Análise do Software Paralelo AHF para Identificação de Estruturas em Cosmologia

Ana Luísa V. Solórzano[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. *O estudo da formação de estruturas do Universo a partir de dados cosmológicos suscita a criação de programas paralelos. Um deles é o AHF, que é paralelizado usando MPI e OpenMP. Neste trabalho, investiga-se o seu comportamento em execuções com OpenMP em uma arquitetura multicore. Para o caso considerado, não houve vantagem de desempenho com o uso de OpenMP, o que motiva uma avaliação mais aprofundada de sua paralelização.*

1. Introdução

Análises de grandes quantidades de dados simulados e observacionais têm sido amplamente empregadas em Cosmologia, que estuda a origem, estrutura e evolução do Universo. Um dos aspectos investigados nesta área é a formação de estruturas cósmicas, tais como, estrelas, galáxias e aglomerados de galáxias [Madsen 1996]. Para isso, dado um grande conjunto de partículas proveniente de simulações ou observações, é necessário identificar e classificar grupos de partículas interligadas. Isso é conhecido como *halo finding* [Gill et al. 2004].

O interesse da comunidade científica, aliado à evolução de algoritmos e ferramentas computacionais, fez com que surgissem diversos algoritmos e programas chamados *halo finders*. Este tipo de *software*, muitas vezes, é desenvolvido por especialistas em Cosmologia com conhecimento em programação. Do ponto de vista cosmológico, essa variedade de códigos tem suscitado questões sobre semelhanças e diferenças de seus resultados [Knebe et al. 2013]. Do ponto de vista computacional, observa-se que muitos desses códigos empregam ferramentas de programação como OpenMP [Chapman et al. 2007] ou MPI [Pacheco 1996], porém não é comum encontrar análises sobre seu desempenho em arquiteturas paralelas.

Neste trabalho, analisa-se o desempenho do *software* Amiga Halo Finder (AHF), que é um dos códigos citados por especialistas na área que proporciona a sua paralelização com OpenMP e MPI. Nesta pesquisa exploratória, tem-se como objetivo geral compreender o comportamento do AHF em uma arquitetura paralela *multicore*, usando métricas e critérios não observados em publicações sobre este código.

2. AHF – Amiga Halo Finder

O Amiga Halo Finder é um *software* para encontrar estruturas de *halos* e *sub-halos* em simulações cosmológicas, a partir de propriedades definidas pelo usuário, realizando

execuções seriais ou paralelas [Knebe e Knollmann 2009]. O seu fluxo de trabalho consiste em encontrar as estruturas, identificar partículas relacionadas a elas, remover as não relacionadas e calcular suas propriedades físicas e estatísticas.

Para isso, o programa utiliza um arquivo de texto intitulado `AHF.input`, que é passado por parâmetro em sua execução para definir propriedades quanto à classificação das partículas, e um arquivo de dados proveniente de simulações cosmológicas com ferramentas como Gadget-2[1], AMIGA[2], e outras desde que no fomato ASCII. Para estas, o arquivo de dados deve conter um cabeçalho com seus parâmetros cosmológicos, que pode ser inserido utilizando um *script* em Shell disponibilizado pelo *software*. Entretanto, como o *script* insere alguns valores padrões relacionados a esse tipo de simulação, é necessário configurá-los no próprio código para resultados mais precisos.

Cada execução gera arquivos de saída contendo informações diversas, sendo o principal deles um arquivo de log que relata os valores de entrada, o número de *halos* encontrados, o tempo de execução do algoritmo *halo finder*, o tempo de execução total do programa e o consumo de memória. Os demais arquivos listam os parâmetros utilizados, as propriedades de cada *halo* e as partículas associadas a ele.

Diferente de outros *halo finders* que predefinem os grupos de partículas a serem processados em paralelo, o AHF divide o volume de dados entre os processadores durante a sua execução [Kim e Park 2006, Knebe e Knollmann 2009]. Em suas primeiras versões, essa paralelização era feita com o padrão MPI, já a versão atual também disponibiliza a opção com OpenMP e MPI com OpenMP, definida por *flags* de compilação.

Em um trabalho anterior, seus autores reportam o desempenho de execuções paralelas com MPI considerando diferentes valores para o balanceamento de carga entre as máquinas e para o parâmetro que define as zonas de fronteira entre partículas [Knebe e Knollmann 2009]. Entretanto, trabalhos que relatem o desempenho da execução paralela para as novas opções não foram encontrados.

3. Experimentos

Para obter o arquivo de dados observacionais, utilizou-se uma amostra do Consórcio Virgo, proveniente do modelo cosmológico LCDM (Lambda Cold Dark Matter) do projeto de Simulações de Escala Intermediária. A amostra define 1.048.576 partículas, cada uma com massa de 6.86 x 10^{10} Msun/h e parâmetros cosmológicos (Ω_M , Ω_Λ , h, σ_8) = (0.3, 0.7, 0.7, 0.9). Os dados, bem como os seus parâmetros numéricos e cosmológicos, encontram-se disponíveis em: `http://wwwmpa.mpa-garching.mpg.de/Virgo/data_download.html`.

Um ambiente propício para pesquisadores em cosmologia realizarem investigações preliminares sobre o desempenho de seus algoritmos é o de arquiteturas paralelas *multicore*. Esse ambiente é mais comum e acessível em comparação ao uso de *clusters*, que podem gerar filas de espera, por exemplo. Assim, optou-se por realizar 30 execuções do programa para 1, 2 e 4 *threads* com o padrão OpenMP.

