UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

GPUHELP: UM AMBIENTE DE APOIO À EXECUÇÃO DE PROGRAMAS PARALELOS EM ARQUITETURAS DE GPU

Douglas Pires Borges

Santa Maria, RS, Brasil

2014

GPUHELP: UM AMBIENTE DE APOIO À EXECUÇÃO DE PROGRAMAS PARALELOS EM ARQUITETURAS DE GPU

Douglas Pires Borges

Orientador: Profª. Drª. Andrea Schwertner Charão

Santa Maria, RS, Brasil

2014

Pires Borges, Douglas

GPUHelp: Um Ambiente de Apoio à Execução de Programas Paralelos em Arquiteturas de GPU / por Douglas Pires Borges. – 2014.
112 f.: il.; 30 cm.

Orientador: Andrea Schwertner Charão
- Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2014.

1. Programação Paralela em GPU. 2. Ambiente de Apoio à Execução de Programas Paralelos em GPU. 3. OpenCL/CUDA. I. Schwertner Charão, Andrea. II. Título.

E-mail: douglasp.borges@gmail.com

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova

GPUHELP: UM AMBIENTE DE APOIO À EXECUÇÃO DE PROGRAMAS PARALELOS EM ARQUITETURAS DE GPU

elaborada por
Douglas Pires Borges

como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

COMISSÃO EXAMINADORA:

Andrea Schwertner Charão, Drª.
(Presidente/Orientador)

Juliana Kaizer Vizzotto, Drª. (UFSM)

Claudio Schepke, Dr. (UNIPAMPA)

Santa Maria, 07 de Março de 2014.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus, por me permitir ter chegado até aqui, por ter concedido forças e coragem para sempre seguir em frente.

Agradeço aos meus pais, meu irmão, meus queridos avós e ao Sr. Nelcir, pelo apoio, incentivo, carinho e esforço que tiveram de fazer para que eu chegasse até aqui. Obrigado por me darem forças nos momentos difíceis e por acreditarem no meu potencial. Nunca terei palavras suficientes para expressar meu amor, carinho e gratidão para com todos vocês.

A minha orientadora, professora Andrea Charão, uma excelente pessoa na qual tive a oportunidade de trabalhar e conviver. Obrigado pelo apoio, incentivo, carinho, compreensão e por fazer com que eu buscasse sempre o meu melhor para construir esta Dissertação. Graças a seus conselhos e orientações pude amadurecer e evoluir. Este trabalho apenas tornou-se possível graças a suas considerações e sugestões.

Ao professor Benhur Stein pelas dicas, conselhos, por sempre ouvir o que eu tinha a dizer, por discordar algumas vezes, o que sempre inspirava-me a buscar uma melhor solução para resolver as dificuldades que surgiram ao longo do caminho. A professora Márcia Pasin, pelos conselhos, conversas e pelas dicas no Latex. A professa Juliana Vizzotto e ao professor Claudio Schepke, por participarem da avaliação deste trabalho.

Aos meus colegas e amigos do LSC, Maurício, Fábio e Rafael. Obrigado por sua amizade, companheirismo e apoio, sempre que necessário. Aos alunos concluintes do curso de Ciência da Computação da UFSM, por participarem do processo de avaliação deste trabalho.

Ao longo do Mestrado, tive a oportunidade de conquistar bem mais do que o título de Mestre. Pude conquistar amizades as quais pretendo levar por toda a vida. Encontrei pessoas que sempre se fizeram presentes para me apoiar, incentivar, aconselhar e brigar, quando necessário fosse. Amigos que não me abandonaram em momento algum. Aos meus amigos: Giullianno Olivar, Otávio, Cezar, Guilherme, Fábio, Matheus, Daniel, Rafael, Guilherme C., George e claro, ao meu grande amigo Maurício. Agradeço a vocês pelo bem mais precioso que me concederam: sua amizade. A uma grande amiga, Aline Mello, pelo apoio, carinho, dedicação, paciência e incentivo. Por sempre me fazer encontrar ou criar forças, quando estas pareciam já não existir.

A todas aquelas pessoas, que contribuíram para este trabalho de forma direta ou indireta, mesmo não tendo sido citadas neste singelo agradecimento, meu sincero obrigado.

Serei eternamente grato, por todo carinho, força e incentivo que recebi de todos vocês.

Foi por todos vocês que cheguei até aqui. E, sem duvidas, é por vocês que seguirei em frente!

“Não importa o tamanho da montanha, ela nunca poderá tapar o sol.”
— PROVÉRBIO CHINÊS

RESUMO

Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

GPUHELP: UM AMBIENTE DE APOIO À EXECUÇÃO DE PROGRAMAS PARALELOS EM ARQUITETURAS DE GPU

AUTOR: DOUGLAS PIRES BORGES
ORIENTADOR: ANDREA SCHWERTNER CHARÃO
Local da Defesa e Data: Santa Maria, 07 de Março de 2014.

Frente às complexas dificuldades que envolvem as aplicações científicas, pesquisadores buscam novos meios de otimizar o processamento destas, utilizando-se de novos conceitos e paradigmas em programação paralela e distribuída. Uma alternativa emergente a este cenário, é a utilização de GPUs *(Graphics Processing Unit)* devido a seu alto poder computacional. Contudo, juntamente com os benefícios advindos da utilização de tais técnicas, tem-se diversas e complexas questões relacionadas ao ensino e aprendizado das mesmas. Desse modo, pesquisadores passaram a dedicar esforços para obter um melhor resultado no ensino destas áreas. Assim, surgiram os ambientes de apoio ao ensino de programação paralela. Tais ambientes provêem um conjunto de ferramentas para o desenvolvimento e teste de aplicações, aprimorando assim a experiência educacional. Entretanto, as pesquisas atuais focam em ambientes de apoio a programação paralela para arquiteturas de CPU, não existindo assim, ambientes de apoio voltados as arquiteturas de GPU. A inexistência de tais ambientes tem impacto negativo, durante o processo de aprendizado, comprovado em diferentes pesquisas científicas. Neste contexto, este trabalho apresenta um ambiente de apoio a programação paralela em GPU, intitulado GPUHelp. O GPUHelp proporciona aos usuários uma solução completa para o desenvolvimento e teste de códigos para arquiteturas de GPU, o CUDA e OpenCL, mesmo para aqueles usuários que não possuem placas gráficas em seus computadores, o que não era possível até então, visto a necessidade de uma placa gráfica compatível com tais arquiteturas. As avaliações realizadas demonstraram que o GPUHelp é uma solução viável com aplicabilidades distintas nos cenários de ensino e treinamento de programação paralela em GPU.

Palavras-chave: Programação Paralela em GPU, Ambiente de Apoio à Execução de Programas Paralelos em GPU, OpenCL/CUDA.

ABSTRACT

Post-Graduate Program in Informatics
Federal University of Santa Maria

GPUHELP: AN ENVIRONMENT SUPPORTING TO EXECUTION OF PARALLEL PROGRAMS FOR GPU ARCHITECTURES

AUTHOR: DOUGLAS PIRES BORGES
ADVISOR: ANDREA SCHWERTNER CHARÃO
Defense Place and Date: Santa Maria, March 07st, 2014.

Faced with complex problems that involve scientific applications, researchers are looking for new ways to optimize the processing of these, using new concepts and paradigms for parallel and distributed programming. An emerging alternative to this scenario is the use of GPUs (Graphics Processing Unit) due to its high computational power. However, along with the benefits from the use of such techniques has been diverse and complex issues related to teaching and learning from them. Thus, researchers began to devote efforts to obtain better results in teaching these areas. So, the environments to support teaching of parallel programming have emerged. Such environments provide a set of tools for the development and testing of applications, thereby improving the educational experience. However, the current researches focuses on environments supporting teaching parallel programming for CPU architectures, not exist environments to teaching support teaching oriented architectures GPU. The absence of such environments has a negative impact, proven in various scientific researches. In this context, this work presents an environment for supporting parallel programming in GPU, called GPUHelp. The GPUHelp provides to users a complete solution for developing and codes test for GPU architectures, the CUDA and OpenCL, even for those users that do not have graphics cards on their computers, which was not possible before, given the need to graphics card compatible with such architectures. Evaluations have shown that GPUHelp is a feasible solution with different applicability scenarios in education and training on parallel programming GPU.

Keywords: Parallel Programming on GPU. Environment Supporting Execution of Parallel Programs on GPU. OpenCL/CUDA.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CPU	*Central Processing Unit*
CUDA	*Compute Unified Device Architecture*
GB	*Gigabyte*
GPGPU	*General Purpose computing on Graphics Processing Units*
GPU	*Graphics Processing Unit*
HD	*Hard Disk*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
JAR	*Java ARchive*
JDK	*Java Development Kit*
JRE	*Java Runtime Environment*
JVM	*Java Virtual Machine*
LSC	Laboratório de Sistemas de Computação
MB	*Megabyte*
OpenCL	*Open Computing Language*
PDF	*Portable document format*
RMI	*Remote Method Invocation*
SDK	*Software Development Kit*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
UFSM	Universidade Federal de Santa Maria

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto e Motivação

A incessante busca por um maior desempenho computacional vêm impactando no surgimento de novas tecnologias, além de novas metodologias para desenvolvimento de *software* e projeto de *hardware*. A computação de alto desempenho *(High Performance Computing)* é uma área em constante evolução. Tal área vem sendo empregada em diferentes pesquisas científicas e de engenharias, visando solucionar problemas computacionais que envolvam alto poder de processamento. Embora o poder computacional disponível atualmente seja sem precedentes, cientistas e pesquisadores necessitam de maior desempenho e velocidade para conduzir suas pesquisas, fazendo-se necessária a execução de milhões de instruções por segundo.

A programação paralela apresenta-se como uma forma de solucionar vários problemas computacionais que envolvam alto poder de processamento e grande quantidade de recursos computacionais de diferentes áreas científicas (PACHECO, 2011) (HWU; KIRK, 2011) (MATTSON; WRINN, 2008). De forma simplificada, pode-se definir a área de programação paralela como sendo uma técnica que se utiliza de vários recursos computacionais, visando, entre outras questões, uma redução no tempo necessário para resolução de determinado problema (PACHECO, 2011) (MATTSON; WRINN, 2008). Por outro lado, esta área pode ser considerada complexa, pois engloba uma série de premissas que devem ser corretamente compreendidas pelo programador, que envolvem desde questões de raciocínio lógico a questões de compreensão do problema como um todo, caso contrário, os resultados da aplicação paralela tendem a não ser satisfatórios (GEBALI, 2011) (MORON; RIBEIRO; SILVA, 1998) .

A crescente mudança para a programação paralela contribui para o surgimento de um novo cenário computacional: onde as arquiteturas tendem a evoluir seu poder de processamento para uma grande quantidade de instruções por segundo, enquanto que o *software* evolui de maneira concorrente e paralela às arquiteturas físicas disponíveis. Isto cria uma nova necessidade: extrair todo o poder computacional fornecido pelas atuais arquiteturas paralelas. A programação de alto desempenho é uma área capaz de extrair um grande poder de processamento de uma arquitetura que possa fornece-lo, mediante situações favoráveis de *hardware* e *software*.

Frente a este cenário, pesquisadores recorreram a novas metodologias com o objetivo de conseguir um aumento no poder computacional. Uma das técnicas utilizadas é a aplicação de GPUs *(Graphics Processing Units)* para a resolução de variados tipos de problemas, em

diferentes áreas científicas. Tal fato somente é possível porque as GPUs atuais conseguem ir muito além de apenas processar tarefas visuais. Estas tornaram-se poderosos processadores maciçamente paralelos (HWU; KIRK, 2011). A empregabilidade de GPUs na resolução de diferentes problemas científicos e de engenharias, resultou no surgimento do movimento das GPGPUs *(General Purpose Computing on Graphics Processing Unit)* (GPGPU, 2013), que trata da combinação de GPUs e CPUs para a solução de diferentes problemas computacionais complexos e que exijam grande poder computacional.

As áreas de programação paralela, e em especial a de programação paralela em GPUs, enfrentam algumas dificuldades que precisam ser contornadas constantemente pela comunidade científica. Uma delas está relacionada ao ensino destas áreas, enquanto que a outra, a utilização e a prática delas propriamente ditas.

A primeira dificuldade acerca da área de programação paralela e de GPUs tem relação com o ensino destas. A utilização de programação paralela não é uma prática simples. Segundo Jie (LIU; WU; MARSAGLIA, 2012), entre as diferentes subáreas da computação, a que apresenta um maior nível de dificuldade é a programação. Em se tratando de programação paralela em GPUs, a complexidade torna-se ainda maior (HWU; KIRK, 2011). Diversos autores (GIACAMAN, 2012) (DELISTAVROU; MARGARITIS, 2011) (HWU; KIRK, 2011) (MAROWKA, 2008) concordam que a complexidade da área de programação paralela também está presente no cenário educacional, e que ensinar programação paralela não é uma tarefa fácil (DELISTAVROU; MARGARITIS, 2011) (MAROWKA, 2008). A computação paralela foi desenvolvida há décadas e ainda continua sendo uma habilidade subdesenvolvida pela grande maioria dos programadores (GIACAMAN, 2012). Este, entre outros motivos, é um limitador no interesse de alguns acadêmicos pela utilização de programação paralela em GPU (GIACAMAN, 2012) (BARNEY; LIVERMORE, 2013) (MAROWKA, 2008).

Finalizando as dificuldades acerca da área de programação paralela em GPUs, um último fator – e talvez o mais importante – está relacionado à aplicação prática da área. Para tornar-se possível o desenvolvimento e especialmente a realização dos testes de aplicações em GPUs, é necessária a disponibilidade de uma placa gráfica dedicada, sendo este um fator limitador para a utilização de programação paralela em unidades de processamento gráfico. Atualmente, uma grande parte dos dispositivos que são comercializados, sejam eles portáteis ou de mesa, possuem GPUs compartilhadas, ou seja, fazem uso da memória e processador do dispositivo, tendo seu poder limitado por terem de compartilhar recursos com outras funcionalidades (NVIDIA,

2013b), não sendo compatíveis para programação paralela em GPUs.

Frente ao atual cenário computacional sem precedentes, diversas pesquisas tem focado na resolução dos problemas acerca do ensino e de maneiras de prover um melhor suporte a este, nas áreas de programação paralela e de programação paralela em GPU. Acerca do ensino, pesquisas como (FREITAS, 2012) (GIACAMAN, 2012) (IPARRAGUIRRE; FRIEDRICH; COPPO, 2012) (LIU; WU; MARSAGLIA, 2012) (MAROWKA, 2008) (MATTSON; WRINN, 2008) concordam que o ensino de programação paralela de forma teórica é inútil, os autores concluem que as lições práticas são importantes, porém, com o objetivo de aumentar a experiência e a capacidade de compreensão dos problemas a serem resolvidos, os estudantes devem ter seções de prática intensas.

Outras pesquisas focam em ferramentas de apoio ao ensino de programação paralela, sendo que estudos demonstram, conforme apresentado em (FREITAS, 2012) (GIACAMAN, 2012) (DELISTAVROU; MARGARITIS, 2010) (OLIVEIRA; CARISSIMI; TOSCANI, 2001) (MAROWKA, 2008) (MATTSON; WRINN, 2008), que ambientes de apoio ao ensino de programação paralela possuem um papel fundamental no ensino de tal área, tendo contribuições que vão desde uma melhor compreensão desta à despertar o interesse de alunos pela adoção da área em atividades acadêmicas futuras. Frente a necessidade de ambientes de apoio ao ensino de programação paralela, surgiram ferramentas como o StarHPC (IVICA; RILEY; SHUBERT, 2009), o ambiente apresentado em (DELISTAVROU; MARGARITIS, 2011) e o ambiente apresentado em (PARK et al., 2000). Projetar e desenvolver ambientes de apoio ao ensino de programação, em especial de programação paralela, demandam tempo e esforço (IVICA; RILEY; SHUBERT, 2009). Existem ainda, uma série de dificuldades acerca do desenvolvimento de tais ambientes, entre elas: custos de *hardware*, gerenciamento, pesquisa e projeto de ferramentas, entre outros (IVICA; RILEY; SHUBERT, 2009).

Este trabalho apresenta o GPUHelp, um ambiente que tem como objetivo atuar como ferramenta de apoio ao ensino da área de programação paralela em arquiteturas de GPUs. Tal ambiente possui suporte aos principais *frameworks* de programação paralela em arquiteturas de GPU, o OpenCL (KHRONOS, 2013) e o CUDA (CUDA, 2013b).

1.2 Objetivos e Contribuição

O objetivo deste trabalho é desenvolver um ambiente capaz de prover os recursos necessários para alunos e professores trabalharem com programação paralela em GPU, visando difundir

a utilização de tal área provendo uma infraestrutura completa para sua utilização. A alternativa desenvolvida oferece um ambiente composto de três partes, sendo elas: 1) um ambiente cliente, acrescido de 2) uma aplicação para execução remota de códigos em 3) um ambiente servidor. O ambiente cliente tem por objetivo prover os recursos e ferramentas necessários para que o usuário possa desenvolver seus códigos para arquiteturas de programação paralela em GPU. Já o ambiente servidor visa solucionar o problema que alguns usuários enfrentam de não possuírem placas gráficas dedicadas, pois este possui placa gráfica compatível com programação paralela em arquiteturas de GPU. Por fim, a aplicação para execução remota de códigos atua como uma ferramenta que conecta o ambiente cliente ao ambiente servidor, de forma transparente ao usuário, permitindo que este teste seus códigos no ambiente servidor.

As principais contribuições deste trabalho são, respectivamente, o desenvolvimento de um ambiente completo para o desenvolvimento e testes de aplicações paralelas para arquiteturas de GPU, com foco nas arquiteturas OpenCL e CUDA, bem como, a disponibilização de tal ambiente à comunidade científica da Universidade Federal de Santa Maria - UFSM, inicialmente. Outras contribuições podem ser citadas, como a possibilidade de executar o código desenvolvido no ambiente local do usuário, em um servidor remoto com placa gráfica com suporte a programação paralela em arquiteturas de GPU. A aplicação para execução remota de códigos é outra contribuição do trabalho proposto. Através de tal aplicação, usuários que não possuem placa gráfica em seus *desktops* e/ou *notebooks*, que antes não poderiam testar seus códigos, agora podem, através da utilização de tal ferramenta. A aplicação realiza a execução dos códigos no ambiente servidor, retornando na tela destes o resultado da execução de seus algoritmos.

A arquitetura desenvolvida é capaz de atender à demandas distintas dos usuários, representando uma alternativa considerável para atender as necessidades da comunidade científica em geral, bem como da UFSM, provendo um ambiente completo para desenvolvimento e testes de aplicações paralelas com foco em placas gráficas.

Adicionalmente, o trabalho também apresenta um estado da arte sobre o ensino de programação paralela além de ferramentas e ambientes de apoio ao ensino desta área. Ao decorrer do texto, são apresentados conceitos sobre programação paralela e em GPUs, bem como, os principais *frameworks* para programação paralela em GPUs, demonstrando a importância e a necessidade de uma maior atenção a esta área.

1.3 Organização do Texto

O presente trabalho está organizado em sete capítulos. O capítulo 2 apresenta uma revisão de literatura sobre os principais assuntos relacionados a solução proposta neste trabalho. Neste capítulo objetiva-se apresentar os principais conceitos da área de programação paralela, algumas diferenças acerca das arquiteturas de CPU e GPU, programação paralela em GPU, bem como suas principais arquiteturas: o OpenCL e o CUDA.

Já o capítulo 3 discute o estado da arte sobre o ensino de programação paralela, alguns ambientes de apoio ao ensino de tal área, além de apresentar algumas pesquisas atuais sobre GPUs, relacionando tais pesquisas ao trabalho aqui apresentado, com foco no apoio ao ensino de programação paralela em GPU.

O capítulo 4 descreve a arquitetura do ambiente GPUHelp. O objetivo deste capítulo é apresentar ao leitor os principais conceitos e funcionalidades, de forma abstrata, relacionadas ao ambiente.

No capítulo 5 é apresentado o desenvolvimento do ambiente proposto neste trabalho. O capítulo é iniciado com uma breve análise das tecnologias utilizadas e que compõem o ambiente, seguindo com o desenvolvimento da aplicação propriamente dita. O fechamento do capítulo é realizado com a apresentação das funcionalidades do ambiente.

O capítulo 6 apresenta os resultados obtidos com o desenvolvimento do GPUHelp. São demonstrados os resultados da avaliação qualitativa do ambiente. São apresentados ainda, os dados coletados através da apresentação do mesmo a alunos em fase de conclusão do curso de Ciência da Computação da UFSM. As informações apresentadas neste capítulo servem de indicadores dos resultados atingidos com o desenvolvimento deste trabalho.

Por fim, o capítulo 7 conclui este trabalho, apresentando as considerações finais e direcionamentos para trabalhos futuros. O apêndice A apresenta o manual de utilização da ferramenta para execução remota de códigos.

2 FUNDAMENTAÇÃO

Este capítulo apresenta um referencial teórico sobre os principais conceitos relacionados a este trabalho. Inicialmente, é introduzida a área de computação de alto desempenho, bem como, os conceitos sobre programação paralela e arquiteturas paralelas. Em seguida, é apresentada a área de programação paralela em arquiteturas de GPUs. Em seguida, são demonstradas as principais características das GPUs além de um breve histórico do surgimento das mesmas. É realizado ainda, de forma sucinta, um comparativo entre as arquiteturas de CPU e GPU, com foco em questões relacionadas ao desempenho. Por fim, são apresentadas as arquiteturas de programação paralela em GPU, o CUDA e o OpenCL.

2.1 Computação de Alto Desempenho

A busca por um maior desempenho computacional nas diferentes pesquisas científicas em computação de alto desempenho *(High Performance Computing)* contribui para o desenvolvimento de novas tecnologias de *software* e *hardware*, sendo que a criação destas impacta no surgimento de algoritmos ainda mais complexos que exigem um poder computacional ainda maior (STALLINGS, 2005) (BUYYA et al., 1999).

Através da popularização dos computadores pessoais, a computação de alto desempenho passou a se utilizar não mais de processadores desenvolvidos especificamente para a computação científica, mas sim de processadores destinados a uso geral (STALLINGS, 2010) (TANENBAUM, 2010). Isso impactou em uma maior empregabilidade desta técnica em diferentes áreas, tendo em vista a redução dos custos a serem investidos em tecnologia necessária e os ganhos obtidos com a utilização da computação de alto desempenho (COULOURIS; DOLLIMORE; KINDBERG, 2007) (TANENBAUM, 2010). Dois termos caracterizam a computação de alto desempenho: a programação paralela, que refere-se ao *software* paralelo e a arquitetura paralela, que refere-se a arquitetura de *hardware* paralelo.

2.1.1 Programação Paralela

Dentro das diferentes subáreas da computação, uma das áreas que apresentam um maior nível de dificuldade é a programação (LIU; WU; MARSAGLIA, 2012). Atualmente, a computação científica exige um alto poder de processamento, fazendo-se necessário novos paradigmas para a resolução de tais demandas (GIACAMAN, 2012) (MAROWKA, 2008) (SEBESTA,

2005).

Neste contexto, surge a área de programação paralela, possibilitando que diversos processos cooperem entre si para executar simultaneamente, de forma coordenada, com o objetivo de resolver um problema específico em comum (GEBALI, 2011) (OLIVEIRA; CARISSIMI; TOSCANI, 2001) (FOSTER, 1995). Esta área da computação tem como característica prover ao utilizador um alto poder de processamento, bem como a resolução dos mais variados problemas científicos, de forma mais eficiente que a programação normalmente aplicada na resolução dos problemas do dia a dia (COULOURIS; DOLLIMORE; KINDBERG, 2007) (MAROWKA, 2008). Porém, a computação paralela possui um alto nível de dificuldade, exigindo dedicação e uma lógica apurada, além do conhecimento e domínio da arquitetura em questão (DELISTAVROU; MARGARITIS, 2011) (MAROWKA, 2008) (MATTSON; WRINN, 2008). A principal motivação da programação maciçamente paralela é de que as aplicações desfrutem de um aumento contínuo em sua velocidade nas gerações de *hardware* futuras (HWU; KIRK, 2011).

Diversas são as motivações para se utilizar programação paralela (PACHECO, 2011), como por exemplo, a possibilidade de resolver diferentes tipos de problemas computacionais grandes ou apenas não ter que se sujeitar as limitações físicas (na velocidade de transmissão das informações através dos componentes) e econômicas (segundo (NVIDIA, 2013b) é mais barato juntar vários processadores menos velozes do que produzir um processador mais rápido) da computação sequencial (BARNEY; LIVERMORE, 2013) (SEBESTA, 2005).

2.1.2 Arquiteturas Paralelas

Um computador paralelo, de forma simplificada, pode ser definido como sendo um conjunto de processadores capazes de trabalhar de forma coordenada para resolver um dado problema (STALLINGS, 2005) (COULOURIS; DOLLIMORE; KINDBERG, 2007) (FOSTER, 1995). Um sistema paralelo pode ser classificado por diversas formas, uma delas é pela taxonomia de Flynn (STALLINGS, 2005) (FOSTER, 1995).

Flynn (FLYNN, 1972) classifica as arquiteturas de máquina da seguinte maneira: SISD, SIMD, MISD e MIMD. A Figura 2.1 apresenta a classificação das arquiteturas de computadores segundo Flynn (FLYNN, 1972).

Uma arquitetura SISD *(Single Instruction stream and Single Data stream)*, segundo Flynn (FLYNN, 1972) é uma arquitetura que possibilita a execução de uma única instrução em um único conjunto de dados, ou seja, não há paralelismo. Esta arquitetura corresponde ao modelo

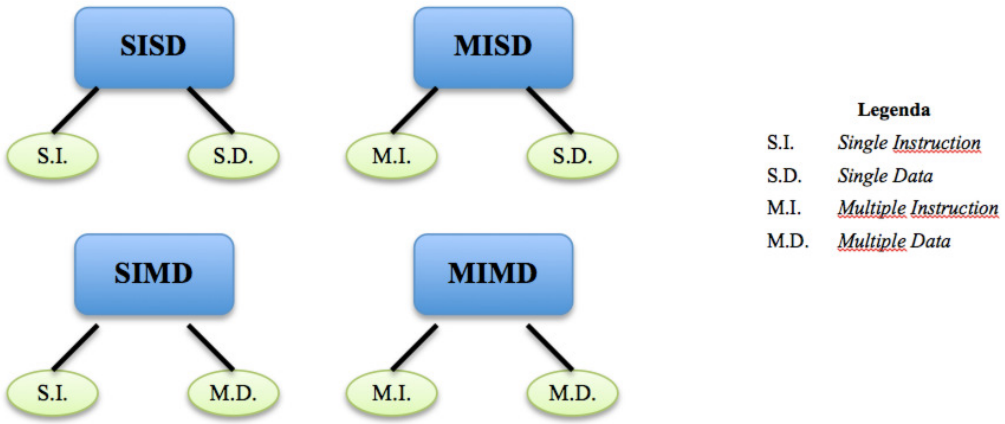


Figura 2.1: Classificação das arquiteturas de computadores segundo Flynn 1972

de von Neumann, e foi substituída por máquinas do tipo SIMD e MIMD a pouco mais de uma década.

Já máquinas SIMD *(Single Instruction stream and Multiple Data stream)* para (FLYNN, 1972), são arquiteturas onde uma unidade de processamento pode executar uma mesma instrução e pode operar em múltiplos conjuntos de dados. Este modelo pode ser empregado em aplicações que necessitem executar a mesma operação em grandes vetores ou matrizes, tirando vantagem deste tipo de arquitetura.

Enquanto que máquinas MISD *(Multiple Instruction stream and Single Data stream)* caracterizam-se por serem arquiteturas em que múltiplas unidades de processamento executam diferentes instruções de forma simultânea, em um mesmo conjunto de dados (FLYNN, 1972).

Por fim, Flynn (FLYNN, 1972) classifica as arquiteturas MIMD *(Multiple Instruction stream and Multiple Data stream)* como arquiteturas onde múltiplas unidades de processamento podem executar diferentes instruções de forma simultânea, em diferentes conjuntos de dados.

2.2 Programação Paralela em GPU

O poder computacional disponível atualmente aumentou de forma considerável nos últimos anos (CUDA, 2013c). De forma concorrente a tal evolução, os *softwares* passaram a exigir uma maior capacidade de processamento (FLYNN; RUDD, 1996) (NVIDIA, 2013b).

Frente a tal necessidade, pesquisadores passaram a utilizar as GPUs *(Graphics Processing Units)* na resolução dos mais variados problemas, com o objetivo de tirar proveito do grande poder de processamento oferecido por tais arquiteturas (NVIDIA, 2013b). Isso somente se

tornou possível, devido ao fato de que as GPUs atuais conseguem ir além de apenas processar tarefas de vídeo. Tais arquiteturas conseguem atuar como poderosos processadores paralelos devido a sua arquitetura maciçamente paralela (CUDA, 2013c) (CUDA, 2013d).

Até meados de 2006, as placas gráficas eram difíceis de programar e exigiam que os desenvolvedores possuíssem conhecimentos aprofundados de APIs gráficas (CUDA, 2013b).

Segundo Kirk e Whu (HWU; KIRK, 2011), os pesquisadores têm atingido ganhos de velocidade superiores a 100x para algumas aplicações adaptadas para GPUs. E a tendência é que cada vez mais aplicações tenham seus códigos reescritos para códigos paralelos (MAROWKA, 2008) a serem executados em GPUs, com o objetivo de tirar proveito do poder computacional destas arquiteturas (CUDA, 2013a).

