Proposta de Paralelização do *Skyline Matrix Solver* utilizando MPI

Arthur F. Lorenzon[1], Márcia C. Cera[1], Fábio D. Rossi[2]

[1]Universidade Federal do Pampa (UNIPAMPA) – Campus Alegrete
Av. Tiarajú, 810 – CEP: 97546-550 – Alegrete – RS – Brasil

[2]Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga 6681 – Prédio 32 (PPGCC) – Porto Alegre – RS – Brasil

af.lorenzon@gmail.com, marciacera@unipampa.edu.br, fabio.diniz@acad.pucrs.br

Resumo. *A eficiência de aplicações paralelas em ambientes com memória distribuída depende da forma como o trabalho é dividido entre os nós computacionais. Nesse sentido, propomos um modelo paralelo de distribuição do trabalho entre os processos para a aplicação Skyline Matrix Solver. A implementação paralela baseia-se no padrão MPI-1. Nossos testes alcançaram uma redução de cerca de 90% no tempo de execução com 32 processos.*

1. Introdução

Neste trabalho, propomos a implementação paralela do problema *Skyline Matrix Solver* em MPI-1. Ele faz parte da suíte *Cowichan Problems* [Wilson and Bal 2007] e está sendo estudado dentro do contexto de um projeto de pesquisa que visa a implementação de aplicações com criação dinâmica de processos [Lorenzon et al. 2012]. Nosso objetivo é desenvolver uma versão paralela do *Skyline Matrix Solver* propondo um modelo de distribuição da carga de trabalho entre os processos.

Para tanto, utilizamos o MPI (*Message Passing Interface*) [Gropp et al. 1998] que é a biblioteca padrão para a transmissão de mensagens em arquiteturas paralelas de memória distribuída, especificando primitivas de comunicação ponto a ponto e coletivas. O MPI permite a extração do paralelismo explícito, onde a divisão da carga de trabalho entre os processos fica a cargo do programador. Portanto, a forma com que ela é distribuída entre os processos influencia diretamente na eficiência de uma aplicação.

Este trabalho está organizado como segue: a Seção 2 apresenta os trabalhos relacionados além de contextualizar sobre o problema alvo e o MPI. O modelo de distribuição da carga de trabalho, assim como o algoritmo paralelo são apresentados na Seção 3. Nossos resultados experimentais são apresentados na Seção 4 e a Seção 5 possui uma breve conclusão e nossos trabalhos futuros. Por fim, são apresentadas as referências utilizadas no desenvolvimento deste trabalho.

2. *Skyline Matrix Solver*

O *Skyline Matrix Solver* consiste em resolver a etapa da decomposição de um sistema de equações lineares do tipo $Ax = b$ em matrizes A do tipo *skyline*. Ela consiste de uma matriz $N \times N$ onde existem constantes r_i e c_j, de tal modo que $1 \leq r_i \leq i$

```
1  for(i = 1; i <= N; i++){
2       Cálculo do elemento U_ii
3       for( j = i+1; j < N; j++)
4            Cálculo do elemento U_ij
5       for( j = i+1; j < N; j++)
6            Cálculo do elemento L_ji
7  }
```

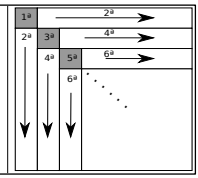

Figura 1. Pseudocódigo sequencial e ilustração das etapas da Decomposição LU.

e $1 \leq c_j \leq_j$. A constante r_i contém a primeira posição da coluna do primeiro valor não nulo da linha i a esquerda da diagonal principal e c_j possui a posição da linha do primeiro valor não nulo da coluna j a direita da diagonal principal [Wilson and Bal 2007].

Segundo Press et al. [Press et al. 2007], um modo muito utilizado para a solução deste tipo de sistema de equações lineares é a Decomposição-*Lower Upper* (Decomposição LU). Nela, a matriz A é o produto de duas matrizes L e U. A matriz L contém os elementos situados abaixo da diagonal principal e a matriz U os elementos da diagonal principal mais os elementos situados acima dela.

