Expansão do projeto MPI.NET com a primitiva de comunicação Bsend

Ismael Stangherlini[1] , **Nicolas Maillard**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)

{istangherlini,nicolas}@inf.ufrgs.br

Abstract. *The parallel programming focused on high performance applications has reached a considerable importance in several areas of Computer Science nowadays. In this context, MPI implementations and interfaces for message passing systems between processes have been and are being developed in many parts of the globe, such as MPI.NET. This paper aims to propose an expansion to this implementation with a point-to-point communication primitive called Bsend, which is contained in the MPI standard.*

Resumo. *A programação paralela voltada a aplicações de alto desempenho tem conquistado uma importância considerável em diversas áreas da Ciência da Computação atualmente. Nesse contexto, implementações e interfaces MPI para sistemas de trocas de mensagens entre processos foram e estão sendo desenvolvidas em diversas partes do globo, tal qual a MPI.NET. Este trabalho tem a finalidade de propor uma expansão a esta implementação com a primitiva de comunicação ponto-a-ponto chamada Bsend, a qual está contida no padrão MPI.*

1. Introdução

A programação paralela trata-se de um modelo de programação em que tarefas complexas são divididas em tarefas menores a fim de serem executadas paralelamente (ou concorrentemente), realizando-se vários cálculos de forma simultânea em diferentes núcleos de processamento. Dentro desse contexto, criou-se a norma MPI (*Message Passing Interface*), a qual é uma especificação padrão para um protocolo de comunicação utilizado para realizar a interação entre diferentes processos em um aplicativo paralelo. A comunicação realizada entre esses processos ocorre, portanto, através da troca de mensagens.

Diversas implementações para diferentes linguagens de programação foram produzidas visando fornecer as facilidades do MPI, tais como o MPICH, o Open MPI e o LAM/MPI [1]. Porém, mesmo através da utilização de um protocolo padronizado de comunicação, a produção de aplicações paralelas pode ser considerada como uma tarefa de razoável complexidade, uma vez que o ser humano tende a raciocinar através de uma lógica seqüencial de eventos.

Baseado no princípio de construir uma camada de abstração entre o MPI e o aplicativo, o *Open Systems Laboratory* da Universidade de Indiana desenvolveu uma interface para uma implementação nativa do MPI, sendo tal interface denominada

MPI.NET e voltada para a linguagem de programação C#. Essa linguagem orientada a objetos foi projetada para possuir diversas facilidades relativas ao modo de programação e a abstração de conceitos [2]. Embora a maioria das funcionalidades presentes na especificação MPI estivessem contidas no MPI.NET, essa interface não provia meios para que o programador utilizasse um modelo de comunicação de envio de mensagens com o uso de buffers definidos pelo próprio usuário. Esse modelo de envio de mensagens é especificado pela primitiva de comunicação MPI_Bsend, juntamente com outras duas primitivas de definição e tratamento de buffers: a MPI_Buffer_attach e a MPI_Buffer_detach.

Este trabalho tem a finalidade de expandir a interface MPI.NET com a implementação da primitiva Bsend, a fim de que a mesma esteja disponível para a programação paralela com MPI na linguagem de programação C#.

2. A Solução MPI.NET

A solução apresentada pelo MPI.NET contém uma razoável quantidade de classes e abstrações. De forma geral, três dessas classes tiveram de ser modificadas para incorporar parte da nova solução, as quais serão descritas abaixo.

2.1. A Classe Communicator

Essa classe encapsula o conceito de pertinência de processos a um mesmo grupo, permitindo que mensagens sejam enviadas entre os mesmos através de chamadas a métodos públicos pertencentes a um determinado objeto dessa classe. Assim sendo, todas as funções MPI relativas à comunicação propriamente dita entre processos encontram-se traduzidas como métodos da classe Communicator.

2.2. A Classe Environment

Essa classe provê a inicialização, finalização e a disponibilidade de operações de consulta ao ambiente MPI. Da mesma forma que na especificação MPI o usuário deve utilizar a primitiva MPI_Init para inicializar o ambiente, na solução MPI.NET, deve-se instanciar um objeto da classe Environment para se obter tal funcionalidade.

2.3. A Classe Unsafe

Essa classe provê acesso direto a biblioteca MPI, onde as funções efetivamente implementadas são importadas e utilizadas pelo MPI.NET. Trata-se de um módulo importante para realizar a conexão da interface com o MPI nativo.

3. A Implementação da Nova Primitiva

A primitiva de comunicação MPI_Bsend existente na especificação MPI consiste no envio de mensagens através da utilização de um buffer definido pelo usuário. Assim sendo, no momento em que esta primitiva é executada por um processo, os dados a serem enviados são copiados para o buffer do usuário e a rotina retorna ao ponto seguinte a primitiva [3]. Trata-se de uma vantagem significativa nos casos em que existe uma grande ocupação do buffer do sistema, uma vez que é permitido um certo controle da quantidade de memória utilizada pelo MPI.

3.1. A Classe BufferMPI

Geralmente, a definição de buffers pelo usuário em linguagens como C se dá através da alocação de memória para um array de uma determinada quantidade de bytes. Como no C# arrays são objetos, eles são considerados alvos do Garbage Collector, artifício de gerenciamento de memória comumente encontrado em linguagens orientadas a objeto. Esse fator é um problema em potencial, uma vez que a implementação nativa do MPI exige que a área de memória utilizada pelo buffer do usuário seja fixa. Isso impede que seja feita a utilização natural de arrays em C# para a definição de buffers, já que os mesmos podem ser deslocados na memória pelo Garbage Collector.