Kirk e Whu (HWU; KIRK, 2011) demonstram que em 2009, a razão entre GPUs e CPUs com múltiplos núcleos para a vazão máxima de cálculo era de 10 para 1. Esse fenômeno é demonstrado na Figura 2.2. O ritmo de crescimento da velocidade das GPUs continua a aumentar incessantemente.

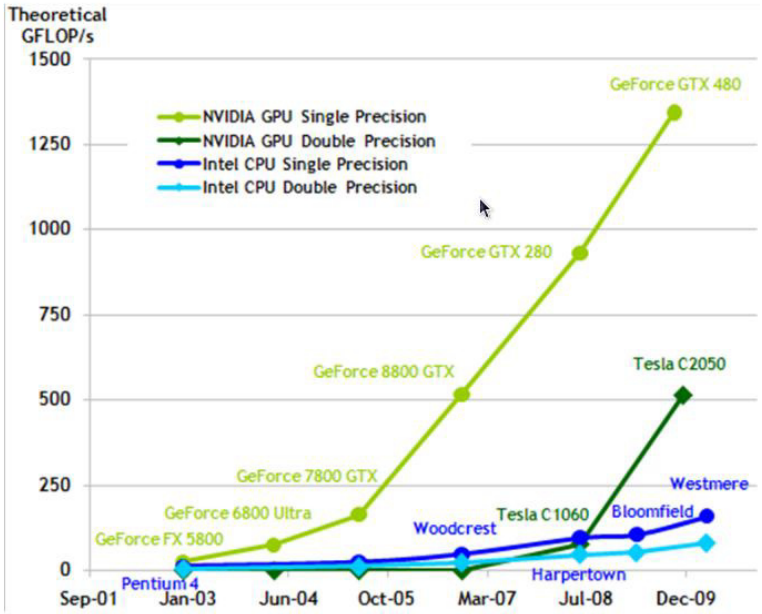


Figura 2.2: Diferencial crescente de performance entre GPUs e CPUs (CUDA, 2013a)

Atualmente, essa grande diferença de desempenho tem motivado muitos desenvolvedores de aplicações a mudarem os trechos computacionalmente pesados de seu *software* para execução em GPU (NVIDIA, 2013b) (KHRONOS, 2013). É fato que estas partes também sejam o principal alvo da programação paralela: quanto mais trabalho a ser realizado, maior a oportunidade de dividir o trabalho entre trabalhadores atuando paralelamente (PACHECO, 2011)

(SILVEIRA; SILVEIRA JR.; CAVALHEIRO, 2010a) (FOSTER, 1995).

Porém, a programação paralela tem como característica o fato de ser uma área complexa (MAROWKA, 2008). Em se tratando de programação paralela em GPUs, a complexidade torna-se ainda maior. Uma das justificativas, relacionada a área de GPUs, tem foco nas arquiteturas disponíveis para programação paralela em GPUs. Tanto os *frameworks* CUDA quanto OpenCL, vêm de projetos distintos, e exigem certa compreensão de seus fundamentos. Isso pode ser algo difícil, inicialmente, para aquelas pessoas acostumadas com programação paralela em arquiteturas de CPU. Outra característica está atrelada a complexidade na tarefa de dividir o problema para ser processado de forma simultânea nos vários processadores disponíveis nas GPUs atuais.

A seguir, serão apresentados conceitos das GPUs *(Graphics Processing Units)*, as GPGPUs *(General Purpose Computing on Graphics Processing Units)* e as arquiteturas para programação paralela em GPUs: OpenCL e CUDA.

2.2.1 GPU *(Graphics Processing Unit)*

Uma GPU *(Graphics Processing Unit)* é um dispositivo dedicado a realizar operações gráficas, presente em *desktops*, *notebooks* e *smartphones* (CUDA, 2013c). As GPUs tem conquistado seu espaço na área de programação de alto desempenho (NVIDIA, 2013b) (CUDA, 2013c). Cientistas e pesquisadores têm dedicado esforços na utilização de arquiteturas de GPGPU *(General Purpose Computing on Graphics Processing Unit)* para a resolução dos mais variados problemas envolvendo diferentes áreas científicas (GPGPU, 2013) (HWU; KIRK, 2011). Elas geralmente são adequadas para cálculos de alta-performance, que apresentam paralelismo de dados para explorar a ampla arquitetura SIMD *(Single Instruction Multiple Data)* da arquitetura das GPUs (HWU; KIRK, 2011).

As GPUs podem ser classificadas de forma simplificada em dois grupos: placa de vídeo dedicada, sendo aquela cuja função é trabalhar com alto poder de processamento visual e demais tarefas relacionadas, tendo em sua arquitetura seu próprio processador e memória, e as placas de vídeo de memória compartilhada, que utilizam parte da memória RAM do computador para executar as atividades que envolvam processamento visual (NVIDIA, 2013a) (INTEL, 2013).

Atualmente, embora a comercialização de *desktops* e *notebooks* esteja em alta crescente, uma parte considerável destes dispositivos ainda possuem placas gráficas compartilhadas (NVIDIA, 2013b), que são bem menos poderosas do que as placas gráficas dedicadas (INTEL, 2013).

Através da utilização de linguagens de alto nível, aplicativos acelerados por GPUs conseguem executar partes sequenciais de seus trabalhos na CPU – que é otimizada para performance com um único segmento *(thread)* – ao mesmo tempo em que aceleram o processamento em paralelo da GPU. Esta técnica é conhecida como computação com GPU (CUDA, 2013c).

Uma GPGPU *(General Purpose Computing on Graphics Processing Unit)* (GPGPU, 2013) utiliza-se do poder das GPUs não apenas para realizar tarefas gráficas, mas também para tarefas como processamento de imagens, inteligência artificial, cálculo numérico, visão computacional entre outros (CUDA, 2013c) (KHRONOS, 2011). GPGPUs podem ser aplicadas na resolução de variados problemas paralelos complicados, como por exemplo, modelagem climática, computação dinâmica de fluídos entre outros.

2.2.2 Um Breve Histórico das GPUs

As primeiras GPUs foram projetadas como aceleradores gráficos, suportando somente *pipelines* de função fixa específicos (CUDA, 2013b).

Ao final da década de 90, surge a primeira GPU da empresa NVIDIA, época em que o *hardware* começou a tornar-se cada vez mais programável. Desde então, pesquisadores e desenvolvedores passaram a dedicar uma maior atenção ao assunto, dando início assim, ao movimento das GPGPUs *(General Purpose Computing on Graphics Processing Units)* (GPGPU, 2013).

Porém, as GPGPUs deste período eram complexas e existiam poucas pessoas aptas a trabalhar com tal arquitetura (HWU; KIRK, 2011). Frente a tal necessidade pesquisadores da Universidade de Stanford reuniram-se para melhorar a arquitetura das GPUs.

Em 2003, era anunciado o primeiro modelo de programação de ampla adoção a aprimorar o poder da linguagem de programação C com construções de paralelismo de dados, o Brook, desenvolvido por uma equipe de pesquisadores liderada por Ian Buck (NVIDIA, 2013b). Usando diversos conceitos, o compilador Brook ajudou a revelar a GPU como um processador de propósito geral em uma linguagem de alto nível (CUDA, 2013b).

Logo após seu revolucionário lançamento, a empresa NVIDIA, percebeu a amplitude e o potencial do projeto, convidando assim, Buck a juntar-se à empresa, com o propósito de combinar um *hardware* extremamente rápido, a ferramentas intuitivas de *software* e *hardware*, dando início ao desenvolvimento de uma solução cujo propósito era acoplar a linguagem C de forma nativa, nas GPUs. Dessa forma, em 2006, a NVIDIA apresentava o CUDA, a primeira solução no mundo para computação de propósitos gerais em GPUs (NVIDIA, 2013b) (CUDA, 2013a).

2.2.3 CPUs X GPUs

As arquiteturas de GPUs *(Graphics Processing Unit)* e CPUs *(Central Processing Unit)* vêm de projetos distintos. A arquitetura de uma CPU é otimizada para o desempenho de código sequencial. Já a arquitetura de uma GPU, é modelada para trabalhar com uma quantidade de cálculos computacionalmente grande (HWU; KIRK, 2011). A Figura 2.3 representa as diferenças nas arquiteturas dos dois microprocessadores.

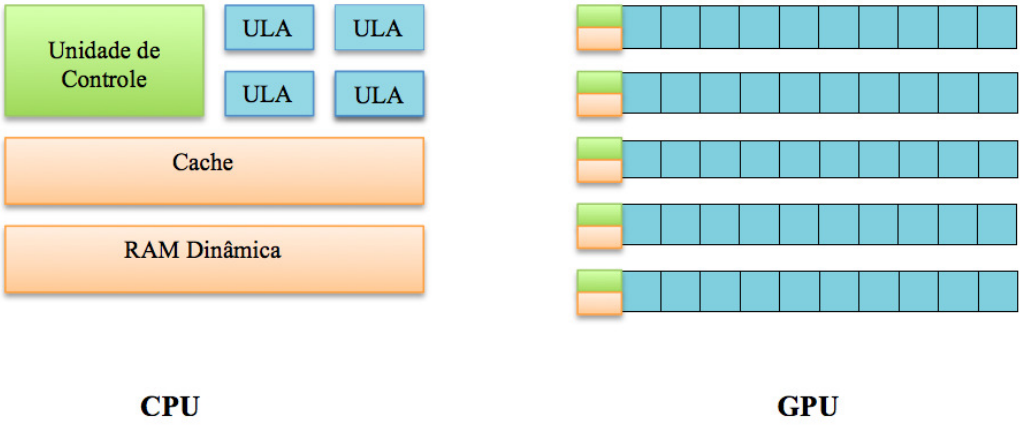


Figura 2.3: Diferença na filosofia de projeto entre CPU e GPU

Computação com GPU é o uso de uma GPU em conjunto com uma CPU para acelerar aplicações científicas e de engenharias de uso geral (CUDA, 2013c). A combinação CPU + GPU resulta em uma poderosa arquitetura, isto porque as CPUs são compostas de alguns núcleos otimizados para o processamento serial, enquanto que as GPUs consistem de milhares de núcleos menores e mais eficientes, projetados para desempenho paralelo. A combinação CPU + GPU, conhecida como GPGPU, é caracterizada por partes do código sendo executado na CPU e partes na GPU (CUDA, 2013c) (CUDA, 2013a).

A Figura 2.4 representa um comparativo entre estas duas arquiteturas e a quantidade de cálculos de ponto flutuante por segundo realizada por cada uma delas (CUDA, 2013d). A partir de meados de 2003, a GPU já se mostrava mais poderosa do que a CPU, sendo que em 2005, esta diferença ficou ainda maior (CUDA, 2013c). Vários são os exemplos (OPENEYE, 2013a) (MOLECULE, 2013) (CHARMM, 2013) que comprovam que a GPU pode atuar em conjunto com a CPU, ficando a cargo desta última apenas tarefas relacionadas ao fluxo de execução do programa (NVIDIA, 2013b) (CUDA, 2013e).

As GPUs evoluíram para um processador de vários núcleos, tornando-as assim, uma opção interessante para o desenvolvimento de aplicações em sistemas paralelos (SILVEIRA; SIL-

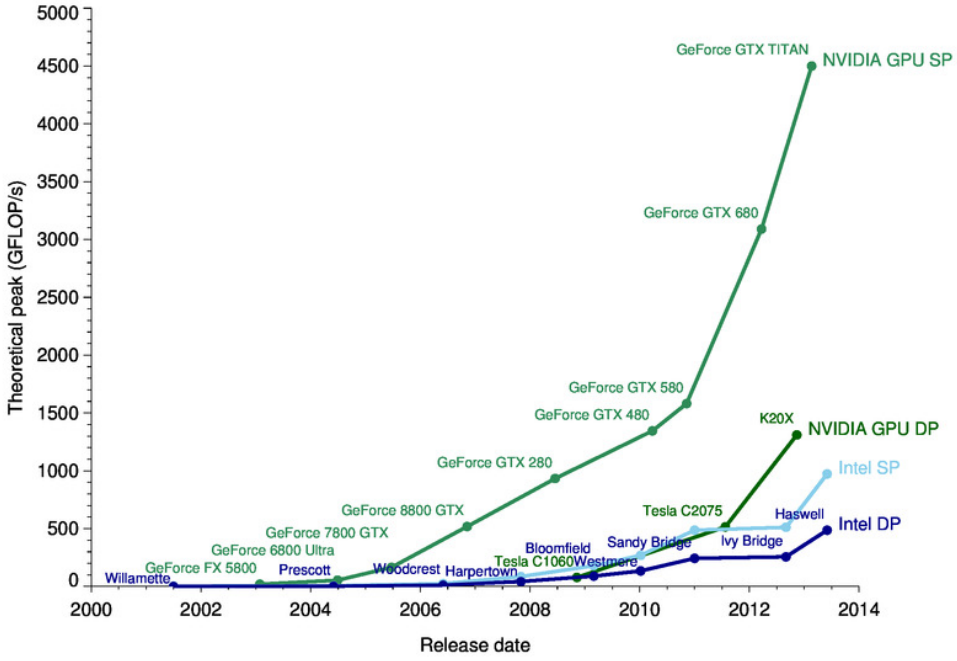


Figura 2.4: Comparativo de performance entre CPU e GPU (NVIDIA, 2013a)

VEIRA JR.; CAVALHEIRO, 2010a) (BERILLO, 2008). Pode-se perguntar por que existe uma diferença de desempenho tão grande entre GPUs e CPUs com múltiplos núcleos. A resposta está na Figura 2.5, que apresenta um comparativo entre a quantidade de processadores de uma GPU e uma CPU (CUDA, 2013c).

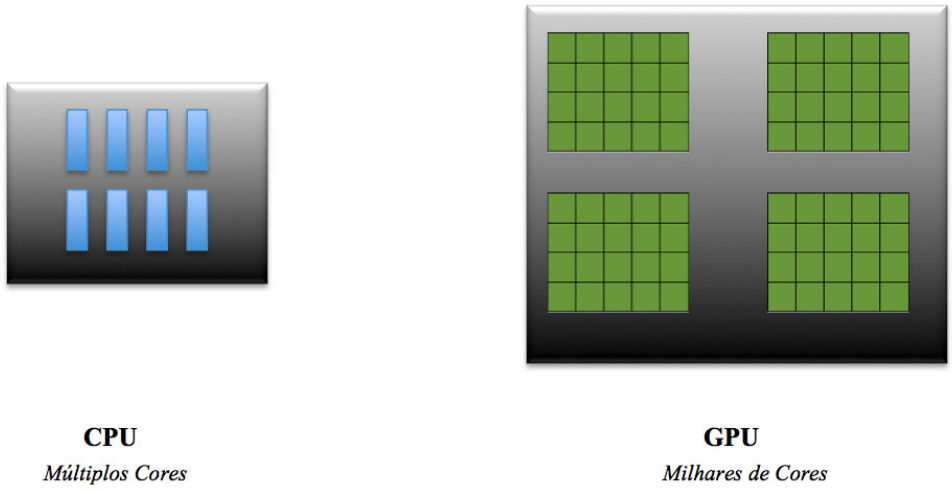


Figura 2.5: Comparativo entre processadores *(cores)* das CPUs e das GPUs

Contudo, Kirk e Whu (HWU; KIRK, 2011) alertam para o fato de que as GPUs são projetadas como mecanismos de cálculo numérico, e estas não funcionarão bem em algumas tarefas em que as CPUs são naturalmente projetadas para funcionarem bem, como por exemplo, apli-

cações que necessitem de espera por resultados, ou ainda, aplicações sequenciais; dessa forma, o correto é esperar que a maioria das aplicações utilize tanto CPUs quanto GPUs, executando trechos sequenciais na CPU e partes com alto índice de paralelismo nas GPUs.

É importante observar que o desempenho não é o único fator decisivo quando os desenvolvedores de aplicações selecionam os processadores para rodar suas aplicações (COULOURIS; DOLLIMORE; KINDBERG, 2007). A adoção e utilização pela comunidade científica é um fator muito importante na escolha da arquitetura a ser utilizada (STALLINGS, 2010). Com o advento das GPUs com muitos núcleos, a comunidade científica despertou forte interesse por tal arquitetura, devido a sua capacidade de processar uma grande quantidade de informações em paralelo (NVIDIA, 2013b) (NVIDIA, 2013a).

2.2.4 Aplicações Aceleradas por GPUs

De acordo com (HWU; KIRK, 2011), apesar de seu alto desempenho demonstrado em aplicações paralelas, os processadores-núcleo da GPU são originados de um projeto relativamente simples. Técnicas mais contundentes serão introduzidas a cada nova arquitetura, para aumentar a utilização real das unidades de cálculo. Devido ao alto poder computacional fornecido pela utilização de GPUs, muitas aplicações estão sendo reescritas para arquiteturas de GPU (CUDA, 2013e).

Algumas aplicações possuem um aumento considerável em seu desempenho, quando traduzidas para arquiteturas de GPUs (CUDA, 2013e) (CUDA, 2013d). ROCS (OPENEYE, 2013b) é uma aplicação para comparação e busca de similaridade de formato para estruturas moleculares, ganhando um aumento na execução de até 3000x em algumas suas rotinas (OPENEYE, 2013a) (CUDA, 2013e). VMD (ILLINOIS, 2013a) foi projetado para modelagem, visualização e análise de sistemas biológicos, obteve um *speedup* de aproximadamente 125x (ILLINOIS, 2013a) (CUDA, 2013e). Outras aplicações também se beneficiaram do poder de processamento oferecido pelas GPUs, sendo o caso de (YONGCHAO, 2013) (PLIMPTON; THOMPSON; CROZIER, 2013) (VALIEV et al., 2010) (ILLINOIS, 2013b). Em computação de alto desempenho, *speedup* é o termo utilizado para avaliar o ganho de desempenho em programas paralelos, sendo obtido através do cálculo entre o tempo de execução paralelo e o tempo de execução sequencial (SEBESTA, 2005) (COULOURIS; DOLLIMORE; KINDBERG, 2007). A Tabela 2.1 apresenta algumas aplicações que tiveram seus códigos reescritos para arquiteturas de GPUs bem como o *speedup* obtido em cada situação.

Tabela 2.1: Algumas aplicações otimizadas pela utilização de arquiteturas de GPU

Aplicação	***Speedup***
Abalone	4 a 29 vezes
AMBER	4 a 29 vezes
CHARMM	4 a 29 vezes
FastROCS	800 a 3000 vezes
VMD	125 vezes

Frente a tais exemplos, torna-se visível os ganhos positivos ao se utilizar computação com GPU para a realização dos mais variados tipos de tarefas.

2.3 Arquiteturas de Programação Paralela em GPUs

A constante evolução das arquiteturas para programação paralela resultou em uma maior facilidade no desenvolvimento de aplicações GPGPU, impactando assim, no aumento de sua utilização.

A seguir serão apresentadas duas maneiras de explorar as capacidades de computação paralela das unidades de processamento gráfico (GPUs) e unidades centrais de processamento (CPUs). Durante a escrita desta Dissertação, duas interfaces de programação de aplicação (APIs) destacavam-se em utilização: CUDA e OpenCL.

2.3.1 CUDA

CUDA (CUDA, 2013b) é uma plataforma de computação paralela e um modelo de programação desenvolvido pela empresa NVIDIA (NVIDIA, 2013a). Esta arquitetura possibilita aumentos significativos na performance computacional tirando proveito do poder de processamento das atuais GPUs (CUDA, 2013a).

Segundo (HWU; KIRK, 2011), para um programador CUDA, a aplicação computacional é constituída de forma geral de um *host* (hospedeiro), sendo este uma CPU tradicional, e um ou mais *devices* (dispositivos), sendo estes, processadores massivamente paralelos.

Na arquitetura CUDA, o *host* e os *devices* possuem espaços de memória separados. Isso reflete a realidade de que os *devices* normalmente são placas de *hardware* que possuem sua própria memória de acesso aleatório dinâmica (DRAM) (HWU; KIRK, 2011).

Ainda segundo (HWU; KIRK, 2011), a estrutura de um programa CUDA é composta de uma ou mais partes que são executadas no *host* (CPU) ou em algum *device* (GPU). As partes que envolvem pouco ou nenhum paralelismo, são executadas no código do *host*. Já as partes

que envolvem uma certa quantidade de paralelismo, são executadas no código do *device*.

Através da literatura analisada, pode-se perceber que a arquitetura CUDA tem se mostrado eficiente e tem proporcionado um *speedup* considerável nas aplicações em que é aplicada (CUDA, 2013d).

2.3.2 OpenCL

O OpenCL *(Open Computing Language)* (KHRONOS, 2013) é o primeiro padrão livre para programação paralela multiplataforma. A arquitetura OpenCL consegue prover um aumento na velocidade de aplicações em diferentes áreas, estando presente em áreas científicas, de engenharias e medicina (KHRONOS, 2011) (KHRONOS, 2013).

O OpenCL foi desenvolvido com base na necessidade de um padrão de desenvolvimento de aplicações de alto desempenho para a grande variedade de plataformas de computação paralela em constante crescimento (SILVEIRA; SILVEIRA JR.; CAVALHEIRO, 2010b) (HWU; KIRK, 2011). O desenvolvimento do OpenCL foi iniciado pela Apple e continuado pelo Grupo Khronos (KHRONOS, 2011).

Modelos de programação paralela heterogênea CPU/GPU, como CUDA, normalmente são específicos para uma plataforma, um fornecedor de *hardware* em específico. Este fator torna-se uma forte limitação, visto o limite imposto ao desenvolvedor ao obriga-lo a acessar o poder computacional das CPUs, GPUs e outros tipos de unidades de processamento a partir de uma única base de código fonte multiplataformas (KHRONOS, 2013) (SILVEIRA; SILVEIRA JR.; CAVALHEIRO, 2010a).

Existem algumas semelhanças entre as arquiteturas CUDA e OpenCL (KHRONOS, 2013) (SILVEIRA; SILVEIRA JR.; CAVALHEIRO, 2010b). Contudo, o OpenCL possui um modelo de gerenciamento de plataforma e *device* mais complexo, refletindo seu suporte a portabilidade e multiplataforma (HWU; KIRK, 2011) (SILVEIRA; SILVEIRA JR.; CAVALHEIRO, 2010a). De forma comparativa a arquitetura CUDA, o OpenCL modela um sistema de computação heterogêneo como um *host* e um ou mais *devices* OpenCL. Sendo o *host* uma CPU tradicional que executa o programa *host*.

Um programa OpenCL consiste em duas partes: *kernels* que executam em um ou mais *devices* OpenCL e um programa *host* que gerencia a execução dos *kernels* (AMD, 2013b). Alguns recursos do OpenCL são opcionais e podem não ser aceitos em todos os *devices*, de modo que um código OpenCL desenvolvido para ser portável, deve evitar o uso destes recursos

(HWU; KIRK, 2011).

A arquitetura OpenCL possui um modelo de gerenciamento de *device* muito mais complexo que a arquitetura CUDA. Tal complexidade vêm do fato de o OpenCL suportar múltiplas plataformas de *hardware*.

O padrão OpenCL apresenta-se como uma forte tendência a substituir a arquitetura CUDA em futuro não muito distante, visto sua característica portável e sua heterogeneidade (KHRONOS, 2013) (KHRONOS, 2011).

2.4 Considerações

Neste capítulo foram apresentados os principais conceitos necessários para a compreensão do ambiente desenvolvido neste trabalho. Foram apresentados conceitos sobre a área de programação paralela e de programação paralela em arquiteturas de GPU. Foi realizado um comparativo entre o poder de processamento das arquiteturas de CPU e de GPU. Foram apresentados ainda, os conceitos históricos das GPUs, bem como as aplicações aceleradas e o ganho de desempenho com a utilização destas arquiteturas. Por fim, apresentou-se as arquiteturas para programação paralela em GPU, o CUDA e o OpenCL.

O Capítulo 3 apresenta as principais pesquisas relacionadas ao desenvolvimento deste trabalho: o ensino de programação paralela além dos ambientes de apoio ao ensino de programação paralela disponíveis atualmente.

3 TRABALHOS CORRELATOS

Este capítulo apresenta pesquisas relacionadas ao foco deste trabalho. Inicialmente, é demonstrado o estado da arte com relação ao ensino da área de programação paralela. Em seguida, são apresentados os ambientes de apoio ao ensino de programação paralela disponíveis atualmente. Por fim, são apresentados os trabalhos envolvendo a utilização de GPUs. Tais pesquisas possuem relação direta ou indireta com o propósito deste trabalho: o desenvolvimento de um ambiente de apoio à área de programação paralela em arquiteturas de GPU.

3.1 Ensino de Programação Paralela

Embora a programação paralela apresente-se como uma maneira eficiente de solucionar problemas que envolvam grande poder computacional, esta é uma área que destaca-se também por sua complexidade. Ensinar programação paralela não é uma tarefa simples. A seguir, serão apresentadas pesquisas relacionadas ao ensino de programação paralela.

3.1.1 Revisão de Trabalhos

Em sua pesquisa, (MAROWKA, 2008) apresenta um *framework* para um curso introdutório de programação paralela voltado a cursos de ciência da computação em níveis de graduação. A disciplina foi projetada com foco na flexibilidade: seções teóricas com tópicos fundamentais para o ensino de programação paralela, complementadas por seções práticas, que envolvem desde questionários a problemas que devem ser solucionados através do uso de algoritmos. O autor organiza a disciplina em dez seções teóricas e três laboratórios práticos, onde são trabalhados os aspectos teóricos fundamentais para a disciplina, bem como, as seções práticas que visam fornecer uma melhor compreensão de como cada conteúdo funciona e deve ser implementado de forma prática.

Já o trabalho desenvolvido por Mattson e Wrinn (MATTSON; WRINN, 2008), tem o objetivo de abordar os erros cometidos ao longo dos anos no ensino de programação paralela. A pesquisa inicia com um histórico sobre a computação paralela. Os autores afirmam que problemas relacionados à programação paralela são abordados, em computação de alto desempenho, há pelo menos 25 anos, chegando a um resultado de que hoje, há ainda apenas uma pequena fração de desenvolvedores especializados em escrever códigos paralelos. Os autores mencionam que, fora algumas exceções, somente estudantes de pós-graduação e outra pequena parte de

desenvolvedores projetam *software* paralelo. Mesmo frente a aplicações que envolvam cálculos intensivos, onde técnicas de programação paralela podem ser aplicadas de forma eficiente, os desenvolvedores profissionais não escrevem *software* paralelo.

Na pesquisa desenvolvida por Freitas (FREITAS, 2012), este discute a importância da correta introdução da disciplina de programação paralela em cursos de graduação. O autor reforça a importância da utilização de programação paralela nos processadores *multicores* atuais. Além disso, realiza uma explanação sobre a necessidade de serem aplicados novos esforços nas técnicas de ensino de programação paralela em cursos de graduação. O artigo descreve um experimento em um curso de Ciência da Computação durante dois anos. O principal assunto trabalhado é o momento mais apropriado para a introdução de modelos de programação paralela a fim de melhorar na qualidade de aprendizagem.

O trabalho desenvolvimento por (GIACAMAN, 2012) detalha a experiência no ensino de programação paralela em um curso de ciência da computação. O objetivo principal é de introduzir os estudantes ao desenvolvimento de complexas aplicações de *software*. O curso visa expor aos estudantes os princípios de programação, *frameworks*, técnicas de teste de *software* e aplicações *multi threading*. A pesquisa brevemente retrata algumas das analogias acerca da técnica de programação paralela e descreve uma abordagem eficiente para o desenvolvimento de habilidades em computação paralela. Acredita-se que uma abordagem teórica acompanhada de exemplos práticos tem um maior impacto para o aprendizado dos conceitos de programação paralela. Dessa forma, isso não apenas aumenta as chances de obter-se um maior sucesso no ensino e aprendizagem de tais conceitos, mas apresenta uma experiência positiva na aplicação de técnicas de computação paralela em necessidades futuras, por parte dos alunos, auxiliando assim, tanto em seu desenvolvimento profissional quanto acadêmico. O retorno positivo obtido por parte dos alunos que foram avaliados ao longo da pesquisa revela também que a combinação entre conceitos teóricos e demonstrações de código sendo executadas ao vivo colaboram fortemente para um melhor resultado ao final do curso.

Frente ao lançamento das novas arquiteturas de microprocessadores a partir de 2003, e ao lançamento de arquiteturas para programação paralela em GPUs a partir de 2007, os autores Iparraguirre, Friedrich e Coppo (IPARRAGUIRRE; FRIEDRICH; COPPO, 2012), decidiram em 2008 ensinar Computação Paralela e Distribuída aos alunos dos cursos de Engenharia Elétrica e da Computação na Universidade Bahía Blanca, Argentina. A partir de 2011, estes passaram a ofertar a disciplina de Processamento Paralelo, de forma eletiva para alunos de tais cursos.

Após uma pesquisa realizada em conjunto com os alunos, os autores reavaliaram e reestruturaram a estrutura curricular da disciplina, com o objetivo de tornar o aprendizado mais eficiente. A mudança impactou na alteração da ordem em que as arquiteturas de programação paralela seriam trabalhadas, criando uma nova sequência de apresentação: após as seções introdutórias à disciplina, os alunos eram apresentados as arquiteturas OpenMP, Pthreads, OpenCL e, por fim, MPI.

3.1.2 Considerações

Nas pesquisas apresentadas em (MAROWKA, 2008) (MATTSON; WRINN, 2008) (GIACAMAN, 2012) (IPARRAGUIRRE; FRIEDRICH; COPPO, 2012), fica evidente a necessidade do ensino de programação paralela através de constante prática. Seções teóricas são importantes, porém, ensinar programação paralela sem uma abordagem prática é inútil (MAROWKA, 2008). Os autores (IPARRAGUIRRE; FRIEDRICH; COPPO, 2012) demonstram que a correta abordagem das seções práticas de programação paralela faz com que os alunos obtenham uma melhor compreensão das premissas envolvendo tal área, despertem um maior interesse por esta e venham a utilizar tais conceitos em oportunidades futuras, na resolução de variados tipos de problemas.

