O Modelo Síncrono BSP para Computação Paralela

Raphael Y. de Camargo
Ricardo Andrade

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo, Brasil

São Paulo, 23 de novembro de 2007

Introdução

- Computação Paralela
  - Computação realizada por múltiplos processadores simultaneamente para solucionar um problema computacional
  - Comunicação entre processadores pode ser feita por *troca de mensagens* (MPI e BSP) ou *memória compartilhada* (BSP)

Introdução

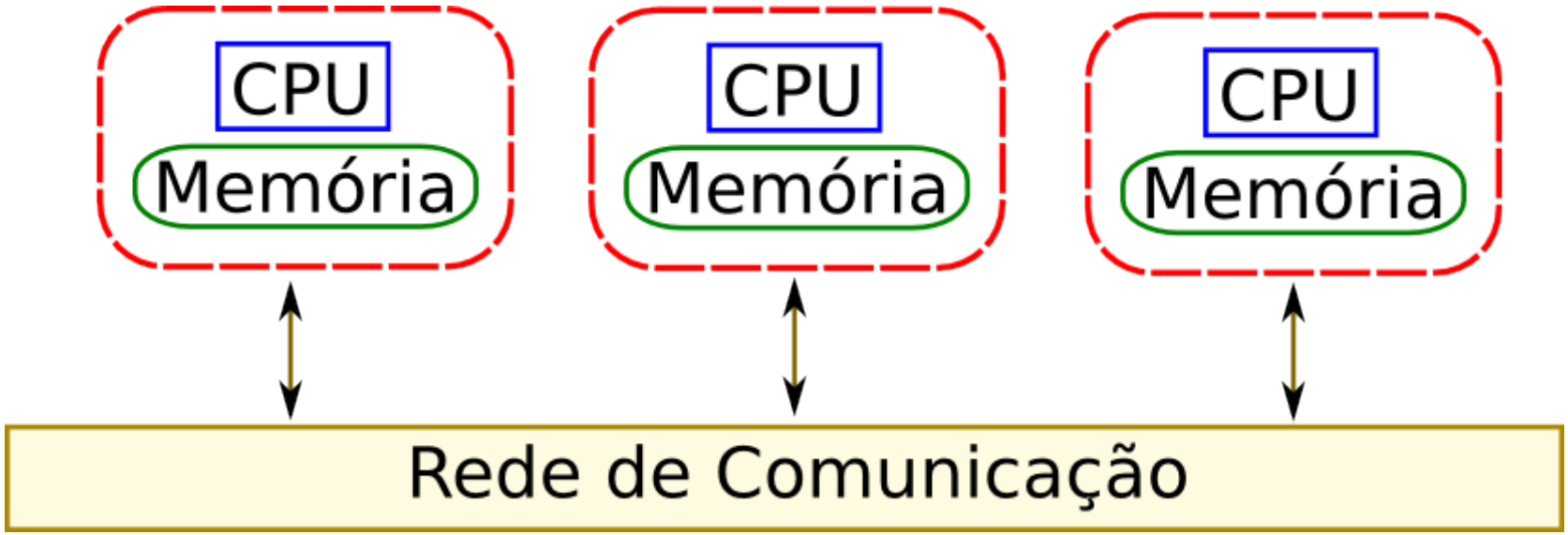
- Motivação para computação paralela
  - Muitos problemas complexos que exigem um alto poder computacional
    - Previsão do tempo, mercado financeiro, etc.
  - Aumento de velocidade de processadores esbarra no alto consumo de energia, geração de calor e dificuldades na miniaturização de componentes
  - É muito mais barato usar dois processadores em paralelo do que dobrar a velocidade de um processador

Introdução

- Programas paralelos
  - São mais difíceis de programar que os seqüenciais:
    - Coordenação e/ou sincronização dos processos
    - Comunicação entre processos
    - Distribuição de trabalho entre os processadores
- Modelos de programação
  - Fornecem maneiras de estruturar os programas
  - Facilitam o desenvolvimento de programas paralelos

Computador Paralelo BSP

- Múltiplos processadores com memória privada e conectados por uma rede de comunicação
- Processadores se comunicam através da rede de comunicação

