MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

PLATAFORMA WEB PARA EXPERIMENTOS COM ALGORITMO FRIENDS-OF-FRIENDS PARALELO HÍBRIDO PARA CLASSIFICAÇÃO DE OBJETOS ASTRONÔMICOS

Ana Luisa Veroneze Solórzano (UFSM, Bolsista PIBIC/CNPq)
Haroldo de Campos Velho (LABAC/COCTE/INPE, Orientador)
Andrea Schwertner Charão (Informática-UFSM, Orientadora)

Relatório Final de Projeto de Iniciação científica (PIBIC/CNPq/INPE).

URL do documento original:
<http://urlib.net/xx/yy>

INPE
São José dos Campos
2018

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6923/6921
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

PLATAFORMA WEB PARA EXPERIMENTOS COM ALGORITMO FRIENDS-OF-FRIENDS PARALELO HÍBRIDO PARA CLASSIFICAÇÃO DE OBJETOS ASTRONÔMICOS

Ana Luisa Veroneze Solórzano (UFSM, Bolsista PIBIC/CNPq)
Haroldo de Campos Velho (LABAC/COCTE/INPE, Orientador)
Andrea Schwertner Charão (Informática-UFSM, Orientadora)

Relatório Final de Projeto de Iniciação científica (PIBIC/CNPq/INPE).

URL do documento original:
<http://urlib.net/xx/yy>

INPE
São José dos Campos
2018

RESUMO

Este trabalho tem como objetivo continuar o desenvolvimento de uma plataforma Web multiusuário, que permita a configuração e a execução remota de experimentos de classificação de objetos astronômicos sobre uma arquitetura paralela híbrida (*multicore*/GPU), com o algoritmo Friends-of-Friends (FoF). Para isso, foram realizados estudos e execuções para a compreensão da plataforma e para a paralelização do FoF, inicialmente para execução em CPU e posteriormente para execuções em GPU. Os métodos aplicados, bem como resultados e pesquisas sobre assuntos correlatos, são apresentados neste relatório.

Palavras-chave: Friends-of-Friends. Computação Híbrida. Plataforma Web. Halo Finder.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

FoF	–	Friends-of-Friends
AHF	–	Amiga Halo Finder
UFSM	–	Universidade Federal de Santa Maria
MPI	–	Message-Passing-Interface
OpenMP	–	Open Multi-Processing
OpenACC	–	for Open Accelerators
GPU	–	Graphics Processing Unit
CPU	–	Central Processing Unit
RAM	–	Random Access Memory

SUMÁRIO

Pág.

1 OBJETIVOS DO TRABALHO

Os objetivos iniciais deste projeto consistem na utilização de uma plataforma Web para a execução remota de experimentos com o algoritmo Friends-of-Friends (FoF) paralelo. Para isso, pretende-se partir de um portal Web anteriormente implementado e realizar otimizações em sua estrutura, de modo que, ao final, seja possível hospedá-lo em uma arquitetura paralela híbrida composta por processador *multicore* e GPU.

A primeira etapa do trabalho exige estudos preparatórios sobre o algoritmo Friends-of-Friends em suas implementações de complexidade O(n^2) e O(nlogn) e versões paralelas com as bibliotecas MPI e OpenMP. Além disso, são necessários estudos sobre conceitos e técnicas fundamentais de programação paralela e arquiteturas paralelas híbridas, de modo a optar pelas melhores escolhas de paralelização.

A segunda etapa consiste na continuação de uma plataforma Web para a execução de experimentos com o algoritmo FoF implementada anteriormente utilizando o *framework* Django. Esta etapa exige estudos sobre desenvolvimento Web, em especial sobre Django, e *softwares* relacionados à implementação de uma aplicação remota, como Celery e Redis, aprofundamento na linguagem de programação Python, utilizada com Django, e demais estudos sobre as funcionalidades implementadas e possíveis melhorias.

A terceira etapa consiste em adaptar o código do FoF para execução em um ambiente de computação híbrida. Para isso, deve-se estudar a paralelização do algoritmo utilizando o padrão OpenACC, primeiramente buscando mínima modificação possível do código original do FoF versão O(n^2) inserindo as diretivas para compilação com OpenACC.

Por fim, pretende-se hospedar a plataforma no cluster Lacibrido do LAC/INPE para as execuções remotas de todas as versões do FoF. As execuções irão coletar métricas de execução e avaliar o desempenho das execuções realizadas por um mesmo usuá-

rio. A partir dessas informações, espera-se suscitar análises sobre as possibilidades de novas implementações do algoritmo ou de utilização de novas tecnologias Web para implementação da plataforma. Ao final, ela será disponibilizada para a comunidade de astronomia e astrofísica, uma ação realizada junto ao Instituto Nacional de Ciência e Tecnologia de Astrofísica (INCT-A, CNPq).

2 INTRODUÇÃO

Análises de grandes quantidades de dados simulados e observacionais têm sido amplamente empregadas em Cosmologia, responsável por estudar a origem, estrutura e evolução do Universo. Um dos aspectos investigados nesta área é a formação de estruturas cósmicas, como estrelas, galáxias e aglomerados de galáxias (Madsen et al., 1996). Para isso, é preciso classificar um grande conjunto de partículas provenientes de simulações ou observações, identificando grupos de partículas interligadas.

O interesse da comunidade científica, aliado à evolução de algoritmos e ferramentas computacionais fez com que surgissem diversos algoritmos e programas para executar essa tarefa. Muitas vezes, esses *softwares* são desenvolvidos por especialistas em Cosmologia com conhecimento em programação, e por isso acabam não priorizando o desempenho do algoritmo, sendo extremamente custoso para processar um grande volume de dados.

