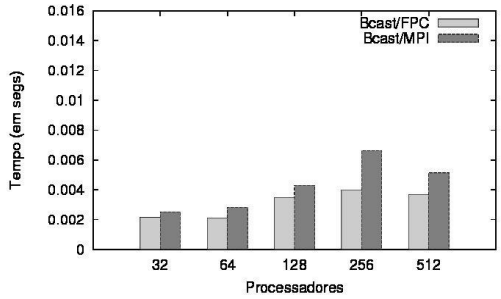
5.3. Análise de Desempenho

A Figura 6 compara o tempo de parede entre a rotina original de *broadcast* (Bcast/MPI) e a rotina com compressão (Bcast/FPC) para a configuração linear dos dados. O melhor desempenho foi obtido com mensagens com 32K pontos-flutuantes. O tempo de execução, para esse caso, foi sete vezes melhor que a rotina *broadcast* original em 64 processadores. No pior desempenho, o *broadcast* com compressão foi apenas 1.1 melhor quando a mensagem enviada para 32 processadores tinha o maior tamanho. Por outro lado, para 512 processadores, seu desempenho foi aproximadamente duas vezes melhor com mensagens de 128K. Esse desempenho está fortemente relacionado às altas taxas de compressão obtidas para

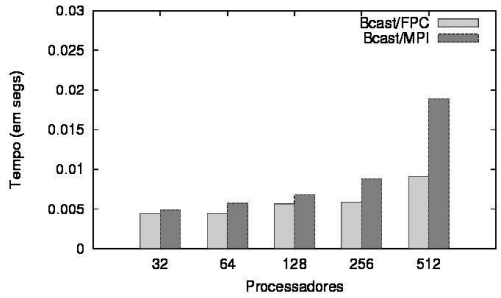
esse tipo de dados, conseqüentemente, impactando favoravelmente a redução do tempo de transmissão.



(a) 32K



(b) 64K



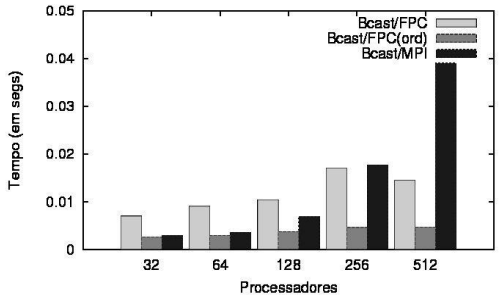
(c) 128K

Figura 6: Comparação entre o Bcast/MPI e Bcast/FPC– Dado Linear

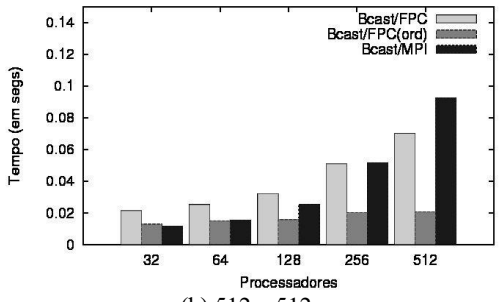
Considerando os custos individuais do FPC e da transmissão dos dados comprimidos, o FPC consome em média 1.02×10^{-3} segundos para mensagens com 32K. Isso corresponde a aproximadamente 93% do custo total do Bcast/FPC em 32 processadores. Com o aumento do número de processadores, o custo da comunicação torna-se mais significativo para o tempo total. Para 512 processadores, o custo do FPC corresponde a 18% do custo da rotina Bcast/FPC. Para mensagens com 64K e 128K números de ponto-flutuante os custos do FPC são 1.9×10^{-3} e 3.9×10^{-3} segundos, respectivamente.

Os resultados representados na Figura 7 ilustram os tempos de execução, quando os dados transmitidos foram gerados pela função exponencial. Para mensagens com ordenação original, o Bcast/FPC ganhou apenas quando 512 processadores foram empregados, independente do tamanho da mensagem. Esse ganho chega ser 2.7 vezes melhor que o esquema original para mensagens contendo 256×256 números de ponto flutuante. Neste caso, as taxas de compressão não foram suficientemente grandes para compensar o custo da compressão e transmissão. Nota-se que, para números de processadores inferiores a 256, o custo de transmissão da mensagem comprimida chega-se no melhor caso, a aproximadamente 83% do custo de transmissão da mensagem original.