Giacaman (GIACAMAN, 2012) aprimorou a maneira de ensinar programação paralela introduzindo demonstrações de código sendo executadas *ao vivo*, dentro das seções teóricas sobre a disciplina. Para (FREITAS, 2012) a disciplina de programação paralela deve ser trabalhada após as disciplinas de programação em C e arquitetura de computadores. Dessa forma, o aluno terá uma visão mais clara dos conceitos acerca da programação paralela.

Para (MAROWKA, 2008) (MATTSON; WRINN, 2008) (GIACAMAN, 2012) a ausência de um ambiente de programação paralela onde os alunos pudessem treinar os exemplos trabalhados ao longo da disciplina teve um impacto negativo. Os autores (MAROWKA, 2008) (MATTSON; WRINN, 2008) (GIACAMAN, 2012) e (FREITAS, 2012) salientam a importância no desenvolvimento de ambientes de programação paralela. Tais ambientes devem ser disponibilizados aos alunos, despertando assim, seu interesse pela disciplina de programação paralela. Já (MATTSON; WRINN, 2008) reforça que mesmo ao longo de 25 anos da área de programação paralela, uma série de erros continuam a se repetir, e, segundo o autor, um dos principais é a ausência de ambientes de programação paralela a serem disponibilizados aos alunos da disciplina.

Ao serem observadas as pesquisas acima, torna-se evidente a necessidade de um ambiente

de apoio ao ensino e prática da técnica de programação paralela. Tal fator é comprovado cientificamente ao longo destas e de outras pesquisas relacionadas à área de programação paralela. A seguir, serão apresentados alguns ambientes desenvolvidos para o ensino de programação paralela.

3.2 Ambientes de Apoio ao Ensino de Programação Paralela

Embora atualmente uma grande diversidade de técnicas para um melhor ensino de programação paralela venham sendo desenvolvidas, alguns resultados obtidos não alcançam o objetivo esperado. A programação paralela além de exigir uma correta fundamentação teórica, necessita de prática constante. Neste sentido, nos últimos anos, a comunidade científica passou a investir em ambientes de apoio ao ensino de programação paralela. A seguir, são apresentadas pesquisas cujo foco são ambientes de apoio ao ensino de programação paralela.

3.2.1 Revisão de Trabalhos

Na pesquisa desenvolvida por Park, Kapadia, Figueiredo, Eigenmann e Fortes (PARK et al., 2000), os autores focam na deficiência de ferramentas de programação paralela no período de desenvolvimento de seu trabalho. Visando corrigir tal dificuldade, estes propõem a construção de uma ferramenta para prática de programação paralela. Os autores levaram em consideração alguns fatores cruciais durante o desenvolvimento de sua pesquisa: a necessidade de baixar, instalar e configurar *softwares* para a prática de programação e a necessidade de ter sempre ao alcance a máquina que possua tais *softwares* instalados e configurados. Para os autores, estas necessidades tornavam a prática de programação desconfortável e problemática. Com base nisso, os autores projetaram e desenvolveram um ambiente de programação paralela que pode ser executado através de navegadores *web*, facilitando assim, o desenvolvimento das aplicações. A solução apresentada por eles fornece ferramentas para o desenvolvimento e testes dos códigos, sem a necessidade de *download* ou instalação de qualquer tipo de *software*. O ambiente descrito foi implantado na Universidade de Purdue.

Ivica, Riley e Shubert (IVICA; RILEY; SHUBERT, 2009) apresentam a ferramenta StarHPC em seu trabalho, sendo este um ambiente desenvolvido como ferramenta de apoio ao ensino de programação paralela para cursos do MIT *(Massachusetts Institute of Technology)* (MIT, 2013). O ambiente desenvolvido pelos autores provê uma imagem virtual de um ambiente pré configurado para o desenvolvimento de programas paralelos nas arquiteturas OpenMP e MPI. Dessa

forma, professores das disciplinas de processamento paralelo preocupam-se apenas com o ensino dos conceitos e fundamentos desta área, devido a existência de um ambiente projetado para o desenvolvimento e testes de algoritmos paralelos. O ambiente é hospedado em uma arquitetura de computação em nuvem, alugada da empresa Amazon (AMAZON, 2013a). O ambiente virtual disponibilizado para os estudantes é composto por *softwares* para a construção de algoritmos, ferramentas de *debugging* e *profiling*, ambiente gráfico e uma instância da ferramenta VMWare Player (VMWARE, 2013) para virtualização do ambiente disponibilizado.

A solução StarHPC (IVICA; RILEY; SHUBERT, 2009) é composta por três elementos principais, conforme ilustrado pela Figura 3.1, sendo eles: uma imagem de máquina virtual utilizada pelos alunos para o desenvolvimento de códigos em seus computadores; *scripts* de administração e gerenciamento do Cluster EC2 (AMAZON, 2013b) da Amazon e uma imagem de máquina virtual executando no ambiente de computação em nuvem. Conforme descrito, a ferramenta StarHPC envolve custos para o MIT por se tratar de uma solução proprietária e fechada.

Delistavrou e Margaritis (DELISTAVROU; MARGARITIS, 2010) somam esforços em uma pesquisa cujo foco é descobrir e apresentar ferramentas que possam auxiliar no ensino e aprendizado de programação paralela no cenário atual. A pesquisa enfatiza o grande avanço desta área atualmente, demonstrando que tal avanço não é baseado apenas nas atuais linguagens para programação paralela ou nas recentes arquiteturas desenvolvidas, mas também pela indústria, com o lançamento de seus processadores *multi-core* que superam os antigos processadores tanto em capacidade de processamento quanto em questões de consumo e gerenciamento de energia. Inicialmente em seu trabalho, os autores realizam uma pesquisa sobre as atuais ferramentas de *software* disponíveis, classificando-as segundo alguns critérios pré definidos. Em seguida, os resultados de tal pesquisa são apresentados, classificando as ferramentas analisadas de acordo com suas funções principais.

O trabalho desenvolvido por Delistavrou e Margaritis (DELISTAVROU; MARGARITIS, 2011) surge da necessidade de um melhor aproveitamento das arquiteturas paralelas disponíveis atualmente no ensino de programação paralela em cursos de graduação. Em sua pesquisa, os autores integram estado da arte, ferramentas *open source* e diversas ferramentas em um ambiente de programação utilizado como ferramenta de apoio ao ensino de programação paralela nas arquiteturas OpenMP e MPI. Estes reforçam a ideia de atualizar os currículos dos cursos de computação, especialmente a disciplina de programação paralela, bem como a inserção de tal

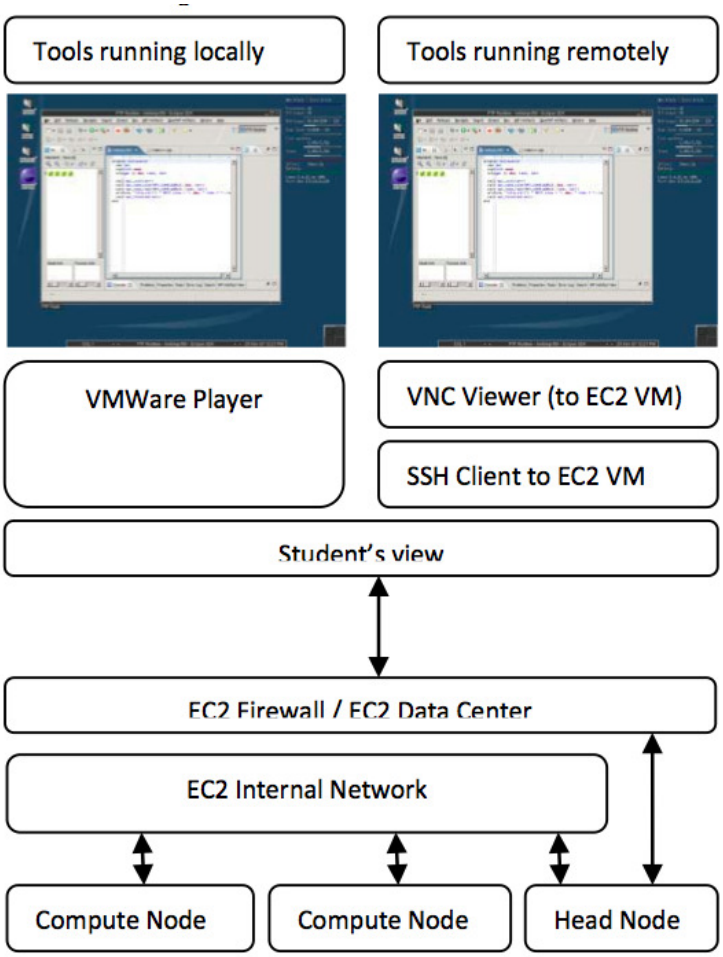


Figura 3.1: Arquitetura do ambiente StarHPC (IVICA; RILEY; SHUBERT, 2009)

disciplina em diversas áreas, como física e biologia, entre outras.

3.2.2 Considerações

Em (PARK et al., 2000) os autores apresentam um ambiente para prática de programação paralela desenvolvido por eles. Tal ambiente é executado em navegadores *Web* suportando os modelos OpenMP e MPI. A justificativa no projeto de um ambiente *Web*, segundo os autores, é de que na época da pesquisa, os computadores portáteis eram de difícil acesso e alto custo. Atualmente, o cenário computacional está repleto de tecnologias móveis e *notebooks* ultra portáteis, sem abrir mão de um alto poder de processamento.

Já em (DELISTAVROU; MARGARITIS, 2010) os autores concentram esforços em uma pesquisa cujo foco é descobrir e classificar ferramentas que possam ser utilizadas na prática de apoio ao ensino de programação paralela. Tal pesquisa tem o objetivo de pré selecionar as principais ferramentas que possam vir a ser utilizadas por eles, em uma pesquisa futura, visando

a criação de um ambiente de programação paralela baseado inteiramente em ferramentas *open source* e *softwares* já desenvolvidos. O trabalho apresentado em (DELISTAVROU; MARGARITIS, 2011) é a sequência de sua pesquisa anterior, colocando em prática o desenvolvimento do ambiente para treinamento de programação paralela. O ambiente projetado pelos autores, comporta a utilização das arquiteturas OpenMP e MPI. Estes reuniram os principais *softwares* para a criação, teste e execução de algoritmos paralelos, desenvolvidos nas arquiteturas mencionadas anteriormente. Devido ao fato de terem utilizado unicamente soluções já existentes, os autores apenas projetaram tal ambiente, e o disponibilizaram a seus alunos, sem nenhuma ferramenta própria.

A ferramenta StarHPC (IVICA; RILEY; SHUBERT, 2009) caracteriza-se como sendo uma imagem de máquina virtual, a ser utilizada pelos alunos do MIT, para treinamento da disciplina de programação paralela com suporte as arquiteturas OpenMP e MPI. Tal ambiente é utilizado como ferramenta de apoio ao longo da disciplina de programação paralela pelos professores, oferecendo um ambiente a parte do sistema operacional nativo do aluno. O ambiente StarHPC acarreta em custos para o MIT, que são pagos mensalmente a empresa Amazon, pelo aluguel da plataforma de computação em nuvem desta, onde o StarHPC está hospedado. A ferramenta StarHPC, conta ainda com um ambiente servidor, onde os alunos podem conectar-se via SSH, e rodar seus algoritmos com o objetivo de comparar o desempenho entre o ambiente local e o Cluster alugado da empresa Amazon.

Os ambientes desenvolvidos em (PARK et al., 2000) e (IVICA; RILEY; SHUBERT, 2009), possuem propósitos e limitações semelhantes: ambos são ambientes para prática de programação paralela, porém, restritos a utilização em suas respectivas Universidades.

Ao longo desta pesquisa, não foram encontrados ambientes de programação paralela com foco em GPUs. A ausência de ambientes de programação paralela para arquiteturas de GPUs impacta em prejuízos negativos a comunidade científica em geral, tendo em vista o poder computacional oferecido pelas atuais arquiteturas de GPUs. Dessa forma, o ambiente proposto neste trabalho, apresenta-se como uma possível solução para as necessidades evidenciadas ao longo deste capítulo: a falta de ambientes livres de apoio ao ensino da técnica de programação paralela em GPUs.

3.3 Pesquisas Relacionadas a Área de GPUs

Os benefícios advindos da utilização da área de GPUs fazem-se presentes em diferentes áreas científicas. Através da utilização de tais arquiteturas, os pesquisadores conseguem obter aumentos de desempenho consideráveis nas aplicações rescritas para as arquiteturas de GPU. Infelizmente, na área de programação paralela em GPUs não existem ainda pesquisas focadas no cenário educacional, como é o caso do GPUHelp. As pesquisas a seguir envolvem a aplicação de GPUs em diferentes contextos.

3.3.1 Revisão de Trabalhos

Os atuais avanços das GPUs têm proporcionado que tal arquitetura atue em diferentes áreas científicas. O trabalho apresentado em (CROIX; KHATRI, 2009) realiza uma análise das arquiteturas de GPU, bem como dos *frameworks* para programação paralela em tais arquiteturas, o CUDA e o OpenCL. Os autores abordam conceitos sobre a correta tradução do código escrito para arquiteturas de CPU, para GPU. São trabalhados ainda, conceitos sobre o modo de pensar necessário para programação paralela em GPU, além dos possíveis problemas envolvendo tal questão. Segundo os autores, o desempenho da aplicação portada ou desenvolvida para arquiteturas de GPU possui relação direta com o propósito de tal aplicação, bem como, da maneira em que esta será programada. Os autores alertam ainda para o fato de que nem todas as aplicações desenvolvidas para arquiteturas de GPU conseguirão obter o *speedup* esperado, tendo em vista o propósito das atuais arquiteturas de GPU: trabalhar com uma imensa quantidade de dados em paralelo. Caso a aplicação não permita isso, o desempenho pode ser inferior à mesma aplicação escrita para arquiteturas de CPU.

Em (SANTOS, 2010) é apresentado um método para acesso as GPUs através de virtualização, o VirtCUDA. Para tornar-se possível o acesso a placa gráfica, por meio de uma aplicação virtualizada, o autor utiliza-se de uma camada de consistência entre a aplicação e o *driver* do dispositivo, denominada *wrapper*. As funcionalidades do *wrapper* podem variar desde um simples redirecionamento de tarefas a um complexo gerenciamento do dispositivo. Através da utilização de tal método, é possível ainda, ocultar as características de *hardware* do sistema, como por exemplo, a quantidade de GPUs existentes. Dessa forma, as aplicações não sabem quantas ou quais são as GPUs existentes. O *wrapper* foi aplicado para disponibilizar o acesso as GPUs pelas máquinas virtuais. Desse modo, tal camada é responsável por encaminhar as requisições das aplicações que estão sendo executadas nas *virtual machines* para o *driver* nativo do

dispositivo, possibilitando assim, a execução das aplicações. A avaliação do trabalho apresenta o problema envolvendo a virtualização, um custo na performance das aplicações testadas.

Atualmente, aplicações que combinam arquiteturas de CPU-GPU vem chamando a atenção de diversos pesquisadores devido ao alto poder computacional obtido com a combinação de tais arquiteturas. Porém, o consumo de energia necessário para utilização de tais arquiteturas tornou-se alvo de pesquisas na área científica. Em (MA et al., 2012) é apresentada uma proposta de gestão de energia para arquiteturas heterogêneas de CPU-GPU. Até então, as pesquisas na área de gestão de energia concentravam esforços em técnicas voltadas a arquiteturas de CPU ou GPU. A pesquisa dos autores visa solucionar a deficiência na área de energia para arquiteturas heterogêneas, apresentando o GreenGPU. Tal *framework* baseia-se na carga de trabalho para cada arquitetura, de CPU e GPU, com o objetivo de fazer com que ambas terminem a execução de determinada tarefa ao mesmo tempo. Como resultado disso, o consumo de energia diminui, visto que uma arquitetura não precisa esperar pela outra, já que ambas estão sendo coordenadas para evitar o desperdício de energia.

Com a crescente utilização de GPUs em diversas áreas, pesquisadores começaram a rescrever seus códigos para tais arquiteturas. Porém, muitas vezes um esforço considerável é investido e o benefício advindo com isso é relativamente pequeno, ou seja, não há nenhuma garantia que o desempenho obtido será o esperado. Visando reduzir esse risco, foram criados modelos de desempenho com o objetivo de projetar o possível desempenho a ser obtido. Porém, as pesquisas atuais focam em medir apenas o tempo de execução da GPU. Na pesquisa apresentada em (BOYER; MENG; KUMARAN, 2013), os autores concentram esforços no desenvolvimento de um *framework* para modelagem de desempenho da GPU, que prevê tanto o tempo de execução do *kernel* quanto o tempo envolvendo a transferência dos dados. O *framework* desenvolvido inclui um analisador de dados de uso de uma sequência de *kernels* para determinar a quantidade de dados que precisam ser transferidos e um modelo de performance PCIe para determinar o tempo que os dados irão levar para serem transferidos.

3.3.2 Considerações

Na pesquisa apresentada em (CROIX; KHATRI, 2009), os autores realizam uma análise das arquiteturas de GPU disponíveis atualmente, além dos *frameworks* disponíveis para programação paralela em GPU, o CUDA e o OpenCL. São trabalhados conceitos relacionados ao modo de pensar necessário para programação paralela em GPU, além dos desafios envolvendo

tal questão.

A virtualização é uma área com diferentes aplicações. Porém, em se tratando de virtualizações de GPUs, o contexto muda. Atualmente, grande parte das soluções para virtualização que possuem suporte a emulação de GPUs são pagas. As soluções *open source* possuem algumas limitações, não contemplando tal necessidade. Dessa forma, em (SANTOS, 2010) é apresentada uma proposta de acesso a GPU através de uma camada de virtualização denominada *wrapper*. O autor utiliza-se da ferramenta Virtual Box, com as devidas alterações, e da arquitetura CUDA, para criar o VirtCUDA. Os resultados do trabalho demonstram que o VirtCUDA apresenta uma perda no desempenho das aplicações, causado pelas questões relacionadas à virtualização da placas gráficas.

Em sua pesquisa, (MA et al., 2012) apresenta um *framework* para gestão de arquiteturas heterogêneas de CPU+GPU, o GreenGPU. Tal pesquisa é inovadora na área, visto que até então, os trabalhos concentravam esforços apenas em arquiteturas de CPU ou GPU. Como resultado, o *framework* consegue obter uma redução no consumo de energia de até 21%. Os autores realizaram os testes com a arquitetura CUDA, na linha de GPUs GeForca da NVIDIA e em processadores Phenom II da AMD.

A pesquisa apresentada em (BOYER; MENG; KUMARAN, 2013) apresenta um *framework* para medir o desempenho de aplicações escritas para GPUs. O *framework* possibilita a modelagem do tempo de execução além do tempo de transferência dos dados. O modelo prevê a sobrecarga de transferência de dados com um erro de apenas 8%, e a inclusão do tempo de transferência de dados pode vir a reduzir o erro na aceleração de GPU em até 9%.

4 GPUHELP - APRESENTAÇÃO DA ARQUITETURA

Este capítulo tem o objetivo de descrever a proposta de solução, denominada GPUHelp, para ambientes de programação paralela em GPU. Sebesta (SEBESTA, 2005) classifica um ambiente de programação como sendo um conjunto de ferramentas utilizadas ao longo do desenvolvimento de *software*. Tal conjunto pode consistir em somente um editor de texto ou uma série de ferramentas e funcionalidades integradas. No contexto deste trabalho, classifica-se como ambiente uma infraestrutura completa que envolva todas as etapas de desenvolvimento e testes de algoritmos ao usuário. O objetivo é oferecer uma infraestrutura completa de *software* e aplicativos para a utilização ao longo de todo o processo de escrita e testes de códigos das arquiteturas de GPU, com foco nas arquiteturas CUDA e OpenCL. As seções a seguir descrevem a ideia e a arquitetura da solução proposta neste trabalho. Cabe salientar que, caso o leitor deseje compreender especificamente o funcionamento do ambiente, este pode avançar diretamente ao Capítulo 5, que trata da construção do ambiente e do desenvolvimento da aplicação.

4.1 A Solução Proposta

O principal objetivo do GPUHelp é oferecer um ambiente para a prática de programação paralela, livre, de fácil acesso e disponível a comunidade acadêmica. As possibilidades de utilização do GPUHelp vão desde um sistema operacional nativo a ser instalado no dispositivo utilizado pelo usuário, até a utilização de apenas a ferramenta de execução remota de códigos. Os componentes básicos do GPUHelp são apresentados nas Seções 4.2.1, 4.2.2 e 4.2.3.

É importante salientar que o GPUHelp também pode ser utilizado como sistema operacional virtualizado. O principal objetivo de utilizá-lo de forma virtualizada, tem foco nos diferentes tipos de testes que os utilizadores possam vir a executar, sem causar danos ou riscos ao sistema operacional real. Caso ocorra alguma falha ou comportamento inesperado durante o processo de virtualização, é necessário apenas ser criada uma nova máquina virtual com outra imagem do GPUHelp.

Com o desenvolvimento de tal ambiente, os professores podem concentrar esforços no conteúdo teórico, sem terem de se preocupar com configurações de máquinas de laboratórios, ou ainda, conflitos de horários, visto que, as ferramentas a serem utilizadas pelo professor da disciplina tendem a ser instaladas em um único laboratório.

A Figura 4.1 apresenta uma visão abstrata, sintetizada, do ambiente proposto neste trabalho.

Pode-se observar a composição básica da proposta: um ambiente servidor acessado via aplicação para execução remota de códigos disponível no ambiente cliente, sendo que neste ainda estarão disponíveis ferramentas para escrita e desenvolvimento de códigos, além dos *drivers* para programação paralela em arquiteturas de GPU instalados e configurados.

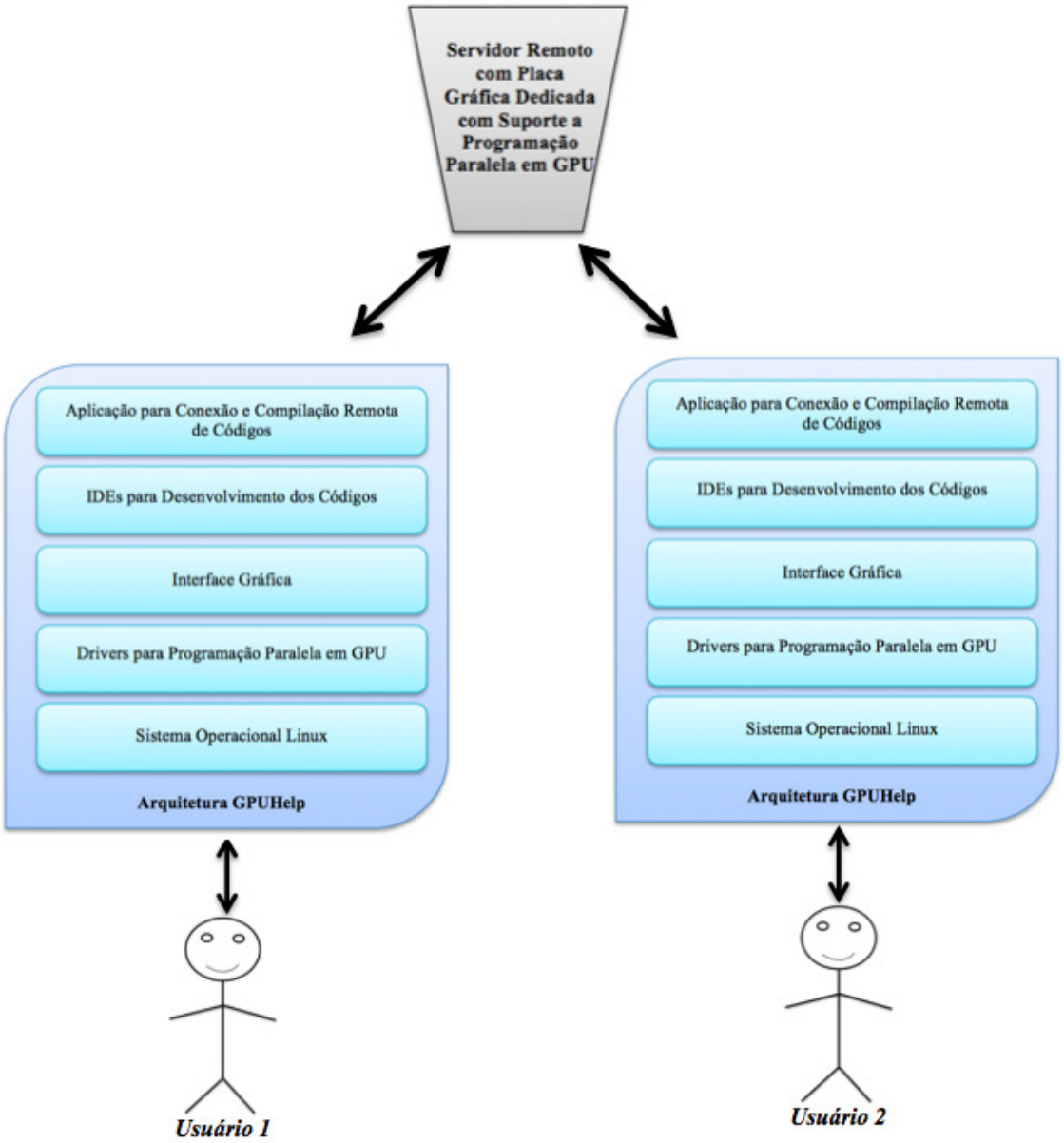


Figura 4.1: Infraestrutura do GPUHelp

4.2 Uma Visão Geral da Arquitetura

A seção anterior descreveu a ideia básica do GPUHelp. Já as seções a seguir, tem o objetivo de detalhar algumas características do ambiente desenvolvido neste trabalho. A arquitetura básica do GPUHelp é apresentada na Figura 4.2. Como pode ser observado, o GPUHelp é

caracterizado por possuir um ambiente cliente e um ambiente servidor.

O ambiente cliente é formado por um sistema operacional acrescido de *drivers* para programação paralela em GPU para as arquiteturas CUDA e OpenCL, editores de código e uma aplicação para conexão e execução remota dos códigos no ambiente servidor. A ideia é oferecer ao usuário um ambiente completo, facilitando assim, a aplicabilidade tanto no desenvolvimento de códigos quanto outras tarefas a serem realizadas por este.

Já o ambiente servidor é constituído de uma máquina com placa gráfica dedicada a computação paralela em GPU, com os *drivers* para programação paralela em GPUs instalados, configurados e com a correta configuração para execução e testes de códigos escritos pelos usuários. A conexão entre o ambiente servidor e o ambiente cliente se dá unicamente através da aplicação para execução remota de códigos. O usuário não possui qualquer tipo de acesso ou controle sobre o ambiente servidor.

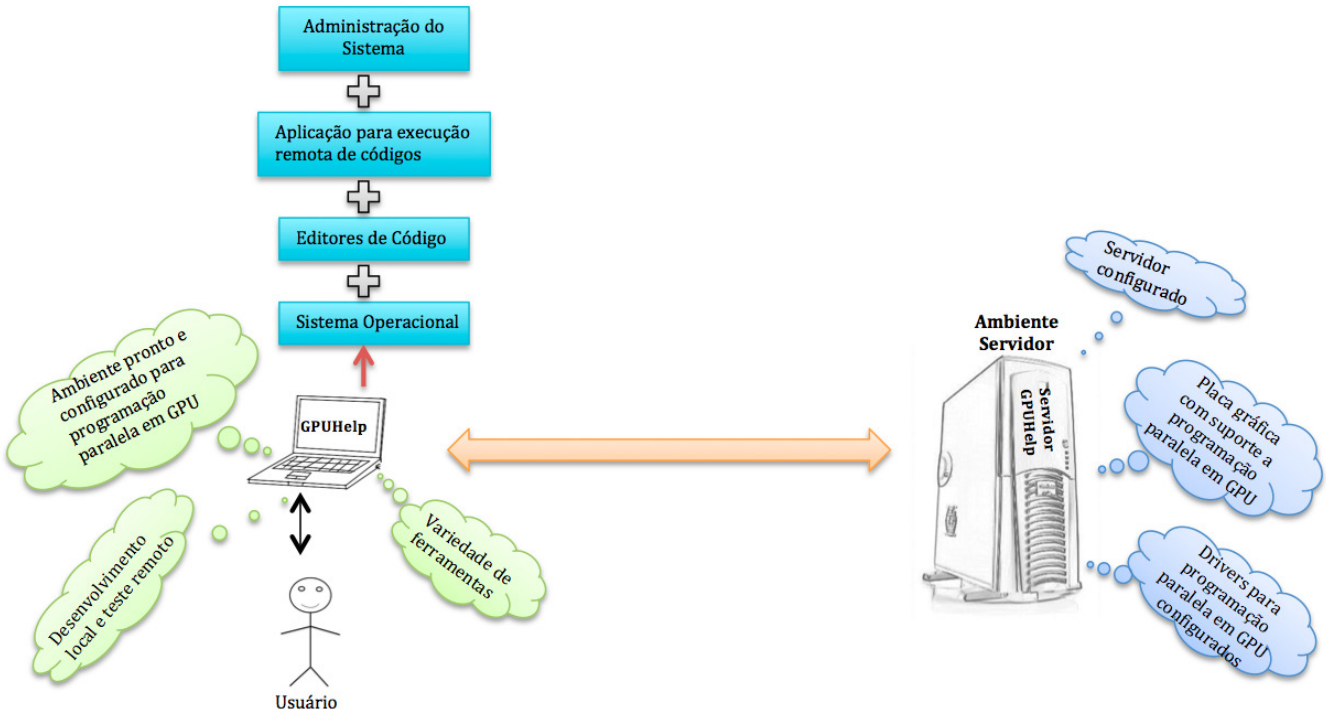


Figura 4.2: Arquitetura do GPUHelp

4.2.1 Sistema Operacional e Ferramentas Disponíveis para os Usuários

Um dos objetivos do GPUHelp é ser uma solução inteiramente livre. Isso está presente em todas as etapas da construção do ambiente. Adotou-se a utilização única e exclusivamente de ferramentas baseadas em *software* livre, que pudessem ser disponibilizadas sem violar quais-

quer de seus direitos. A Seção 5.2.2 detalha as características do sistema operacional utilizado no GPUHelp.