Um desses algoritmos é o Friends-of-Friends (Huchra and Geller, 1982; CARETTA et al., 2008), que apesar de eficiente para a sua tarefa, não é indicado para o processamento de muitos dados em sua versão sequencial. Visto que simulações completas podem ter informações sobre milhões de partículas, o desenvolvimento de programas eficientes e com um baixo tempo de execução é primordial.

A utilização de ambientes de computação composto por CPUs *multicore* é uma das maneiras de minimizar esse alto tempo de execução. Para isso, os algoritmos devem ser paralelizados de forma a explorar ao máximo o uso dos nodos computacionais disponíveis na máquina. Além disso, o uso de computação híbrida, composta de uma CPU e um dispositivo acelerador, vem demonstrando ser uma opção vantajosa para resolver problemas que lidam com amplas quantidades de dados, sendo a GPU uma plataforma de aceleração bastante utilizada.

A escolha dos usuários sobre qual *software* optar para a classificação de partículas

pode ser desafiadora, visto que cada um adota sua própria documentação e muitas vezes têm acesso restrito. Esse cenário motiva o uso de plataformas Web para a execução de algoritmos sequenciais e de alto desempenho de forma remota e de fácil acesso à comunidade científica.

Os capítulos seguintes deste relatório apresentam a fundamentação teórica (Capítulo 3) os métodos e materiais utilizados durante a pesquisa (Capítulo 4) para alcançar os objetivos descritos no Capítulo 1 além de outros objetivos não incluídos como iniciais, mas que agregaram ao trabalho. No Capítulo 5 são apresentadas as análises e resultados obtidos até o momento, e o Capítulo 6 apresenta as conclusões.

3 FUNDAMENTAÇÃO TEÓRICA

A área de computação de alto desempenho explora principalmente o uso de CPUs e GPUs para o processamento de programas que lidam com uma grande quantidade de dados. Inclusive, em muitos casos, o elevado custo computacional para execução de um algoritmo pode ser fator limitante para que sejam realizadas execuções contínuas ou sobre um volume de dados completo.

Um cenário que lida com problemas de grande envergadura é a análise computacional de dados astronômicos provenientes de simulações cosmológicas e de observatórios virtuais. Considerando sua importância para, por exemplo, prever comportamentos similares na formação de grandes estruturas observadas no Universo, como galáxias e aglomerados de galáxias (Ruiz, 2009), o desenvolvimento de programas eficientes e com um baixo tempo de execução é primordial.

Um dos algoritmos mais utilizados em simulações para classificar objetos astronômicos é o Friends-of-Friends (FoF), que identifica quais objetos estão em interação gravitacional (CARETTA et al., 2008; Huchra and Geller, 1982). Para isso, utilizam-se dados provenientes de simulações cosmológicas, ou de imagens de observações astronômicas que contém dados cosmológicos sobre cada objeto.

A classificação é realizada considerando-se um cenário de N-corpos (objetos ou partículas) em atividade gravitacional, e um raio de interação gravitacional (raio de percolação definido pelo usuário. Partículas presentes à esfera definida pelo raio são consideradas "amigas"e classificadas em um mesmo grupo. Da mesma maneira, seguindo a regra de que "qualquer partícula amiga de minha amiga também é minha amiga"termina-se a classificação, e o algoritmo apresenta ao final o número de grupos

encontrados.

O primeiro trabalho de paralelização do FoF sugeriu a utilização da biblioteca MPI (Message-Passing-Interface) (Pacheco, 1997), que permite a troca de mensagens entre os processadores. Neste trabalho, Ruiz et al. (Ruiz, 2009) definiram um processador mestre para fazer a leitura do arquivo de entrada e balancear a carga, que seria distribuída entre os processadores utilizando rotinas da biblioteca MPI. Cada processador faria o processamento de seu grupo de partículas utilizando o FoF e ao final retornaria os grupos identificados ao processador mestre, que realizaria um pós-processamento para verificar partículas de um mesmo grupo que poderiam ter executado em processadores distintos. Por fim ele também geraria um arquivo de saída.

O segundo trabalho de paralelização do FoF sugeriu a utilização da biblioteca OpenMP (BERWIAN et al., 2017). O OpenMP é uma biblioteca para a exploração do paralelismo através do uso de diretivas de compilação, permitindo executar uma aplicação em vários fluxos e após, se necessário, novamente em um único fluxo (Chapman, 2007). Neste trabalho, os autores abordaram duas estratégias de paralelização em laços de repetição distintos, porém na mesma porção de código, responsável por agrupar as partículas.

Apesar de trazerem desempenho, ambas versões paralelas com OpenMP enfrentaram alguns desafios devido a lógica do algoritmo. O primeiro é o fato de que a execução do FoF é completamente sensível aos dados de entrada, visto que cada simulação encontrará partículas em diferentes posições do ambiente analisado, e com isso o agrupamento delas será distinto para cada arquivo de entrada. Outro desafio foi devido às estruturas condicionais presentes na porção de código que faz o agrupamento, um fato limitante para a paralelização do código.

