Geração de Grafo de Fluxo de Dados de Programas MPI para o Escalonamento Automático

Guilherme P. Pezzi, Rafael E. Silva, Nicolas Maillard

Instituto de Informática - Universidade Federal do Rio Grande do Sul
{pezzi, resilva, nmaillard}@inf.ufrgs.br

Introdução

Este pôster apresenta a implementação de um protótipo de uma biblioteca de geração de fluxo de dados de programas MPI que será utilizada para o escalonamento automático de processos MPI. O objetivo dessa biblioteca é melhorar o desempenho de aplicações MPI.

No estudo teórico realizado em [SIL 2004], verificou-se a possibilidade de particionar eficientemente grafos de fluxo de dados de algoritmos em fases através de ferramentas de particionamento adicionando pouco custo no tempo de execução do algoritmo. A razão custo de particionamento sobre o volume de dados é $1/n$ e para grandes valores de n o custo de particionamento se torna desprezível.

Com base nisso, foi dado início à implementação do protótipo de uma biblioteca chamada "Biblioteca de Escalonamento de TArefas MPI" (β-MPI), para auxiliar no escalonamento de processos de aplicações MPI [MPI 95].

Justificativa

O objetivo da biblioteca β-MPI é realizar o escalonamento automático de aplicações MPI. Para isso será gerado o grafo de fluxo de dados a partir de uma execução inicial de uma fase da aplicação. Após a geração do grafo, pode-se determinar uma estratégia mais eficiente para o escalonamento dos processos através das informações contidas no grafo para que, nas execuções futuras, o desempenho da aplicação seja melhorado.

A biblioteca utilizada na etapa de geração do grafo de fluxo de dados é a β-MPI. Já para escalonar os processos, será realizado o particionamento do grafo utilizando-se uma biblioteca de particionamento de grafos.

Particionamento de grafos

Segundo [SCH 2000], dado um grafo $G = (V, E)$ o problema do particionamento de grafo é distribuir V vértices e/ou arestas em k subconjuntos distintos chamados subdomínios ou partições, onde cada subdomínio tem aproximadamente o mesmo número de vértices.

Existem diversos algoritmos de particionamento de grafos [SCH 2000], dentre eles destacam-se aqueles que seguem o paradigma multinível. Neste algoritmo, um grafo é

primeiramente reduzido, através do agrupamento de vértices, formando um grafo relacionado menos refinado. Depois, é calculado o particionamento do grafo reduzido. E, por fim, o grafo vai sendo expandido até chegar ao seu nível superior, ou seja, o grafo original. Em cada passo da expansão a partição vai sendo refinada.

Essa técnica tem implementação em duas ferramentas já conhecidas: *Jostle* e *Metis* [KAR 95]. Inicialmente não foi encontrada nenhuma grande diferença entre elas, por isso, optou-se iniciar a implementação utilizando a ferramenta *Metis*.

A biblioteca β-MPI

No caso de um programa MPI, o *rank* do processo é usado como identificador de um vértice. As mensagens entre os processos e o volume de dados transferidos definem as arestas e seus pesos. Uma vez que apenas uma aresta entre dois vértices é permitida, a biblioteca tem que rastrear todas as mensagens trocadas e calcular a soma de todos os dados trocados nelas para construir o grafo.

Sobrecarga das primitivas MPI

Para poder rastrear os dados trocados, foram sobrecarregadas as primitivas do MPI. Em primeiro lugar, é preciso sobrecarregar algumas rotinas do MPI tais como o `MPI_Comm_rank`, o `MPI_Init` ou o `MPI_Finalize`, para poder recuperar o número de processos bem como dados gerais sobre o grafo (*i.e.* o tempo de execução). Além disso, também foram sobrecarregadas as chamadas das primitivas de troca de dados para definir as arestas. A implementação é feita de modo que o código de retorno das chamadas MPI seja preservado.

As primitivas do MPI utilizadas para troca de dados são as seguintes: `MPI_Send`, `MPI_Recv`, `MPI_ISend` e `MPI_IRecv`. Apesar do HPL usar intensamente comunicações globais, ele não chama diretamente as primitivas do MPI tipo `MPI_Bcast`, pois reimplementa-as baseando-se nas comunicações ponto-a-ponto. Isso garante um melhor controle do desempenho dos algoritmos de broadcast no HPL e tem como efeito colateral diminuir o trabalho de sobrecarga.

Tecnicamente, a β-MPI se apresenta como um arquivo `beta-mpi.h` que sobrecarrega as primitivas MPI, usando o precompilador C para chamar as novas versões implementadas na `libbeta-mpi.a`, com a qual será feita a ligação. Na biblioteca, cada primitiva sobrecarregada analisa a mensagem enviada ou recebida, atualiza uma estrutura de dados da biblioteca e, depois, efetua a chamada MPI original para garantir a execução correta do programa.

Na sobrecarga do `MPI_Finalize`, a biblioteca imprime na saída o grafo no formato do *Metis*.