Desenvolvedores costumam ter suas preferências e familiaridades com determinados editores de código. Sendo assim, com o objetivo de tornar a experiência de utilização do GPUHelp a mais amigável possível com os diversos tipos de utilizadores, optou-se por disponibilizar os principais editores de códigos atualmente disponíveis, conforme pode ser visto na Seção 5.2.3. Desse modo, um usuário que já está familiarizado a sua ferramenta de desenvolvimento habitual, não precisará ter de aprender a utilizar um novo editor, pois possivelmente terá o seu disponível no ambiente.

Cabe salientar que as ferramentas de *software* disponíveis no ambiente cliente são apenas as que já estarão pré configuradas, porém, caso o usuário tenha preferência por alguma outra distinta das que compõem o ambiente, o mesmo poderá instalar a ferramenta de sua preferência, sem impactar no correto funcionamento da arquitetura.

Com a utilização do GPUHelp como ambiente completo, o utilizador terá disponível além dos *drivers*, os exemplos que são instalados com estes para o desenvolvimento e testes dos códigos. A Figura 4.3 apresenta a composição das ferramentas do GPUHelp.

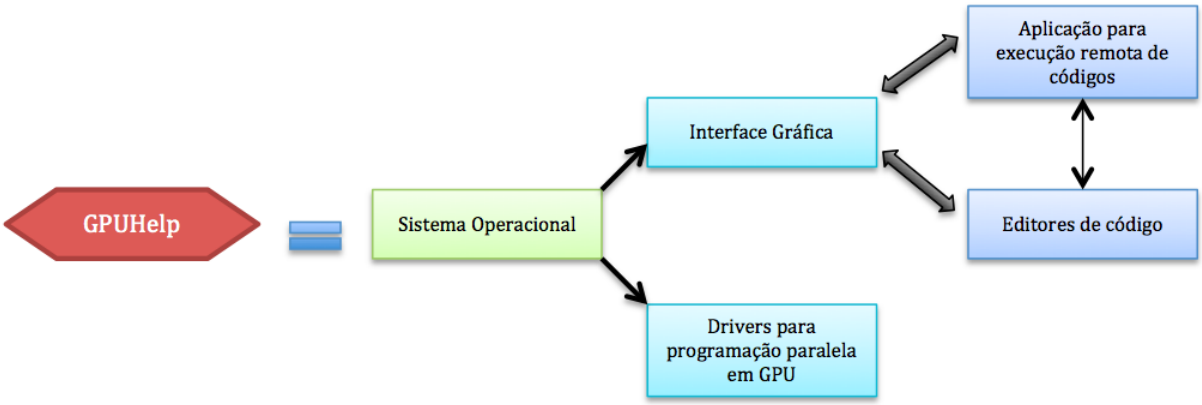


Figura 4.3: Sistema operacional e demais ferramentas que compõem a arquitetura GPUHelp

4.2.2 Aplicação para Conexão e Execução Remota de Códigos

Durante o projeto deste trabalho, pensou-se em uma ferramenta que fosse essencial para os utilizadores, algo que atuasse diretamente no problema-foco da construção do GPUHelp: a ausência de placas gráficas em uma quantidade considerável de dispositivos comercializados atualmente. Seguindo tal pensamento, criou-se a aplicação para execução remota de códigos. Tal aplicação possui suporte para as arquiteturas OpenCL e CUDA, ambos *frameworks* para

programação paralela em GPUs. A Figura 4.4 ilustra as funcionalidades da aplicação para execução remota de códigos.

Por caracterizar-se como um ambiente de apoio ao ensino de programação paralela em GPU, o GPUHelp oferece uma aplicação com interface simples, intuitiva e amigável. A aplicação foi projetada visando os diferentes tipos de usuários em programação paralela: usuários iniciantes e usuários experientes. Dessa forma, cada tipo de usuário terá uma interface distinta, com suas próprias funcionalidades, adequadas ao seu nível de conhecimento na área. As Seções 5.4.2 e 5.4.3 apresentam as interfaces de ambos os tipos de usuários disponíveis na aplicação. Já a Seção 5.4 apresenta de forma detalhada a ferramenta para execução remota de códigos.

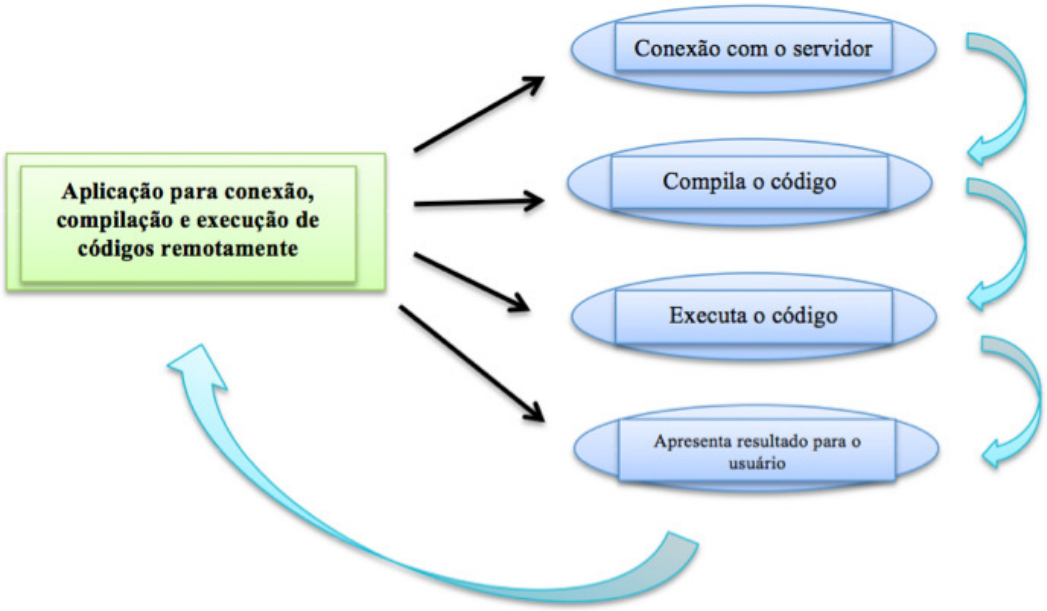


Figura 4.4: Funcionalidades da aplicação para conexão, compilação e execução remota de códigos do GPUHelp

4.2.3 Ambiente Servidor

Uma das etapas relacionadas ao desenvolvimento de qualquer algoritmo, seja ele sequencial ou paralelo, é o teste do código. Em programação paralela de GPUs, algumas arquiteturas, como o CUDA, por exemplo, necessitam de placas gráficas de fabricantes em específico, não permitindo heterogeneidade de dispositivos para possibilitar o teste dos códigos desenvolvidos. Este é um fator limitador que envolve os utilizadores que possuem dispositivos com placas gráficas compartilhadas, pois impede que estes consigam testar seus códigos, impossibilitando uma parte fundamental da construção do algoritmo: o teste deste.

Desta forma, adicionou-se à arquitetura GPUHelp um servidor com placa gráfica compatível com as principais arquiteturas de programação paralela em GPU: o CUDA e o OpenCL. O ambiente servidor é disponibilizado ao usuário através da aplicação para execução remota de códigos, conforme ilustrado na Figura 4.4 da Seção 4.2.2. No ambiente servidor, os *drivers* para programação paralela em GPU estão instalados e configurados. A Figura 4.5 demonstra as funções que são desempenhadas pelo servidor, além de apresentar as funções da aplicação para execução remota de códigos. A Seção 5.6 detalha as características do ambiente servidor.

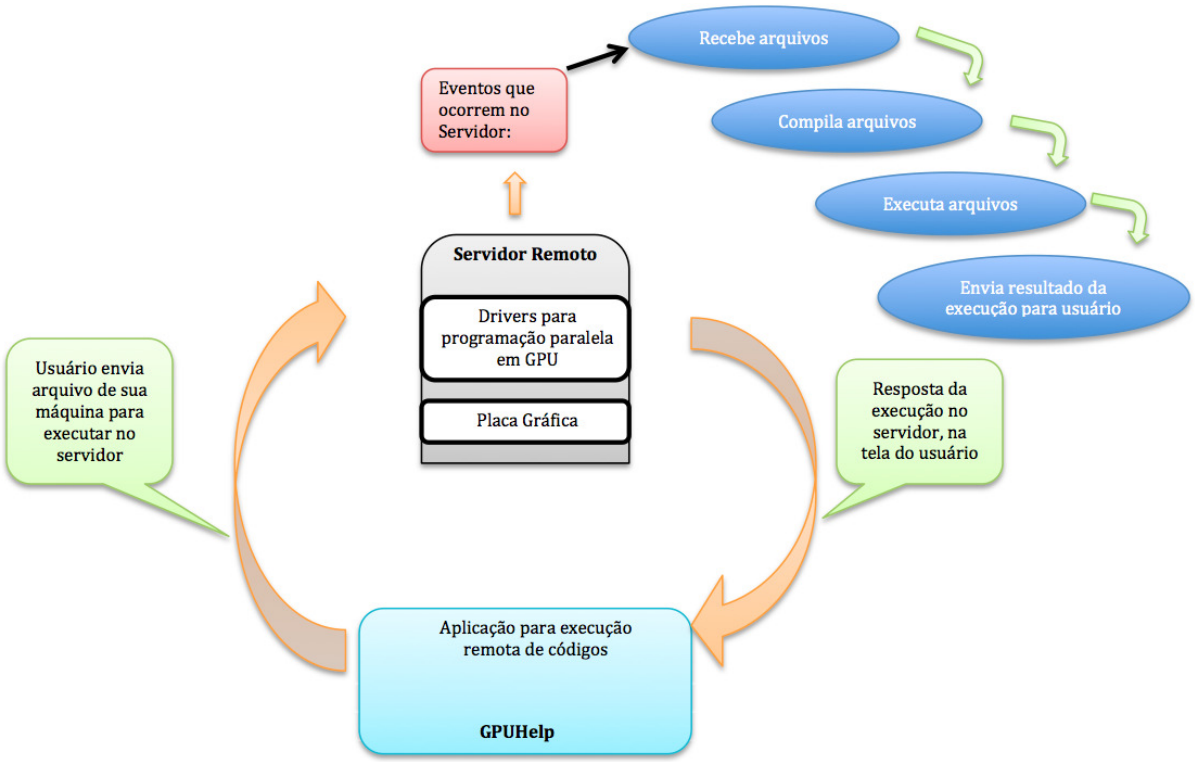


Figura 4.5: Ambiente servidor e aplicação para execução remota de códigos

4.2.4 Interfaces Pela Visão do Usuário

A interface com o usuário é realizada através da interface gráfica, juntamente com os editores de código e a aplicação para conexão e execução remota de códigos quando o utilizador optar pelo ambiente completo, conforme ilustrado na Figura 4.6(A), ou interface gráfica da aplicação quando o usuário optar pela utilização apenas da aplicação para conexão e execução remota de códigos, conforme ilustrado na Figura 4.6(B). A interface do usuário é responsável pela comunicação com o servidor remoto, além da entrega dos resultados da execução dos códigos diretamente na tela do utilizador. Um dos objetivos do GPUHelp é fornecer uma interface simples para o usuário, objetivando a facilidade de utilização.

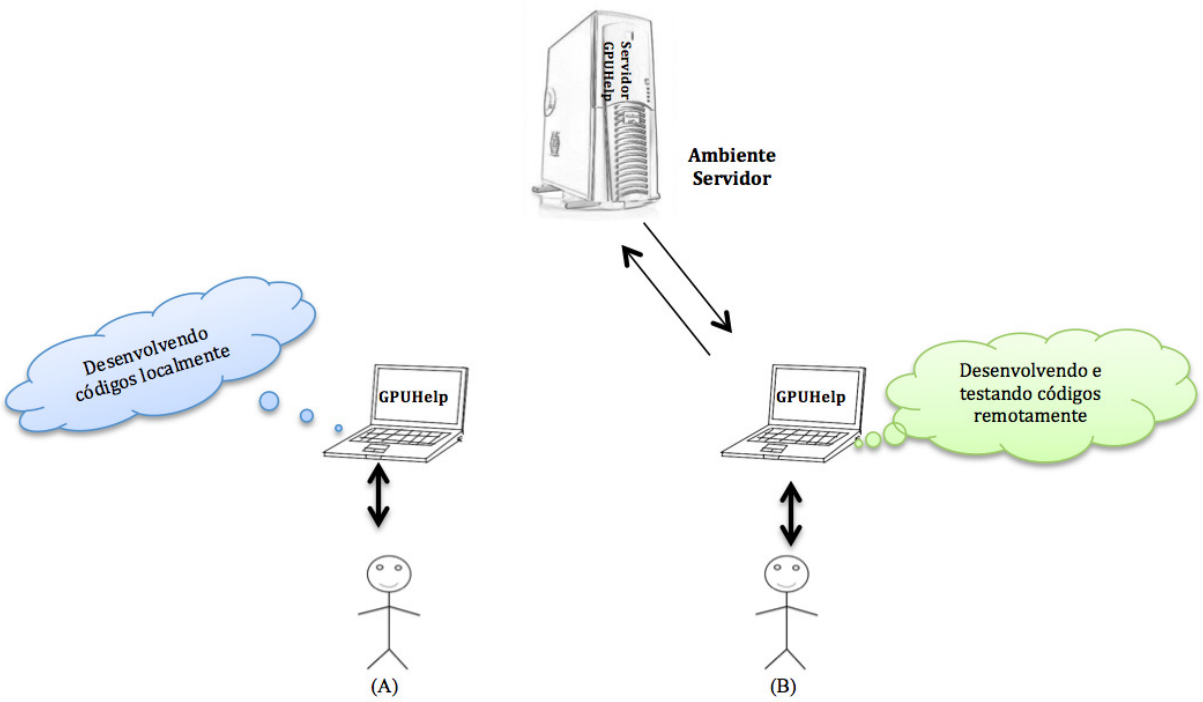


Figura 4.6: Interfaces a nível de usuário, sendo que em (a) o usuário está escrevendo seus códigos localmente, enquanto que em (b) o usuário está testando os códigos que desenvolveu em sua máquina com o GPUHelp no ambiente servidor, através da aplicação para execução remota de códigos

4.2.5 Questões Relacionadas à Segurança

O objetivo do GPUHelp é atuar como ferramenta de apoio no ensino de programação paralela em GPU. Porém, isso não diminui os cuidados com a segurança que o ambiente deve possuir.

A segurança envolvendo o ambiente servidor foi um dos fatores que contribuiu pelo desenvolvimento de uma aplicação para conexão e execução remota dos códigos da forma como descrito até aqui, com o uso de autenticidade baseada em confidencialidade. Apenas os usuários que devem ter acesso ao servidor, poderão acessá-lo. A ideia é preservar a integridade do ambiente servidor, contribuindo para que a máquina não tenha sua segurança comprometida.

Outra questão a ser levada em consideração é a distribuição do ambiente GPUHelp, bem como da aplicação para conexão e execução remota, quando utilizada de forma separada do ambiente. A disponibilização do ambiente deverá ser feita pelo professor da disciplina, acordando o melhor momento para disponibilizar o ambiente ou a ferramenta aos alunos de forma segura.

4.3 Caso de Uso e Aplicabilidades do GPUHelp

Com o objetivo de melhor ilustrar o propósito da arquitetura projetada como ferramenta de apoio ao ensino de programação paralela em GPU, segue um exemplo prático. A figura 4.7 apresenta o caso de um professor que precisa preparar aulas para a disciplina de programação paralela em GPU. Supõe-se que neste momento o professor esteja iniciando a preparação de exemplos práticos a serem utilizados em sua disciplina. Supõe-se também, que o professor possui interesse em trabalhar sua disciplina com uma abordagem teórica e prática de forma paralela, e que o mesmo tem interesse em uma maneira de disponibilizar um meio para os alunos desenvolverem e testarem seus códigos ao longo da disciplina ministrada por ele.

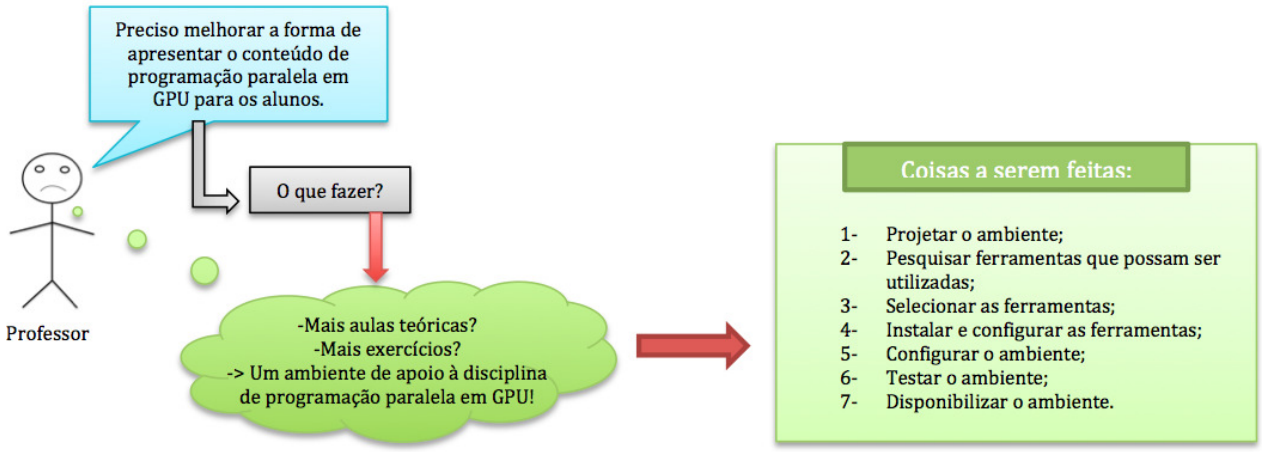


Figura 4.7: Professor preparando aulas para a disciplina de programação paralela em arquiteturas de GPU

Uma das possíveis soluções é a compra de um ambiente para atuar como ferramenta de apoio no ensino da disciplina. Neste trabalho, desconsidera-se esta possibilidade, tendo em vista que a mesma envolve custos para a Universidade e precisa que tal proposta tenha um projeto desenvolvido e apoio dos órgãos administrativos, além de uma série de outras questões burocráticas. Outra opção, seria que o próprio professor desenvolvesse tal ambiente. Entretanto, tal opção também não pode ser levada em consideração, tendo em vista as diferentes atividades que requerem a atenção de um professor. Frente as diferentes atividades que envolvem a rotina acadêmica de um professor, fica claro que este não dispõe do tempo necessário para projetar e construir tal ambiente. Pode-se dizer que a construção de tal ferramenta pode ser considerada uma tarefa relativamente complexa, independentemente da solução proposta, podendo ser esta uma solução constituída basicamente de ferramentas simples e já disponíveis, até o desenvolvimento de aplicações específicas que envolvam necessidades distintas para cada cenário

predeterminado. Ou seja, tanto na possibilidade de compra quanto na de desenvolvimento de tal ambiente, o professor terá de investir algum tempo na prototipação e acompanhamento do desenvolvimento de tal atividade.

Há ainda outra possível saída a ser adotada pelo professor. A de auxiliar e instruir cada aluno durante um longo e desgastante processo de instalação e configuração de *software*, que envolve desde a instalação de determinado sistema operacional, editor de código e configuração dos *drivers* das arquiteturas de programação paralela – levando em consideração o fato de que todos os alunos possuam em suas máquinas, placas de vídeo dedicadas, com suporte a programação paralela em GPU. Acredita-se que esta opção também deva ser desconsiderada, baseado na alta carga de trabalho a ser aplicada ao longo deste árduo processo, obtendo resultados distintos nas diferentes máquinas de cada aluno.

Cenários como os descritos anteriormente são alvos da solução proposta. A aplicação desenvolvida neste trabalho baseia-se na necessidade de um ambiente de apoio para a disciplina de programação paralela, com foco nas arquiteturas de GPUs. Tal ambiente pode ser administrado e gerenciado de forma a adaptar-se as necessidades de cada turma de programação paralela em GPU que o professor venha a trabalhar, visto que este terá acesso administrativo ao ambiente. Desse modo, o professor pode disponibilizar funcionalidades distintas dentro do ambiente ao longo de cada semestre.

O aluno, conforme ilustrado pela Figura 4.8, precisa apenas efetuar o *download* do ambiente desenvolvido, conforme visto na Seção 4.2. Isto simplifica o processo de trabalho do professor e provê recursos para que o aluno possa obter uma melhor experiência e um maior aprendizado na área de programação paralela em GPU. Com a solução proposta neste trabalho, o professor pode dedicar seu tempo a atividades mais pertinentes em sua rotina e que estejam relacionadas a uma melhor apresentação do conteúdo teórico da disciplina, além de técnicas para melhor apresentar e demonstrar de forma prática o conteúdo trabalhado, tendo em vista a existência de um ambiente de programação paralela em GPU que pode ser oferecido a todos os alunos da disciplina, provendo assim, recursos que antes eram inexistentes a alguns alunos, colaborando no processo de aprendizagem do conteúdo e despertando o interesse destes pela área.

As seções a seguir objetivam descrever as diferentes maneiras de utilização do GPUHelp. A ideia é ilustrar aos usuários as diferentes aplicabilidades do ambiente, permitindo-os decidir qual irá atender da melhor forma sua necessidade.

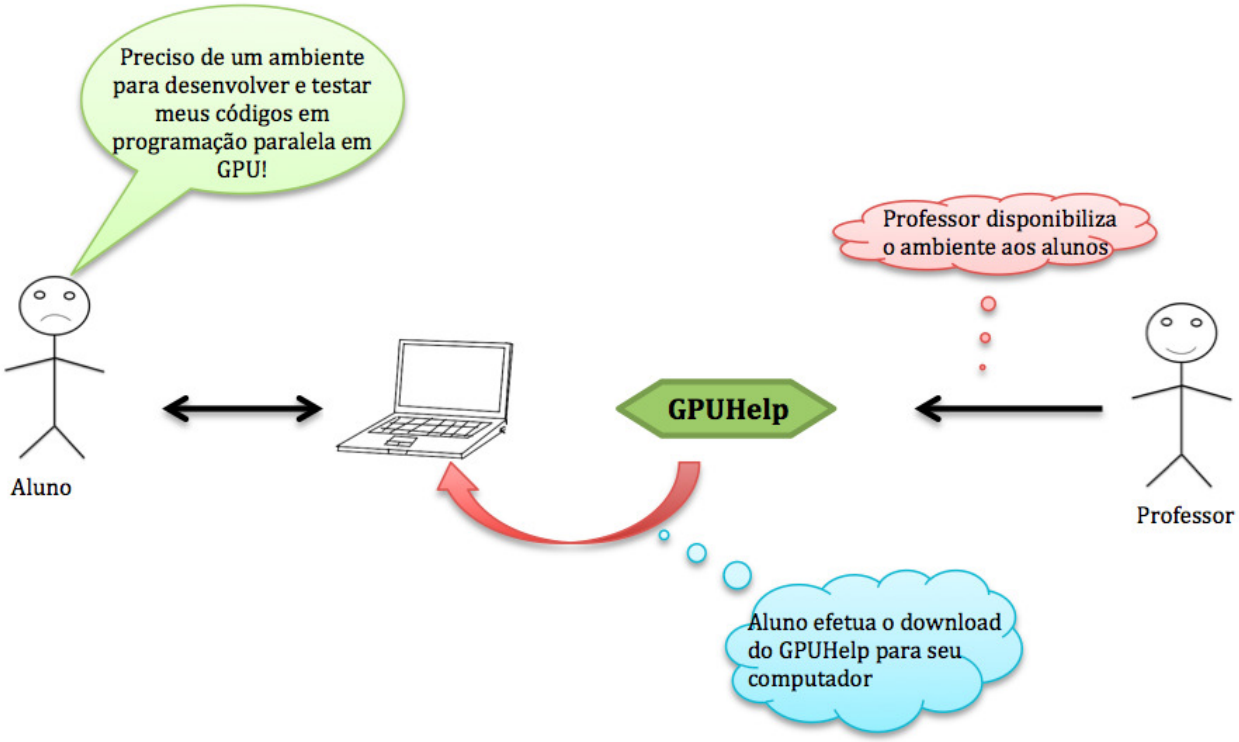


Figura 4.8: Professor disponibiliza o ambiente aos alunos

4.3.1 Aplicabilidades do GPUHelp

Tendo em vista as diferentes possibilidades de aplicação do GPUHelp, pode-se definir como o cenário ideal para utilização de tal ambiente, aquele onde o GPUHelp é utilizado como ferramenta de apoio pelo professor da disciplina de programação paralela em GPU, com foco nas arquiteturas CUDA e OpenCL, conforme ilustrado pela figura 4.9. Desse modo, o professor pode apresentar exemplos de códigos durante as aulas teóricas, com o objetivo de demonstrar de forma prática o conteúdo teórico trabalhado. Através da utilização do ambiente, os alunos possuem uma ferramenta para acompanhar os códigos apresentados pelo professor da disciplina, podendo realizar testes e acompanhar a demonstração dos códigos trabalhados. O capítulo 3 desta dissertação apresenta diferentes pesquisas que concluem que a utilização de uma intensa abordagem prática tem forte impacto para uma melhor compreensão dos conceitos de programação paralela. Desta forma, a utilização do GPUHelp pode contribuir para uma melhor compreensão dos conceitos da área de programação paralela em GPU.

Dando continuidade aos cenários onde o GPUHelp pode ser aplicado, pode-se mencionar a aplicabilidade do ambiente em oficinas, mini cursos ou cursos ministrados pelos professores ou monitores da disciplina. O ambiente pode ser utilizado ainda por alunos com experiência na

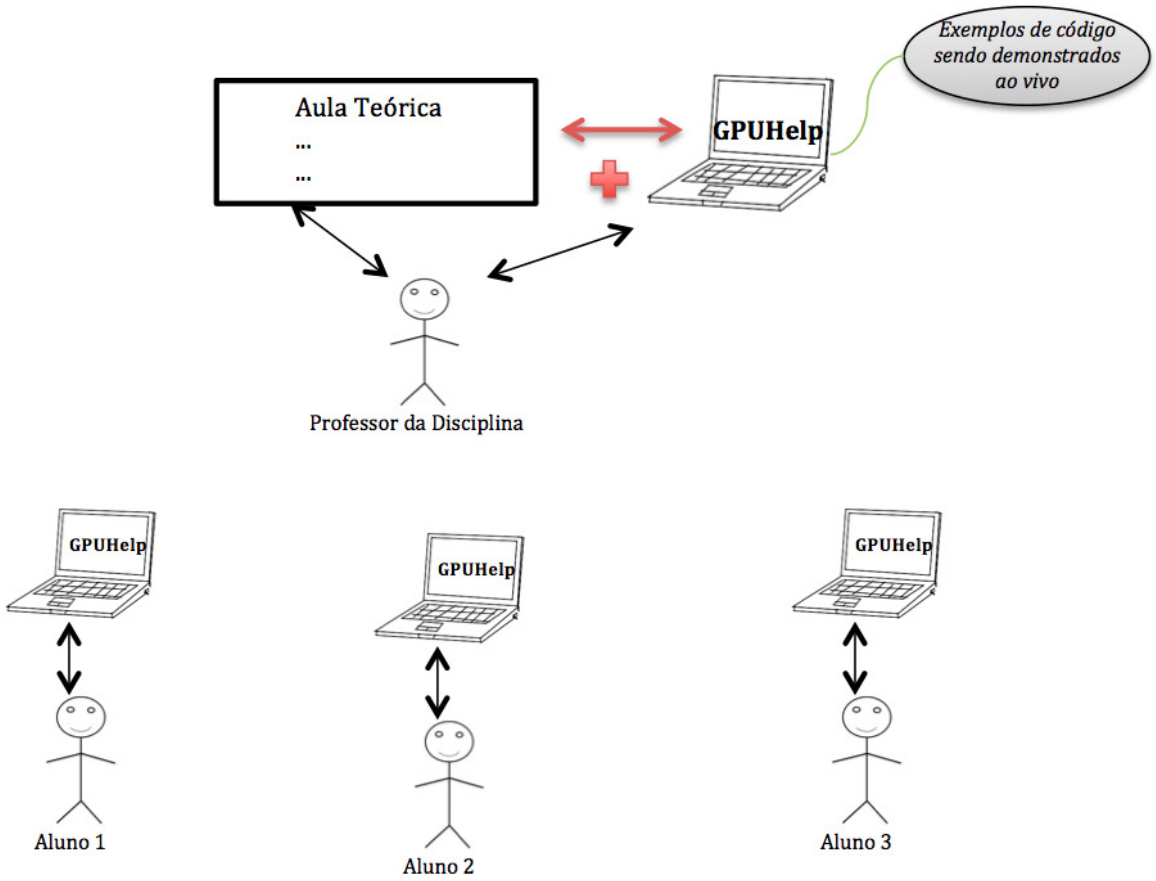


Figura 4.9: Cenário de aplicação do GPUHelp como ferramenta de apoio para a disciplina de programação paralela em GPU

área que desejam repassar seus conhecimentos e experiências a seus colegas. Em todas estas possíveis aplicações, o GPUHelp possui a estrutura para prover os recursos necessários para atuar como ferramenta de apoio na disciplina de programação paralela com foco em arquiteturas de GPU.