Além disso, em 2015, Madalosso et. al propuseram uma nova implementação do algoritmo FoF de complexidade O(nlogn) (MADALOSSO et al., 2015). Essa versão foi implementada com base no algoritmo de Barnes-Hut (BARNES and HUT, 1986) que implementa um método de estrutura de dados hierárquica conhecido como Octree (WIKIPEDIA,). Nesse método, os N-corpos de entrada foram alocados de acordo com suas posições em um espaço físico tridimensional (um cubo) e a partir disso foram feitas subdivisões recursivas do sólido em 8 quadrantes utilizando uma estrutura de árvore, com 8 ponteiros para cada nó (um para cada quadrante). Cada nó da árvore pôde ser classificado como "cheio", "vazio"ou "parcialmente preenchido". Mesmo diminuindo a complexidade do FoF, os autores citam limitações como a possibilidade

de *overhead* nas estruturas utilizadas dependendo do tamanho da amostra de dados.

4 MÉTODOS E MATERIAIS

Esta seção descreve as etapas de pesquisa realizadas, bem como os materiais e métodos utilizados. Na seção 4.1 é apresentado o algoritmo Friends-of-Friends, que exigiu estudos sobre as suas diferentes implementações e análises quanto ao seus desempenhos. Na seção 4.2 é apresentado o trabalho realizado sobre a plataforma Web, abordando estudos sobre a sua implementação local, melhorias e métodos utilizados e desafios sobre a hospedagem da plataforma em um ambiente de computação em nuvem. Em 4.3, apresenta-se um estudo sobre um *software* correlato ao FoF e, por fim, em 4.4 as estratégias de paralelização do FoF com OpenACC para um ambiente de computação híbrida.

Para todas execuções do FoF utilizou-se dados de uma simulação de N-corpos do Consórcio Virgo[1] O trabalho no projeto disponibilizou de acesso à um servidor do Laboratório de Sistemas de Computação da UFSM composto por:

- Processador Intel® Xeon® E5620 com quatro cores físicos e oito virtuais; cache L1 de 32KB, cache L2 de 256KB e cache L3 de 12MB, e 12 GB de memória
- GPU NVIDIA GeForce GTZ Titan X
- Sistema operacional: Debian 9, versão do Linux 4.9.0-2
- Compilador gcc versão 7.3.0
- Compilador da Portland Group (PGI) edição Community (NVIDIA,) versão 17.4-0

4.1 Friends-of-Friends

O Friends-of-Friends (FoF) é um algoritmo de classificação de objetos astronômicos para identificar objetos em interação gravitacional (Huchra and Geller, 1982; CARETTA et al., 2008). Para isso, utilizam-se dados provenientes de simulações cosmológicas, ou de imagens de observações astronômicas.

[1] http://www.mpa-garching.mpg.de/Virgo/data_download.html

A classificação é realizada considerando-se um cenário de N-corpos em atividade gravitacional. O FoF classifica cada um desses corpos (objetos ou partículas) considerando um raio de interação gravitacional (raio de percolação), definido pelo usuário. Partículas presentes à esfera definida pelo raio são consideradas "amigas"e classificadas em um mesmo grupo. Ao final, o algoritmo apresenta o número de grupos encontrados.

O algoritmo inicia recebendo por linha de comando o arquivo de dados de entrada e o raio de percolação a ser utilizado no cálculo da distância. Ao ler o arquivo ele armazena suas informações em *arrays*: um chamado `igru`, que armazenará os grupos de partículas, e os `x`, `y` e `z` para armazenar a posição de cada partícula. O processamento principal do Algoritmo 4.1 é a etapa de classificação composta por três laços `for` aninhados.