A fim de padronizar o modo de criação de buffers pelo usuário, foi inserida uma nova classe na solução MPI.NET: a classe BufferMPI. A instanciação de um objeto dessa classe possui a capacidade de alocar uma determinada quantidade de bytes de memória não-gerenciada pelo Garbage Collector. Assim sendo, o usuário informa o número de bytes que deseja alocar através do construtor da classe. Em seguida, a implementação do construtor utiliza-se do método Marshal.AllocHGlobal do espaço de nome System.Runtime.InteropServices da plataforma .NET para alocar a quantidade de memória não-gerenciada desejada. A criação de buffers para utilização na primitiva Bsend restringe-se a instanciação de objetos da classe BufferMPI, portanto.

3.2. O Bsend

Uma vez que os métodos referentes à comunicação ponto-a-ponto encontram-se descritos na classe Communicator, foi realizada a inserção de um novo método para o envio de mensagens através do modo *buffered*: o *Bsend*. A criação desse método seguiu os mesmos princípios da implementação da primitiva Send. Dessa forma, diferentes tratamentos são fornecidos conforme o tipo de dado a ser enviado pelo MPI.NET.

O tratamento dos dados a serem enviados ocorre de uma maneira particularmente simples. No caso do dado ser de algum tipo simples da linguagem C# como *int*, por exemplo, associa-se o dado a um tipo MPI existente – MPI_INT, no caso. Por outro lado, se o dado não possui um tipo MPI nativo associado como, por exemplo, um objeto ou uma estrutura, é necessária a realização da serialização do dado, a fim de que o mesmo possa ser enviado como um stream de bytes. É importante notar que, nessa situação , dois envios são feitos: um contendo o tamanho de bytes serializados do stream e outro contendo os dados propriamente ditos. Em ambos os casos de tratamento, a implementação do Bsend realiza chamadas à função nativa MPI_Bsend importada pela classe Unsafe.

3.3. As Alterações Finais

Além das alterações na classe Unsafe para que as primitivas MPI_Bsend, MPI_Buffer_attach e MPI_Buffer_detach fossem importadas da implementação nativa do MPI, tornou-se necessária a definição do modo pelo qual o usuário informaria ao ambiente sua intenção de utilização de um buffer próprio para o envio de mensagens com o Bsend. Essa questão pôde ser solucionada adicionando-se os métodos *BufferAttach* e *BufferDetach* à classe Environment.

O primeiro método recebe como parâmetro um objeto da classe BufferMPI e realiza a chamada nativa ao MPI_Buffer_attach com o vetor alocado por esse objeto. Já

o segundo método não recebe parâmetros e retorna um objeto BufferMPI contendo o buffer que estava sendo utilizado anteriormente pelo ambiente. Além disso, ele realiza a chamada nativa ao MPI_Buffer_detach, desanexando o buffer antigo do MPI.

4. Avaliações de Desempenho

Diversos testes foram realizados utilizando-se a primitiva Bsend, incluindo-se neles o envio de uma quantidade razoável de dados de tipos simples e abstratos do C#, como instanciações de objetos e estruturas. Todas as execuções obtiveram sucesso na sua realização. Além disso, foram efetuados testes com um programa seqüencial de multiplicação de matrizes para realizar uma comparação entre C e C#, a fim de que se possa ter uma noção do overhead existente com o C# - influência direta no desempenho de aplicativos paralelos. O resultado da execução para diferentes tamanhos de matriz pode ser observado na Figura 1. Os testes foram rodados em uma máquina Intel Pentium D (2.99 Ghz e 3.00 Ghz) com 1GB de memória RAM no Windows XP SP3.

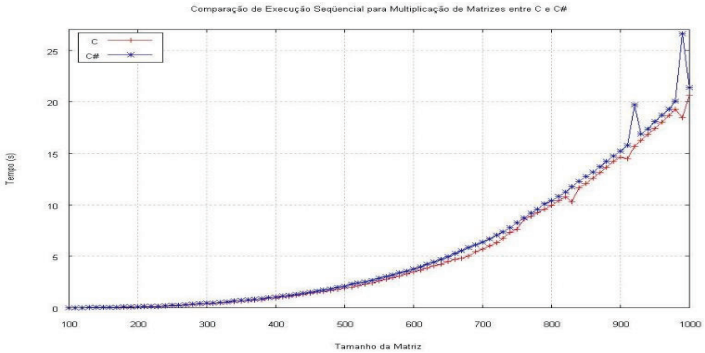

Figura 1. Comparação de multiplicação de matrizes entre o C e o C#

5. Conclusões

A inclusão de novas primitivas existentes na especificação padrão MPI na interface MPI.NET traz como vantagem uma solução mais robusta e possivelmente mais aceita para ser utilizada pela linguagem C# na plataforma .NET em aplicações de alto desempenho, associadas a programação paralela.

Dentro desse contexto, o trabalho desenvolvido na implementação da primitiva Bsend na Universidade Federal do Rio Grande do Sul foi parcialmente apoiado pela Microsoft e pôde ser realizado com sucesso.

Referências

[1] Gropp, W., Lusk, E. and Skjellum, A. (1999). "Using MPI: Portable Parallel Programming with the Message-Passing Interface". The MIT Press. 2ª edição.

[2] Deitel, H. M., Deitel, P. J., Listfield, J., Nieto, T. R. and Zlatkina, M. (2003). "C#: Como Programar". Pearson Education. 1ª edição.

[3] Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J. (1996). "MPI: The Complete Reference". The MIT Press. 1ª edição.