Comparação de Estratégias de Paralelização de um Algoritmo Friends-of-Friends com OpenMP

Leonardo Berwian[1], Eric T. Zancanaro[1], Diogo J. Cardoso[1]
Andrea S. Charão[1], Renata S. R. Ruiz[2], Haroldo F. de Campos Velho[2]

[1] Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria

[2]Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais

Resumo. *Neste trabalho, compara-se duas estratégias de paralelização de um algoritmo Friends-of-Friends, usado na análise de grandes conjuntos de dados em Cosmologia. Ambas estratégias de paralelização são voltadas a arquiteturas paralelas com memória compartilhada, usando o padrão OpenMP como ferramenta de programação paralela. Os resultados apontam que o melhor desempenho é obtido com uma estratégia que requer maiores modificações no código-fonte, porém também é possível obter speedup com modificações mínimas.*

1. Introdução

Algoritmos de percolação e agrupamento são muito usados em análises de grandes massas de dados, com o objetivo de identificar grupos de dados que satisfazem determinadas condições. Nesta família de algoritmos, encontram-se aqueles conhecidos pelo termo *Friends-of-Friends*.

A ideia geral de um algoritmo *Friends-of-Friends* (FoF) é identificar grupos aplicando sucessivamente uma regra do tipo "qualquer amigo de meu amigo também é meu amigo". Este tipo de algoritmo é classicamente utilizado em Astrofísica e Cosmologia, para identificar estruturas (galáxias, grupos de galáxias, etc.) em dados observacionais ou simulados, usando a proximidade física como condição [Huchra e Geller 1982].

Os dados de entrada de um algoritmo FoF são geralmente provenientes de grandes bases de dados cosmológicos, que mantém registros de observatórios astronômicos ou de simulações inseridas em pesquisas na área. Considerando esse volume de dados, há um interesse recorrente em soluções de computação de alto desempenho aplicadas a algoritmos FoF.

Neste trabalho, partiu-se de uma implementação sequencial de um algoritmo FoF e investigou-se duas estratégias de paralelização, ambas voltadas para arquiteturas paralelas com memória compartilhada, usando OpenMP.

2. Algoritmo Friends-of-Friends (FoF)

O algoritmo de percolação FoF é um dos métodos mais utilizados em simulações de N-corpos para se determinar estruturas no Universo [Huchra e Geller 1982, Caretta et al. 2008]. Este algoritmo considera uma esfera de raio R ao redor de cada partícula do conjunto total. Se, dentro dessa esfera, existirem outras partículas, elas serão consideradas pertencentes

ao mesmo halo e serão chamadas de amigas. Em seguida, toma-se uma esfera ao redor de cada amiga e continua-se este procedimento usando a regra "qualquer amigo de meu amigo é meu amigo". O procedimento pára quando nenhuma amiga nova pode ser adicionada ao grupo.

Em um trabalho anterior [Ruiz et al. 2009], desenvolveu-se um algoritmo FoF sequencial e implementou-se sua paralelização usando MPI, para execução em *clusters* de computadores e, posteriormente, em grade computacional [Ruiz et al. 2011]. A implementação usando MPI particiona o conjunto de dados e aplica o algoritmo FoF em cada partição, sendo necessária uma etapa de pós-processamento sequencial. No presente trabalho, usou-se como referência a implementação sequencial de [Ruiz et al. 2009] e explorou-se estratégias de paralelização voltadas para arquiteturas *multicore*, usando OpenMP.

3. Paralelização do FoF com OpenMP

A ferramenta OpenMP foi escolhida para este trabalho por ser um padrão voltado para a programação *multithread*, com facilidades para inclusão de seções paralelas em um código já existente.

Analisando-se o programa sequencial, notou-se que a maior parte do tempo de processamento ocorre na função de agrupamento das partículas . Logo, a paralelização do código concentrou-se neste ponto. Por tratar-se de um bloco de código composto por três laços for aninhados (ver Algoritmo 1), é necessária a análise das computações ocorridas dentro de cada um destes laços para garantir a paralelização adequada do processo.

```
for (i = 0; i < N; i++) // Para cada linha do arquivo
{
  k++;
  while (igru[i] != 0) i++;
  igru[i] = k;
  for (j = i; j < N; j++) // Para cada a linha do arquivo a partir da linha i
  {
    if(igru[j] == k)
    {
      for (l = (i + 1); l < N; l++)
      {
        if (igru[l] == 0)
        {
          dist = sqrt((x[j] - x[l])*(x[j] - x[l]) +
                      (y[j] - y[l])*(y[j] - y[l]) + (z[j] - z[l])*(z[j] - z[l]));
          if (dist <= rperc)
          {
            igru[l] = k;
          }
        }
      }
    }
  }
}
```

Algoritmo 1. Trecho de código da versão sequencial

3.1. Estratégia 1: Paralelização do laço mais interno

A primeira estratégia partiu da ideia de paralelizar o algoritmo com o menor número de modificações no código original possível. Portanto, analisando os 3 laços apresentados, o mais interno foi identificado como o mais propício a ser paralelizado, pois não envolve nenhuma dependência de dados.

A paralelização foi realizada com a diretiva `#pragma omp parallel for`, utilizando escalonamento `guided` e criando uma variável LocalDist para garantir a consistência do cálculo da distância pelas *threads*. O escalonamento utilizado é uma variação do escalonamento dinâmico, porém onde os primeiros blocos de trabalho são grandes, enquanto os posteriores vão diminuindo gradativamente de tamanho. Esta estratégia foi escolhida pois o cálculo possui condicionais que o tornam sensível à entrada de dados.