O laço mais externo percorre todas as partículas criando um novo grupo, se a partícula atual ainda não está classificada, senão, se ela já pertence a um grupo, passa-se para a próxima. Para novos grupos, calcula-se a distância entre uma partícula e as outras usando a fórmula da distância $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$. Se a distância entre elas é menor do que o raio de percolação definido pelo usuário a nova partícula é considerada sua amiga e classificada no mesmo grupo.

```
for (i = 0 ; i < N ; i++){
  k++;
  while (igru[i] != 0 )i++;
  igru[i] = k;
  for (j = i ; j < N ; j++){
    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 4.1 - Versão sequencial do FoF

Após, o algoritmo escreve em um arquivo de saída as informações iniciais sobre as partículas e apresenta o número de grupos encontrados e o número de partículas por grupo. As paralelizações do FoF relatadas na seção 3 utilizando MPI e OpenMP fizeram processamento sobre a porção de código apresenta em 4.1.

4.2 Plataforma Web

As abordagens apresentadas nesta subseção utilizaram como base a plataforma Web desenvolvida pelo Otávio Madalosso (MADALOSSO et al., 2016), disponível em: `https://github.com/Madalosso/TG`.

4.2.1 Implementação local

O primeiro contato com esse portal foi desafiador, visto que o aluno o havia implementado há mais de um ano, utilizando diferentes versões dos nove programas auxiliares necessários para a sua execução, sendo algumas obsoletas atualmente. Assim, realizaram-se diversos testes iniciais com diferentes versões dos programas até encontrar as que funcionassem corretamente, ainda sem alterações no código fonte do aluno.

Posteriormente, precisou-se entender o funcionamento do portal e pontuar pendências quanto à sua funcionalidade, como problemas com o cadastro do usuário, validação de cadastro via e-mail, *download* dos resultados gerados pela execução do algoritmo e demais funcionalidades do portal em relação a interatividade com o usuário. Outro desafio sobre o portal foi quanto a utilização do *framework* Django, o qual estudou-se através de tutoriais online e pela criação de pequenos projeto pessoais, de modo a conseguir interpretar o código do portal já iniciado.

Por fim, conseguiu-se executar o portal localmente e resolver as pendências observadas após alterações no código fonte do aluno. Como trabalho seguinte, sugeriu-se hospedar o portal em um serviço de computação em nuvem, visando observar o seu comportamento neste ambiente.

4.2.2 Implementação em nuvem

A computação em nuvem vem sendo uma alternativa muito utilizada para usufruir de serviços e de recursos computacionais virtualizados. Nesses ambientes, o usuário

deixa de se preocupar com características sobre a infraestrutura contratada, como custo de energia, manutenções e habilitações de acessos a serviços, e passa a se preocupar apenas com a parte de desenvolvimento do seu trabalho. Existem 3 tipos de serviço de nuvem:

- Infraestrutura como serviço: na qual o provedor hospeda a infraestrutura virtualizada;
- Plataforma como serviço: na qual o provedor oferece a infraestrutura e a plataforma/ambiente de execução virtualizados;
- Software como serviço: na qual o provedor provê a infraestrutura, a plataforma e a aplicação virtualizados;

Apesar de seus distintos benefícios para o usuário, a utilização desses ambientes requer a contratação, em sua maioria mediante a pagamento, dos recursos utilizados e está vulnerável a um tempo de inatividade do provedor. Além disso, deve-se considerar que existe o compartilhamento de recursos entre máquinas virtuais, ou seja, vários usuários estão utilizando um mesmo recurso ao mesmo tempo, o que pode comprometer o desempenho de uma aplicação que necessitaria de um recurso dedicado.

O serviço de Plataforma se apresenta como o mais viável para implementação de um portal web no ambiente de computação em nuvem, onde o usuário pode hospedar suas próprias aplicações utilizando diversas linguagens de programação, bibliotecas e outras ferramentas que o servidor suporte. Existem diversas plataformas para computação em nuvem disponíveis, porém escolheu-se utilizar o Heroku[2], por disponibilizar diversos tutoriais quanto as suas funcionalidades, possibilitar acesso remoto à máquina utilizada (com limitações) e, principalmente, por oferecer um plano básico gratuito de tempo ilimitado, o que a maioria das outras ferramentas não oferece.

4.2.2.1 Heroku

O Heroku é uma Plataforma como Serviço com suporte oficial a diversas linguagens de programação, a vários *frameworks* (incluindo Django) e com mais de 170 *add-ond* disponíveis para auxílio na execução da aplicação. A plataforma utiliza a ideia de "dynos", que são contêiners Unix isolados e virtualizados, no qual os aplicativos são executados. Nos dynos a aplicação recebe o ambiente de execução necessário para

[2]https://www.heroku.com/

sua utilização, além de lidar com requisições Web e processamento de trabalhos em segundo plano. Existem também os "web-dynos", que fazem a hospedagem da aplicação na Web com domínio do Heroku e os "one-off-dynos", que são dynos temporários criados para executar comandos e tarefas esporádicas, acessados via uma sessão interativa com um interpretador de comandos na nuvem.

