Análise Comparativa do Uso de MPI e Sockets Aplicados na Convolução de Imagens*

Epifanio Dinis Benitez†, Gerson Geraldo H. Cavalheiro

Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
São Leopoldo - RS - Brasil
{epifanio,gersonc}@exatas.unisinos.br

Introdução

Um dos objetivos da programação paralela e distribuída é diminuir o tempo de processamento em aplicações com grande demanda computacional. Para que isso seja possível, é necessário mapear a concorrência da aplicação no paralelismo da arquitetura.

A disponibilidade de aglomerados de computadores (*clusters*) multi-processados (SMP), motiva estudantes e pesquisadores a desenvolver aplicações paralelas e distribuídas para obter ganho de desempenho em relação à execução seqüencial. Nestes ambientes de Processamento de Alto Desempenho (PAD) existem dois níveis de paralelismo que devem ser explorados de maneira eficiente para obter ganho de desempenho, que são os níveis intra-nó e entre-nodos.

A exploração do paralelismo entre-nodos em um aglomerado consiste em distribuir as tarefas entre os nodos, utilizando ferramentas para a comunicação dos dados para, viabilizar a interação entre as tarefas. A comunicação pode ser realizada por bibliotecas de troca de mensagens tais como MPI (*Message Passing Interface*) ou Sockets (descritor de arquivos padrão do UNIX). Para a exploração do paralelismo intra-nó pode ser utilizado o padrão de *Threads* POSIX, mapeando a concorrência da aplicação no ambiente multi-processado (SMP).[CAV 2002] [DIV 2001]

O objetivo deste trabalho é comparar as ferramentas de comunicação para o Processamento de Alto Desempenho (PAD) em aglomerados de computadores (*clusters*). Para este estudo, foi desenvolvida uma aplicação que realiza um método de tratamento em imagens: a operação de convolução. A convolução possui um algoritmo altamente paralelo onde podem ser explorados os dois níveis de paralelismo. Para esta aplicação foram utilizadas as duas ferramentas clássicas de comunicação, MPI e Sockets, que oferecem um baixo nível de abstração para as interações entre tarefas. Toda comunicação é realizada de forma explícita disponibilizando, no entanto, um alto grau de flexibilidade, motivando o estudo e utilização destas ferramentas.

Para realizar a comparação foram definidos alguns aspectos considerados essenciais em uma ferramenta de comunicação, tais como requisitos para iniciar a comunicação, quais os protocolos de comunicação permitidos, quais os mecanismos para o envio e recebimento de mensagens, como realizar a sinronização do envio e recebimento de mensagens e como finalizar a comunicação em um aglomerado de computadores.

* Apoio: FAPERGS
† BIC-FAPERGS

Ferramentas

MPI [PAC 97] é uma biblioteca de comunicação que utiliza o mecanismo de troca de mensagens. O padrão de rotinas foi desenvolvido por Universidades, empresas e grupos de pesquisa e está disponível no MPI Fórum [1].

Para ativar o MPI, é necessário realizar a criação da máquina virtual especificando quais os nodos que estarão disponíveis para o processamento (é aconselhável que seja o mesmo número de nodos presentes no algomerado), que será fixo durante toda a execução do programa. A compilação de um programa MPI e a criação e desativação da máquina virtual são realizados através de comandos específicos desta bilioteca.

O modelo de programação mais utilizado com esta ferramenta é o SPMD (*Single Program, Multiple Data*), no qual o mesmo programa está presente em todos os nodos, e cada um deles é selecionado para executar um trecho do programa. Também é oferecido o modelo de programação MPMD (*Multiple Program, Multiple Data*).

Existe um amplo conjunto de rotinas do tipo *send/receive* para realizar a comunicação dos dados, e estão divididos em dois grupos: bloqueantes e não-bloqueantes. As rotinas bloqueantes travam o fluxo de execução até receber a confirmação de que a comunicação foi realizada com sucesso; isto garante a integridade dos dados no receptor. As rotinas não-bloqueantes não esperam a confirmação de recebimento e seguem normalmente o fluxo de execução. Internamente o MPI utiliza os protocolos TCP/IP e UDP em suas rotinas de comunicação.