Outra maneira de utilização do GPUHelp é como um ambiente para treinamento de programação paralela em arquiteturas de GPU de forma geral, conforme demonstrado na figura 4.10. Dessa forma, o utilizador tem a liberdade de desenvolver e testar seus códigos, tanto para fins acadêmicos quanto para fins de aperfeiçoamento na prática de programação paralela em GPU.

As variadas possibilidades de aplicação do GPUHelp comprovam a flexibilidade e a ampla variedade de utilização do ambiente, bem como suas contribuições como ferramenta de apoio no ensino de programação paralela em arquiteturas de GPU. As seções a seguir apresentam outras possibilidades de utilização do ambiente GPUHelp.

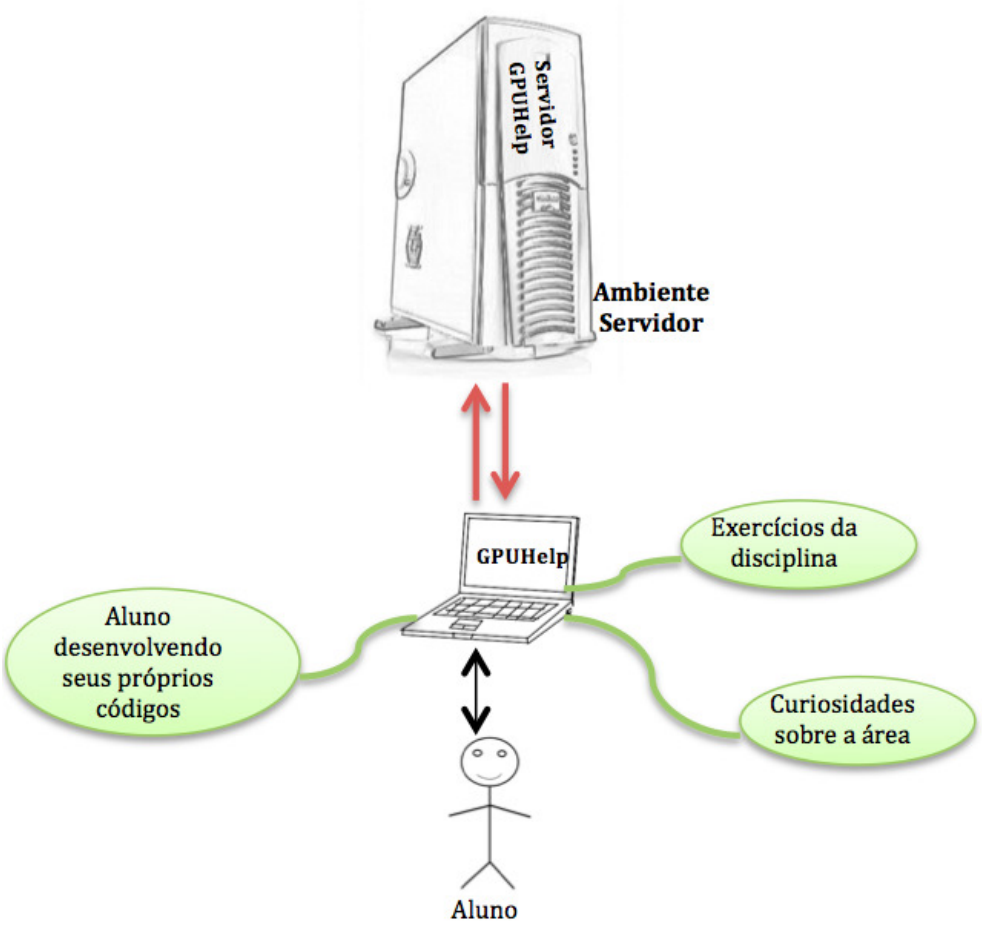


Figura 4.10: Aluno utilizando o GPUHelp como um ambiente para treinamento na área de programação paralela em GPU

4.3.2 Sistema Operacional Nativo

Considera-se como a forma de utilização recomendada para o GPUHelp, a instalação deste ambiente como um sistema operacional nativo na máquina do utilizador. Com isso, o usuário terá um melhor desempenho em todas as ferramentas utilizadas, além de um melhor gerenciamento do sistema operacional como um todo. Esta utilização é reforçada devido ao fato de o GPUHelp possibilitar a adição de novas ferramentas e aplicações ao ambiente, sem prejudicar sua estrutura básica. Sendo assim, o usuário pode configurá-lo da maneira que melhor se adequar ao seu meio de utilização. A Figura 4.11 ilustra o ambiente GPUHelp instalado como um sistema operacional nativo na máquina de determinado usuário.

Por meio da instalação do GPUHelp como sistema operacional nativo, a instalação e configurações de *softwares* e ferramentas que antes era necessária, agora não é mais, visto que o ambiente já está previamente configurado. Sendo assim, o utilizador pode investir seu tempo desenvolvendo e testando os códigos, ao invés de ter de realizar atividades longas e complexas

de instalação e configuração de *softwares*.

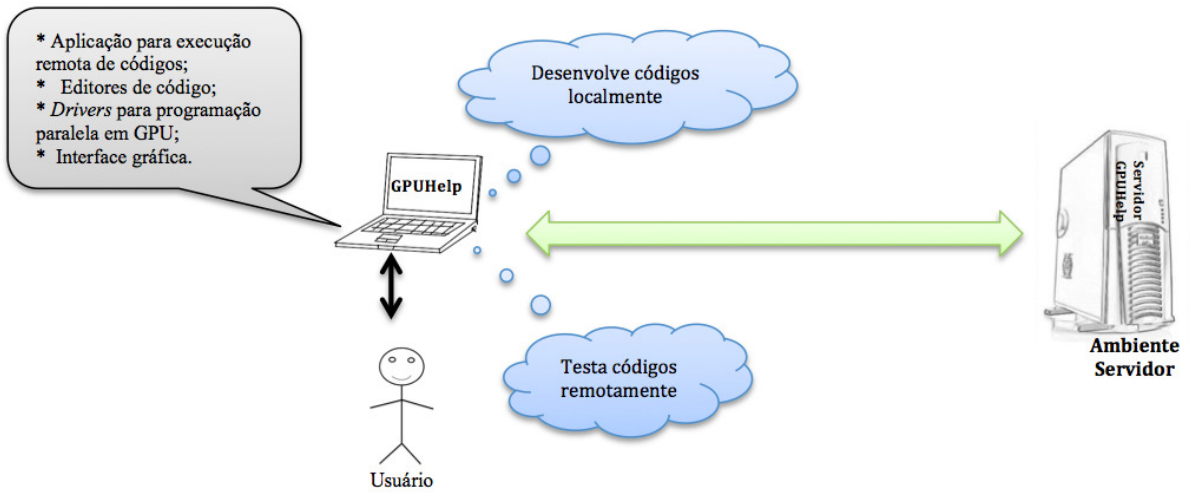


Figura 4.11: GPUHelp como sistema operacional nativo na máquina do usuário

4.3.3 Sistema Operacional Virtualizado

Dentre as diferentes situações que envolvam cada um dos usuários com suas próprias necessidades, existem aqueles utilizadores que não podem ou simplesmente não querem instalar o GPUHelp como um sistema operacional nativo em suas máquinas. Uma possível solução a ser adotada por estes é a de utilizar o GPUHelp como um ambiente virtual. A Figura 4.12 ilustra este cenário.

Com o uso de virtualização torna-se possível para aqueles usuários que não podem instalar o ambiente de forma nativa, utilizar os recursos do GPUHelp através de uma ferramenta de virtualização. Desta forma, estes usuários também teriam o GPUHelp disponível em suas máquinas.

A utilização da técnica de virtualização impacta em considerações que devem ser previamente analisadas pelo usuário. Pode-se destacar como uma delas a dificuldade acerca da virtualização de placas gráficas. Atualmente, somente é possível virtualizar dispositivos gráficos através da utilização de soluções proprietárias. De outro modo não é possível, visto que as soluções livres atualmente ainda não comportam a virtualização de GPUs. Outra questão que deve ser levada em consideração é o desempenho. Ao virtualizar um sistema operacional, o desempenho deste é reduzido, dependendo do *hardware* disponível na máquina do utilizador. Estas, entre outras questões, devem ser levadas em consideração no momento da escolha pela forma que se utilizará o GPUHelp: de forma nativa ou virtual.

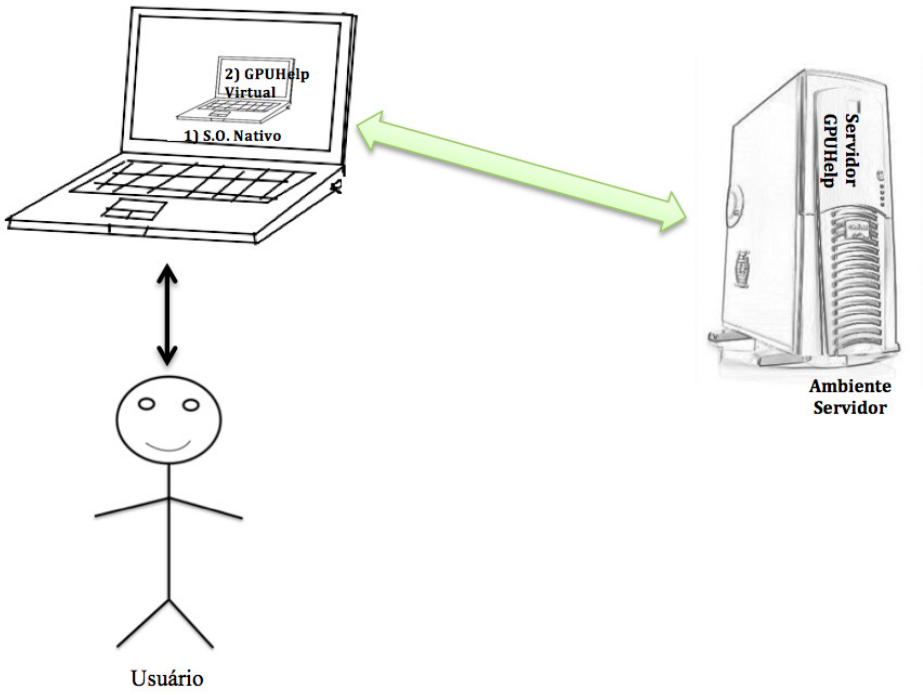


Figura 4.12: GPUHelp como sistema operacional virtual na máquina do usuário

4.3.4 Como Ferramenta para Execução Remota de Códigos

Esta é a forma de distribuição simplificada do GPUHelp. Pode-se dizer que esta é a forma mais objetiva da ferramenta, visto que esta disponibiliza apenas a aplicação para conexão e execução remota de códigos. Esta forma, foca nos usuários que possuem seus ambientes de desenvolvimentos de códigos (IDEs) configurados e prontos para uso, necessitando apenas de uma maneira para testar os códigos desenvolvidos. A Figura 4.13 apresenta este cenário.

Desse modo, o usuário continua utilizando as ferramentas de desenvolvimento que possui em sua máquina. Neste cenário, pressupõe-se que o aluno possua as ferramentas necessárias para a escrita de códigos para arquiteturas de GPU, necessitando apenas da aplicação para realizar os testes.

Como mencionado anteriormente, um dos objetivos do GPUHelp é a flexibilidade nos meios de utilização. Disponibilizar apenas a aplicação para conexão e execução remota dos códigos aos usuários, concede a estes a liberdade para que escolham em qual ambiente preferem desenvolver seus códigos e treinar os conceitos de programação paralela em GPU. Neste sentido, pode-se dizer que a aplicação seria um componente a ser agregado ao sistema operacional do usuário, disponível sempre que necessário.

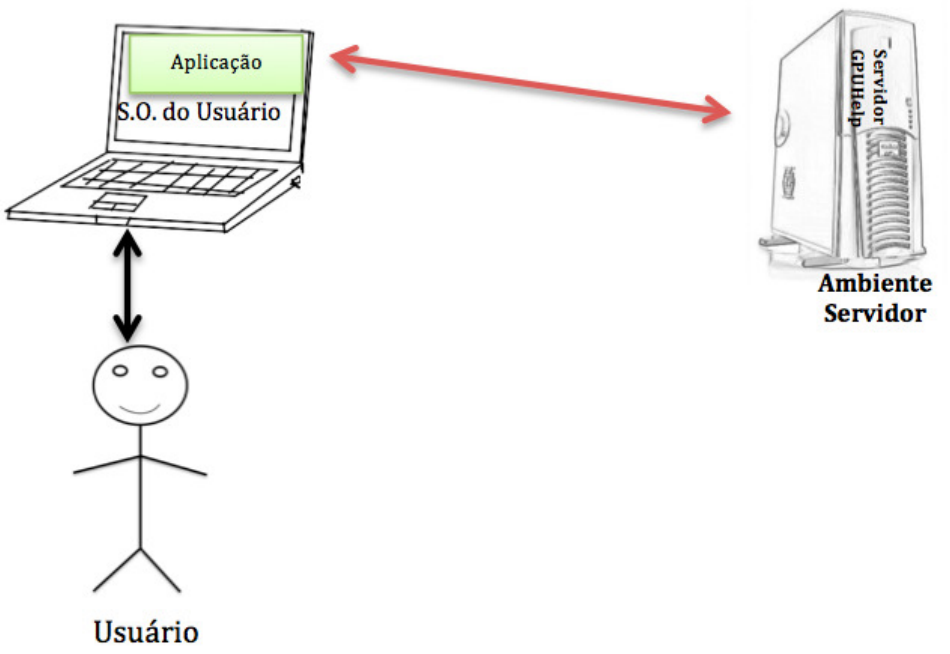


Figura 4.13: Aluno utilizando apenas a aplicação para conexão e execução remota de códigos em sua máquina

4.4 Considerações

Neste capítulo o GPUHelp foi apresentado de forma abstrata, uma demonstração em alto nível do ambiente, suas ferramentas e a aplicação para execução remota de códigos. A ideia é preparar o leitor para o Capítulo 5, onde são apresentados os detalhes e tecnologias utilizadas no desenvolvimento do ambiente GPUHelp. Devido ao nível de abstração utilizado no Capítulo 4, torna-se possível ao leitor compreender o objetivo do ambiente, suas características e funcionalidades, esclarecendo assim, dúvidas relacionadas ao projeto e arquitetura deste.

5 DESENVOLVIMENTO E IMPLEMENTAÇÃO

5.1 Introdução

Após ser apresentada a estratégia para o ambiente desenvolvido, pode-se discutir agora o desenvolvimento da arquitetura proposta. Este capítulo objetiva descrever a construção e a implementação do GPUHelp. Inicialmente são detalhadas as tecnologias utilizadas. A seguir, apresenta-se o projeto da arquitetura do GPUHelp. É detalhado ainda, o desenvolvimento da aplicação para execução remota de códigos. São apresentadas as características do ambiente servidor, além dos detalhes acerca do ambiente completo. Na sequência, discute-se as questões relacionadas à segurança do ambiente, bem como as limitações do mesmo. O capítulo é encerrado com as considerações acerca do desenvolvimento do GPUHelp.

O código fonte da aplicação para execução remota de códigos está disponível *online*, no SVN do Google Code, através deste endereço: `https://code.google.com/p/gpuhelp/`. Desse modo, novos utilizadores podem fazer *download* da aplicação e utilizá-la a mesma em suas Instituições. A ferramenta possui código fonte aberto, ou seja, permite alteração e adaptação pelos usuários, conforme suas necessidades. Devido ao tamanho do ambiente GPUHelp completo, o mesmo não está disponível *online* para *download*, devendo ser obtido diretamente com os administradores do ambiente.

5.2 Análise das Tecnologias Utilizadas

5.2.1 Arquitetura

A arquitetura proposta ao longo das Seções 4.1 e 4.2, tem como principal objetivo proporcionar simplicidade, eficiência e flexibilidade na utilização de um ambiente livre de apoio ao ensino de programação paralela em GPU. A ideia na utilização de ambientes de apoio baseia-se nas diferentes pesquisas acerca da complexidade da área, bem como das dificuldades relacionadas ao ensino e aprendizagem desta técnica.

Professores e alunos enfrentam algumas limitações acerca da área de programação paralela em GPUs: a necessidade de uma placa gráfica para os testes dos códigos é uma delas. Alguns usuários não possuem tal recurso em seus computadores, impossibilitando assim, o teste dos códigos desenvolvidos. Com o GPUHelp, os desenvolvedores terão a sua disponibilidade uma aplicação que executa os códigos desenvolvidos por eles remotamente em um servidor com

placa gráfica com suporte a programação paralela em GPU.

Dentro deste contexto, uma das metas principais na implementação do GPUHelp foi focar na simplicidade e flexibilidade da arquitetura proposta (Seções 4.1 e 4.2). Adicionalmente, buscou-se utilizar ferramentas comumente utilizadas por desenvolvedores de códigos nas diferentes arquiteturas e linguagens existentes, reduzindo potenciais trabalhos redundantes, e, consequentemente, aumentando a produtividade destes, visto não ser necessária a instalação e configuração de *software* adicional.

Para o desenvolvimento da aplicação para execução remota de códigos, foi utilizada a linguagem Java (ORACLE, 2013a). Entre as diferentes linguagens de programação disponíveis atualmente, Java destaca-se pela sua simplicidade e por ser portável, tornando-se assim amplamente utilizada. Dessa forma, a linguagem Java é ideal para este trabalho, tendo em vista os diferentes tipos de usuários, com seus sistemas operacionais distintos. Com o desenvolvimento da aplicação em Java, o usuário poderá utilizar uma aplicação portável, que funcione em sistemas Linux, MAC e Windows.

Já no ambiente servidor, buscou-se utilizar recursos e tecnologias disponíveis no Departamento de Linguagens e Sistemas de Computação (LSC) da UFSM, não sendo necessária a aquisição de bens e/ou materiais de consumo para o desenvolvimento deste trabalho. As características da máquina servidor são descritas de forma mais aprofundada na Seção 5.6. As características e a definição do sistema operacional a ser utilizado no ambiente GPUHelp são descritas na Seção 5.2.2, onde é apresentada a distribuição Linux escolhida para compor o ambiente.

5.2.2 Sistema Operacional

Para o desenvolvimento do GPUHelp foi escolhido o sistema operacional GNULinux devido a sua liberdade, flexibilidade, e, principalmente, por este ser um sistema operacional gratuito, livre, de código aberto e utilizado por uma vasta comunidade de usuários. Desse modo, o GNULinux é a alternativa ideal para este trabalho, reduzindo implicações legais de desenvolvimento de soluções e dependências de terceiros. Ser livre e possibilitar a distribuição é uma das premissas do GPUHelp, dessa forma, o sistema operacional a ser utilizado também deve cumprir com estes requisitos.

A opção por uma distribuição da família de sistemas operacionais GNULinux justifica-se também pela questão de existirem diversas ferramentas que dão suporte aos desenvolvedores

de códigos nas diferentes linguagens e arquiteturas. Outras características como facilidade e flexibilidade, relacionadas a manutenção do sistema, são importantes também, pois caso seja necessário que algum usuário realize determinada configuração, ou ainda, enfrente algum comportamento inesperado pelo sistema, este poderá corrigi-lo sem grandes problemas, devido à natureza simples de instalação e configuração do ambiente.

Após ser realizada uma análise, optou-se pela escolha do GNULinux Ubuntu (CANONICAL, 2013a) (CANONICAL, 2013b), pela simplicidade e facilidade de uso desta distribuição. A distribuição Ubuntu pode ser utilizada tanto por usuários avançados quanto iniciantes, visto sua facilidade de utilização e vasta documentação na *Internet*. Outro importante fator que levou a escolha pelo sistema Ubuntu, é a compatibilidade com os *drivers* das arquiteturas CUDA e OpenCL.

A instalação dos *drivers* para programação paralela em arquiteturas de GPUs é um processo com certo nível de complexidade. Novos usuários com pouca experiência em sistemas Linux e instalação e configuração de ferramentas, podem vir a encontrar certas dificuldades. Dessa forma, a ideia é disponibilizar o ambiente com todos os *drivers* para programação em GPU instalados, configurados e prontos para uso. Juntamente com os *drivers*, estão disponíveis exemplos para arquiteturas de GPU, que poderão ser estudados pelos alunos.

5.2.3 Ferramentas para Desenvolvimento de Códigos

Ao longo do projeto de elaboração do GPUHelp, buscou-se realizar uma seleção de ferramentas e aplicações que pudessem adicionar funcionalidades a arquitetura desenvolvida neste trabalho. Procurou-se pensar em quais seriam mais úteis e imprescindíveis aos utilizadores. Como o foco da arquitetura proposta são professores e alunos, ou seja, um foco acadêmico e educacional, a meta principal foi focar em tecnologias conhecidas e utilizadas por estes. Dessa forma, projetou-se um conjunto de ferramentas principais que devem compor o ambiente.

Seguindo esta linha de pensamento, podem ser citadas algumas ferramentas durante a prototipação do ambiente. Todas as ferramentas tem como base para sua escolha, parâmetros como o grau de conhecimento necessário para utilização, a disponibilidade de documentação, a produtividade, o tipo de licença (preferencialmente gratuitas e de código aberto), o tipo de distribuição (livre, preferencialmente) e a flexibilidade.

Para desenvolverem códigos, cada programador possui uma IDE de sua preferência. Dessa forma, foram adicionadas as principais IDEs para desenvolvimento de códigos disponíveis atu-

almente: Eclipse, Cobeblocks e Netbeans.

O Eclipse (ECLIPSE, 2013a) é uma IDE para desenvolvimento de códigos Java, que suporta também, várias outras linguagens através de *plugins* como C, C++, Python, Plataforma Android, ColdFusion entre outros. A ferramenta Eclipse foi desenvolvida em Java, e esta segue o modelo *open source* de desenvolvimento de *software*. O projeto Eclipse teve início com a empresa IBM, que desenvolveu a primeira versão do produto e o doou como *software* livre a comunidade (WHITE, 2012). Em 2011, o Eclipse já era a IDE para desenvolvimento Java mais utilizada no Mundo (KABANOV, 2011).

Netbeans (ORACLE, 2013b) caracteriza-se por ser uma IDE de código aberto para desenvolvedores C, C++, Java, entre outros. O Netbeans pode ser executado em diferentes sistemas operacionais, como Mac, Linux e Windows. Em 1996 o Netbeans teve seu projeto iniciado, em 1999 o projeto tornava-se proprietário, porém, em 2000 a empresa Sun liberou seu código fonte para a comunidade, tornando-o assim, uma ferramenta *open source* (ORACLE, 2013b).

A ferramenta CodeBlocks (CODEBLOCKS, 2013) é uma IDE para desenvolvimento de códigos em C e Fortran, que possui código fonte aberto, ou seja, é uma ferramenta *open source*. Codeblocks é multiplataforma, podendo ser executado em sistemas Linux, Windows e Mac OS.

5.3 Desenvolvimento do GPUHelp

Para o desenvolvimento da aplicação para execução remota de códigos foi utilizada a linguagem Java, juntamente com o *software* Eclipse (ECLIPSE, 2013a), na versão Juno (ECLIPSE, 2013b), de 64 bits para MAC OS X. Foram instaladas também, a JRE e a JDK do Java, para possibilitar o correto funcionamento da ferramenta Java.

Foram adicionadas ao Eclipse Juno as *libraries* (bibliotecas) JTattoo (JTATTOO, 2013) e Jpedal (JPEDAL, 2013). O JTattoo consiste de um conjunto de funcionalidades *Swing* que possibilitam ao desenvolvedor aprimorar a aparência visual de sua aplicação, tornando assim, sua interface mais intuitiva e amigável. Já o Jpedal *(Java PDF Extraction and Decoding Library)* é uma biblioteca Java para apresentar o conteúdo de arquivos com a extensão *".pdf"*. A versão utilizada da biblioteca Jpedal é a versão *open source*.

Para conexão entre cliente e servidor utilizou-se o RMI *(Remote Method Invocation)*, que possibilita chamadas remotas em aplicações desenvolvidas em Java. O package "Rmi" contém os arquivos que possibilitam esta funcionalidade. O RMI é capaz de prover as funcionalidades de uma plataforma de objetos distribuídos. Graças a arquitetura RMI torna-se possível que

um objeto ativo em determinada *Virtual Machine* Java possa interagir com objetos de outras JVMs, independente da localização destas Máquinas Virtuais Java. A API RMI é capaz de fornecer ferramentas para que seja possível ao utilizador projetar e desenvolver uma aplicação sem precisar preocupar-se com detalhes de comunicação entre os diversos elementos *(hosts)* de um sistema.

A aplicação para execução remota de códigos necessita que o utilizador possua a versão 7.0 (ou superior) do java em seu *desktop* ou *notebook* para poder utilizá-la. Isso faz-se necessário devido as novas tecnologias que são adicionadas a cada atualização da JRE e JDK do Java, além, é claro, das funcionalidades implementadas na aplicação, sendo uma das principais sua interface.

5.4 Aplicação para Execução Remota de Códigos: Visão do Usuário

5.4.1 Interface Inicial

A aplicação para execução remota de códigos tem por objetivo possuir uma interface simples e funcional, conforme apresentado na Figura 5.1, a tela inicial da aplicação. Nesta interface, o usuário pode decidir qual é seu nível de experiência em se tratando de programação paralela em GPU: usuário iniciante ou usuário experiente.

A interface inicial da aplicação conta ainda com uma ajuda ao usuário que não sabe qual opção selecionar. As Figuras 5.2 e 5.3 demonstram ao usuário as características de cada uma das aplicações, tornando possível a este selecionar seu nível de experiência em programação paralela em arquiteturas de GPU.

5.4.2 Tela Usuário Inicial

Após a interface inicial apresentada na Figura 5.1, o usuário tendo escolhido a opção usuário inicial, será direcionado para a interface apresentada na Figura 5.4, que apresenta a interface para usuários iniciais em programação paralela em arquiteturas de GPU. Nesta tela, o usuário pode selecionar códigos prontos, para as arquiteturas CUDA e OpenCL, que estão disponíveis nos repositórios do CUDA e do Grupo Khronos, para analisar e executar, observando os resultados.