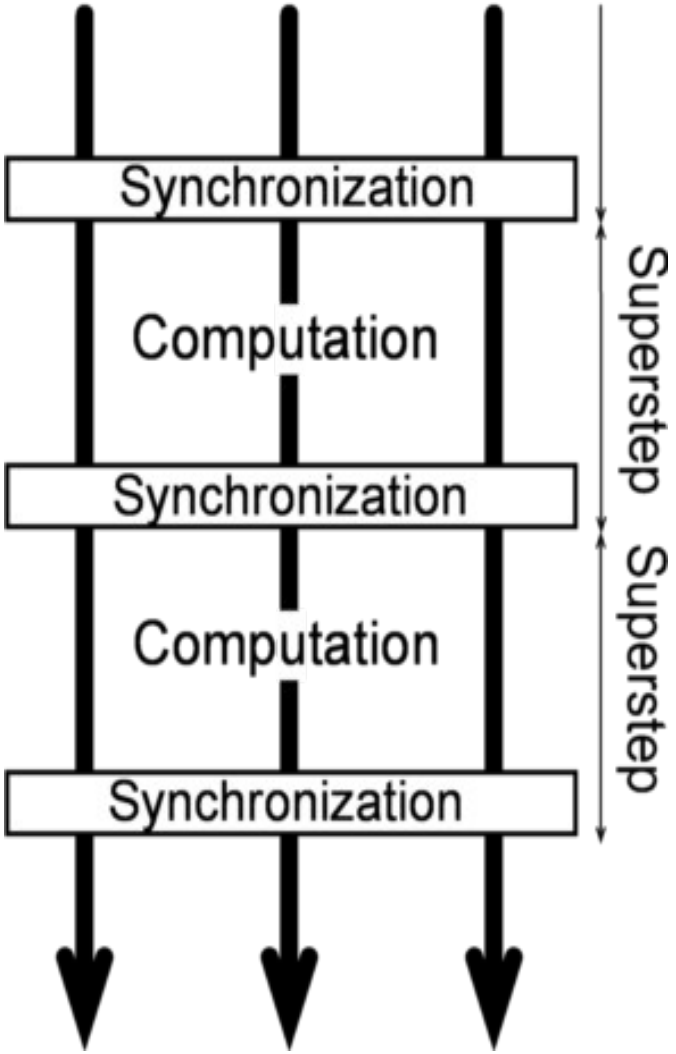


Computador Paralelo BSP

- Simula uma memória compartilhada distribuída
  - Cada processador pode acessar:
    - Memória local (acesso rápido)
    - Memória compartilhada de outro processador (acesso lento)
  - Processadores podem se comunicar por mensagens
- Programas BSP são portáveis
  - Não dependem da rede de comunicação utilizada
  - Modelo pode ser implementado em diferentes arquiteturas de computadores

Modelo de Programação BSP

- Aplicações paralelas
- Execução em *superpassos*
  - Computação e sincronização
  - Mensagens trocadas na fase de sincronização
- Modelo síncrono
  - Processadores periodicamente sincronizam sua execução
  - Facilita o controle da concorrência



Modelo de Programação BSP

- Fase de computação
  - Cada processador realiza uma ou mais operações computacionais
  - Cada processador informa os dados remotos deseja acessar e escrever e as mensagens que deseja enviar
- Fase de sincronização
  - Mensagens entre processadores são trocadas
  - Comunicação só é efetivada após esta fase

Modelos de Comunicação

- DRMA: *Distributed Remote Memory Addressing*
  - Simula uma memória compartilhada distribuída
  - Leitura da memória local ou remota
  - Escrita na memória local ou remota
- BSMP: *Bulk Synchronous Message Passing*
  - Comunicação é realizada através de troca de mensagens entre os processos
  - Envio de mensagens para uma fila de mensagens em um processo remoto
  - Leitura de mensagens da fila de mensagens local

BSPlib do InteGrade

- API baseada na BSPlib de Oxford
  - Programas escritos para a BSPlib de Oxford podem rodar no InteGrade com uma simples recompilação do código fonte
- API bastante simples e enxuta
  - Contém apenas 20 funções
- Implementação realizada sobre CORBA
  - Permite a portabilidade para diferentes plataformas
  - Processos de uma mesma aplicação podem ser executados em máquinas de diferentes plataformas

API da BSPlib

- Funções de controle
  - Inicialização e término da execução
    - `bsp_begin(nprocs), bsp_end()`
  - Obtenção de informações sobre a execução
    - `bsp_pid(), bsp_nprocs()`
  - Delimitação dos superpassos
    - `bsp_sync()`
- Funções de comunicação
  - Comunicação DRMA (memória compartilhada)
  - Comunicação BSMP (troca de mensagens)