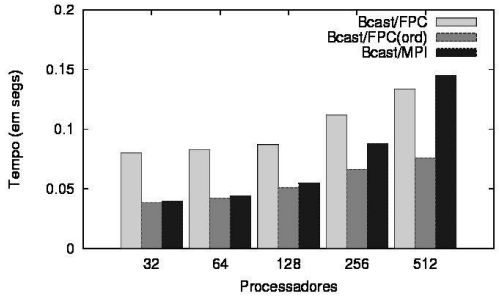
O desempenho do Bcast/FPC melhora quando consideramos os dados ordenados. Nesse caso, chega-se a um desempenho 8.3 vezes melhor na transmissão de 256×256 números de ponto flutuante em 512 processadores.



(a) 256 x 256



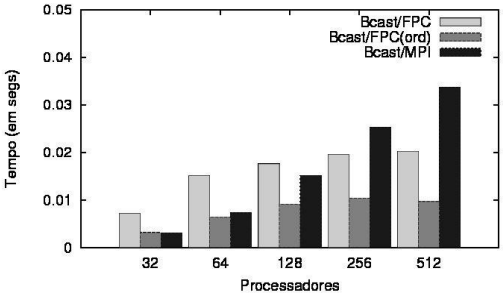
(b) 512 x 512



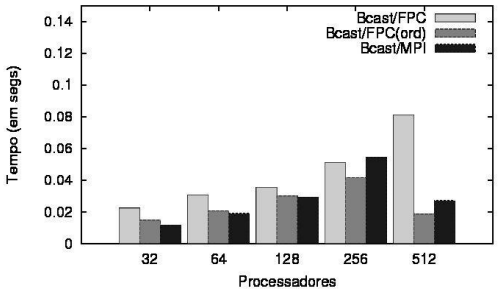
(c) 1024 x 1024

Figura 7: Comparação entre o Bcast/MPI e Bcast/FPC – Dado Exponencial.

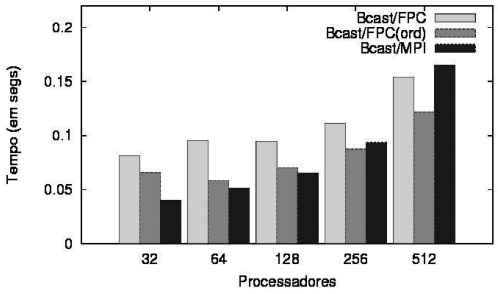
Finalmente, a Figura 8 mostra os resultados para a configuração de dados trigonométrica. Para mensagens com 256 × 256 números de ponto-flutuante, Bcast/FPC foi 1.28 vezes mais rápido que o Bcast/MPI com ordenação natural e 2.4 vezes mais rápida com os dados ordenados crescentemente quando executado em 256 processadores. Em 512 processadores, o ganho foi de 1.6 e 3.5 melhor, considerando os dados na ordenação original e crescente, respectivamente. Para mensagens com 512 × 512 números de ponto-flutuante, o Bcast/FPC obteve melhor desempenho com a ordenação original apenas quando executado em 256 processadores. Considerando os dados ordenados, o desempenho foi 1.3 e 1.45 melhor em 256 e 512 processadores. Por fim, considerando mensagens com 1024 × 1024 números de ponto-flutuante, o Bcast/FPC é melhor que o *broadcast* original em 512 processadores, independente do tipo de ordenação dos dados. Em 256 processadores, o desempenho foi melhor apenas com os dados ordenados crescentemente.



(a) 256x256



(b) 512 x 512



(c) 1024 x 1024

Figura 8: Comparação entre o Bcast/MPI e Bcast/FPC – Dado Trigonométrico

Como visto, essa última configuração de dados apresenta as taxas de compressão mais baixas e os piores resultados. Considerando-se mensagens com 1024 × 1024 números de ponto-flutuante com ordenação original, o custo dos procedimentos de compressão e descompressão correspondeu a 0.040 segundos. Para 32 processadores, esse valor foi superior ao tempo consumido pelo *broadcast* padrão. Em 64 processadores, o custo do FPC foi de 44% do custo total do Bcast/FPC e o custo de transmissão foi praticamente o mesmo do que a rotina original. Em 128 processadores, o custo do FPC foi de 42% em relação ao custo total. Já o custo de transmissão correspondeu a 83% do tempo consumido pelo esquema original. Em 256 e 512 processadores, o custo do FPC foi 36% e 25% do custo total, respectivamente. Além disso, as taxas de transmissão das mensagens comprimidas corresponderam a 76% e 69% do custo de transmissão da mensagem original. Note-se que, para essa configuração de dados, a redução da latência da comunicação da mensagem comprimida compensou o custo da compressão apenas quando executado sobre 512 processadores.