A Figura 5.5 apresenta alguns dos códigos disponíveis aos usuários para as arquiteturas CUDA (CUDA, 2013d) e OpenCL (KHRONOS, 2011), disponíveis nos repositórios destes. O objetivo disto é apresentar códigos prontos, para que os alunos possam ter um contato ini-

Figura 5.1: Interface inicial da aplicação para execução remota de códigos em arquiteturas de GPU

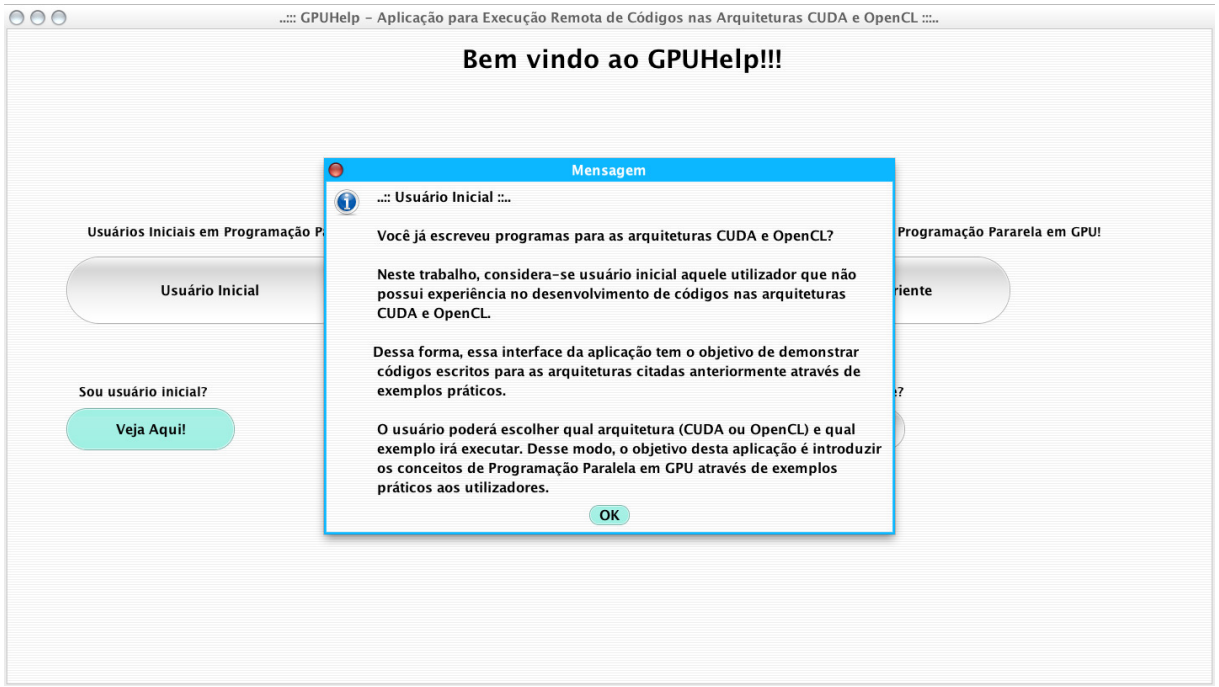


Figura 5.2: Texto explicativo para a interface usuário inicial

cial com a programação paralela em tais arquiteturas. Através da análise de tais exemplos, os usuários podem concentrar esforços em compreender os paradigmas necessários para a programação paralela em GPU, sem precisarem desenvolver códigos inicialmente. Para compor os exemplos disponíveis na interface para usuários iniciantes, selecionou-se códigos iniciais,

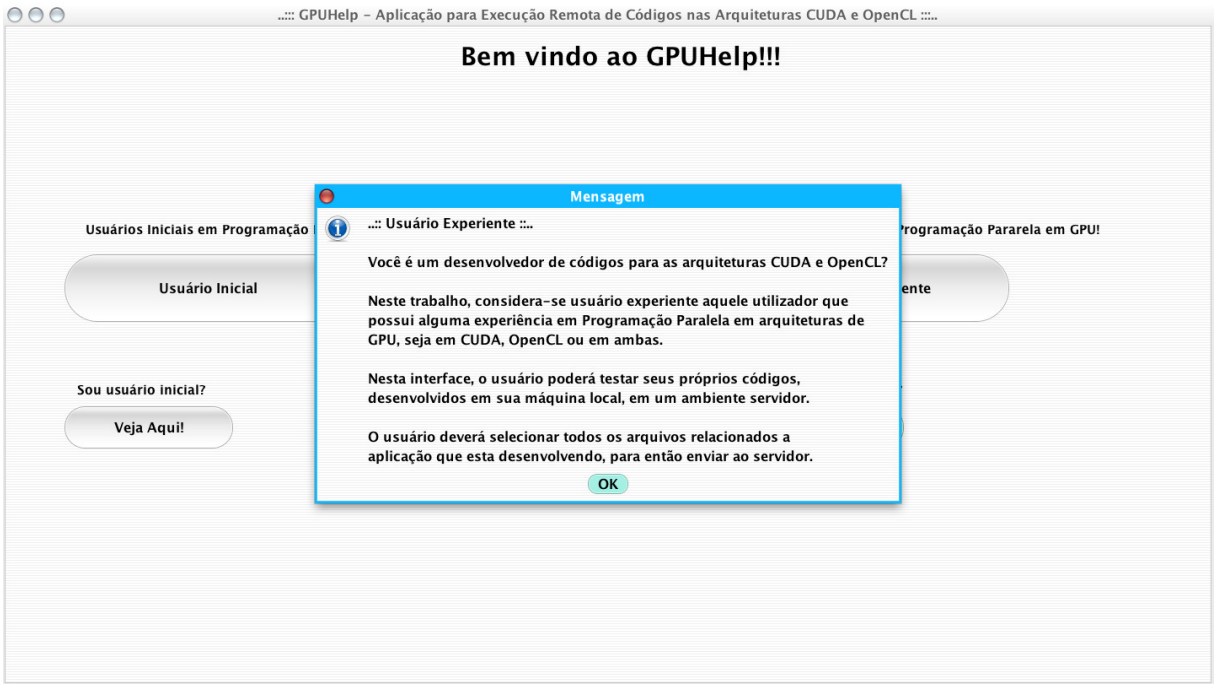


Figura 5.3: Texto explicativo para a interface usuário experiente

Figura 5.4: Interface para usuários iniciais em programação paralela em arquiteturas de GPU

como por exemplo, o *"Hello-World"* para as arquiteturas CUDA e OpenCL, além de códigos para multiplicação de matrizes e exemplos de soma de números, entre outros.

Já a Figura 5.6 apresenta o editor de códigos disponível na interface para usuários iniciais. Através desta funcionalidade, torna-se possível ao usuário testar os exemplos disponíveis na aplicação, podendo obter novos resultados além de realizar testes variados. Com a opção

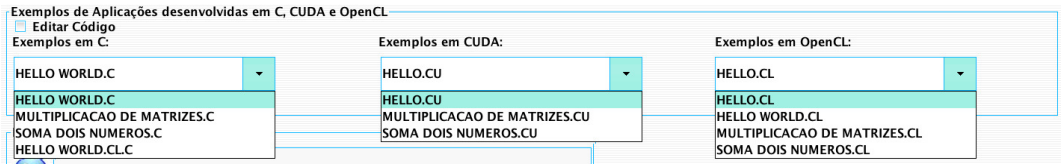


Figura 5.5: Exemplos disponíveis na aplicação para alunos iniciantes em programação paralela em arquiteturas de GPU

de edição de códigos, o usuário poderá compreender o funcionamento dos códigos disponíveis editando-os e realizando experimentos diversos. Enquanto que a Figura 5.7 apresenta o resultado da execução de um dos exemplos disponíveis na aplicação para usuários iniciais.

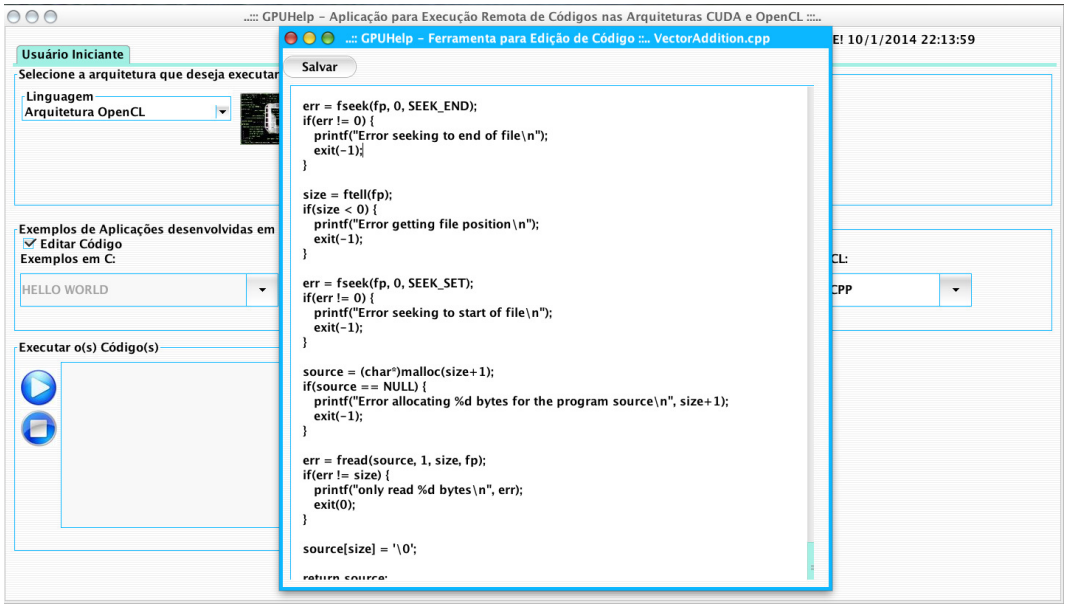


Figura 5.6: Ferramenta para edição de exemplos disponíveis na interface usuário inicial

5.4.3 Tela Usuário Experiente

Caso o usuário da aplicação já tenha conhecimento ou experiência em programação paralela em arquiteturas de GPU, este deve selecionar a opção Usuário Experiente, na Figura 5.1. A Figura 5.8 apresenta a interface para Usuários Experientes da aplicação.

Diferentemente da aplicação para Usuários Iniciantes em programação paralela em GPU, a aplicação para Usuários Experientes permite que o usuário selecione códigos de seu computador para executar no ambiente servidor. Esta aplicação, para aqueles que possuem placas gráficas em suas máquinas, pode vir a servir como uma forma de comparar o desempenho da aplicação executando em sua placa gráfica local e na placa gráfica disponível no ambiente servidor. A Figura 5.9 ilustra o processo de seleção de arquivo(s) na máquina do usuário. O seletor

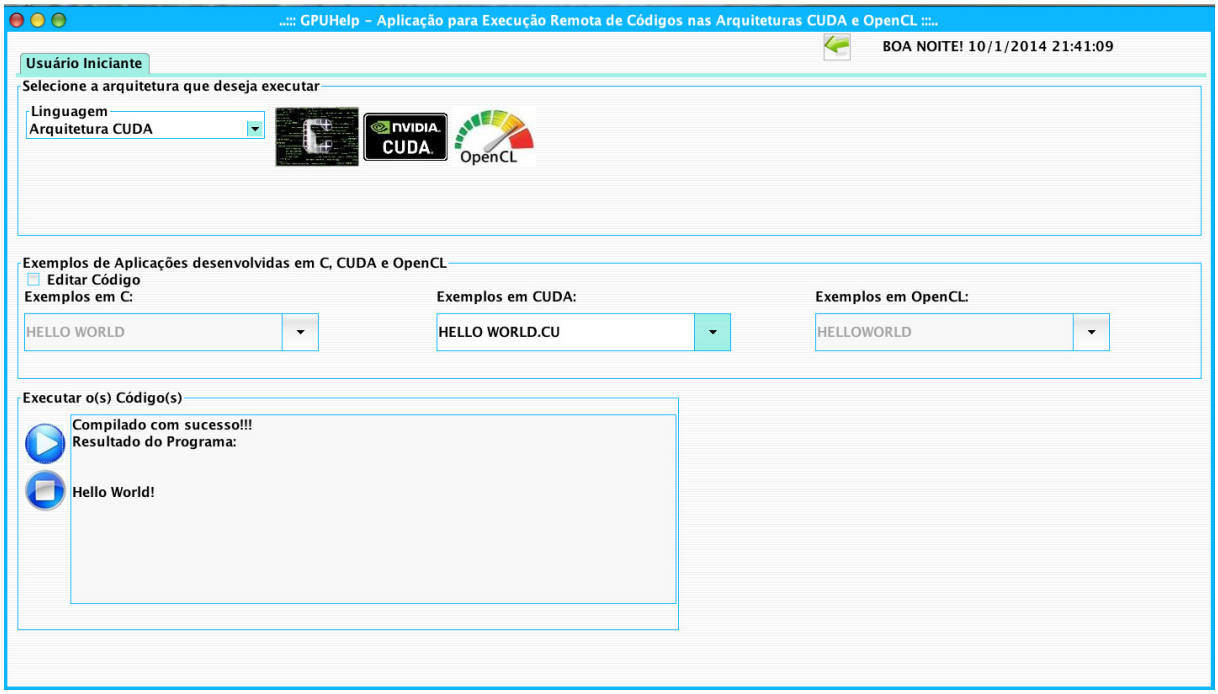


Figura 5.7: Resultado da execução de determinado algoritmo selecionado pelo usuário apresentado na aplicação

de arquivos possui filtro de seleção, permitindo apenas as extensões *".c"*, *".cl"* e *".cu"*, das arquiteturas OpenCL e CUDA. Já a Figura 5.10 apresenta a funcionalidade de o usuário informar uma *flag* personalizada para compilar e executar seus códigos. Através deste recurso, torna-se possível que o usuário possa testar diferentes tipos de códigos com parâmetros específicos. A única premissa existente é de que o utilizador informe a *flag* de forma correta, visto que a aplicação não possui um analisador e organizador de *flags*.

A Figura 5.11 apresenta o resultado da execução de determinado arquivo do computador do usuário no ambiente servidor, retornando o resultado de tal execução na aplicação do cliente. Caso o código possua algum erro de sintaxe ou de programação, o resultado da execução no servidor será apresentado na aplicação para que o usuário realize as devidas alterações necessárias para o correto funcionamento do código.

5.5 Aplicação para Execução Remota de Códigos: Detalhamento do Código

A aplicação para execução remota de códigos foi desenvolvida utilizando o princípio cliente-servidor, sendo que o primeiro possui as ferramentas necessárias para escrita e teste de códigos no ambiente servidor, enquanto que o segundo possui os requisitos físicos necessários para proporcionar o teste de tais códigos.

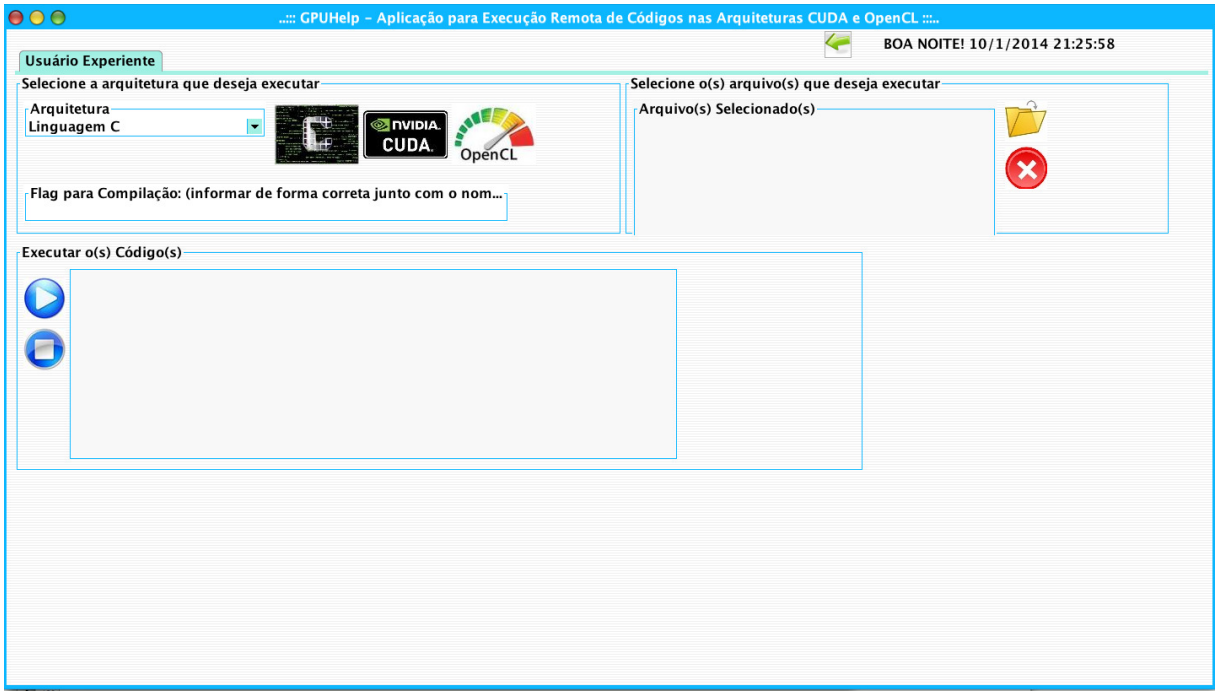


Figura 5.8: Aplicação para usuários com experiência em programação paralela em arquiteturas de GPU

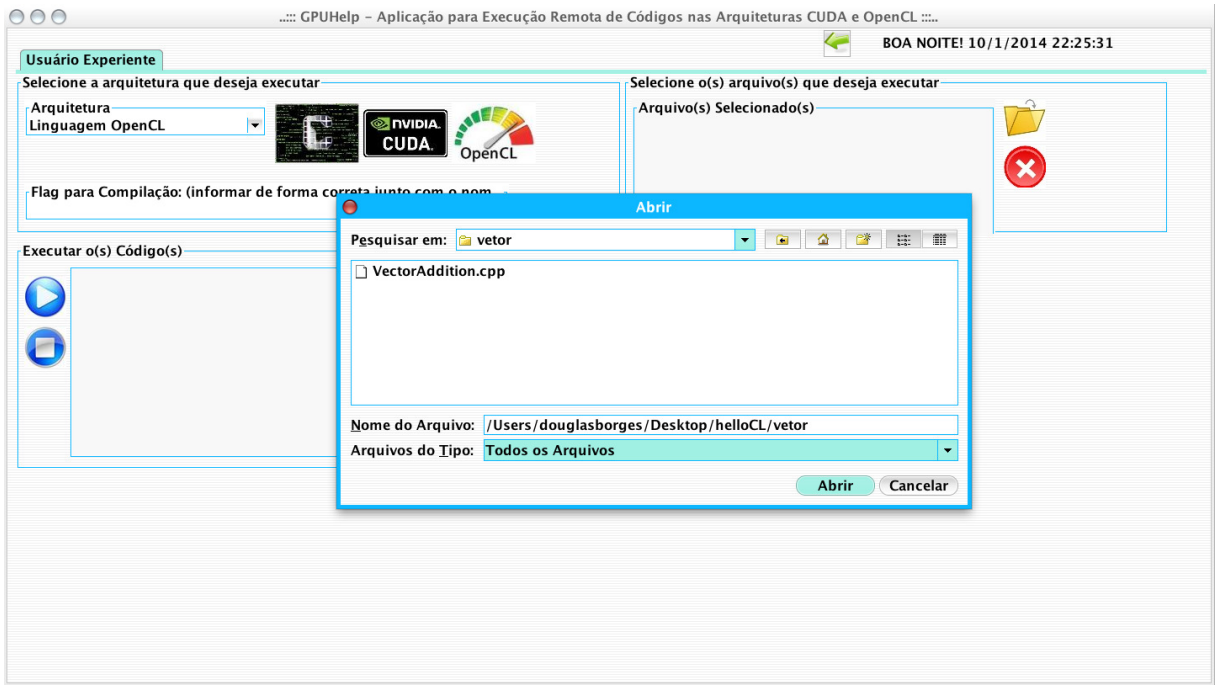


Figura 5.9: Usuário selecionando arquivo(s) do seu computador para execução no ambiente servidor

As seções a seguir detalham o desenvolvimento das principais funcionalidades da aplicação para execução remota de códigos. Caso algum administrador tenha interesse em analisar o código fonte da aplicação, este não encontrará problemas, visto que as classes estão comentadas com suas respectivas funcionalidades.

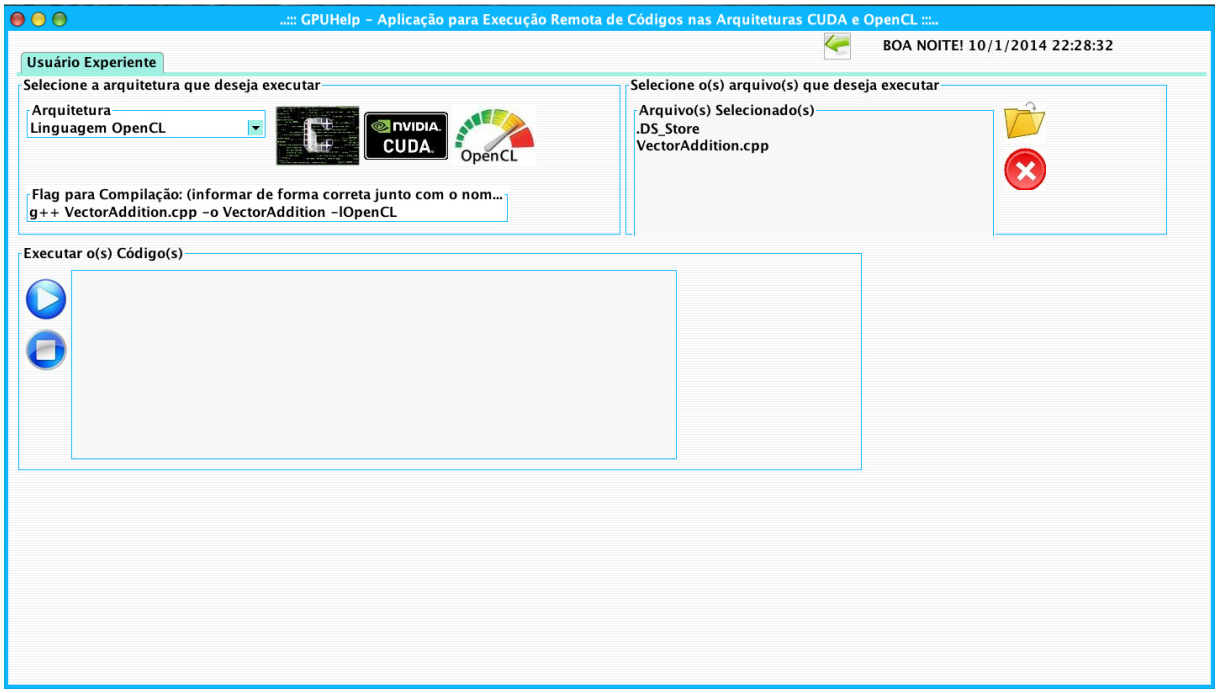


Figura 5.10: Usuário informando uma *flag* personalizada para executar seu código

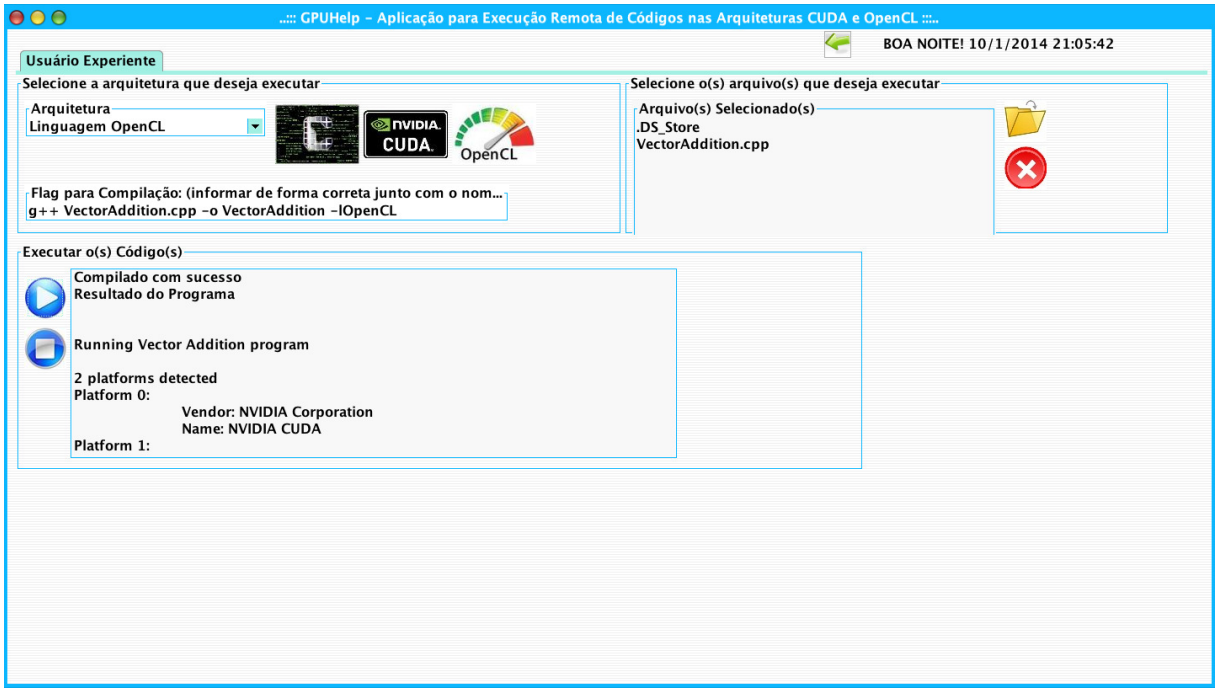


Figura 5.11: Resultado da execução de algoritmo desenvolvido pelo usuário no ambiente servidor

5.5.1 Desenvolvimento das Interfaces

A aplicação para execução remota de códigos é iniciada através da classe *Principal.java*, no *package* Inicio. Esta classe possui as configurações iniciais da aplicação, além de ser responsável por coletar os endereços IP e MAC da máquina do utilizador. A classe *Principal.java* é

responsável ainda por criar uma pasta de trabalho para o usuário, identificar o sistema operacional que este está utilizando e instanciar uma comunicação com o ambiente servidor.

A interface inicial da aplicação é construída na classe *TelaInicial.java*. Essa interface é composta por três painéis, sendo o primeiro para selecionar a arquitetura que deseja executar, o segundo para selecionar os exemplos disponíveis na aplicação e por fim o terceiro, cuja funcionalidade é apresentar o resultado da execução do código na tela do usuário.

Já a interface Usuário Experiente possibilita ao usuário selecionar diversos arquivos, que serão transferidos pela rede, conforme apresentado na Seção 5.5.2. Após transferir os arquivos, a aplicação envia a informação para o servidor RMI com o nome do projeto e o nome da máquina que deseja executar a aplicação. É importante salientar que o usuário deve selecionar todos os arquivos que compõem seu projeto, caso contrário, no momento da execução do código, este não encontrará as referências que não foram enviadas. Desse modo, recomenda-se que o usuário selecione a pasta que contém seu projeto. A pasta de trabalho é criada no computador do usuário com a seguinte sintaxe: *"GPUHelp-Endereço MAC"*, por exemplo: GPUHelp-002365985XYZ (*Mac Address* omitido).

5.5.2 Conexão e Envio de Arquivos ao Servidor

Esta seção reúne as informações referentes a conexão com o ambiente servidor, envio de arquivos, retorno do resultado da execução do código na interface cliente da aplicação, além da passagem da *flag* de compilação ao ambiente servidor.

A aplicação servidor é constituída pelas classes *Servidor.java* e *ServidorServicos.java*. O *Servidor.java* registra a RMI, sendo que a classe *ServidorServicos.java* tem como interface a classe *InterfaceServico.java*, cuja funcionalidade é proporcionar a comunicação entre o cliente e o servidor.

O envio dos arquivos ao servidor, inicialmente, era realizado utilizando o protocolo FTP. Alguns problemas foram encontrados ao utilizador tal metodologia, sendo que o principal foram as restrições encontradas nas redes acadêmicas. Entende-se a necessidade de tais restrições e bloqueios, tendo em vista que as redes Institucionais têm fins acadêmicos. Ao testar o GPUHelp na rede da UFSM, a conexão e envio de arquivos ao servidor não era possível, sendo que em redes domésticas era. Devido a isso, foi realizada uma análise na rede, onde constatou-se os bloqueios existentes nas portas. O FTP utiliza a porta 21, para envio e recebimento de arquivos. Tal porta é bloqueada na rede alunos, mas liberada na rede professores. Ao testar o GPUHelp na

rede alunos, a conexão ao servidor, bem como o envio dos arquivos, não era possível. Porém, ao ser testado na rede professores, a aplicação funcionava. Frente a tal dificuldade, criou-se uma nova metodologia para envio e recebimento dos arquivos. Ao invés de serem enviados arquivos ao servidor, implementou-se no código uma função que transforma os arquivos em *strings*, para envio ao servidor utilizando uma estrutura FIFO *(First In First Out)*, sendo que dessa forma, passou a ser utilizado o protocolo TCP. O servidor recebe tal *string*, montando-a na ordem correta, gerando assim, o arquivo a ser executado. Com isso, o envio de arquivos pela rede tornou-se mais rápido e transparente, visto que os pacotes são tratados como mensagens de texto, que precisam apenas serem montadas no ambiente servidor, de modo que o envio é feito de forma ordenada. Para tornar possível o envio/recebimento dos arquivos, em forma de *strings*, utilizou-se nas classes *usuarioI* e *usuarioE*, que são as classes para o ambiente cliente, a função *getfile*, enquanto que na classe *ServidorServicos* é utilizada a função *setfile*, ou seja, no ambiente servidor. Utilizou-se um *StringBuilder*, por suportar um tamanho maior de dados, quando comparado com uma *string* comum. Ao final de todo o processo, o RMI cria um *Object* e o envia pela rede, em forma de texto, sendo que para finalizar o processo, é realizada a conversão *.toString*, nativa no Java.

Após receber, compilar e executar o resultado, o servidor envia o resultado pelo *Objeto Scanner* do Java ao usuário. A classe *Scanner* faz parte do *package java.util.Scanner*. Através dessa classe é possível realizar diversas interações, entre usuário e aplicação (Java), como por exemplo, capturar o que o usuário digita no teclado, posição do mouse, entre outros.

Ainda relacionado ao ambiente servidor, existem os arquivos, *Prop.ini* e *server.policy*. O arquivo *Prop.ini* possui as informações do servidor que serão replicadas para o cliente. Já o arquivo *server.policy*, de forma resumida, possui as permissões do cliente.

Na funcionalidade de passagem de *flags*, o usuário deve informar a *flag* de maneira correta, bem como o nome do(s) arquivos(s) que deseja executar. As *flags* estão incorporadas ao código que executa no ambiente servidor, como uma passagem de parâmetros.

5.6 Detalhamento do Ambiente Servidor

5.6.1 Características da Máquina Servidor

Uma das dificuldades acerca do desenvolvimento do GPUHelp está relacionada ao ambiente servidor. Para que o GPUHelp funcione conforme o esperado, faz-se necessária uma máquina física, com placa gráfica com suporte a programação paralela em arquiteturas de GPU, para

atuar como ambiente servidor. Tal máquina encontra-se no LSC da UFSM.

Porém, existem questões administrativas acerca da utilização de patrimônio público, que são compreensíveis, pois zelam pelo bem do item em questão. O foco deste trabalho não é entrar em detalhes acerca das questões administrativas que envolvem o uso e disponibilização de bens Institucionais.

Desse modo, como alternativa a contornar tais questões, para o desenvolvimento do GPUHelp, utilizou-se uma máquina *desktop* com placa gráfica com suporte a programação paralela nas arquiteturas de GPU. As características da máquina são descritas na Tabela 5.1.

Tabela 5.1: Características da máquina servidor

Componente	***Memória***
Memória	4 GB
Processador	Intel Core2Quad Q8200 2.3 GHz
Hard Disk	500 GB
Placa Gráfica	NVIDIA GT 218 [GeForce 310]
Sistema Operacional	Linux Ubuntu
CUDA	*Driver* CUDA Versão 5.5
OpenCL	*Driver* OpenCL Versão 2.0

5.6.2 Aplicação Servidor

As classes integrantes da aplicação para o ambiente servidor são: *Servidor.java*, *Servidor-Servicos.java*, do *packag*e Servidor; a classe *InterfaceServicos.java*, do *package* interfaceRMI e, por fim, a classe *TransfereArquivos.java*, do *package* Utilidades.

