Paralelização do algoritmo Mergesort em um *cluster* de computadores

Jean P. S. Dias[1], Ana P. Canal[1]

[1]Curso de Ciência da Computação – Centro Universitário Franciscano (UNIFRA)
Rua dos Andradas, 1614 – 97.010-032 – Santa Maria – RS – Brazil

jean182@gmail.com, apc@unifra.br

Resumo. *Este artigo descreve a implementação do algoritmo de ordenação Mergesort, para execução paralela no cluster de computadores existente no Centro Universitário Franciscano - UNIFRA. Para tal utilizou-se a biblioteca MPI, responsável pela comunicação entre os diferentes processos.*

1. Introdução

Determinados problemas, ao serem tratados computacionalmente em máquinas seqüenciais, tornam o processamento demorado, pois exigem grande capacidade computacional de memória ou de processamento. Como alternativa, há a computação paralela. A divisão de um dado problema, em problemas de menor complexidade, possibilita que as partes resultantes possam ser executadas em paralelo, ou seja, em processadores diferentes, a fim de diminuir o tempo de execução (DE ROSE, 2006).

Os algoritmos de ordenação têm como objetivo facilitar a busca e a recuperação de determinado dado. Um exemplo de classificação de dados é a ordenação de uma lista telefônica. Se esta lista possuir milhões de números, será longo o tempo gasto para ordená-la, como também o tempo para procurar algum número específico na mesma (ZIVIANI, 2002). A alternativa para este problema é ordenar esta lista telefônica, porém dependendo do tamanho de números existentes na mesma, uma máquina seqüencial se torna ineficaz na resolução, sendo necessária a computação paralela. Em outras palavras em um algoritmo de ordenação seqüencial, conforme aumenta o tamanho da entrada, maior será o tempo de execução para ordenar os dados.

No presente trabalho, foi implementado em paralelo o método de ordenação Mergesort, para a execução no *cluster* de computadores existente no Centro Universitário Franciscano - UNIFRA. Foi utilizada a biblioteca MPI – *Messsage Passing Interface* para a comunicação entre os diferentes processos.

2. Mergesort

O algoritmo Mergesort baseia-se na filosofia de "dividir e conquistar". Este método de ordenação divide recursivamente a seqüência de *n* elementos na metade, gerando duas novas seqüências que são ordenadas, independentemente, em um novo passo recursivo. A partir de toda seqüência dividida em partes, as partes são recombinadas duas a duas, reordenando seus elementos e obtendo a nova seqüência ordenada (CORMEN, 2002).

3. Mergesort Paralelo

O algoritmo Mergesort é um dos métodos de ordenação mais eficientes que se conhece, conforme CORMEN (2002), sua complexidade é *O*(n log n). Na implementação paralela do Mergesort, os dados do tipo *float* são particionados na metade e enviados aos diferentes processos. Após estar completa a divisão de dados, cada processo utilizando a função *mergesort* que ordena recursivamente sua porção de dados, e em seguida estes são recombinados de dois a dois através da função *merge*, gerando a nova seqüência ordenada. O algoritmo de ordenação Mergesort paralelo foi implementado para ser executado em dois, três e quatro processos. A Figura 1 ilustra o funcionamento do Mergesort na execução paralela em dois processos.