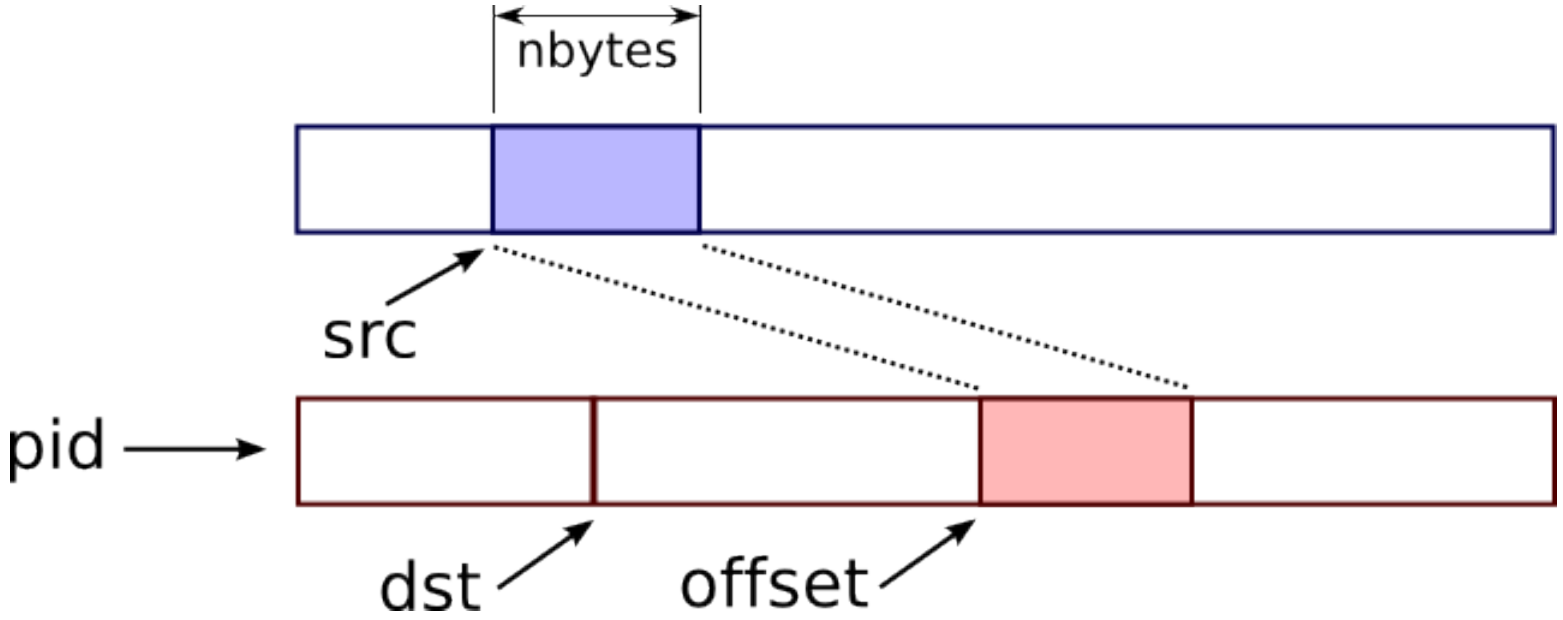
Comunicação DRMA

- `void bsp_push_reg (void *ident, int size)`
  - Registra o compartilhamento de um endereço de memória de tamanho `size` apontado por `ident`.
  - Um endereço registrado pode ser acessado por processos remotos tanto para a escrita como leitura de dados.

- `void bsp_pop_reg (void *ident)`
  - Remove o registro do endereço de memória apontado por `ident`.