Cada um dos nodos presentes inicializados pelo MPI possuem um identificador associado, tornando possível a comunicação entre quaisquer dois ou mais nodos. A sincronização das mensagens entre os nodos deve ser realizada especificando qual é o par de nodos que participa da comunicação, e em cada uma das partes, para qualquer rotina de *send* realizada, deve haver uma rotina *receive* correspondente.

Sockets [STE 98] [EVA 94] é um mecanismo de comunicação que utiliza descritores de arquivos padrão do UNIX. Um descritor de arquivo é um número inteiro associado a um arquivo aberto, que pode ser uma conexão em rede, Pipe, FIFO, entre outros. Isto quer dizer que para realizar a comunicação em uma rede, é necessário utilizar um descritor de arquivo.

Os Sockets podem ser utilizados para a comunicação em rede local, aglomerados ou para a Internet, utilizando o protocolo TCP/IP que garante a ordem e recebimento dos dados enviados, ou ainda utilizando o protocolo UDP.

A implementação distribuída com Sockets utiliza o modelo Cliente-Servidor. Para realizar a comunicação com os clientes o Servidor deve ter sido inicializado. Isto consiste em atribuir um endereço IP e uma porta a um Socket. Após a inicialização, o Servidor passa ao modo passivo (fica bloqueado), escutando a porta de comunicação até a chegada de uma requisição para uma conexão. A cada requisição um descritor é criado para identificar o canal de comunicação com o cliente. Através do descritor é possível utilizar rotinas do tipo *read/write* para o envio e recebimento de mensagens.

Não existe um mecanismo padrão para gerenciar as conexões estabelecidas, portanto é importante a criação de um que armazene e manipule os descritores dos clientes, pois a cada requisição eles são sobrepostos.

[1] ver http://www.mpi-forum.org

A sincronização entre os clientes e o servidor pode ser realizada com ou sem a utilização de alguma biblioteca auxiliar: simplesmente é necessário que para cada rotina de *read/write* no cliente, deve haver uma rotina *write/read* no Servidor para este cliente. O encerramento do canal de comunicação deve ser realizado através de rotinas específicas de Sockets, para as quais deve ser passado o descritor da conexão.

Implementação Concorrente em Aglomerado

A implementação concorrente em aglomerados foi explorada em dois níveis, intra-nó e entre-nodos. No nível intra-nó a concorrência foi mapeada no ambiente multi-processado (SMP) pela multi-programação leve (*multithreading*), seguindo o padrão POSIX para Threads nas implementações com MPI e Sockets.

A estratégia utilizada para o processamento distribuído foi *split-compute-merge*, que explora o paralelismo entre-nodos presente em arquiteturas de Processamento de Alto Desempenho. O processo principal realiza o *split*, que consiste em distribuir um bloco de dados de entrada para cada nodo do aglomerado; o processamento desses dados é realizado na etapa *compute*, e a devolução dos dados calculados para o processo principal é denominado *merge*.

No MPI cada nodo tem um **rank** (ID) associado que é utilizado para distribuir as tarefas entre os nodos. O nodo *master* tem **rank = 0**. O trecho de código abaixo mostra como o MPI reconhece o master e o escravo.

if (myrank == 0) { /* código do master */ }

else { /* código do escravo */ }

Através desse identificador, cada nodo identifica a parte do código que ele deverá executar, apesar de cada máquina possuir a cópia do mesmo programa, caracterizando portanto o modelo SPMD. O nó mestre deve ler a imagem e a máscara para poder calcular os dados que serão enviados aos nodos. Cada nodo deve receber a máscara, um bloco da imagem, o limite inferior e superior e o número de threads que irão processar os dados. Após a distribuição dos dados, o nó mestre aguarda os resultados: quando recebidos, eles são organizados através dos limites inferiores e superiores para gerar a imagem de saída.