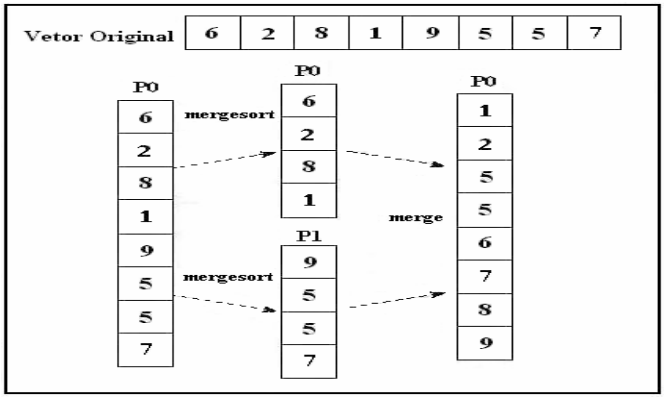

Figura 1: Mergesort Paralelo – 2 Processos

Na execução com dois processos, o processo 0 divide seus dados ao meio e envia a segunda metade para o processo 1. Após ocorre a ordenação em cada processo e a seguir o processo 0 faz a recombinação das duas partes ordenadas, resultando na seqüência final ordenada. Na execução com três processos, o funcionamento ocorre da mesma forma, porém o processo 1 ao receber a sua porção de dados, faz uma nova divisão ao meio, enviando a segunda metade ao processo 2. Depois da divisão completa dos dados, cada processo faz a ordenação dos mesmos. Em seguida, o processo 1 faz a recombinação de seus dados com os do processo 2, e envia esta parte ordenada ao processo 0, que faz a recombinação final, resultando na seqüência ordenada.

Finalmente na execução com quatro processos, o funcionamento segue o mesmo modelo das execuções com dois e três processos, porém um dos processadores, ficará com uma carga maior que os outros dois. O processo 0 divide seus dados e envia ao processo 1, após o processo 0 e o processo 1 dividem seus dados ao meio e enviam ao processo 2 e processo 3 respectivamente. Cada um ordena sua porção de dados. Com os dados ordenados dentro de cada processo, o processo 0 recombina seus dados com os dados do processo 2, e o processo 1 faz a recombinação com os dados do processo 3, gerando assim duas partes ordenadas. O processo 1 envia sua parte ordenada ao processo 0, que faz a recombinação final, gerando a nova seqüência ordenada.

4. Resultados

O ambiente de testes utilizado na obtenção de resultados foi o *cluster* do tipo *Beowulf* do Centro Universitário Franciscano. Este *cluster* é composto por três máquinas *Pentium* IV 2.4 Ghz. O nó principal possui 512 MB de memória RAM, e os demais

possuem 256 MB. Os computadores são interligados através de uma tecnologia *Gigabit Ethernet* nas placas de rede e no *switch*.

Para programação paralela do algoritmo foi utilizada a biblioteca MPI – *Messsage Passing Interface*. Pelo uso de suas primitivas, foi possível realizar a comunicação entre os diferentes processos da aplicação e a alocação destes processos aos processadores do *cluster*. Na execução do algoritmo com até três processos, houve um bom balanceamento da carga do sistema, pois cada processo foi alocado a um processador. Já com quatro processos, um dos processadores obteve um processo a mais que os demais, consequentemente, isto influenciou negativamente no desempenho, como pode ser observado nas medidas do *speedup* e da eficiência.

Após a implementação do algoritmo Mergesort paralelo, foram realizados uma série de testes de tempo de execução do algoritmo seqüencial e paralelo. Calculou-se o *speedup* e a eficiência para a avaliação do desempenho do mesmo, a partir da média de cinco execuções, pois foi observada pouca dispersão nos valores de tempo de execução.

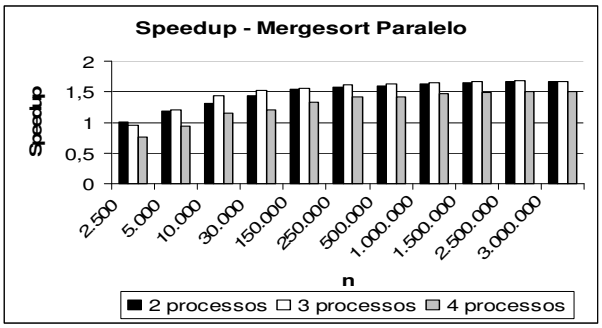

Figura 2: *Speedup* – Mergesort Paralelo

Na Figura 2 está o gráfico relativo ao *speedup* obtido na implementação do método de ordenação Mergesort paralelo, com a variação do tamanho dos dados. Verificou-se que o *speedup* ficou bem abaixo do *speedup* ótimo nas execuções com três e quatro processos, que deveriam ser próximos a 3 e 4, respectivamente. Na execução paralela em dois processos, foi obtido desempenho para todas as entradas testadas. O valor 1,660284 de *speedup* obtido na execução com 2.500.000 de elementos indicou a obtenção de melhor desempenho.