Visando otimizar as configurações de conexão entre o ambiente servidor e a aplicação para execução remota de códigos (disponível no ambiente cliente), criou-se um arquivo responsável por manter as informações de conexão com o ambiente servidor. O objetivo de tal arquivo é evitar que, caso haja necessidade de ser alterado o endereço IP da máquina servidor, não seja necessário gerar uma nova compilação do código inteiro da aplicação, para então gerar o arquivo *GPUHelp.jar*, que é disponibilizado aos usuários. Tal arquivo é o *prop.ini*, e deve ficar junto com o *.jar* do GPUHelp para que haja comunicação entre ambiente servidor e cliente. Ao ser criada tal metodologia, o professor/administrador pode alterar as configurações de conexão no arquivo *prop.ini* e enviar aos utilizadores, que devem apenas substituir pelo arquivo que possuíam até então. Deve-se alertar aos utilizadores para que não alterem tal arquivo, pois caso isso aconteça, resultará na perda de conexão entre o ambiente cliente e o ambiente servidor. As

configurações e alterações devem ficar a cargo apenas dos administradores do ambiente e da ferramenta para execução remota.

5.7 GPUHelp: Aplicação Completa

5.7.1 GPUHelp: Utilizado como Ambiente Completo

Na Figura 5.12 o GPUHelp é apresentado como sistema operacional nativo no computador de determinado usuário, conforme descrito na Seção 4.3.2. Neste cenário de utilização, o usuário tem a seu alcance todas as ferramentas disponíveis no ambiente, além da aplicação para execução remota de códigos. Já a Figura 5.13 ilustra o editor Eclipse (a), o editor Netbeans (b) e o editor CodeBlocks (c), disponíveis no ambiente GPUHelp, como ferramentas para desenvolvimento de códigos, conforme apresentado na Seção 5.2.3.

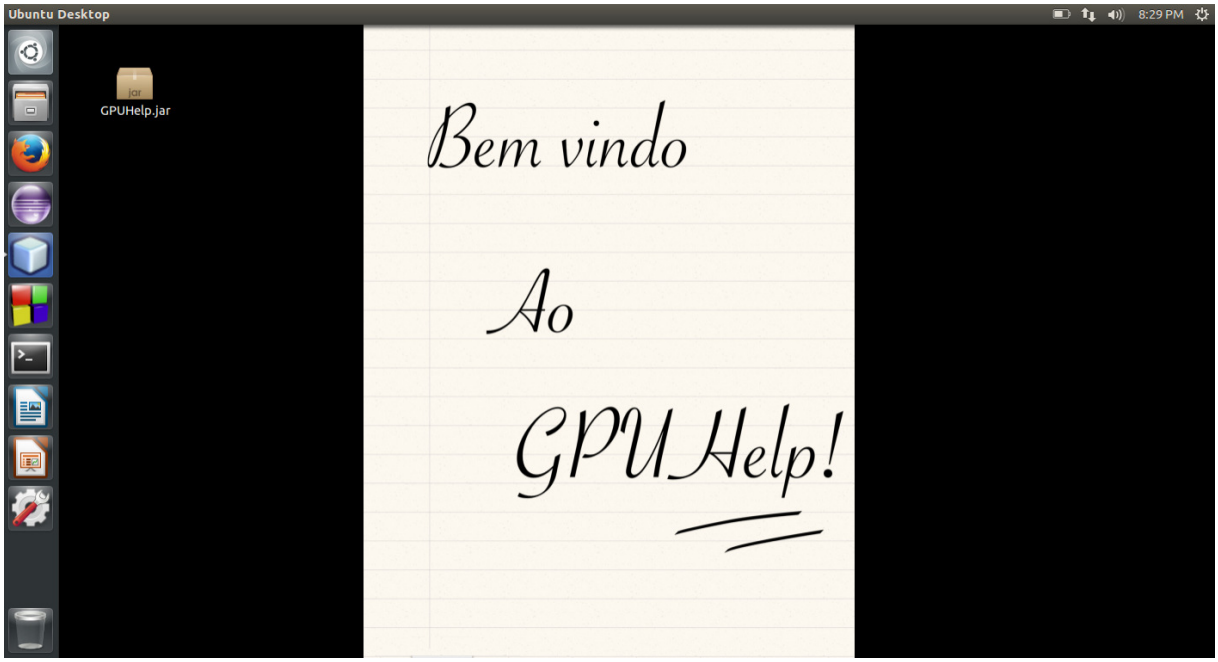


Figura 5.12: GPUHelp sendo utilizado como sistema operacional nativo

A Figura 5.14 ilustra o GPUHelp como sistema operacional virtualizado, conforme descrito na Seção 4.3.3, em um sistema operacional MAC OS X, versão 10.9 – Mavericks, da Apple. A ferramenta utilizada para virtualizar o ambiente foi o *software* Parallels Desktop 9. Conforme descrito anteriormente, a imagem disponibilizada do GPUHelp pode ser virtualizada ainda em sistemas Windows e Linux.

5.7.2 GPUHelp: Utilizado como Ferramenta para Execução Remota de Códigos

A Seção 4.3.4 apresenta a utilização de apenas a ferramenta para execução remota de códigos, disponível no ambiente GPUHelp. Esta utilização tem foco naqueles usuários que possuem as ferramentas para desenvolvimento de códigos configuradas e prontas em seus *desktops* ou *notebooks*. Dois tipos de usuários são foco da utilização de apenas a ferramenta para execução remota: aqueles que não possuem placa gráfica em seus computadores, e aqueles que as possuem, mas pretendem utilizar a ferramenta como comparativo de performance entre seus computadores e o ambiente servidor. A Figura 4.13 ilustra em alto nível a utilização de apenas da aplicação no computador de determinado usuário. Já a Figura 5.15 apresenta a interface inicial da aplicação para execução remota de códigos, sendo executada em um sistema operacional MAC OS X.

A interface para usuário iniciante, selecionada a partir da interface inicial, pode ser visualizada na Figura 5.16. A Figura apresenta a tela para usuário inicial sendo executada na plataforma Linux.

Por fim, a interface para usuários experientes pode ser visualizada na Figura 5.17, sendo executada na plataforma Windows.

5.7.3 Questões de Segurança

Visando preservar a segurança envolvendo o ambiente servidor, desenvolveu-se a ferramenta para execução remota de códigos. Tal ferramenta, realiza o envio de apenas códigos desenvolvidos para as arquiteturas CUDA e OpenCL, através de um filtro que limita as extensões dos arquivos. Porém, não foi implementado nenhum analisador de códigos em sua estrutura.

Outro fator que envolve a segurança do ambiente é a disponibilização do mesmo. Sendo assim, acredita-se que o ambiente deva ser disponibilizado pelo professor diretamente aos alunos da disciplina, ou aqueles que possuam interesse em utilizá-lo ou a aplicação para execução remota de códigos.

Acredita-se no uso ponderado e consciente do ambiente e da aplicação, com foco apenas para fins educacionais. Sendo assim, não é o objetivo do GPUHelp abordar questões de segurança relativamente complexas e que devem ser construídas envolvendo seus utilizadores. O motivo de não terem sido aplicadas determinadas premissas de segurança ao ambiente é o fato de que alguns utilizadores podem optar por utilizá-lo como sistema operacional nativo. Dessa forma, acredita-se não ser possível restringir os utilizadores em suas próprias máquinas

de utilizarem os recursos que consideram necessários.

5.7.4 Limitações da Arquitetura

Algumas limitações foram identificadas ao longo do desenvolvimento do GPUHelp. A primeira delas está relacionada ao tamanho da imagem do ambiente. Não foi possível criar uma imagem *.iso* com um tamanho menor, com base no fato de que todos os componentes instalados são ferramentas necessárias para o correto funcionamento do propósito na qual o ambiente foi projetado.

Outra limitação acerca da arquitetura está relacionada ao ambiente servidor. Atualmente, o GPUHelp conta com apenas uma máquina servidor. Tal máquina é responsável por executar os códigos para os usuários do ambiente cliente do GPUHelp, mais especificamente da aplicação para execução remota de códigos. Caso esta máquina enfrente alguma falha de *hardware* ou problema de conectividade, os usuários não poderão testar seus códigos no ambiente servidor.

Em se tratando da aplicação para execução remota de códigos, esta possui algumas limitações em sua arquitetura. A primeira delas tem relação com o envio dos códigos ao ambiente servidor. Não foi implementado nenhum algoritmo que faça a análise dos códigos a serem enviados. O único filtro que existe na aplicação é com relação aos arquivos que podem ser enviados ao ambiente servidor.

Outra limitação acerca da aplicação está relacionada aos códigos que o usuário pode testar. Alguns códigos não podem ser testados, como por exemplo: códigos que resultem em gráficos, tabelas ou saídas que demandem determinada organização na apresentação dos dados; códigos que resultem em figuras, animações ou outras funções gráficas.

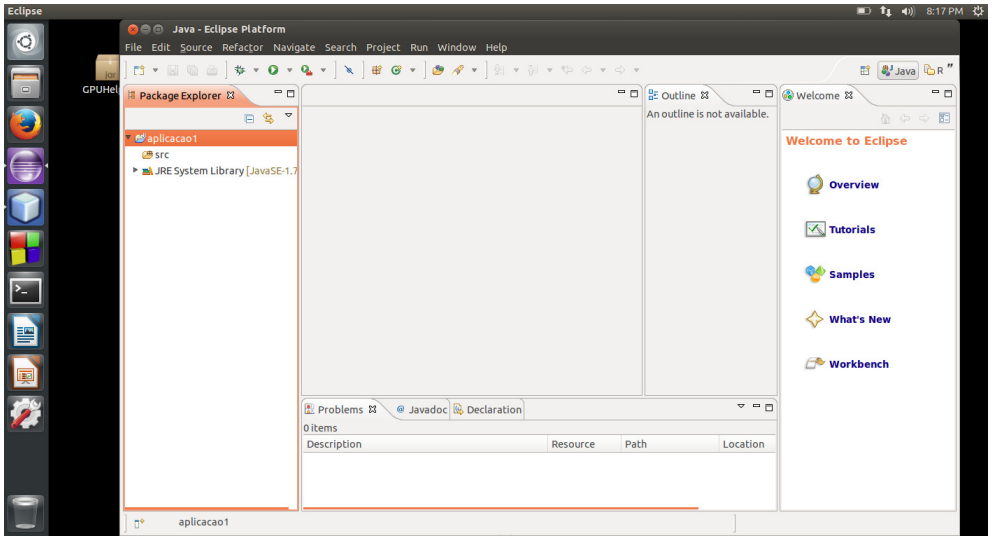
Com relação a aplicação para usuários iniciais, esta apresenta códigos para as arquiteturas CUDA e OpenCL. Buscou-se colocar códigos que tenham a mesma funcionalidade, ou seja, similares, porém, escritos em CUDA e OpenCL, para que os usuários possam compreender as diferenças e particularidades acerca de cada arquitetura. A limitação atrelada a isso, é o baixo número de códigos existentes que são similares. Existem sim, vários códigos nos repositórios das arquiteturas CUDA e OpenCL. Entretanto, como citado anteriormente, buscou-se a similaridade de funcionamento nestes, o que resultou no baixo número de códigos compatíveis.

Outro problema relacionado as arquiteturas CUDA e OpenCL tem relação com a portabilidade destas. Os códigos OpenCL podem ser testados em diferentes máquinas, com placas gráficas de diversos fabricantes, sendo este um padrão portável. Porém, a arquitetura CUDA

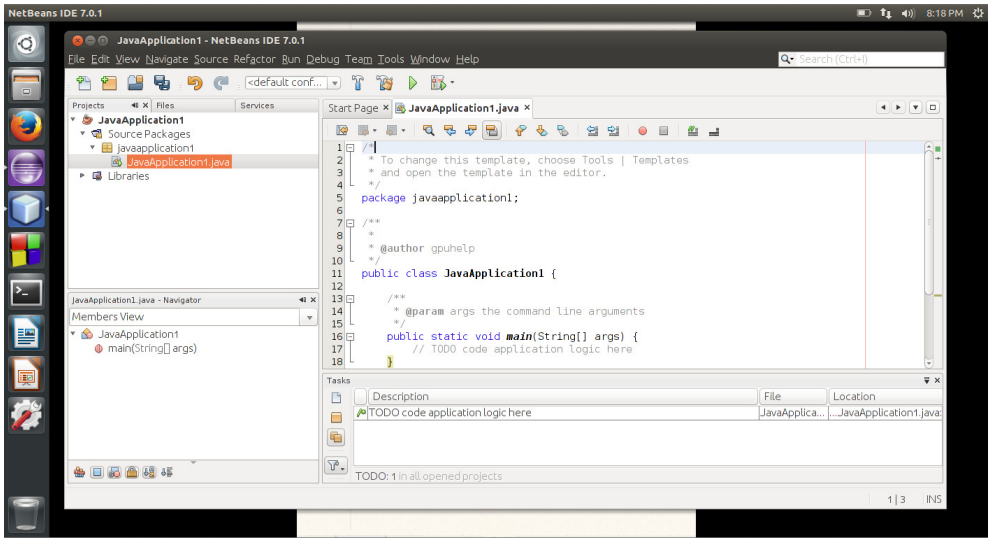
possibilita os testes dos códigos apenas em máquinas que possuam placa gráfica da NVIDIA. Dessa forma, esta arquitetura não oferece uma portabilidade de códigos a seus utilizadores, sendo uma arquitetura proprietária, atrelada a seu fabricante.

5.8 Considerações

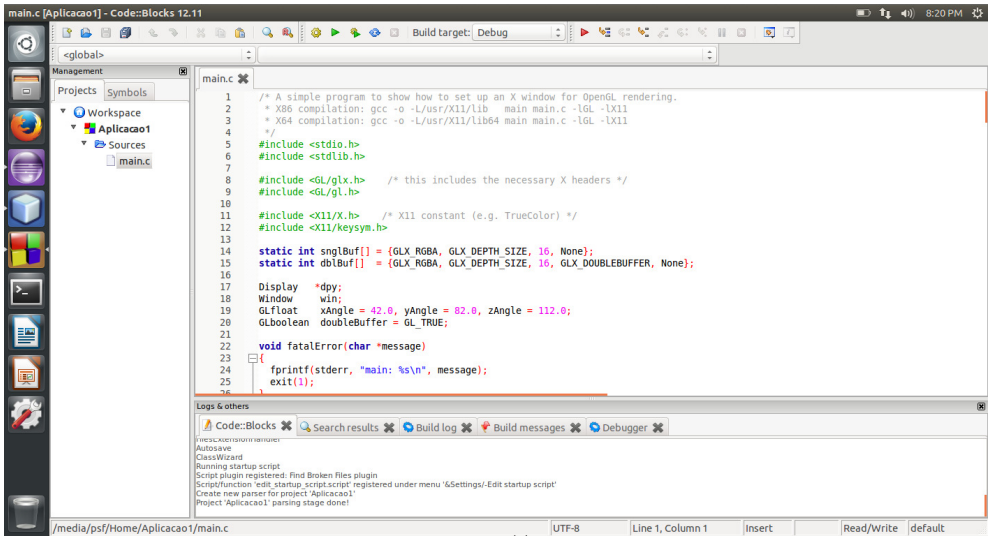
Ao longo do desenvolvimento do GPUHelp buscou-se utilizar ferramentas conhecidas e amplamente utilizadas pela comunidade acadêmica. O Capítulo 5 descreveu o desenvolvimento do GPUHelp. Foram apresentadas as tecnologias utilizadas, as características do ambiente, além de ter sido descrito o desenvolvimento da aplicação para execução remota de códigos, detalhado o ambiente servidor e apresentado o ambiente completo, em diferentes sistemas operacionais, além da aplicação para execução remota de códigos. O Capítulo 6 apresenta os testes realizados com o ambiente GPUHelp e um comparativo deste com o ambiente StarHPC (IVICA; RILEY; SHUBERT, 2009), bem como os resultados obtidos.



(a) Eclipse



(b) Netbeans



(c) CodeBlocks

Figura 5.13: Editores de códigos disponíveis para os utilizadores do GPUHelp, Eclipse (a), Netbeans (b) e CodeBlocks (c)

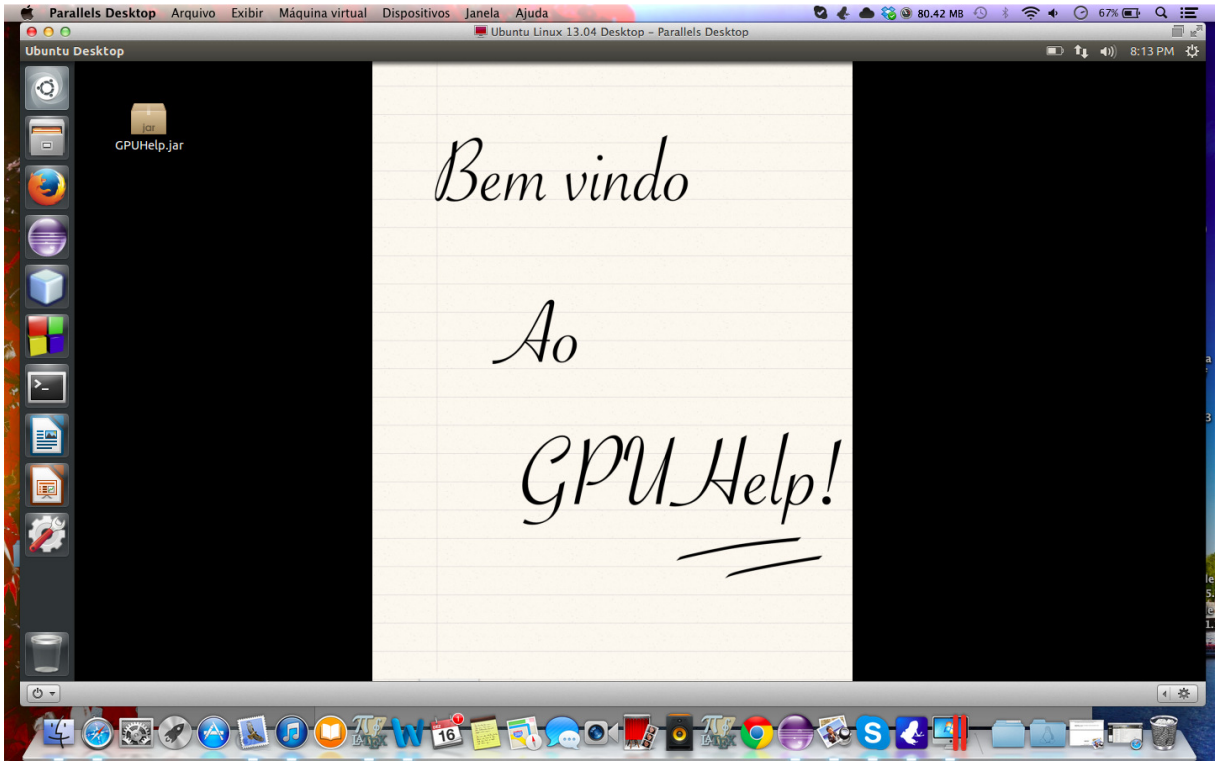


Figura 5.14: GPUHelp virtualizado em um sistema operacional MAC OS X

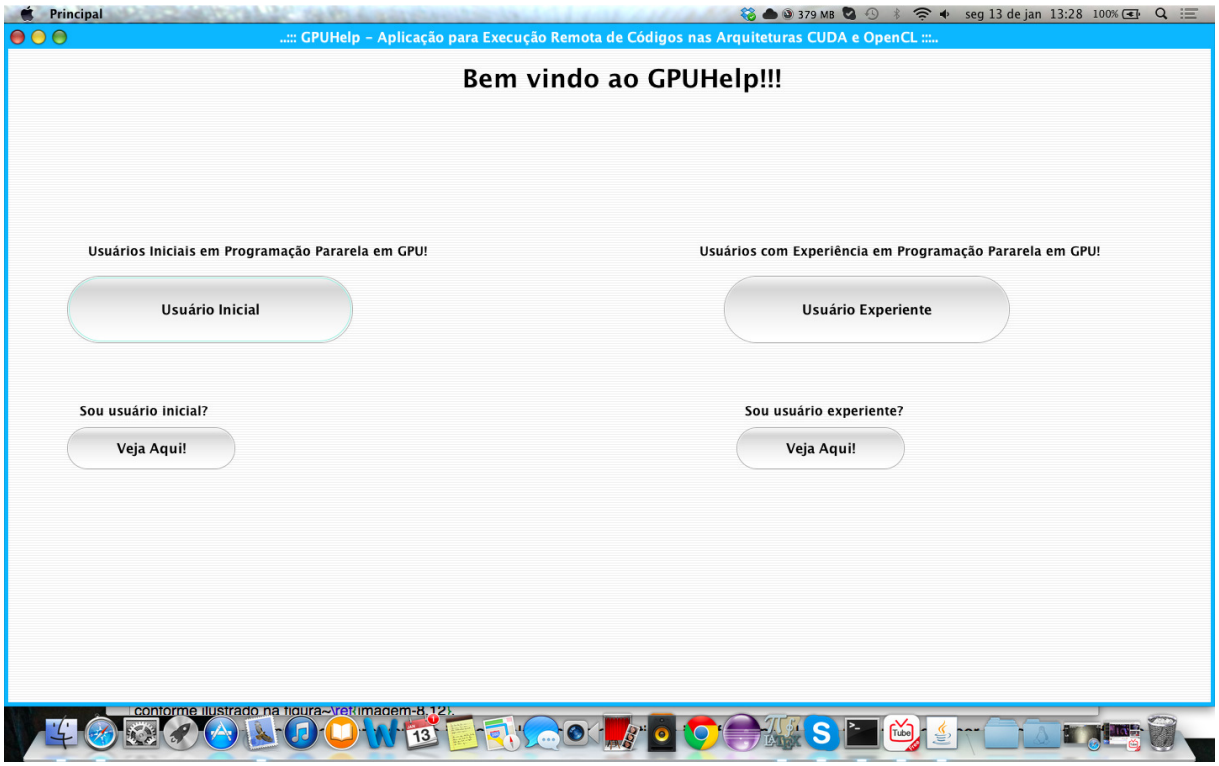


Figura 5.15: Interface inicial da aplicação para execução remota de códigos no sistema operacional MAC OS X

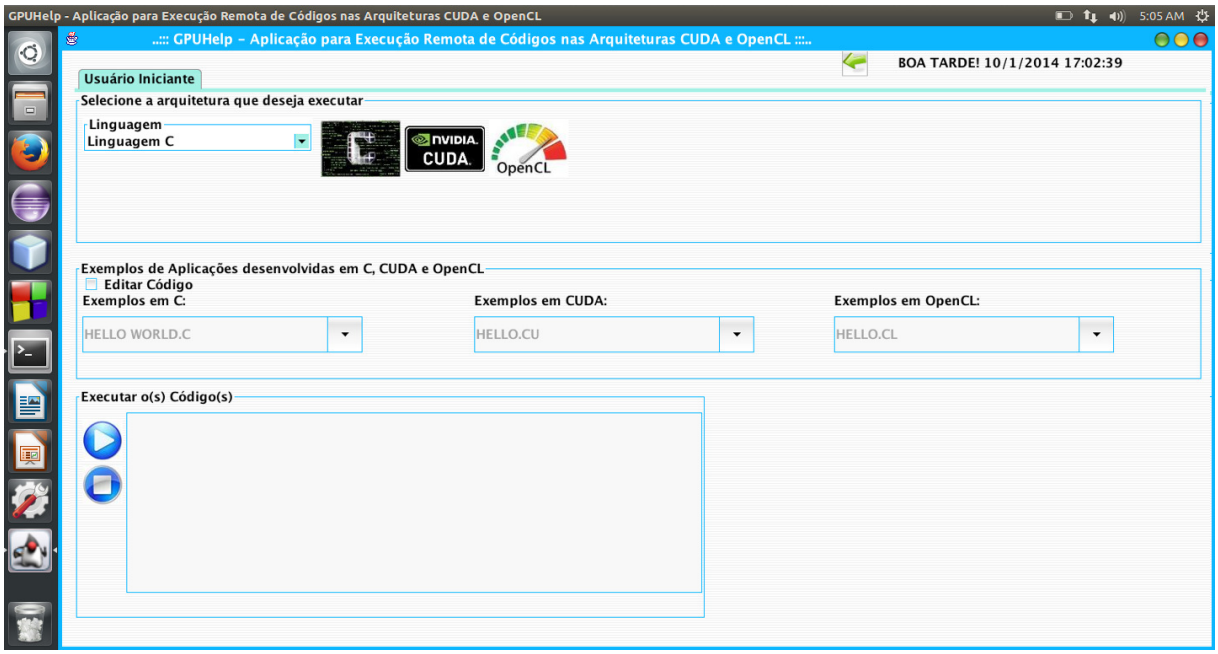


Figura 5.16: Aplicação para usuários iniciantes executada no sistema Linux

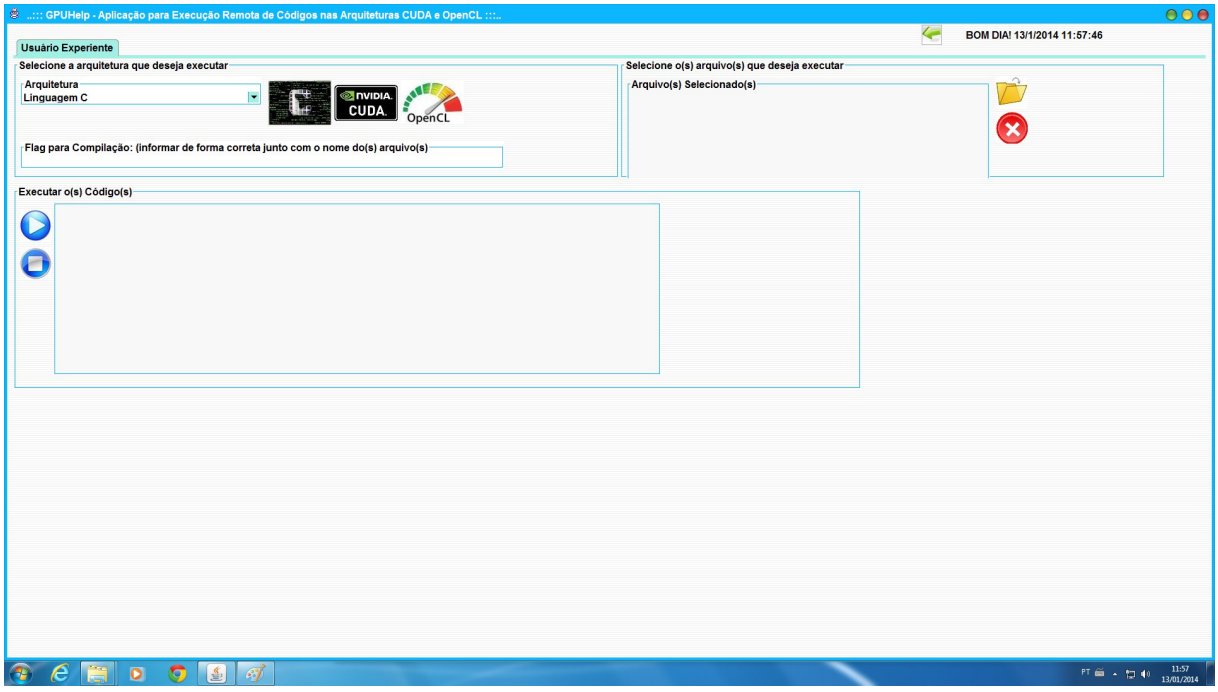


Figura 5.17: Aplicação para usuários experientes sendo executada na plataforma Windows

6 AVALIAÇÃO E RESULTADOS

Este capítulo apresenta os resultados obtidos a partir do desenvolvimento do GPUHelp. A Seção 6.1 apresenta uma avaliação comparativa entre os ambientes StarHPC (IVICA; RILEY; SHUBERT, 2009) e GPUHelp. Enquanto que as considerações acerca da avaliação feita pelos alunos do curso de Ciência da Computação da UFSM são apresentadas na Seção 6.2. Por fim, são apresentadas algumas considerações com relação à avaliação deste trabalho.

6.1 Análise Qualitativa

6.1.1 Comparativo das Características

O desenvolvimento de ambientes de apoio ao ensino de programação paralela, sejam estes para arquiteturas de CPU ou GPU, é uma tarefa relativamente complexa. Dessa forma, os ambientes geralmente são desenvolvidos por profissionais contratados pelas Instituições e estas limitam o acesso da ferramenta a sua comunidade acadêmica. Este é o caso da ferramenta StarHPC (IVICA; RILEY; SHUBERT, 2009), cujo foco é a área de programação paralela para as arquiteturas OpenMP e MPI. Outras características acerca do ambiente StarHPC são apresentadas na Seção 3.2.

Até o momento do desenvolvimento deste trabalho não existem ambientes de apoio ao ensino de programação paralela para arquiteturas de GPU. A ideia inicial era comparar o GPUHelp com outro ambiente de apoio ao ensino com suporte as arquiteturas de GPU. Porém, devido à inexistência de tal ambiente, buscou-se comparar as características principais do ambiente desenvolvido neste trabalho com o ambiente StarHPC. Esperava-se poder incluir nas comparações o ambiente desenvolvido por (DELISTAVROU; MARGARITIS, 2011). Porém, tal ambiente, mesmo sendo baseado inteiramente em soluções de *software* livre, é restrito a Universidade na qual foi projetado pelo autor.

