Desenvolvimento de uma Biblioteca para Programação Paralela - BPP

Juliano F. C. Ciocari, Ana P. Canal

Centro Universitário Franciscano (UNIFRA)
Rua dos Andradas, 1614 – Centro – Santa Maria/RS, (55) 3220-1200
influenzi@gmail.com, apc@unifra.br

Introdução

As arquiteturas baseadas em *clusters* de computadores têm sido bastante utilizadas para processamento de alto desempenho. Para a construção de programas paralelos nessas máquinas é necessário o uso de primitivas básicas de comunicação, baseadas no envio e recebimento de mensagens. Neste texto é abordado o desenvolvimento de uma biblioteca para programação paralela, a BPP. Esta biblioteca tem por objetivo prover as primitivas básicas para comunicação através de trocas de mensagens, bem como para sincronização dos processos. Serão apresentadas suas principais características quanto à estrutura, implementação e interface de programação.

A Biblioteca Proposta: BPP

A BPP é uma biblioteca para programação em ambientes com memória distribuída, característica dos *clusters* de computadores, que possibilita a programação paralela baseada no modelo de trocas de mensagens [FOS 95]. A BPP é constituída de três módulos: módulo de Rede, módulo de Comunicação e Sincronização e o módulo de Gerência.

O módulo de Rede faz uso de *sockets* para enviar ou receber dados, sendo responsável por toda parte de comunicação, no nível de rede, entre diferentes processos. No módulo de Comunicação e Sincronização estão todos os procedimentos referentes à comunicação da biblioteca. E o módulo de Gerência é responsável pelo controle local e remoto dos processos. A organização em módulos é puramente lógica, para facilitar a execução e entendimento da implementação da BPP.

A BPP fornece recursos para trocas de mensagens, permitindo a interação de diferentes processos. Isso envolve, inicialmente, operações de envio e recebimento de mensagens. As operações implementadas até o momento são na forma síncrona, fazendo com que as trocas sejam realizadas diretamente entre os processos. Desta forma, quando um processo (transmissor) executar a primitiva de envio ele ficará bloqueado até que outro processo (receptor) receba sua mensagem.

Também é função da BPP gerenciar todos os processos referentes à determinada aplicação, desde sua criação até seu término. Isso é feito através da execução de comandos do sistema operacional em cada uma das máquinas da arquitetura, neste caso, um *cluster* de computadores.

Implementação

A BPP é codificada na linguagem de programação ANSI C sobre o sistema operacional GNU/LINUX 2.4.29 [LIN 05], usando *sockets* TCP/IP. A biblioteca *Pthreads* [IEE 95] também é usada para fazer o controle dos fluxos de execução, internos à biblioteca.

Na atual versão da biblioteca estão implementadas as rotinas de envio e recebimento de mensagens síncronas (*BPP_Send* e *BPP_Recv*), de inicialização e finalização (*BPP_Init* e *BPP_Finalize*), a rotina de sincronização (*BPP_Barrier*) e a de medida de tempo (*BPP_Time*). A tabela 1 identifica as tarefas executadas por cada uma das primitivas da BPP.

BPP_Init	Inicializa as estruturas básicas da biblioteca; Cria um novo *thread* que aguarda por conexões na respectiva porta do processo e espera por mensagens.
BPP_Finalize	Libera a memória usada nas estruturas da biblioteca; Finaliza o *thread* que espera por conexões.
BPP_Send	Conecta ao processo passado como parâmetro e envia uma mensagem; O processo remetente permanece bloqueado até que a mensagem seja recebida.
BPP_Recv	Procura na estrutura específica de mensagens recebidas da biblioteca por uma mensagem; O processo receptor permanece bloqueado até que encontre a mensagem.
BPP_Barrier	Bloqueia os processos até que todos executem esta mesma primitiva. Faz o controle através do envio de mensagens entre os diversos processos.
BPP_Time	Retorna o tempo decorrido desde o início da execução do processo que a chamou.

Tabela 1 – Tarefas de cada primitiva da BPP

Para haver transferência de dados entre os processos são definidos alguns tipos básicos de dados, como: Caracteres (*BPP_CHAR*) *char*; inteiros (*BPP_INT*) *int*; ponto flutuante de precisão simples (*BPP_FLOAT*) *float*; e ponto flutuante de precisão dupla (*BPP_DOUBLE*) - *double*. Para a definição de identificadores de processos é usado o tipo *BPP_proc*, que define uma variável do tipo *short int*.

As duas estruturas de dados principais da biblioteca são a *BPP_Cluster* e a *BPP_msgs_recebidas*, conforme a tabela 2. A primeira mantém uma lista completa com todas as máquinas que participarão da aplicação, com endereço e identificador de cada uma. A segunda possui todas as mensagens recebidas pelo processo no decorrer do tempo, é nela que a chamada *BPP_Recv* busca por mensagens.

```
struct BPP_ips {
  BPP_proc *proc;
  char *end;
}*BPP_cluster;
```

```
struct BPP_msgs_recebidas {
  BPP_proc id;
  void *msg;
  struct BPP_msgs_recebidas *prox;
} *BPP_lista
```

Tabela 2 – Tarefas de cada primitiva da BPP

Além destas primitivas, existem alguns *scripts* implementados em *BASH scripting*, para facilitar o uso da biblioteca. Um deles é o *bppcc* que deve ser usado para compilar os códigos fontes. Ele executa o compilador com os parâmetros corretos para o processo de link-edição com a BPP. O outro é o *bpprun*, usado para executar os programas, no *cluster* de computadores, depois de compilados. Ele se encarrega de passar o comando de execução para todas as máquinas, através do RSH - *Remote Shell*.