Na literatura, Dichter et al. [Dichter et al. 2001] realizam um estudo sobre a paralelização da decomposição da matriz LU em sistemas de memória distribuída. O trabalho paraleliza o método de Gauss utilizando a biblioteca PVM (*Parallel Virtual Machine*). Já Paudel *and* Amaral [Paudel and Amaral 2011] realizam a paralelização do problema *Skyline Matrix Solver* utilizando a linguagem de programação X10. Portanto, nosso trabalho se torna complementar a estes, pois aborda a paralelização do método de *Doolittle* [Press et al. 2007] utilizando o padrão MPI.

3. Implementação Paralela do Algoritmo

Um algoritmo muito utilizado para realizar a Decomposição LU em matrizes *skyline* é o Método de *Doolittle*. Isto porque ele não realiza o cálculo de posições da matriz que possuam valor zero, já que para esta posição o resultado será sempre zero. Seu pseudocódigo na versão sequencial é apresentado à esquerda da Figura 1. Nele, a linha 2 realiza o cálculo do elemento da diagonal principal, definida como a 1ª etapa do método. O valor obtido para esta posição será utilizado para computar a 2ª etapa, que consiste do cálculo dos elementos da linha e coluna (diferentes de zero) da matriz U e L respectivamente (linha 4 e 6). O algoritmo repete este procedimento até a computação total dos elementos. Ainda na Figura 1, a representação ao lado do pseudocódigo ilustra de forma gráfica a ordem de execução das etapas.

A versão paralela do algoritmo foi desenvolvida em MPI-1 num ambiente SPMD (*Single Program, Multiple Data*) utilizando o modelo de programa Fases Paralelas. A distribuição de tarefas entre os processos segue o modelo de divisão igualitária de linhas e colunas e a Figura 2 ilustra como a aplicação está organizada. Nela, o pseudocódigo (situado à esquerda) descreve o algoritmo em alto nível de abstração e a matriz (situada à direita) apresenta a estratégia utilizada na distribuição das tarefas entre 4 processos para uma matriz 8×8.

Para melhor compreender o pseudocódigo apresentado na Figura 2, dividimos

```
1   Cálculo da 1ª linha e coluna
2   for(i = 2; i <= N; i++){
3       Cálculo do elemento U_ii
4       if(rank < numProc/2){
5           for(j = i+1; j < N; j++)
6               Cálculo do elemento U_ij
7       }else{
8           for(j = i+1; j < N; j++)
9               Cálculo do elemento L_ji
10      }
11      Broadcast elementos calculados
12  }
```

	1	2	3	4	5	6	7	8
1								
2			0	0	0	1	1	1
3		2		0	0	1	1	1
4		2	2		0	0	1	1
5		2	2	2		0	1	1
6		3	3	2	2		0	1
7		3	3	3	3	2		0
8		3	3	3	3	3	2	

Figura 2. Pseudocódigo paralelo e ilustração da distribuição das tarefas.

ele em quatro fases: *(i)* Inicialmente, todos os processos (neste caso, 4) calculam a primeira linha e coluna da matriz (linha 1 do pseudocódigo); *(ii)* Todos os processos calculam o elemento da diagonal principal (linha 3); *(iii)* Se o *rank* (identificador) do processo for menor do que a metade do número total de processos, caberá ao processo computar os elementos correspondentes a matriz U (linha 4 a 6). Caso contrário, o processo calculará seus elementos correspondentes à matriz L (linha 8 e 9). Por exemplo, na segunda linha da matriz, o processo 0 calculará os elementos U_{23}, U_{24} e U_{25} e o processo 1 os outros três elementos da linha. No segundo caso, o processo 2 calculará os elementos L_{32}, L_{42} e L_{52} e o processo 3 os outros três elementos da coluna 2; *(iv)* Após o cálculo de seus elementos, cada processo envia seus dados computados aos demais processos através da primitiva de comunicação coletiva `MPI_Bcast` (operação síncrona, onde um processo envia dados para um grupo de processos). As fases *(ii)*, *(iii)* e *(iv)* são repetidas até que todos os elementos da matriz tenham sido computados.