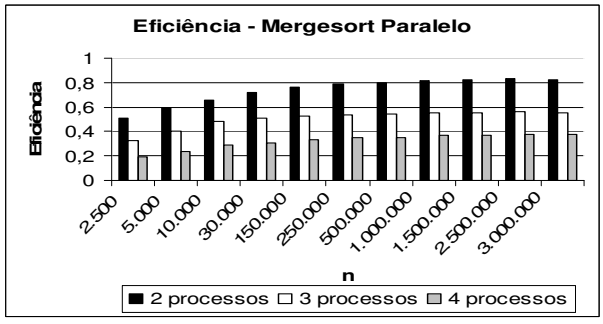

Figura 3: Eficiência – Mergesort Paralelo

Na Figura 3 tem-se o gráfico da eficiência. Verificou-se que com o aumento do número de processos, a eficiência do algoritmo diminui. O método de ordenação

paralelo Mergesort tem a sua melhor eficiência, quando executado com dois processos e uma entrada de dados de tamanho 2.500.000 com o valor de 0,830142, confirmando o ganho de desempenho indicado pelo *speedup*. Apenas com quatro processos, obteve-se baixa eficiência para todos os tamanhos de entrada, mas em termos de tempo de execução, este tempo é menor que o tempo de execução seqüencial, porém conforme a eficiência, não houve um bom aproveitamento dos recursos computacionais nestas execuções.

5. Conclusões e Trabalhos Futuros

A partir deste trabalho, conclui-se que o Mergesort paralelo implementado mostrou um desempenho satisfatório na execução com dois processos. Nos casos com três e quatro processos, constatou-se que com o aumento do número de processos, o desempenho do algoritmo diminui, devido à influência da comunicação na troca de mensagens.

Foi realizada uma comparação com o algoritmo Quicksort também em paralelo implementado, para alguns valores de entrada. O desempenho do Quicksort foi inferior ao Mergesort, devido ao tipo de particionamento dos dados, que devido à escolha do elemento pivô, gera sub-seqüências de tamanho diferentes. Para a paralelização do método de ordenação Mergesort, foram verificados alguns trabalhos relacionados como os desenvolvidos por JUNIOR (2002) e COSTA (2006) e FOSTER(1995).

Foi constatado que a paralelização de métodos de ordenação é uma forma de classificar grande quantidade de dados, porém o uso de diversos elementos de processamento em uma arquitetura paralela, pode não alcançar o desempenho esperado devido à grande comunicação necessária.

Como trabalhos futuros, pretende-se fazer a implementação paralela de outros métodos de ordenação e verificar a escalabilidade dos algoritmos. Também se pretende verificar novas formas de particionamento dos dados e avaliar o desempenho nestas situações.

Referências Bibliográficas

Cormen, T. (2002), Algoritmos: teoria e prática. Rio de Janeiro: Campus.

Costa, L. S.; Sander, L. V. Geração de Arranjos de Sufixos em Paralelo. Disponível em: http://homepages.dcc.ufmg.br/~nivio/cursos/pa06/seminarios/seminario15/seminario15.pdf . Acesso em: 18/01/2008.

De Rose, C. A. F. (2006), Fundamentos de Processamento de Alto Desempenho. In: Caderno dos Cursos Permanentes. CRAD-RS. Porto Alegre: SBC.

Foster, I. Designing and Building Parallel Programs. 1995. Disponível em http://www-unix.mcs.anl.gov/dbpp/text/book.html . Acesso em: 18/01/2008.

Junior, W. F. P. Hiperquicksort: uma análise prática com implementação em MP. Disponível em: http://www.comp.ufla.br/monografias/ano2002/Hiperquicksort_uma_analise_pratica_com_implementacao_em_MP.pdf . Acesso em: 18/01/2008.

Ziviani, N. (2002), Projeto de Algoritmos com implementações em Pascal e C. São Paulo: Pioneira Informática, 2002.