The Bitonic Merge Sort Algorithm Using C and PVM Implementation

Jonas Francisco da Silva[1], Sérgio Francisco Tavares de Oliveira Mendonça[1]

Jones Oliveira Albuquerque[2] - orientador

[1]Departamento de Física e Matemática – Universidade Federal Rural de Pernambuco (UFRPE) - Alunos do Curso de Graduação Licenciatura em Computação

[2]Departamento de Física e Matemática – Universidade Federal Rural de Pernambuco (UFRPE) - Coordenador e Professor do Curso de Graduação Licenciatura em Computação

R. Dom Manoel de Medeiros, s/n - Dois Irmãos

CEP 52.171-900 - Recife/PE - Brazil F: 55 81 3302.1011 - reitoria@ufrpe.br

jonasfsilva@yahoo.com.br (1), sergiomend onca@yahoo.com (1), joa@ufrpe.br (2)

Abstract. *This article describes the implementation of Bitonic Merge Sort algorithm. In general way, a bitonic sequence is a sequence which contains at least a peak and a valley; said in other words, is a sequence that has elements in ascendant order followed by elements in descendant order and vice versa.*

Resumo. *Este artigo descreve a implementação do algoritmo Bitonic Merge Sort. De uma forma geral uma seqüência bitônica é uma seqüência que contém pelo menos um pico e um vale, ou seja, que possua elementos na ordem ascendente seguidos de elementos seguidos na ordem descendentes ou vice-versa. Em outras palavras, seqüências com altos e baixos, ou picos e vales.*

1. Introdução

O grande objetivo da ordenação é classificar os dados de modo a facilitar a localização de qualquer membro para um determinado conjunto de dados. A ordenação de dados ocorre em diversos setores e pode ser encontrada em listas de endereços, bibliotecas, dicionários, catalogações, informações estatísticas anuais, etc.

A técnica de ordenação tem uma maior ocorrência em computadores seriais e através dela podem-se resolver diversos problemas por meio de diferentes algoritmos.

A ordenação tem uma importância adicional para os desenvolvedores de algoritmos paralelos. Ela é freqüentemente utilizada para realizar permutações de dados em computadores de memória distribuída. Estas operações de movimentos de dados podem ser usadas para resolver problemas de teoria dos grafos, computação geométrica e processamento de imagem em tempo próximo ao ótimo [Quinn 1994].

2. Bitonic Merge Sort

O algoritmo Bitonic Merge Sort é a base para algoritmos de tempo de ordenação polilogarítmico de complexidade $O\ (log^2\ n)$ [Batcher 1968]. O seu princípio funcional baseia-se no compara-troca, onde dois elementos são comparados e se estiverem fora de ordem, serão trocados para obter a ordem classificada.

Para se aplicar este algoritmo se faz necessário ter-se uma seqüência bitônica. Se os elementos não formam uma seqüência bitônica esta deverá ser transformada em uma e em seguida esta sequência poderá ser classificada. Um único passo compara-troca pode dividir a seqüência bitônica em duas seqüências bitônicas, confome o Lema: Se *n* é par então *n/2* comparadores são suficientes para transformar uma seqüência bitônica de *n* valores $a_0, a_1, a_2, ..., a_{n-1}$ em duas seqüências bitônicas de valores *n/2* assim esquematizados:

$\min(a_0, a_{n/2})$, $\min(a_1, a_{n/2+1}),\ldots, \min(a_{n/2-1}, a_{n-1})$ e
$\max(a_0, a_{n/2})$, $\max(a_1, a_{n/2+1}),\ldots, \max(a_{n/2-1}, a_{n-1})$ de tal forma que nenhum valor na primeira seqüência é maior que algum valor na segunda seqüência.

3. Usando Paralelismo

O Bitonic Merge (fusão bitônica) sempre compara elementos cujos índices diferem em exatamente um bit. Uma vez que os processadores estão conectados usando o modelo de vetor de processadores no hipercubo e, se seus índices diferem em exatamente um bit, fica fácil implementar o algoritmo **Bitonic Merge Sort** neste modelo. Neste caso os processadores substituem os comparadores. Em vez de rotear pares de elementos por comparadores, os processadores roteam os dados para os processadores adjacentes, onde os elementos são comparados e trocados se necessários.

4. Implementação

BITONIC MERGE SORT (HYPERCUBE PROCESSOR ARRAY)

```
Global    d    (Distance between elements being compared)
Local     a    (One of the elements to be sorted)
          t    (Element retrieved from adjacent processor)
begin
    for i ← 0 to m-1 do
        for j ← i downto 0 do
            d ← 2^j
```

```
                for all P_k where 0     k     2^m - 1 do
                    if k mod 2d < d then
                    t ⇐ [k + d]a  {Get value from adjacent processor}
                        if k mod 2^(i+2) < 2^(i+1) then
                            [k + d]a ⇐ max(t, a) {Sort low to high…}
                            a ← min(t, a)
                        else
                            [k + d]a ⇐ min(t, a) {… or sort high to low}
                            a ← max(t, a)
                        endif
                    endif
                endfor
            endfor
        endfor
end
```

5. Referências Bibliográficas

www.epm.ornl.gov/pvm/pvm_home.html

www.sbc.org.br/reic/edicoes/2001e1/cientificos/Avaliando_o_Bitonic_Merge_Sort_utilizando_PVM.pdf

[Batcher 1968] Batcher, K. E. Sorting Networks and their applications. In proceedings of the AFIPS Spring Joint

[Quinn 1994] Quinn, Michael J. - Parallel Computing: Theory and Practice. McGRAW-HILL, 1994.