Cada novo projeto de aplicativo no Heroku é associado a um repositório remoto, sendo via Git a principal forma de lançar ele na plataforma. São oferecidos planos pagos e gratuitos, no qual o pago oferece acesso a mais dynos e outras ferramentas que auxiliam no desenvolvimento e o plano mais caro oferece acesso à uma máquina dedicada, e o plano gratuito oferece acesso à um web-dyno com 512MB de memória RAM, e o seu acesso via one-off-dyno tem a conexão interrompida após trinta minutos de inatividade.

O fluxo de trabalho na plataforma funciona da seguinte maneira: o usuário tem o código de sua aplicação e todos os programas necessários para a sua execução; o sistema do Heroku lê um arquivo intitulado "Procfile"que contém os comando que a plataforma deve utilizar para executar a aplicação; o usuário faz a submissão de seu projeto para a nuvem (processo chamado de "deploy") utilizando comandos git; o Heroku compila o código, interpretando o Procfile e, se tudo correr bem, disponibiliza a aplicação para o usuário. A Figura 4.1 representa este processo.

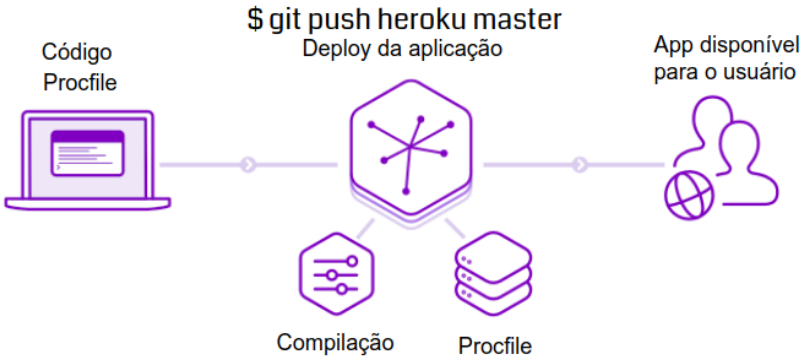

Figura 4.1 - Fluxo de trabalho no Heroku

4.2.2.2 Implementação

Para implementação do portal Web deste trabalho no Heroku foi necessário primeiro criar o projeto para o Heroku dentro de um ambiente virtual que permite criar am-

bientes isolados de desenvolvimento com a linguagem Python. Ele é necessário para que não hajam conflitos entre a instalação das bibliotecas necessárias pro funcionamento do portal em específico e outras bibliotecas instaladas no computador. Após, criou-se o arquivo Procfile e iniciou a fase de deploys para a aplicação.

Apesar dos primeiros deploys terem sido concluídos com sucesso, o portal apresentou erros. Um deles acusou que os arquivos estáticos, que definem o aspecto visual da página Web, não estavam corretamente apresentados. Para solucionar esse problema, precisou-se utilizar o *middleware* WhiteNoise[3]. Outro problema encontrado foi que a plataforma utiliza por padrão a última versão do Python, mas o código do portal é escrito em uma versão anterior, assim, para o correto funcionamento foi preciso definir a versão correta em novo arquivo chamado "runtime.txt"interpretado pelo Heroku.

Os próximos deploys acusaram outros erros, pois como o portal recebe uma requisição (execução do FoF), processa os dados enviados e depois disponibiliza os resultados ao usuário, ele precisaria de mais de um dyno para que esse fluxo ocorresse corretamente, o que não é possível ao plano gratuito. Esse novo dyno seria um "worker-dyno", responsável por realizar trabalhos em segundo plano, no caso lançando a execução do Celery. Uma solução encontrada foi a de usar um novo dyno dentro do já disponibilizado, fazendo com que, ao invés de o Procfile definir os comandos para executar o portal diretamente, ele chamasse outro Procfile, denominado ProcfileFree, que lança um web-dyno para o Gunicorn[4], que é um servidor Python que permite executar aplicações concorrentemente, e um worker para o Celery.

Com isso, o portal começou a funcionar corretamente, porém apresentou algumas limitações devido ao ambiente virtualizado. A primeira delas foi o limite do tamanho do arquivo de dados de entrada a ser processado pelo FoF. Uma solução encontrada foi utilizar o Gevent[5], que permite ao Gunicorn executar *micro-threads* mais leves, que auxiliam nas requisições Web pesadas, principalmente àquelas limitadas por entrada e saída como o FoF, que recebe entradas de dados com mais de 20MB.

4.2.2.3 Resultados

Apesar de funcional, a utilização do ProcfileFree não agregou desempenho equivalente a utilização de dynos distintos em um plano pago, pois ainda pode comprome-

[3]http://whitenoise.evans.io/en/stable/
[4]http://gunicorn.org/
[5]http://www.gevent.org/

ter o processamento de arquivos de dados muito grandes, mesmo com a utilização do Gunicorn. Além disso, encontraram-se problemas para manter a consistência do banco de dados utilizado devido a sua hospedagem no servidor do Heroku. Uma alternativa para isso seria utilizar um banco de dados externo, para armazenar e recuperar os dados salvos para cada usuário durante o seu acesso ao portal. Entretanto, esse serviço também comprometeria o seu desempenho. Outras limitações observadas foram quanto à utilização de máquinas compartilhadas, que comprometeram o desempenho das execuções, e o domínio do site pertencente ao Heroku.

Como pontos positivos, a implementação no Heroku se destacou por ser gratuita e abstrair a infraestrutura utilizada como a administração do servidor. Como o Heroku vem sendo bastante utilizado por desenvolvedores ele oferece uma ampla documentação, facilitando o desenvolvimento de iniciantes na área de computação em nuvem. O portal hospedado na nuvem Heroku está disponível em: `https://webfriends.herokuapp.com`, e o código utilizado e os registros de log com orientações sobre a execução local e em nuvem do portal estão em: `https://github.com/anaveroneze/webfriends`.

4.2.2.4 Seminário

No dia 27 de outubro de 2017 ocorreu um seminário sobre o andamento do projeto, junto a uma aluna do curso de Análise e Desenvolvimento de Sistemas da FATEC, que estava desenvolvendo um projeto similar sobre uma plataforma Web para execução de experimentos com algoritmos como o FoF em seu trabalho de conclusão de curso. O projeto dessa aluna era realizar uma nova implementação desse modelo de portal, com base no portal anteriormente projetado [MADALOSSO 2016], visando a execução remota de quaisquer algoritmos correlatos com implementação local utilizando os mesmos programas (Django, Celery, Redis, etc.) em um ambiente *multicore*. Para o seminário, planejou-se sua apresentação à distância, o que resultou em uma ótima experiência, pois além de esclarecer dúvidas e diferenças entre os diferentes portais, pôde-se trocar ideias sobre aspectos da área de computação em geral.