Dois exemplos são o caso do `MPI_Send` e do `MPI_Finalize`. O arquivo de cabeçalho `beta-mpi.h` contém séries de linhas `#define` tipo:

```
#define MPI_SEND(buf, count, datatype, dest, tag, comm) \
     __bMPI_SEND(buf, count, datatype, dest, tag, comm)
```

```
#define MPI_Finalize() __bMPI_Finalize()
```

Já a implementação da biblioteca contém:

```
int __bMPI_Send( void *buf, int count, MPI_Datatype
            datatype, int dest, int tag, MPI_Comm comm ) {
   __add_msg(__rank(), dest, count*sizeof(datatype));
   return MPI_Send(buf, count, datatype, dest, tag, comm);
}
```

A função `__add_msg` atualiza a estrutura de dados interna da biblioteca que terá, neste caso, uma aresta entre o vértice `__rank()` e o vértice `dest`, com peso `count *sizeof(datatype)`. A função `__rank()` retorna o *rank* do processo, armazenado em uma variável inicializada durante a primeira chamada ao `MPI_Comm_rank`.

A função de finalização apenas chama uma função interna `__gatherGraph()` antes de chamar o `MPI_Finalize` normal. A função `__gatherGraph()` varre o grafo para eliminar as arestas paralelas entre 2 vértices e reúne, num único *buffer*, as informações locais de cada processo. Feito isso, então é gerado o grafo correspondente no formato do *Metis*.

Formato do grafo na biblioteca Metis

No *Metis*, os vértices do grafo são numerados de 1 até n e as arestas como listas de vértices, cada um com seu devido peso. Assim, a linha $i = 1 \ldots n$ do arquivo descrevendo o grafo contém uma lista de duplas $(j_1, p_1), (j_2, p_2), \ldots (j_{g_i}, p_{g_i})$, sendo g_i o número de arestas saindo do ou entrando no vértice i, $\{j_1, \ldots, j_{g_i}\}$ e p_j representam as arestas e o peso da aresta (i, j) respectivamente.

No formato do *Metis*, não se pode ter mais de uma aresta ligando dois vértices. De forma que todas as comunicações entre dois processos devem ser agregadas em uma única aresta, que estará representando a soma dos tamanhos de todas as mensagens trocadas.

Utilização da β-MPI

Para compilar uma aplicação MPI utilizando β-MPI, basta incluir `beta-mpi.h` e fazer a ligação com a biblioteca. Para gerar o grafo, basta executá-la normalmente. Uma vez obtido o grafo, é possível particioná-lo com o *Metis* para determinar a melhor alocação dos processos nos processadores disponíveis para uma futura execução.

Inicialmente, o protótipo da β-MPI foi desenvolvido para gerar o grafo do benchmark Linpack (HPL) [PET 2004]. A figura 1 representa o grafo de fluxo de dados gerado pelo HPL, numa configuração de execução com 4 processos. Cabe destacar que este grafo é muito simples para a utilização de um algoritmo de escalonamento e foi gerado apenas para exemplificar o uso da biblioteca.

Conclusão e trabalhos futuros

Com o presente trabalho já é possível gerar o grafo de fluxo de dados de uma aplicação MPI em um formato que pode ser tratado pela biblioteca *Metis*. A validação

```
### graph info begins ###
4   6   1
2   201808  3   271152  4   265464
1   201808  3   236008  4   260512
1   271152  2   236008  4   213696
1   265464  2   260512  3   213696
### graph info ends ###
```

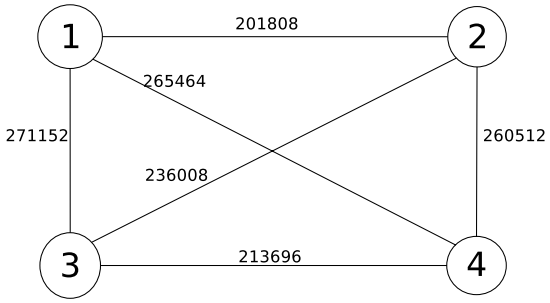

Figura 1: À esquerda está o grafo no formato do arquivo Metis e à direita, a representação do grafo de troca de dados.

prevista consiste em executar o *Linpack* com um grande número de processos, particionar o grafo, alocar os processos e medir o ganho de desempenho. Atualmente, ainda não está feita a integração do *Metis* com o escalonamento de processos, ou seja, falta uma aplicação prática.

Após a integração com a bilioteca de particionamento de grafos, estará implementada uma ferramenta genérica de escalonamento automático de programas MPI. A β-MPI será utilizada futuramente para escalonar os processos dos algoritmos LU [BER 89], gradiente conjugado e FFT [COR 2002].

Referências

[BER 89] BERTSEKAS, D. P.; TSITSIKLIS, J. N. **Parallel and distributed computation – numerical methods**. [S.l.: s.n.], 1989. ISBN 0-13-648759-9.

[COR 2002] SOUZA, W. de (Ed.). **Algoritmos - teoria e prática**. [S.l.]: Editora Campus, 2002.

[KAR 95] KARYPIS, G.; KUMAR, V. **Analysis of multilevel graph partitioning**. Disponível em http://www-users.cs.umn.edu/ karypis/publications/partitioning.html.

[MPI 95] MPI message passing interface standard. Disponível em http://www.mcs.anl.gov/mpi/standard.html.

[PET 2004] PETITET, A. et al. **High-performance linpack benchmark website**. http://www.netlib.org/benchmark/hpl/.

[SCH 2000] DONGARRA, J. et al. (Eds.). **Graph partitioning for high performance scientific simulations**. [S.l.]: Morgan Kaufmann, 2000.

[SIL 2004] SILVA, R. E. **Análise de ferramentas de particionamento de grafos para escalonamento do algoritmo de decomposição lu**.