Interface de Programação

Antes do uso de qualquer primitiva da BPP, deve-se usar a *BPP_Init* que cria um ponto inicial para a comunicação, preparando a biblioteca para as trocas de mensagens. Ela recebe como parâmetro as variáveis *argc* e *argv,* da função *main*, acrescidas de duas variáveis de tipo *BPP_proc* (*short int*), a primeira conterá o identificador do processo e a segunda, o número total de processos participantes da execução. Para encerrar o ponto de comunicação usa-se o procedimento *BPP_Finalize*, que não possui nenhum argumento. Nenhum procedimento da biblioteca deve ser usado fora deste bloco de comunicação.

```
#include <BPP.h>
#include <string.h>
int main(int argc, char *argv[])
{
     BPP_proc id,p;
     char str[10];
     int i;
     double ti,tf;

     BPP_Init(&argc,argv,&id,&p);
     BPP_Barrier(id,p);
     ti=BPP_Time();

     if (id == 1) {
         strcat(str,"MSG!");
         for (i=0; i<100; i++)
                BPP_Send(str,strlen(str)+1,BPP_CHAR,0);
     }
     if (id == 0)
          for (i=0;i<100;i++)
               BPP_Recv(str,5,BPP_CHAR,1);
     BPP_Barrier(id,p);
     tf=BPP_Time();
     printf("%f\n",(tf-ti));
     BPP_Finalize();
}
```

Quadro 1 – Programa Exemplo BPP

Para a comunicação ponto a ponto entre diferentes processos são fornecidas as primitivas *BPP_Send* e *BPP_Recv*. A *BPP_Send* recebe como parâmetro um ponteiro para o endereço de memória dos dados que serão enviados, um inteiro que especifica quantos elementos serão enviados, o tipo de dado que está sendo enviado, de acordo com os tipos da BPP e o número do processo de destino da mensagem. De maneira análoga funciona a *BPP_Recv*, com os seguintes parâmetros: ponteiro para endereço de memória que irá receber os dados, quantidade recebida, tipo de dado da BPP e processo de origem.

A função *BPP_Time* pode ser usada no decorrer do programa, sendo que retorna um valor do tipo ponto flutuante de precisão dupla com o tempo decorrido desde o início da execução do processo. A barreira, *BPP_Barrier*, garante que nenhum processo passe de determinado local antes que todos tenham atingido a barreira e tem como argumentos o identificador do processo e o número total de processos da aplicação.

Todos os programas feitos utilizando a BPP devem incluir no arquivo o cabeçalho <BPP.h>, que define todos os procedimentos e tipos de dados da biblioteca que podem ser utilizados pelo programador. Um exemplo da BPP e suas primitivas

encontra-se no Quadro 1, com um programa que executa cem trocas de mensagens, e calcula o tempo decorrido para executar todas as trocas. Para compilação e execução do código devem ser usados os scripts *bppcc* e *bpprun*, de acordo com o Quadro 2.

```
# bppcc exemplo.c -o exemplo

# bpprun exemplo
```

Quadro 2 – Compilação e Execução na BPP

Desempenho

Para comparação da BBP com a biblioteca MPI – *Message Passing Interface* [MPI 05], foram realizadas medidas do tempo de execução de uma aplicação que troca mensagens entre dois processos, cujos resultados estão na tabela 3. A execução desta aplicação foi feita em um *cluster* de computadores, o *cluster* Unifra, composto por três máquinas Pentium IV 2.4 Ghz, onde o nó principal possui 512 MB de RAM e os demais, 256 MB. Cada máquina tem uma interface de rede Intel padrão *Gigabit Ethernet*, no entanto existe uma limitação de banda imposta pelo *switch*, que opera no padrão *Fast Ethernet*. Com o aumento do número de mensagens a BPP mostrou menor desempenho, provavelmente relacionado à comunicação síncrona implementada. É necessário investigar este aspecto e também realizar comparações sobre a comunicação assíncrona, assim que implementada.

	10 Mensagens	100 Mensagens	1000 Mensagens
BPP	0.412 seg	0.657 seg	2.310 seg
MPI	0.610 seg	0.598 seg	1.677 seg

Tabela 3: Comparativo entre BPP e MPI

Considerações Finais

A BPP é uma biblioteca que permite a programação paralela e tem como principal característica a simplicidade, o que favorece seu uso para fins de estudos acadêmicos. Ela tem mostrado-se estável em relação à troca de mensagens, porém existem ainda trabalhos a serem feitos na sua implementação, como melhorar a rotina de sincronização, buscar a obtenção de desempenho e estende-la, codificando as operações de comunicação assíncronas e comunicação de grupo.

Referências

[FOS 95] FOSTER, I. T. **Designing and Building Parallel Programs: concepts and tool for parallel software engineering.** Addison-Wesley. 1995.

[IEE 95] IEEE. **Guide to the POSIX Open System Environment.** Disponível em: http://standards.ieee.org/reading/ieee/std/posix/1003.0-1995.pdf.

[LIN 05] LINUX. **Linux on line.** Disponível em <http://www.linux.org/> . Acesso em abril de 2005.

[MPI 05] MPI. **The Message Passing Interface Standard.** Disponível em: http://www-unix.mcs.anl.gov/mpi/. Acesso em maio de 2005.