4.2.2.5 Continuação

Como trabalhos futuros no ambiente de computação em nuvem, sugere-se um possível teste de hospedagem do portal no Heroku utilizando a modalidade paga, e reflexões sobre as utilizações mais viáveis do portal hospedado na nuvem, dadas as suas limitações, como por exemplo: utilizá-lo para execuções que demandem menos

processamento ou utilizá-lo para realizar experimentos sobre sugestões para o portal, como diferentes frameworks, bancos de dados e add-ons. Além disso, sugeriu-se observar também outras plataformas como serviço.

4.3 Amiga Halo Finder

Esta etapa da pesquisa consistiu no estudo de um *software* correlato ao FoF, que processa grandes quantidades de dados cosmológicos provenientes de simulações e de observações para investigar a formação de estruturas cósmicas no Universo, como estrelas, galáxias e aglomerados de galáxias. O Amiga Halo Finder (AHF)[6], é um *software*, enquadrado como um halo finder, para encontrar estruturas de halos e sub-halos em dados de simulações cosmológicas junto a propriedades cosmológicas sobre a entrada definidas pelo usuário. O programa permite execuções seriais ou paralelas com MPI e OpenMP.

Em seu site oficial, são disponibilizados o arquivo para *download* do programa, dois artigos publicados por seus desenvolvedores (GILL et al., 2004) e (Knollmann and Knebe, 2009) para serem utilizados como documentação e um arquivo pdf com orientações sobre a plataforma. Essa documentação explica o contexto do AHF, como o compilar e executar e descreve as possibilidades que o usuário tem de alterar a execução do seu experimento utilizando flags de compilação.

Porém, na mesma documentação, os desenvolvedores sugerem que para obter propriedades mais avançadas, o usuário não pode se limitar àquelas orientações, mas sim, deve realizar alterações no código fonte do programa, que consiste em muitos arquivos. Por isso, como primeira pesquisa sobre o AHF, optou-se por investigar o potencial da ferramenta para a paralelização com OpenMP, oferecida nativamente embora não esclarecida na documentação.

Para investigar o comportamento de execuções paralelas com OpenMP no AHF, utilizou-se uma amostra de dados do Consórcio Virgo na máquina *lsc5*. Utilizaram-se como ferramentas auxiliares dois analisadores de perfil de execução: o Understand[7] e o Intel Vtune Amplifier[8].

Com os resultados provenientes dos experimentos e de análises do programa, observou-se que o AHF não apresenta vantagens de desempenho nas execuções paralelas com os dados utilizados em comparação à execução serial. Sugere-se que isso

[6]http://popia.ft.uam.es/AHF/Download.html
[7]https://scitools.com/
[8]https://software.intel.com/en-us/intel-vtune-amplifier-xe

ocorra devido aos autores do trabalho priorizarem o seu bom funcionamento e completude em relação a outros halo finders, oferecendo apenas como fator adicional a possibilidade de sua execução em ambientes paralelos. Como próximos trabalhos, planeja-se um maior estudo do código fonte do programa, principalmente com o intuito de rever e alterar a forma de paralelização para o padrão OpenMP em busca de um maior desempenho, além de realizar testes com o padrão MPI.

4.4 Paralelização com OpenACC

Apesar de implementações anteriores do FoF paralelo com MPI e com OpenMP para ambientes *multicore*, o ambiente de computação híbrida, que é capaz de resolver de forma eficiente problemas que processam grandes quantidades de dados, ainda não havia sido explorado.

Atualmente, existem diversos dispositivos aceleradores, como FPGAs, Intel Xeon Phi e GPU. Optou-se por utilizar a GPU pois além de ser um dos aceleradores mais populares também é o mais acessível no servidor utilizado. Dentre as ferramentas disponíveis para paralelização de programas com GPU, utilizou-se o padrão OpenACC (OpenACC, 2011), que possibilita a distribuição da computação em um ambiente híbrido usando diretivas de compilação no código.

4.4.1 Estratégia 1

A primeira estratégia buscou fazer o mínimo possível de alterações no código original do algoritmo. Para isso, paralelizou-se o laço de repetição mais interno do 4.1, visto que ele não possui dependência de dados. Para isso, foi utilizada a diretiva `#pragma acc parallel`, que descreve uma região do código a ser acelerada com os *kernels* da GPU. Notou-se que a variável `dist`, que armazena o valor da distância entre duas partículas, deve ser privada para evitar valores inconsistentes.

Visto que o laço paralelizado lê os vetores `x`, `y`, `z` e `igru`, e também pode alterar o último, utilizou-se a diretiva `#pragma acc data copyin` antes do laço mais externo, de modo a copiar uma única vez para a GPU os valores dos vetores apenas lidos. O `igru` precisou de maior atenção, pois é modificado antes de ser lido no laço paralelizado e pode ser modificado dentro dele. Esse tratamento gerou duas abordagens semelhantes:

a) A primeira consistiu em utilizar a diretiva `#pragma acc copy`, que copia o vetor da CPU para a GPU antes de entrar na região paralela e da GPU

para a CPU antes de sair, garantindo a consistência dos dados do vetor durante toda a execução.