3.2. Estratégia 2: Divisão da entrada e pós-processamento

A segunda estratégia implementada visava uma paralelização em maior granularidade do algoritmo sequencial, através da paralelização do laço mais externo. Para tal, dividiu-se a entrada em N partições, distribuindo-as para execução simultânea pelas *threads* criadas pelas diretivas OpenMP. Esta estratégia resulta em N coleções de grupos, as quais devem ser unificadas através de um pós-processamento.

O bloco de código executado por cada *thread* é análogo ao âmago do algoritmo sequencial: três laços de repetição onde as partículas são testadas em relação a sua proximidade (dado um raio R). De forma a auxiliar a operação de pós-processamento, foi criada uma classe para representar um grupo, classe esta que contém uma lista com os identificadores das partículas pertencentes, um identificador próprio e um valor com os limites inferiores e superiores de cada uma das três coordenadas do espaço.

Estes valores de limite representam um cubo no espaço, dentro do qual as partículas do grupo existem. Esta representação do espaço permite a comparação entre os grupos sem entrar no mérito da posição individual de cada partícula.

A operação de pós-processamento é realizada de forma sequencial e consiste na análise dos "cubos" de cada grupo, buscando detectar grupos com possíveis intersecções no espaço, considerando ainda o valor do raio de percolação. Caso exista intersecção neste espaço ocupado pelos dois grupos, é possível que exista uma conexão entre os mesmos, sendo necessário analisar as partículas de ambos. Detectada uma conexão entre dois grupos, deve-se unificá-los e atualizar os limites do espaço do novo grupo de forma a refletir o espaço ocupado pela união dos dois grupos.

4. Experimentos e Resultados

Realizou-se experimentos visando avaliar o desempenho das estratégias de paralelização implementadas. Para isso, utilizou-se um servidor com um processador *quad-core* Intel Xeon E5620 2.4 GHz, sendo cada *core* com 2 *threads* (HT), possuindo 12 GB de RAM DDR3. O sistema operacional é Debian 8 e usou-se o compilador GCC 4.6.

Para cada implementação paralela, variou-se o número de *threads* configurando-se a variável de ambiente OMP_NUM_THREADS em 2, 4 e 8. Repetiu-se cada experimento 30 vezes, medindo-se o tempo total de execução e calculando-se a média dos tempos. Utilizou-se o mesmo conjunto de dados de entrada para todas as versões, totalizando 50.000 partículas, com raio de percolação igual a 1. Para esses dados de entrada, o tempo de execução sequencial é de 29,5 segundos. Essa medida foi usada como referência para calcular o *speedup* obtido em cada estratégia.

Na tabela 1, apresentam-se os resultados obtidos. Com a primeira estratégia, houve redução dos tempos de execução, como esperado, porém o *speedup* obtido ficou aquém do

ideal. Isso pode ser explicado pelo desbalanceamento de carga no laço mais interno, que é sensível aos dados de entrada. Além disso, a execução se alterna muitas vezes entre uma região sequencial e as regiões paralelas (modelo *fork-join* do OpenMP).

Com a segunda estratégia, obteve-se um *speedup* superlinear, o que inspira uma investigação mais aprofundada. Essa versão também é sensível ao desbalanceamento de carga, porém há somente um *fork-join*. Nas regiões paralelas, a granularidade é maior do que na estratégia 1. De forma geral, esse resultado é coerente com os resultados obtidos em um trabalho relacionado [Ruiz et al. 2009], em que também se verificou um *speedup* superlinear, explicado pela redução do espaço de busca. Nesse trabalho anterior, no entanto, a região sequencial responsável pelo pós-processamento dependia de troca de mensagens, o que não ocorre na implementação com OpenMP.

Tabela 1. Tempos de execução e speedup de cada estratégia

Número de Threads	Estratégia 1		Estratégia 2	
	Tempo de execução (s)	Speedup	Tempo de execução (s)	Speedup
2	18,16	1,62	11,6	2,5
4	10,21	2,88	5,1	5,7
8	9,40	3,14	3,9	7,5

5. Considerações Finais

Os resultados obtidos neste trabalho mostram que é possível obter diferentes patamares de desempenho com a paralelização do algoritmo FoF em questão, usando OpenMP. Os resultados apontam que o melhor desempenho é obtido com uma estratégia que requer maiores modificações no código-fonte, porém também é possível obter *speedup* com modificações mínimas. Esta investigação ainda precisa ser aprofundada, pois utilizou-se somente parte de um conjunto de dados. Novos conjuntos precisam ser extraídos de bases de dados e convertidos para o formato de entrada utilizado nos programas.

Referências

Caretta, C. A., Rosa, R. R., de Campos Velho, H. F., Ramos, F. M., e Makler, M. (2008). Evidence of turbulence-like universality in the formation of galaxy-sized dark matter haloes. *Astronomy & Astrophysics*, 487(2):445–451.

Huchra, J. P. e Geller, M. J. (1982). Groups of galaxies. I - Nearby groups. *Astrophysical Journal*, 257:423–437.

Ruiz, R. S. R., Campos Velho, H. F., Caretta, C., A., Charão A., S., e Souto R., P. (2011). Grid Environment for Turbulent Dynamics in Cosmology. *Journal of Computacional Interdisciplinary Sciences*, 2:87.

Ruiz, R. S. R., Campos Velho, H. F., e Caretta, C. A. (2009). Parallel algorithm friends-of-friends to identify galaxies and cluster of galaxies for dark matter halos. In *Proceedings...* Workshop dos Cursos de Computação Aplicada do INPE, 9. (WORCAP)., Instituto Nacional de Pesquisas Espaciais (INPE).