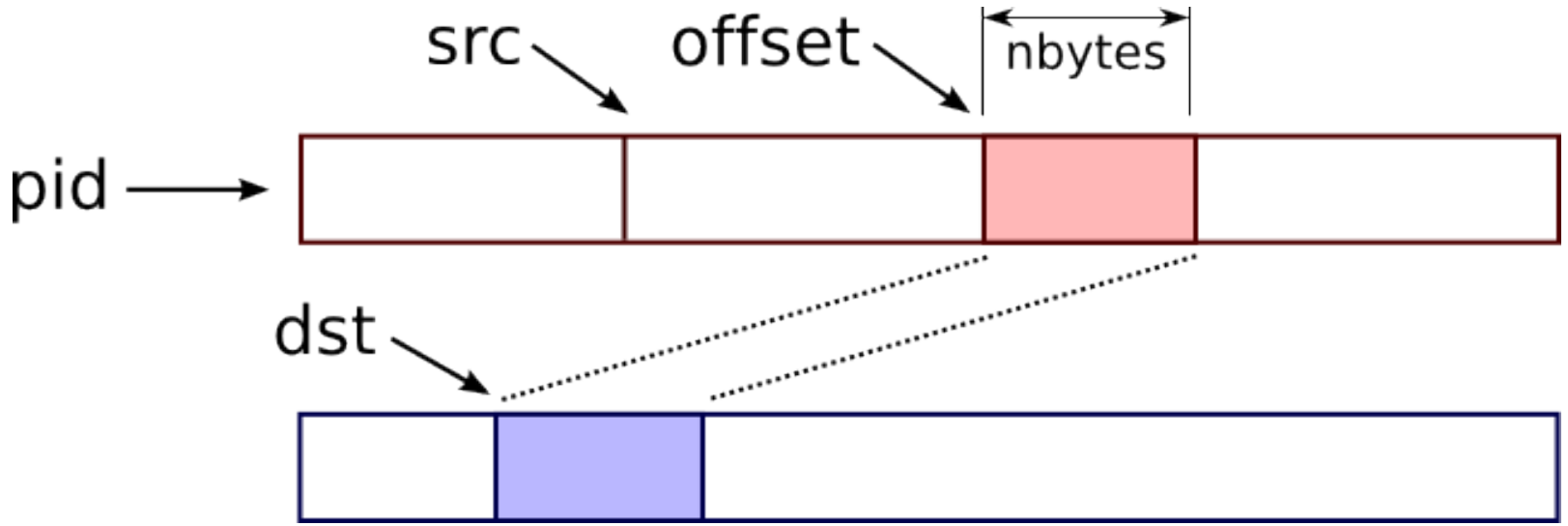
Comunicação DRMA

- `void bsp_put(int pid, void *src, void *dst, int offset, int nbytes)`
  - Escreve `nbytes` contidos no endereço local `src`
  - No endereço remoto `dst` do processo `pid`
  - `offset` descreve o ponto de início da escrita em `dst`



Comunicação DRMA

- `void bsp_get(int pid, void *src, int offset, void *dst, int nbytes)`
  - Lê `nbytes` do endereço remoto `src` do processo `pid`
  - Para endereço local `dst`



Comunicação BSMP

- No modo de comunicação BSMP, cada processo mantém uma fila de mensagens.
  - Processos *remotos* colocam mensagens nesta fila
  - O processo *local* lê mensagens desta fila

- `void bsp_send(int pid, void *tag, void *payload, int nbytes)`
  - Envia uma mensagem de tamanho `nbytes` contida em `payload` e com o identificador `tag` para a fila de mensagens do processo `pid`.
- `void bsp_set_tagsize(int *nbytes)`
  - Define o tamanho da `tag` como `nbytes`.

Comunicação BSMP

- `void bsp_get_tag(int *status, void *tag)`
  - Se houver mensagens na fila, retorna o tamanho da próxima mensagem em `status` e o conteúdo do *tag* da mensagem em `tag`.
- `void bsp_move(void *payload, int nbytes)`
  - Copia o conteúdo da primeira mensagem da fila em `payload`. A variável `nbytes` representa o tamanho do *buffer* apontado por `payload`.
- `void bsp_qsize(int *packets, int *nbytes)`
  - Devolve o número de mensagens na fila em `packets` e o tamanho total das mensagens em `nbytes`.