b) A segunda utilizou a diretiva `#pragma acc update device`, que copia os valores atualizados no `igru` da memória local para o dispositivo após sua alteração no laço mais externo, e `#pragma acc update self`, que atualiza os dados modificados no dispositivo para a memória local após sair da região paralela. Para isso, precisou-se alocar o vetor no acelerador usando a diretiva `#pragma acc data create` antes do laço mais externo.

4.4.2 Estratégia 2

A segunda estratégia buscou considerar maiores alterações no código original do FoF, e para isso reimplementou o código na linguagem C. Para isso também foram utilizadas duas abordagens para implementação do FoF voltado à sua paralelização com OpenACC:

a) A primeira consistiu em balancear a carga total a ser processada (número de partículas) entre os *kernels* da GPU, que realizaram o FoF sequencial sobre elas e após os resultados voltariam à execução em CPU, que faria um pós-processamento para determinar partículas que possam pertencer ao mesmo grupo, mas que foram processadas por *kernels* diferentes.

b) Visto que o maior processamento neste algoritmo está em calcular a distância entre todas as partículas, a segunda estratégia buscou otimizar essa tarefa. Para isso, inicialmente calculou na GPU todas as distâncias entre partículas, armazenando as informações em uma matriz. Após, em CPU, verificaram-se quais partículas pertenciam ao mesmo grupo verificando se a distância entre elas era inferior ao raio de percolação. Se positivo, as partículas eram classificadas como pertencentes ao mesmo grupo. Ao final, o pós-processamento seria sobre a inter-relação entre as partículas, pois se a partícula `A` pertencer ao grupo da partícula `B`, e `B` pertencer ao grupo de `C`, então `A` também pertence ao grupo de `C`, e assim `A`, `B` e `C` pertencem a um mesmo grupo.

5 RESULTADOS E DISCUSSÃO

5.1 Paralelização com OpenACC

5.1.1 Estratégia 1

Inicialmente, a compilação do código original do FoF apresentou erros, acusando sobrecarga de funções. Possivelmente, isso ocorreu porque a `libc` nativa define as funções das bibliotecas declaradas no mesmo momento em que a `libstdc++` também as carrega, assumindo que a libc não o faça. Assim, a solução encontrada foi não utilizar as bibliotecas conflitantes `math` e `stdlib`, adaptando o trecho do código que utilizava a função `sqrt()` para cálculo da distância.

Foram realizadas 30 execuções para três arquivos de entrada com 65536 partículas (Arquivo 1), 174761 partículas (Arquivo 2) e 1048576 (Arquivo 3), o qual foram realizadas 10 execuções, todas utilizando raio de percolação 0.1. Os resultados obtidos para as duas versões são apresentados na Tabela 5.1.

Arquivo	Média (s) serial	Média (s) versão 1	Speedup versão 1	Média (s) versão 2	Speedup versão 2
1	25.25	21.07	1.20	19	1.33
2	162	126.31	1.28	112.40	1.44
3	5120.30	4015.29	1.27	3582.66	1.43

Tabela 5.1 - Análise de desempenho para as duas abordagens com OpenACC

Utilizando o *profiler* nvprof [1], investigou-se a diferença entre os *speedups* das duas abordagens. Na segunda, o *pragma* de atualização de dados da CPU para a GPU apenas sincroniza os valores alterados, enquanto que na primeira, o *pragma* de cópia escreve todo o vetor na GPU a cada iteração do laço, resultando em um maior processamento. Com essa análise, também se constatou que o maior tempo gasto em ambas abordagens, cerca de 60% do tempo total, está no bloqueio da CPU a espera dos *kernels* lançados pelo acelerador, ou seja, tempo em que a GPU está fazendo o seu trabalho.

Os resultados de desempenho mostraram que, com mínima modificação do código original, o algoritmo FoF não explora todo o potencial de uso da GPU, demandando um tempo excedente de execução, pois processa partes do programa na GPU e partes

[1] https://developer.nvidia.com/nvidia-visual-profiler

na CPU alternadamente. Esse fato, sustenta-se na lógica utilizada no algoritmo FoF, que contém várias estruturas condicionais e dependências de dados, o que dificultou a sua paralelização no dispositivo utilizado. Apesar disso, indica-se o OpenACC para investigações iniciais, pois é uma ferramenta simples de ser utilizada e altamente portável.

Além disso, o laço mais interno possui um desbalanceamento de carga, o que levou à utilização de diretivas de escalonamento para determinar a quantidade de dados a serem processados por *kernel* lançado. Entretanto, não obteve-se êxito, visto que a computação no laço paralelizado não depende principalmente das informações sobre cada partículas (suas posições), e não da quantidade de partículas a serem processadas, o que implicaria em cargas distintas para cada arquivo de entrada.

Visto que a segunda abordagem da primeira estratégia teve um melhor desempenho, decidiu-se comparar sua execução com a execução utilizando paralelização com OpenMP relatada em (BERWIAN et al., 2017). Apesar das distintas arquiteturas, uma comparação sobre a execução desses algoritmos para o mesmo conjunto de dados poderia contribuir para uma posterior implementação do FoF.

Foram realizadas 10 execuções dos algoritmo com OpenMP utilizando 2, 4 e 8 *threads* e 10 execuções do algoritmo com OpenACC. Os dados cosmológicos utilizados foram uma parte da simulação do Consórcio Virgo composto por 318133 partículas.

A Figura 5.1 apresenta a média dos resultados das execuções das duas estratégias de paralelização do FoF versão $O(n^2)$ em comparação à média de sua execução sequencial. Observou-se que a abordagem com OpenMP obteve um *speedup* de 2.25 na paralelização com duas *threads*. Entretanto, o aumento do número de *threads* utilizadas não trouxe desempenho à aplicação, com uma média de 166.67 segundos para execução com OpenMP. Para as execuções com OpenACC, obteve-se um *speedup* de 3.22 levando 116.06 segundos.