6. Conclusões

Este trabalho investigou a utilização de um algoritmo de compressão de ponto-flutuante na redução do custo total de transmissão de mensagens longas contendo tais dados. Um novo algoritmo de *broadcast* foi implementado, incorporando a compressão de mensagens na biblioteca MPI. Os resultados mostraram que esse algoritmo pode ser até oito vezes mais rápido que o mecanismo de *broadcast* original. Vale ressaltar que o desempenho desse algoritmo depende da previsibilidade dos dados contidos na mensagem. Verifica-se que quanto mais suave for a distribuição dos números de ponto-flutuante na mensagem maior será a eficiência da compressão dos dados. Consequentemente, menor será

a latência na comunicação. O papel da compressão de números de ponto-flutuante em outras primitivas MPI está atualmente sendo investigado.

Agradecimentos

Este trabalho foi desenvolvido no âmbito do acordo de cooperação entre a COPPE/UFRJ e o *Institute for Computational Engineering and Sciences* (ICES) da Universidade do Texas (UT) em Austin, EUA. Agradecemos M. Burtscher, do ICES por sua paciência e apoio no entendimento de seu algoritmo de compressão de números de ponto flutuante, W. Barth do *Texas Advanced Computer Center* pelo apoio na utilização do RANGER e ao Prof. Graham F. Carey, do ICES pela colaboração contínua ao longo dos anos entre a UT Austin e a COPPE/UFRJ. Este trabalho é apoiado ainda pelo CNPq, ANP e a PETROBRAS.

7. Referências

[1] Top500, www.top500.org. (2009)

[2] MPI Forum. http://www.mpi-forum.org/

[3] M. Burtscher & P. Ratanaworabhan. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Transactions on Computers*, 58(1):18-31. January 2009.

[4] T. Hoefler, C. Siebert & W. Rehm. A practical constant-time MPI broadcast algorithm for large-scale InfiniBand Cluster with Multicast. In *IEEE International Parallel & Distributed Processing Symposium*, 2007.

[5] Michael Brim & Joel Sommers. Efficient MPI broadcast communication and file distribution for local-area clusters. Project Report for CS 739: Distributed System, the University of Wisconsin, Madison.

[6] V. S. Kumar, R. Nanjundiah, M. J. Thazhuthaveetil & R. Govindarajan. Impact of message compression on the scalability an atmospheric modeling application on clusters. *Parallel Computing*, 38:1-16, 2008.

[7] P. Lindstrom & M. Isenburg. Fast and efficient compression of floating-point data. *IEEE transactions on visualization and computer graphics*, 12,(5):1245-1250, 2006.

[8] J. Ke, M. Burtscher & E. Speight. Runtime compression of MPI messages to improve the performance and scalability of parallel applications. In *SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004.

[9] B. Goeman, H. Vandierendonck & K. Bosschere. Differential FCM: increasing value prediction accuracy by improving table usage efficiency. In *Seventh International Symposium on High Performance Computer Architecture*, pp. 207-216, 2001.

[10] Y. Sazeides & J. E. Smith. The predictability of data values. In *30th International Symposium on Microarchitecture*, pp. 248-258, 1997.

[11] A. Faraj, X. Yuan & D. Lowenthal. STAR-MPI: Self tuned adaptive routines for MPI collective operations. In *Proceedings of the 20th Annual International Conference on Supercomputing*. pp. 199 – 208, 2006.

[12] R. Rabenseifner. Automatic MPI Counter Profiling. In *42nd CUG Conference*, 2000.

[13] J. Liu, A. Mamindala & D. Panda. Fast and scalable MPI-Level broadcast using InfiniBand's hardware multicast support, In *Parallel and Distributed Processing Symposium*, 2004.

[14] R. Rabenseifner, Optimization of Collective Reduction Operations, In *International Conference on Computational Science*, 2004.

[15] G. Almási, P. Heidelberger, C. J. Archer, J. Charles, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow & Y. Zheng, Optimization of MPI collective communication on BlueGene/L systems. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pp 253-262, 2005.