4. Resultados Experimentais

A aplicação foi executada em um *cluster* do tipo *Beowulf* contendo 8 nós computacionais interligados por uma rede *Gigabit Ethernet*. Cada nó estava equipado com um processador Intel *Core i5*, com frequência de 3.20 Ghz. Em cada nó, havia 4 *cores*, sendo 2 físicos e 2 lógicos (através da tecnologia *Hyper-Threading*). O Sistema Operacional em uso foi o Debian (*kernel* 2.6.32-5-amd64) e a versão da distribuição do MPI utilizada foi a OpenMPI 1.6. Para ambas as versões (sequencial e paralela), o tempo computado desconsidera as operações de entrada e saída.

Em nossos testes, o número de processos por nó foi sempre igual a 4, desta forma cada *core* recebeu um processo. Utilizou-se esta configuração visando evitar o desperdício dos recursos computacionais disponíveis. Através da execução prévia dos algoritmos, determinou-se o tamanho da matriz de entrada, fixado em 6000×6000, o que permitiu analisar a escalabilidade da aplicação em nossa arquitetura. Realizou-se 30 execuções para cada quantidade de processos definida, obtendo-se a média, em que o desvio padrão foi sempre inferior a $0,1$ segundos.

A Figura 3 apresenta os tempos de execução obtidos para o intervalo entre 4 a 32 processos e os valores de *speedup* correspondentes. Nela, podemos acompanhar que para todas as quantidades de processos, houve ganho em relação a versão sequencial. Observa-se também, que o maior *speedup* alcançado foi de $10,32$ com redução de cerca de 90% no tempo de execução com a utilização de 32 processos.

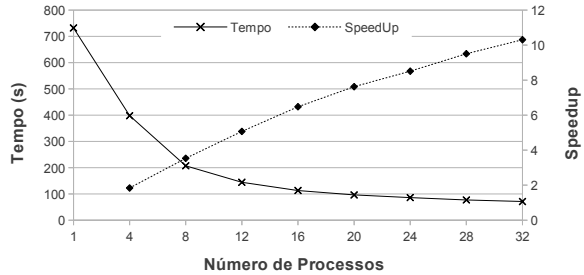

Figura 3. Tempo de execução em segundos (eixo y à esquerda) e speedup (eixo y à direita) para 4 a 32 processos.

Este ganho é alcançado, pois a dependência de dados e a intensa troca de dados existentes são compensadas pelo tempo efetivo de computação (cerca de 70% do total da aplicação).

5. Conclusão e Trabalhos Futuros

Este trabalho objetivou paralelizar o problema *Skyline Matrix Solver*. Para isto, propusemos inicialmente um modelo de distribuição da carga de trabalho entre os processos baseado na distribuição de porções igualitárias de linhas e colunas. Mostramos, que o modelo proposto possibilitou ganhos em relação a versão sequencial.

Identificamos oportunidades futuras de realizar melhorias no escalonamento das tarefas entre os processos, objetivando equilibrar o tempo de computação entre os mesmos. Isto permitirá melhorar a eficiência da aplicação, pois atualmente, os processos que finalizam seu trabalho primeiro ficam bloqueados no `MPI_Bcast` à espera dos demais. Dando seguimento a nossa pesquisa, implementaremos o *Skyline Matrix Solver* com criação dinâmica de processos.

Referências

Dichter, J., Mahmood, A., and Sholl, H. A. (2001). Parallel data distributions for optimum LU decomposition. *I. J. Comput. Appl*, 8(1).

Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., and Snir, M. (1998). *MPI – The Complete Reference*, volume 2, The MPI Extensions, 2nd ed. of *Scientific and Engineering Computation Series*. MIT Press.

Lorenzon, A. F., Cera, M. C., and Rossi, F. D. (2012). Analysing the impact of mpi-2 dynamic process creation to the game of life problem. In *Computer Systems (WSCAD-SSC), 2012 13th Symposium on*, pages 133 –140.

Paudel, J. and Amaral, J. N. (2011). Using the cowichan problems to investigate the programmability of x10 programming system. Technical report, NY, USA.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). Numerical recipes 3rd edition: The art of scientific computing.

Wilson, G. V. and Bal, H. E. (2007). The cowichan experience.