Apesar de ambas estratégias ganharem desempenho, com *speedups* maiores que 2 a partir de mínima paralelização, o *speedup* da estratégia com OpenMP não foi crescente conforme o número de *threads*, e com OpenACC o potencial do acelerador não foi completamente explorado. Mesmo utilizando diferentes arquiteturas, esses resultados incentivaram a continuação apresentada na Estratégia 5.1.2.

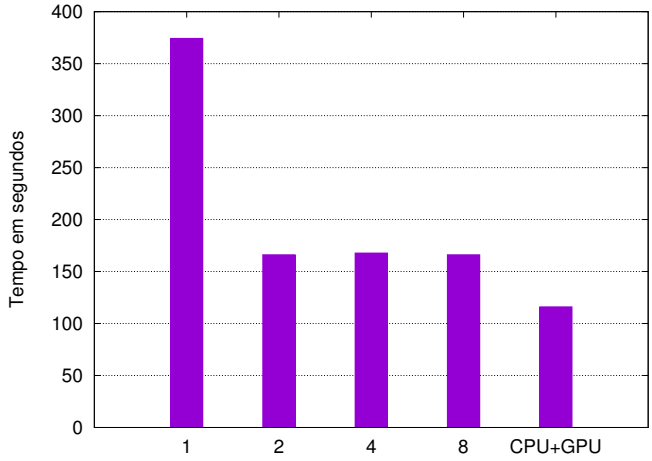

Figura 5.1 - Desempenho da execução paralela do FoF com OpenMP(2, 4 e 8 *threads*) e com OpenACC (ambiente composto por CPU e GPU).

5.1.2 Estratégia 2

A segunda estratégia está sendo implementada, porém, nota-se que a segunda abordagem apresenta-se inicialmente como a mais promissora, pois não processa dados com dependências na GPU, apenas calcula as distâncias, que exigem grande processamento para o número total de partículas. E após, o restante do algoritmo pode explorar as *threads* da CPU.

5.2 Redação de artigos

Os trabalhos sobre a paralelização do FoF com OpenACC (SOLORZANO et al., 2018)e sobre a execução de experimentos com o AHF (SOLORZANO et al., 2018) resultaram em duas publicações apresentadas na XVIII Escola Regional de Alto Desempenho (ERAD/RS) em abril 2018. Os anais do evento estão disponíveis em: `http://www.inf.ufrgs.br/erad2018/anais/ERAD2018.pdf`

6 CONCLUSÕES

Neste trabalho, estudou-se uma plataforma Web para experimentos remotos com o algoritmo Friends-of-Friends e implementações do algoritmo para execução em um ambiente de computação híbrida. Durante esse estudo também foram investigadas outras abordagens, como a hospedagem do portal em um ambiente de computação em nuvem e estudos sobre o AHF, um *software* correlato ao FoF. Os resultados apontam que a hospedagem do portal no servidor local do INPE ainda é a melhor

solução frente à hospedagem em nuvem, que em sua versão gratuita oferece recursos limitados. Além disso, observou-se que o FoF pode explorar o uso de uma GPU como acelerador, porém não faz uso de todo o seu potencial inserindo as diretivas OpenACC com mínima alteração do código. Assim, espera-se que com alguma das duas novas estratégias seja possível fazer melhor uso do acelerador.

REFERÊNCIAS BIBLIOGRÁFICAS

BARNES, J.; HUT, P. A hierarchical O(NlogN) force-calculation algorithm. 1986. In: Nature 324(4). p. 446–449 3

BERWIAN, L.; ZANCANARO E. T.; CARDOSO, D. J.; CHARÃO A. S.; RUIZ, R. S. R.; de CAMPOS VELHO H. F. Comparação de Estratégias de Paralelização de um Algoritmo Friends-of-Friends com OpenMP. In: Anais da XVII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, 2017. p. 279–252. 3, 15

CARETTA, C. A. et al. Evidence of turbulence-like universality in the formation of galaxy-sized dark matter haloes. 2008. Astronomy and Astrophysics, 487(2). p. 445–451. 1, 2, 4

CHAPMAN, B.; JOST, G.; VANDERPAS, R.; KUCK, D. J. Using OpenMP: Portable Shared Memory Parallel Programming. The MIT Presse. 31 outubro 2007. 3

GILL, S. P. D.; KNEBE, A.; GIBSON, B. K. The Evolution of Substructure – I. A new identification method. 2004. Monthly Notices of the Royal Astronomical Society, 351(2). p. 399–409. 11

HUCHRA, J. P.; GELLER, M. J. Groups of galaxies I. Nearby groups. 1982. In: The Astrophysical. 257. p. 423–437. 1, 2, 4

KNOLLMANN, S. R.; KNEBE, A. AHF: Amiga's Halo Finder. 2009. The Astrophysical Journal Supplement. 182(2). p. 608–624. 11

MADALOSSO, O. M.; CHARÃO A. S.; de CAMPOS VELHO H. F; RUIZ, R. S. R. Implementação do algoritmo Friends-of-Friends de complexidade n*logn(n) para classificação de objetos astronômicos. In: Anais da XV Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, 2015. 3

MADALOSSO, O. M.; CHARÃO A. S.; de CAMPOS VELHO H. F; RUIZ, R. S. R. A web portal framework for remote execution of high performance applications in astronomy. 2016. In: Proceedings of the Conference of Computational Interdisciplinary Science. 6

MADSEN, M. S. In the Dynamic Cosmos - exploring the physical evolution of the Universe. 1996. New York, NY, USA. Chapman & Hall. 1

OCTREE. Disponível em: <https://en.wikipedia.org/wiki/Octree>. Acesso em: 15 julho 2018. 3

OPENACC. The OpenACC Application Programming Interface, 10, 2011. Disponível em: <http://www.openacc.org>. Acesso em: 15 julho 2018. 12

PACHECO, P. S. Parallel programming with MPI. 1997. San Francisco: Morgan Kaufmann Publishers, Inc. 3

PGI community edition. Disponível em: <https://developer.nvidia.com/openacctoolkit>. Acesso em: 15 julho 2018. 4

RUIZ, R. S. R.; de CAMPOS VELHO H. F.; CARETTA, C. A. Parallel algorithm Friends-of-Friends to identify galaxies and cluster of galaxies dor dark matter halos. 2009. In: Anais do IX Workshop do Curso de Computação Aplicada (WORCAP). São José dos Campos: INPE. 2, 3

SOLORZANO, A. L V.; CHARÃO A. S.; RUIZ, R. S. R.; de CAMPOS VELHO H. F. Análise do Software Paralelo AHF para Identificação de Estruturas em Cosmologia. In: Anais da XVIII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, 2018. 16

SOLORZANO, A. L V.; CHARÃO A. S.; RUIZ, R. S. R.; de CAMPOS VELHO H. F. Exploração de Computação Híbrida com OpenACC em um Algoritmo Friends-of-Friends para Classificação de Objetos Astronômicos. In: Anais da XVIII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, 2018. 16