A máquina utilizada contém um processador Intel® Core i5-5200U de dois cores físicos e quatro virtuais, com 32KB de cache L1, 256KB de cache L2 e 3MB de cache L3,

[1]http://wwwmpa.mpa-garching.mpg.de/gadget/
[2]http://popia.ft.uam.es/AMIGA/

e 4 GB de memória. Sistema operacional Ubuntu 16.04.3 LTS, versão do Linux 4.4.0-103, e compilador gcc versão 5.4.0.

Visando aprofundar a compreensão do código e de sua execução paralela, foram utilizadas duas ferramentas. A primeira foi a Understand[3], que analisa códigos fonte de projetos e oferece uma API para a investigação, edição e visualização de gráficos sobre o projeto. A segunda foi a Intel VTune Amplifier[4], uma ferramenta da Intel para analisar o perfil de execução de códigos, de modo a identificar gargalos quanto ao uso da CPU em execuções *multithreading*.

4. Resultados e Discussão

A execução dos experimentos resultou nos dados observados na Tabela 1. Os tempos calculados foram obtidos no arquivo de saída de log do programa, o qual relata separadamente o tempo do *halo finder*, que consiste na identificação das estruturas cosmológicas, e o tempo total, que também engloba a manipulação do arquivo de dados e outras computações relacionadas às propriedades das partículas. Visando legitimar esses valores, eles foram comparados a execuções utilizando o comando `time` do Unix.

Threads	Média(s) Tempo *halo finder*	Média(s) Tempo Total
1	24.70	66.71
2	24.56	66.67
4	24.53	66.74

Tabela 1. Médias dos tempos de execução

A partir desses resultados, realizaram-se execuções com o Intel Vtune, onde se observou que o programa faz pouco uso do processamento paralelo durante a sua execução, justificando o desempenho observado. Partindo disso, investigou-se quais funções do AHF requerem um maior tempo de processamento e como as diretivas OpenMP são aplicadas.

4.1. Análise da arquitetura do AHF

Com o Intel Vtune, constatou-se que o maior processamento está na preparação dos dados a serem identificados pelo algoritmo de *halo finding*. Nessa etapa, é aplicada a técnica de AMR (Adaptative Mesh Refinement) para classificar as partículas hierarquicamente em uma estrutura de árvore. Essa é a árvore que será percorrida pelo algoritmo, para definir as subestruturas de *halos* e *sub-halos*.

Com o Understand, analisaram-se as funções que fazem o refinamento dos dados. A mais importante delas é a `AMR_hierarchy`, que recursivamente prepara os dados a serem identificados e invoca a função mais custosa do programa, que refina os dados a partir das densidades das partículas e do número de partículas permitidas por nó da árvore, cujo valor é definido pelo usuário no arquivo `AHF.input`. Essa etapa do programa não faz uso de processamento paralelo.

[3]https://scitools.com/

[4]https://software.intel.com/en-us/intel-vtune-amplifier-xe

4.2. Análise da paralelização do AHF

As diretivas OpenMP são aplicadas em laços utilizados nos arquivos de escrita dos dados de saída e no arquivo responsável por percorrer a árvore. Porém, a maioria dos laços paralelizados processam cargas estáticas e têm grande parte das variáveis utilizadas definidas como privadas, o que interfere no desempenho já que resulta em cópias de variáveis entre as *threads* e pode resultar em *threads* ociosas, dependendo do tamanho de carga a ser distribuída.

Porém, mesmo executando de forma serial, o AHF apresenta um bom desempenho em comparação a outras ferramentas, conforme relatado em [Knebe et al. 2013]: dentre oito ferramentas correlatas, o AHF teve o segundo menor tempo de execução para dois arquivos de dados com tamanhos distintos, sendo até 84 vezes mais rápido comparado ao de pior desempenho.

5. Considerações Finais

Com base nas análises sobre o *software* paralelo AHF em um ambiente *multicore* utilizando OpenMP, observou-se que o programa não apresenta vantagens de desempenho em execuções com 2 e 4 *threads* em comparação à execução serial, para os dados analisados. Estima-se que isso ocorra devido aos autores do trabalho priorizarem o seu bom funcionamento e completude em relação a outros *halo finders*, oferecendo como fator adicional a possibilidade de sua execução em ambientes paralelos. Assim, como trabalhos futuros, planeja-se estudar o código fonte do programa a fim de rever e alterar a forma de paralelização para o padrão OpenMP em busca de um maior desempenho, além de realizar testes com MPI.

6. Agradecimentos

Os autores agradecem ao PIBIC-CNPq-INPE pelo suporte na forma de bolsa de Iniciação Científica para o projeto de nº 800012/2016-0.

Referências

Chapman, B., Jost, G., e van der Pas, R. (2007). *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, USA.

Gill, S. P. D., Knebe, A., e Gibson, B. K. (2004). The evolution of substructure - i. a new identification method. *mnras*, 351:399–409.

Kim, J. e Park, C. (2006). A new halo-finding method for n-body simulations. *ApJ*, 639:600–616.

Knebe, A. et al. (2013). Structure finding in cosmological simulations: the state of affairs. *Monthly Notices of the Royal Astronomical Society*, 435(2):1618–1658.

Knebe, A. e Knollmann, S. R. (2009). Ahf: Amiga's halo finder. *The Astrophysical Journal Supplement*, 182(2):608–624.

Madsen, M. S. (1996). In *The Dynamic Cosmos - Exploring the Physical Evolution of the Universe*, New York, NY, USA. Chapman & Hall.

Pacheco, P. S. (1996). *Parallel Programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.