Com Sockets o modelo de programação para a comunicação é Cliente-Servidor. O servidor se encarrega de verificar quantos clientes estão disponíveis para o processamento, e a partir disto, pode distribuir os dados necessários para que o processamento seja realizado. Esses dados consistem em um bloco da imagem, o limite inferior e superior assim como o número de threads. Após o processamento, os clientes devem enviar os dados ao servidor, que os organiza para formar a nova imagem.

Avaliação de Desempenho

As medidas de desempenho foram realizadas utilizando 3 computadores com processadores XEON bi-processados HT de 2.8 Ghz com 1 Gigabyte de memória RAM, utilizando uma rede de 10 megabits.

Na figura 1, podemos observar que o MPI, representado pela linha superior, obteve menor desempenho tanto na aplicação ping-pong como na aplicação de convolução. Isto pode ser explicado pelo fato de que o MPI possui uma máquina virtual que realiza uma

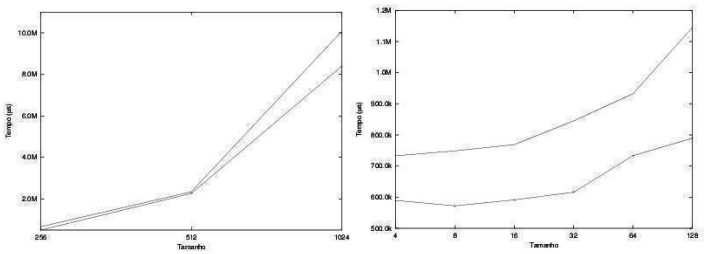

Figura 1: Aplicação Ping-Pong e Aplicação de Convolução

série de testes para verificar se as máquinas estão conectadas, além de executar mecanismos para garantir a comunicação dos dados. O tempo de comunicação, utilizando as duas ferramentas, apresentou um comportamento exponencial com relação ao tamanho da imagem. Sockets, por apresentar primitivas mais próximas ao sistema operacional, mostrou-se eficiente na comunicação com mensagens de diversos tamanhos.

Conclusão

A implementação com a ferramenta de comunicação MPI, por apresentar sua execução através de uma máquina virtual, disponibiliza facilidades para a implementação, pois não foi necessário utilizar qualquer outra rotina para sua inicialização. Por outro lado, Sockets necessitou da declaração de portas e endereço do servidor, que foram realizados através de rotinas específicas. O modelo cliente-servidor apresentou dificuldades na sincronização entre os clientes, e para isso foi desenvolvido um esquema de autenticação, garantindo que os clientes estarão conectados ao servidor antes de começar o envio dos dados. A máquina virtual do MPI lança todos os processos ao mesmo tempo, portanto está garantida a participação de todos os nodos. Pode-se concluir que o modelo de máquina virtual não é uma boa escolha se é desejado obter ganho de desempenho, quando comparado com Sockets, pois a flexibilidade e eficiência são requisitos para computação de alto desempenho. Neste caso, o modelo cliente-servidor mostrou ser a melhor opção para processamento paralelo e distribuído que demande um grande custo computacional.

Referências

[CAV 2002] CAVALHEIRO, G. G. H.; DIVERIO, T. A. (Eds.). **Anais da escola regional de alto desempenho**. [S.l.]: Instituto de Informática da UFRGS, 2002.

[DIV 2001] DIVERIO, T. A.; NAVAUX, P. O. (Eds.). **Anais da escola regional de alto desempenho**. [S.l.]: Instituro de Informática da UFRGS, 2001.

[EVA 94] EVANS, C. D. (Ed.). **Internetworking with tcp/ip**. Englewood Cliffs: Prentice Hall, 1994.

[PAC 97] PACHECO, P. S. (Ed.). **Parallel programming with mpi**. San Francisco: Morgan Kaufmann, 1997.

[STE 98] STEVENS, R. W. (Ed.). **Unix networking programming**: networking apis. [S.l.]: Prentice Hall, 1998.