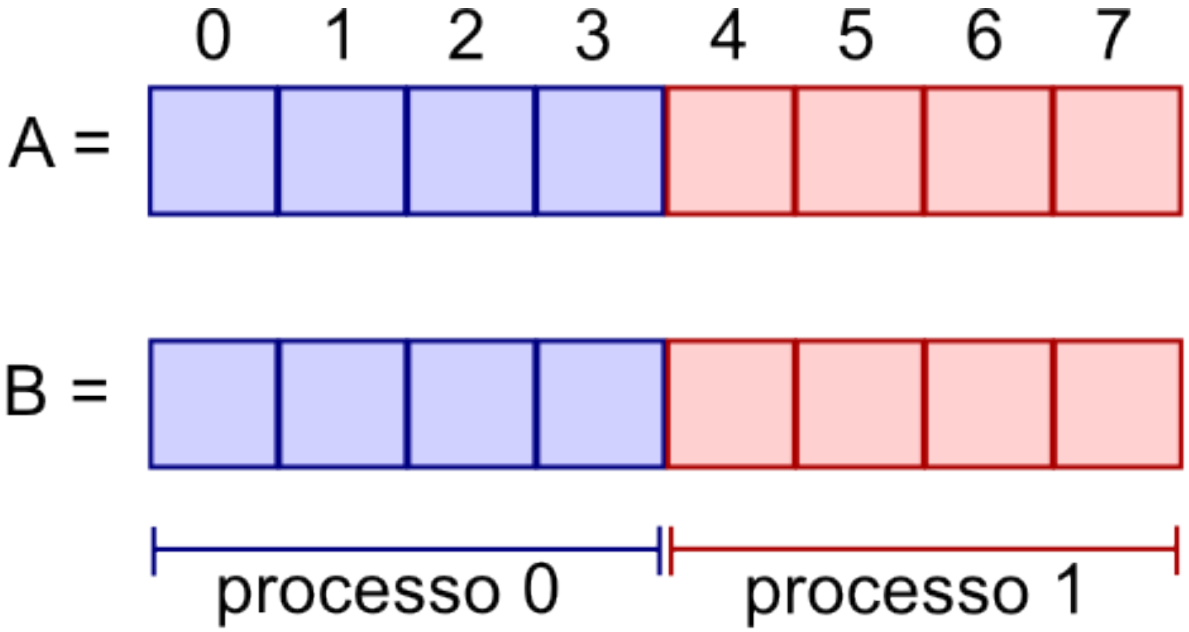
Barreira de Sincronização

- Troca de mensagens (BSMP)
  - As mensagens enviadas do processo A para o processo B ficam disponíveis na fila de mensagem de B apenas após o término do superpasso.

- Memória compartilhada (DRMA)
  - As operações de leitura e escrita em um processo remoto ficam disponíveis somente após a barreira de sincronização
  - Dados ficam armazenados em *buffers* temporários

Exemplo de Aplicação BSP

Produto interno de vetores



produto interno =

a[0]*b[0] + a[1]*b[1] + a[2]*b[2] + a[3]*b[3]

+

a[4]*b[4] + a[5]*b[5] + a[6]*b[6] + a[7]*b[7]

Produto Interno: BSMP (1/2)

```
#include ”BspLib.hpp”

int main(int argc, char **argv) {

  int pid, nprocs = 4;
  double produto;
  double *listaProdutos =
      (double *) malloc(nprocs * sizeof(double));

  bsp_begin( nprocs );
  pid = bsp_pid( );
  produto = calculaProduto( pid, nprocs );

  int tagsize = sizeof(int);
  bsp_set_tagsize( &tagsize );
  for ( int proc = 0; proc < bsp_nprocs( ); proc++)
    bsp_send ( proc, &pid, &produto, sizeof(double) );

  /* continua na proxima pagina */
```

Produto Interno: BSMP (2/2)

```
  /* continuacao */

  bsp_sync ( );

  for ( int proc = 0; proc < bsp_nprocs( ); proc++) {
    int messageSize, source;
    bsp_get_tag( &messageSize, &source )
    bsp_move( &listaProdutos[source], messageSize )
  }

  escreveProduto( listaProdutos );
  bsp_end( );
}
```

Produto Interno: DRMA

```
int main(int argc, char **argv) {

  int pid, nprocs = 4;
  double produto, *listaProd;
  listaProd = (double *)malloc( nprocs*sizeof(double) );

  bsp_begin( nprocs );
  bsp_push_reg( &listaProd, nprocs * sizeof(double) );
  pid = bsp_pid( );

  produto = calculaProduto( pid, nprocs );

  for ( int proc = 0; proc < bsp_nprocs( ); proc++)
    bsp_put(proc, &produto, &listaProd, pid, sizeof(double));
  bsp_sync ( );

  escreveProduto( listaProd );
  bsp_end( );
}
```

Conclusões

- Desenvolver uma aplicação BSP é bastante simples
  - Envolve poucas funções
  - Modelo síncrono evita problemas de concorrência
- Mas possui desvantagens
  - Modelo síncrono gera um desempenho ruim em redes de grande área

Mais informações

- BSP Worldwide
  - http://www.bsp-worldwide.org/
- Sítio do InteGrade
  - http://www.integrade.org.br
- Suporte do InteGrade
  - integrade-support@integrade.org.br