Sendo assim, foram comparadas as características técnicas e de suporte de cada ambiente. Além de características de usabilidade, portabilidade, nível dos usuários finais e administradores, entre outras. Ou seja, as ferramentas tiveram seus propósitos finais abstraídos para esta avaliação, pois ambas foram caracterizadas como ambientes de apoio ao ensino, sem focar nas arquiteturas suportadas por cada um.

A seguir são analisadas algumas características consideradas importantes para os ambientes de apoio ao ensino de programação paralela, tanto em arquiteturas de CPU quanto GPU. Tais

características referem-se a qualidades que podem ser classificadas como difíceis de serem avaliadas e mensuradas, mas que permitem serem discutidas e comparadas quando presentes nos diferentes sistemas. A ideia é classificar cada ambiente com uma escala classificatória.

Gerenciamento: administrador e usuários das soluções devem ter condições de realizarem suas atividades, como alteração de características funcionais e gerenciais do ambiente local, das aplicações que compõem o ambiente e no ambiente servidor (administradores), desenvolvimento e teste de códigos, instalação e configuração de ferramentas adicionais necessárias (usuários), nos ambientes de forma prática, simples e rápida. Acredita-se que os administradores do sistema devem ser capazes de gerenciar e alterar, além de adicionar e/ou remover, as funcionalidades dos ambientes com eficiência e praticidade. Já os usuários finais devem ser capazes de utilizar o ambiente de forma prática e intuitiva, e ainda, que deva ser possível a estes instalar e configurar ferramentas que possam vir a ser necessárias para o desenvolvimento de suas atividades. Desse modo, classifica-se como gerenciamento alto a possibilidade de o administrador e o usuário final poderem ter controle total sobre o sistema, podendo realizar as configurações que considerarem pertinentes. Isso inclui instalar, modificar e configurar ferramentas que atendam as suas necessidades particulares. Para tanto, o sistema deve, principalmente, ser aberto e composto de ferramentas conhecidas pelos utilizadores. Já a flexibilidade gerencial média impacta em um sistema que permita alguma, mas pouca, intervenção por parte dos usuários. Por fim, um sistema com flexibilidade gerencial baixa não permite aos utilizadores nenhum tipo de alteração em sua arquitetura, sendo possível a estes apenas utilizarem seus componentes e ferramentas na forma estabelecida pelos desenvolvedores/administradores do ambiente.

Gerenciamento (por parte dos Administradores):

StarHPC: flexibilidade gerencial **média**. Os administradores do sistema têm domínio sobre as principais funcionalidades do sistema. Porém, parte do gerenciamento e administração da ferramenta fica por conta da empresa Amazon, visto que esta hospeda o ambiente servidor.

GPUHelp: flexibilidade gerencial **alta**. Os administradores do sistema tem total controle sobre o ambiente. Estes podem efetuar alterações no ambiente cliente com praticidade e simplicidade. É possível adicionar, remover ou alterar configurações das ferramentas que compõem o ambiente. A aplicação para execução remota de códigos possui seu código fonte aberto, permitindo que os administradores possam alterar suas funcionalidades. Por fim, o ambiente servidor está hospedado na UFSM, em uma máquina do LSC.

Gerenciamento (por parte dos Usuários Finais):

StarHPC: flexibilidade gerencial **baixa**. a ferramenta não permite nenhuma alteração por parte dos usuários finais. Estes podem apenas utilizar as ferramentas disponíveis no ambiente. A ferramenta disponibiliza apenas um editor de código, o *software* Eclipse. Caso o utilizador faça uso, ou tenha interesse em utilizar outra ferramenta diferente desta, não será possível. O nível de restrições é alto.

GPUHelp: flexibilidade gerencial **alta**. A ferramenta GPUHelp possibilita aos usuários finais o gerenciamento completo do ambiente cliente. O ambiente não permite ao usuário interagir (acessar ou modificar) com o ambiente servidor. O único meio de comunicação com o ambiente servidor é através da ferramenta para execução remota de códigos. Ou seja, o ambiente servidor está protegido, permitindo acesso apenas aos administradores do ambiente. A ferramenta para execução remota de códigos também não permite alterações pelos usuários finais. Porém, o ambiente permite que o usuário possa instalar, alterar ou remover ferramentas que considera necessárias para sua utilização. Isso permite aos utilizadores configurarem o ambiente da maneira que melhor entenderem. Acredita-se que este tipo de direito deva ser concedido aos usuários finais, visto que eles podem vir a utilizar a ferramenta como sistema operacional nativo em suas máquinas.

Escalabilidade: acredita-se que os ambientes de apoio ao ensino devam levar em consideração questões de escalabilidade, onde ao aumentar o número de usuários, não implique em problemas de desempenho e funcionamento do sistema. A utilização de arquiteturas distribuídas, escaláveis, caracteriza-se como uma alternativa que pode vir a amenizar o impacto negativo da escalabilidade em ambientes de apoio ao ensino. Um sistema com escalabilidade alta, permite sua utilização além das redes locais, embora isso possa vir a impactar em ferramentas complementares para suportar a sincronização e replicação dos dados. Já a escalabilidade média compreende o sistema que possui ou permite a utilização de algum recurso que escale a sua aplicação em redes locais. Por fim, a escalabilidade baixa objetiva a aplicação de um sistema local, ou seja, desprovido de recursos que possam tornar possível a sua aplicação em escalas maiores (nas redes locais ou redes de longa distância).

StarHPC: escalabilidade **alta**. A ferramenta StarHPC origina-se de uma arquitetura baseada em computação em nuvem, com alta disponibilidade e performance. Tal ambiente está hospedado em um Cluster da Amazon. Isso impacta em um alto desempenho, permitindo a este suportar uma alta demanda de poder de processamento.

GPUHelp: escalabilidade **baixa**. O ambiente GPUHelp foi projetado para atender todos os

usuários a partir de um único servidor. Isso impacta em um risco, pois caso tal máquina falhe ou fique indisponível, os usuários não terão onde testar seus códigos.

Heterogeneidade: um ambiente de apoio ao ensino deve ser capaz de ser virtualizado em diferentes soluções de virtualização disponíveis atualmente. Ao ser projetado voltado a um único sistema de virtualização, o ambiente limita o utilizador àquela ferramenta. Isso pode ser considerado uma característica negativa por alguns usuários. Heterogeneidade alta abrange aqueles ambientes que podem ser utilizados em diferentes *softwares* de virtualização, de forma transparente aos utilizadores. Já um sistema com heterogeneidade média pode ser classificado como aquele que suporta alguns sistemas de virtualização e que pode ser adaptado, caso necessário, pelos usuários finais para outras ferramentas de virtualização. Por fim, um sistema com heterogeneidade baixa significa que tal ambiente foi projetado com foco em uma única ferramenta para virtualização, como o VirtualBox, por exemplo.

StarHPC: heterogeneidade **baixa**. A ferramenta StarHPC disponibiliza aos seus utilizadores uma imagem de máquina virtual compatível apenas com a solução VirtualBox. Isso limita o usuário a utilizar apenas esta solução.

GPUHelp: heterogeneidade **alta**. O ambiente GPUHelp foi concebido de maneira a suportar diferentes ferramentas de virtualização, ou seja, com uma heterogeneidade alta.

Adaptabilidade: caracteriza-se como adaptável o sistema capaz de ser facilmente adaptado a diferentes cenários e contextos de aplicação. Um sistema com um nível de adaptabilidade considerado bom, prevê em sua arquitetura componentes independentes, ou seja, fracamente acoplados. Dessa forma, novos cenários podem ser rapidamente construídos através da inclusão de novos componentes. Um sistema com adaptabilidade alta é aquele projetado e constituído de ferramentas simples e independentes, ou seja, fracamente acopladas. Já um sistema com adaptabilidade média pode ser considerado aquele que possui uma arquitetura mista, composta de forma parcial por componentes independentes. Por fim, um sistema com baixa adaptabilidade é aquele que possui uma arquitetura mais rígida, única e que não permite alterações, ou seja, projetado sem a previsão de novos componentes.

StarHPC: adaptabilidade **baixa**. O ambiente StarHPC disponibiliza aos usuários uma imagem de máquina virtual compatível apenas com a ferramenta VirtualBox. Adicionalmente, sua arquitetura não prevê a adição ou alteração de novos componentes, por parte dos usuários.

GPUHelp: adaptabilidade **alta**. Uma das características do ambiente GPUHelp é a flexibilidade, obtida graças a utilização de ferramentas com alto índice de adoção atualmente. O

ambiente foi projetado de maneira a prever a adição ou alteração de novos componentes e funcionalidades por parte dos usuários finais. Isso torna o ambiente flexível e adaptável a diferentes cenários e contextos.

Extensibilidade: tal característica prevê a utilização do ambiente em situações distintas da que foi originalmente projetado. Ou ainda, o uso de outras tecnologias. Nesse sentido, uma arquitetura classificada como fracamente acoplada, que é composta por ferramentas flexíveis, contribui para o uso de outras tecnologias ou adaptações futuras a cenários distintos. Um sistema com extensibilidade alta é aquele projetado com foco em atender cenários diversos, composto por ferramentas que proveem uma flexibilidade relativamente alta a seus utilizadores. Já um sistema com extensibilidade média caracteriza aqueles ambientes que suportam determinado tipo de adequação a alguma situação distinta da inicialmente planejada. Finalmente, um sistema com extensibilidade baixa é aquele cujo projeto é direcionado a determinado cenário, podendo ser utilizado somente mediante as situações pré definidas pelos seus desenvolvedores.

StarHPC: extensibilidade **baixa**. A solução StarHPC disponibilizada aos usuários finais não leva em consideração necessidades de uso diferentes das projetadas inicialmente em sua arquitetura. O ambiente possibilita o desenvolvimento de códigos em arquiteturas OpenMP e MPI, sem outras funcionalidades a serem disponibilizadas para os usuários.

GPUHelp: extensibilidade **alta**. Já a solução GPUHelp foi projetada de maneira a proporcionar aos utilizadores uma extensibilidade alta, permitindo que os mesmos possam utilizar a ferramenta em diferentes situações, possibilitando a instalação, configuração e remoção de *software* sempre que necessário. Como exemplo, pode-se citar o caso do aluno que precisa desenvolver um código nas arquiteturas OpenMP ou MPI. O GPUHelp possui foco nas arquiteturas de GPU, ou seja, no CUDA e no OpenCL. Porém, o aluno precisa apenas instalar e configurar o suporte as arquiteturas OpenMP e MPI no ambiente, para que torne-se possível desenvolver e testar códigos escritos para tais arquiteturas.

A Tabela 6.1 sintetiza o comparativo das características apresentadas anteriormente, entre as soluções analisadas para ambientes de apoio ao ensino de programação paralela, StarHPC e GPUHelp. Conforme pode ser observado, existem algumas características similares entre os dois ambientes. A ferramenta StarHPC possui uma melhor escalabilidade quando comparada com o GPUHelp. Contudo, existem alguns itens em que o GPUHelp destaca-se frente o StarHPC, sendo eles o gerenciamento, a heterogeneidade, a adaptabilidade e a extensibilidade.

***Softwares* de Virtualização Suportados:** suportar diferentes ferramentas virtuais é de suma

Tabela 6.1: Comparativo de características das ferramentas StarHPC e GPUHelp

Características	**StarHPC**	**GPUHelp**
Gerenciamento (Administradores)	Média	Alta
Gerenciamento (Usuários Finais)	Baixa	Alta
Escalabilidade	Alta	Baixa
Heterogeneidade	Baixa	Alta
Adaptabilidade	Baixa	Alta
Extensibilidade	Baixa	Alta

importância atualmente. Neste sentido, um ambiente de apoio ao ensino deve ser projetado para permitir o suporte as diferentes ferramentas de virtualização disponíveis no cenário atual. Esta deve ser uma característica transparente ao usuário final. A Tabela 6.2, de forma sucinta, realiza um comparativo entre as ferramentas de virtualização suportadas pelo StarHPC e pelo GPUHelp.

Tabela 6.2: Comparativo das ferramentas de virtualização compatíveis com os ambientes StarHPC e GPUHelp

***Software* para Virtualização**	**StarHPC**	**GPUHelp**
VMWare	Não	Sim
VirtualBox	Sim	Sim
Parallels	Não	Sim

Dando continuidade, serão apresentadas algumas características e limitações acerca dos ambientes StarHPC e GPUHelp. A Tabela 6.3 apresenta um comparativo entre os dois sistemas avaliados.

Solução livre/aberta: definida como a característica de a solução ser livre e aberta, podendo qualquer pessoa utilizá-la, adaptá-la ou alterá-la de acordo com suas necessidades. Essa é uma característica importante para uma grande parte dos utilizadores, além de diferentes cenários de aplicação. Cabe salientar que, o GPUHelp é uma solução livre, porém, ainda não está disponível para alunos externos à Universidade. Isso decorre do fato de o GPUHelp fazer uso de uma máquina servidor que pertence à UFSM, ou seja, um bem público. Dessa forma, precisa-se do consentimento dos órgãos administrativos para que a solução possa ser disponibilizada ao público em geral – o que não é o foco deste trabalho tratar de tal questão, pois envolve uma série de fatores e interesses administrativos além de políticas internas da Instituição.

Dependência de conexão a rede: tal característica define se os sistemas dependem de rede ou Internet para funcionarem. Ambos os sistemas dependem de conexão à *Internet* para

possibilitar o teste dos códigos, porém, o desenvolvimento é feito localmente.

Nível dos usuários administradores: avalia o grau de conhecimento necessário pelos usuários administradores dos ambientes.

Nível dos usuários finais: avalia o grau de conhecimento necessário aos usuários finais dos ambientes.

Múltiplas IDEs para desenvolvimento de códigos: esta característica avalia a(s) ferramenta(s) disponíveis nos ambientes para desenvolvimento de códigos pelos usuários finais.

Usuários possuem direitos administrativos: avalia se os usuários finais possuem direitos administrativos nos ambientes, possibilitando assim, a instalação, alteração e remoção de ferramentas.

Tabela 6.3: Características e limitações das arquiteturas StarHPC e GPUHelp

Características	**StarHPC**	**GPUHelp**
Solução Livre/Aberta	Não	Sim*
Dependência de conexão a rede	Sim para testes de código	Sim para testes de código
Nível dos usuários administradores	Alto	Médio/Alto
Nível dos usuários finais	Médio	Baixo/Médio
Múltiplas IDEs para desenvolvimento de códigos	Não	Sim
Usuários possuem direitos administrativos	Não	Sim

[*]A alunos da UFSM, por enquanto.

Conforme pode ser observado na Tabela 6.1, Tabela 6.2 e na Tabela 6.3, existem vantagens e desvantagens em cada uma das arquiteturas. A escolha por um ambiente ou outro, está relacionado com as necessidades e demandas de cada usuário em particular.

6.1.2 Comparativo de Funcionalidades

A seguir será realizado um comparativo das funcionalidades dos ambientes StarHPC e GPUHelp. O objetivo é demonstrar e avaliar as características funcionais de cada um dos ambientes, apresentando os resultados obtidos na sequência.

Linguagens suportadas:

StarHPC: o ambiente possui suporte as arquiteturas OpenMP e MPI. Não é possível adicionar suporte a outra linguagem.

GPUHelp: o GPUHelp possui suporte as arquiteturas de programação paralela em GPUs, o OpenCL e o CUDA. Além disso, é possível adicionar suporte a outras linguagens, devido à característica de o GPUHelp prover acesso administrativo a seus utilizadores.

Ferramentas para escrita de códigos:

StarHPC: o ambiente conta com a ferramenta Eclipse.

GPUHelp: possui instaladas e configuradas de forma nativa as seguintes ferramentas: Netbeans, Eclipse e CodeBlocks. É possível ainda, que o usuário instale alguma outra ferramenta de sua preferência.

Ferramentas para virtualização:

StarHPC: a imagem disponibilizada possui suporte apenas para a ferramenta VirtualBox.

GPUHelp: a imagem disponibilizada possui suporte as principais ferramentas de virtualização disponíveis no mercado atualmente: VMWare, VirtualBox e Parallels.

Tamanho da Imagem:

StarHPC: o ambiente StarHPC teve removidas as principais ferramentas de seu sistema operacional. Estão habilitadas apenas a ferramenta para edição de códigos (Eclipse) e um navegador *Web*. Além, claro, dos *scripts* de configuração do ambiente e de conexão com o Cluster da Amazon. O tamanho da imagem do StarHPC é 1.5 GB.

GPUHelp: diferentemente da ferramenta StarHPC, o ambiente GPUHelp não teve os componentes do sistema operacional removidos. Pelo contrário, foram adicionadas funcionalidades à arquitetura. Entre elas, podem ser citadas: ferramentas para desenvolvimento de códigos, *drivers* para suporte à programação paralela em arquiteturas de GPU e a aplicação para execução remota de códigos. A imagem do ambiente GPUHelp possui 5.6 GB.

Sistemas Operacionais Suportados:

StarHPC: o ambiente StarHPC possui suporte aos seguintes sistemas operacionais: Windows, MAC e Linux.

GPUHelp: o ambiente GPUHelp possui suporte aos mesmos sistemas operacionais que o StarHPC. Porém, diferentemente do StarHPC, o GPUHelp pode ser instalado e configurado como sistema operacional nativo no *notebook* ou *desktop* do usuário.

Conexão com o Ambiente Servidor:

StarHPC: no ambiente StarHPC, a conexão com o ambiente servidor é realizada através de SSH. Os alunos possuem um usuário/senha para poderem acessar o Cluster da Amazon e executarem seus códigos no ambiente servidor.

GPUHelp: o ambiente GPUHelp conta com uma aplicação para execução remota de códigos, que realiza a conexão com o ambiente servidor de forma transparente aos usuários. Desse modo, não é necessário que os alunos possuam acesso ou contas de usuário ao ambiente servidor

hospedado na UFSM.

Restrições Administrativas nos Ambientes:

StarHPC: o ambiente StarHPC possui um alto índice de restrições. Os alunos podem apenas desenvolver e testar seus códigos no ambiente, não sendo possível instalar, alterar ou remover quaisquer ferramentas.

GPUHelp: já o ambiente GPUHelp concede acesso administrativo aos usuários. Acredita-se que pelo fato de os usuários poderem optar pela utilização de tal ambiente como sistema operacional nativo, não é cabível aplicar altos níveis de restrições no ambiente. Para isso, espera-se que os usuários utilizem a ferramenta de forma consciente, devido ao seu objetivo: atuar como ferramenta de apoio na disciplina de programação paralela.

A Tabela 6.4 sumariza as características comparadas até agora. O objetivo é apresentar ao leitor de forma clara e sucinta tais informações.

Tabela 6.4: Visão geral das características dos ambientes StarHPC e GPUHelp

Características	**StarHPC**	**GPUHelp**
Linguagens Suportadas (de forma nativa)	OpenMP e MPI	OpenCL e CUDA
Linguagens que Podem ter Suporte Adicionado pelo Usuário	Nenhuma	OpenMP, MPI, entre outras
Ferramentas para Virtualização Suportadas	VirtualBox	VMWare, VirtualBox, Parallels
Tamanho da Imagem	1.5GB	5.6GB
Sistemas Operacionais Suportados	Windows, Linux e MAC	Windows, Linux e MAC
Conexão com o Ambiente Servidor	Via SSH	Via Aplicação para Execução Remota de Códigos
Nível de Restrições Administrativas no Ambiente	Alto	Baixo

6.2 Opiniões dos Alunos

Esta seção descreve a entrevista realizada com os alunos em fase de conclusão do curso de Ciência da Computação da UFSM. A entrevista está dividida em três momentos. No primeiro

momento, conforme apresentado na Seção 6.2.1, são coletadas as informações dos alunos a respeito do ensino de programação paralela e de programação paralela em GPU, além de questões técnicas relacionadas a estas áreas e são introduzidos os ambientes de apoio ao ensino de programação paralela. Dando continuidade, o segundo momento, descrito na Seção 6.2.2, apresenta o GPUHelp. Por fim, são coletadas as informações dos alunos acerca do ambiente e da aplicação para execução remota de códigos, conforme apresentado na Seção 6.2.3.

6.2.1 Entrevista sobre Programação Paralela e Programação Paralela em GPUs

Com o objetivo de obter opiniões diferenciadas acerca da área de programação paralela e de GPUs, recorreu-se a uma abordagem qualitativa, concretizada através de entrevistas individuais com 7 alunos em fase de conclusão do Curso de Ciência da Computação da Universidade Federal de Santa Maria – UFSM. Os alunos foram escolhidos por já possuírem conhecimentos sobre a área de programação paralela e de programação paralela em GPUs, adquiridos por meio de disciplinas obrigatórias e optativas. Com isso, entende-se que suas opiniões e percepções seriam mais fundamentadas do que as de alunos sem nenhum conhecimento da área.

A fim de obter resultados que contribuíssem da forma esperada para este trabalho, procurou-se primeiramente focar em questões relacionadas às dificuldades do aprendizado de programação paralela e distribuída. Em seguida, buscou-se conhecer as opiniões e conhecimentos dos alunos acerca dos ambientes de apoio ao ensino de programação paralela e de programação paralela em arquiteturas de GPUs. Foram ainda, apresentadas em alto nível as pesquisas acerca dos ambientes de apoio ao ensino de programação paralela aos alunos, conforme apresentado na Seção 3.2 desta Dissertação. A Tabela 6.5 apresenta o roteiro da entrevista adotado para com os alunos selecionados.

Obteve-se respostas distintas com cada aluno entrevistado. A diversidade de opiniões tornou a pesquisa mais rica e diversificada, contribuindo para a construção do ambiente proposto neste trabalho, o GPUHelp.

Como pode ser observado, o roteiro da entrevista relacionou de forma geral diferentes temas acerca do ensino-aprendizado na área de programação paralela e programação paralela em arquiteturas de GPU, além dos ambientes de apoio ao ensino destas áreas. Esse roteiro fez com que os alunos expusessem seus pontos de vista com relação a estes temas de forma livre. As entrevistas foram gravadas, com a permissão dos alunos, para uma análise e reflexão posterior das respostas obtidas.

Tabela 6.5: Roteiro da entrevista

Número	Questão
1	Considera a área de Programação Paralela uma área complexa? E quanto a área de Programação Paralela em GPUs?
2	Conseguiste atingir os resultados esperados ao longo da disciplina? Por quê?
3	Qual(is) ferramenta(s) utilizou para a escrita de códigos na disciplina?
4	Teve de instalar e configurar ferramentas para desenvolvimento de códigos? Quantas?
5	Encontrou dificuldades com a instalação e configuração das ferramentas? Quais?
6	Possui placa gráfica em seu *notebook* ou *desktop* para o teste de códigos em arquiteturas de GPU?
7	Qual seria a metodologia ideal de ensino adotada pelo professor da disciplina em sua opinião?
8	Utilizou algum ambiente de apoio ao ensino de Programação Paralela ao longo da disciplina?
9	Utilizaria um ambiente de apoio, caso houvesse algum, para a área de Programação Paralela? Em quais situações?
10	Já utilizou alguma distribuição Linux? Caso sim, qual?
11	Quais ferramentas e qual sistema operacional gostaria que fizessem parte da arquitetura de um ambiente para a prática de programação paralela?
12	Caso possua placa gráfica em seu computador, utilizaria igualmente o ambiente? Por quê?
13	Utilizaria uma aplicação para execução remota de códigos em um ambiente servidor com placa gráfica compatível com as arquiteturas CUDA e OpenCL?
14	Caso a aplicação fosse disponibilizada a parte do ambiente completo, qual seria sua preferência de utilização, ambiente completo ou apenas a ferramenta para execução remota?

Ao serem questionados sobre o primeiro tópico, relacionado à complexidade da área de programação paralela, seis dos sete alunos concordaram que a área é complexa. Já quando questionados a respeito da área de programação paralela em GPUs, os sete alunos concordaram que esta área é ainda mais complexa do que a área de programação paralela. Cada aluno, de sua maneira, relatou algum tipo de dificuldade que enfrentou ao longo da disciplina, algumas dificuldades relacionados a questões de compreensão da área, outros com a configuração das ferramentas necessárias e ainda, com relação ao desenvolvimento dos primeiros exemplos de códigos paralelos.

Quando questionados com relação a terem ou não conseguido atingir os resultados esperados ao longo da disciplina, todos os alunos acreditam ter atingido os resultados esperados. Alguns acreditam ter conseguido desenvolver as habilidades necessárias que os permitam utili-

zar os conceitos da área em aplicações e trabalhos futuros.

Já o terceiro item da entrevista, que trata sobre as ferramentas utilizadas para a escrita de códigos trouxe diferentes respostas. Todos os alunos já utilizaram as IDEs Netbeans, Eclipse e CodeBlocks. A grande maioria dos alunos utiliza a ferramenta Netbeans, enquanto que outros estão divididos entre a ferramenta Eclipse e a ferramenta CodeBlocks.

As perguntas de número 4 e 5 estão relacionadas, e questionam os alunos sobre o fato de terem ou não que instalar ferramentas adicionais para o desenvolvimento de códigos, além de perguntar se estes tiveram dificuldades ao longo do processo. Quanto à instalação de *software* adicional, todos os alunos tiveram de instalar alguma ferramenta ou *driver* adicional para conseguir desenvolver e/ou testar os códigos. Já em relação às dificuldades acerca da instalação e configuração de *softwares* e *drivers* necessários, todos os alunos tiveram dificuldades. Os alunos consideram o tempo investido neste tipo de atividade interessante, porém, segundo eles, seria melhor poder investir esforços na prática da área, deixando este tipo de atividade como uma forma de sanar sua curiosidade lendo sobre a instalação, ao invés de dedicar tempo e esforço na instalação e configuração, bem como na compreensão dos erros obtidos – que geralmente surgem, segundo eles – ao longo do processo.

A sexta pergunta do questionário tem impacto fundamental na pesquisa e desenvolvimento deste trabalho. Ao serem questionados sobre terem ou não placas gráficas em seus *desktops* ou *notebooks*, apenas um dos sete alunos possui tal recurso. E ainda, esta é compatível com uma versão obsoleta do CUDA. Dessa forma, torna-se inviável o teste dos códigos em arquiteturas de programação paralela de GPUs pelos alunos.

O sétimo tópico está relacionado à abordagem que os alunos consideram ideal a ser utilizada pelo professor na disciplina de programação paralela e de GPUs. Todos os alunos consideram os conceitos iniciais importantes, a introdução das áreas, seu histórico, características e funcionalidades importantes. Alguns alunos acreditam que deveria ser melhor explorado as áreas em que é possível aplicar os conceitos das áreas de programação paralela e de GPUs através de exemplos mais concretos.

As perguntas 8 e 9 abordam os ambientes de apoio ao ensino de programação paralela. No oitavo tópico, os alunos são questionados sobre já terem ou não utilizado algum ambiente de apoio ao ensino de programação paralela e de GPUs. Nenhum dos alunos já utilizou ou tinha conhecimento sobre tais ambientes, até o momento da entrevista em que lhes foram apresentados alguns ambientes, como o StarHPC (IVICA; RILEY; SHUBERT, 2009) e a pesquisa de (DE-

LISTAVROU; MARGARITIS, 2011). Estes explicam que não fizeram uso de tal ferramenta, por não a terem a sua disposição. Já na nona pergunta, os alunos são questionados se utilizariam ambientes de apoio nas disciplinas de programação paralela e de GPUs. Para esta pergunta, as respostas foram satisfatórias e favoráveis para o desenvolvimento deste trabalho. Todos os alunos utilizariam um ambiente de apoio ao ensino de programação paralela e, principalmente, de programação paralela em GPUs. Segundo eles, as situações de uso seriam principalmente ao longo da disciplina, como ferramenta de apoio, onde o professor poderia demonstrar em tempo real alguns códigos, executando e demonstrando erros e acertos em cada situação, ao invés de exemplos estáticos em apresentações de *slides*, que causam dúvidas sobre o funcionamento e compreensão do código. Outra resposta obtida em cenários de utilização, foi como ferramenta para desenvolverem seus códigos em casa ou em outros lugares, por terem um ambiente que pode ser levado a qualquer lugar, e por permitir ser instalado em seus *notebooks*. Por fim, os alunos utilizariam o ambiente mesmo após o término da disciplina, o que foi um fator surpreendente na entrevista. Segundo eles, utilizariam o ambiente até o momento que tivessem necessidade ou tivessem a sua disposição sua própria máquina com placa gráfica compatível com as arquiteturas de programação paralela em GPUs.

A pergunta seguinte, que trata sobre a utilização de alguma distribuição Linux, todos os alunos já utilizaram alguma distribuição. Em resposta, a maioria deles utiliza a distribuição Ubuntu, enquanto outros utilizam Debian e alguns Fedora.

A décima primeira pergunta trata das ferramentas e sistema operacional que os alunos gostariam que estivesse presente em um ambiente, caso este fosse desenvolvido, como ferramenta de apoio ao ensino de programação paralela em GPUs. Dos alunos entrevistados, cinco escolheram a distribuição Ubuntu, enquanto outro gostaria que fosse uma versão da distribuição Debian e outro uma versão de Fedora. Em relação as ferramentas para o desenvolvimento dos códigos, as ferramentas mencionadas foram: Eclipse, Netbeans e CodeBlocks.

Na pergunta a seguir, se o aluno utilizaria um ambiente de apoio caso possuísse placa gráfica em seu *desktop* ou *notebook*, cinco afirmaram que sim, enquanto um utilizaria caso possuísse espaço de sobra em seu disco rígido e apenas um não utilizaria o ambiente em tal situação.

As perguntas de número 13 e 14, tratam sobre uma aplicação que auxilia no processo de testes, sendo possível através dela, realizar testes em um ambiente servidor com placa gráfica compatível com as arquiteturas de programação paralela em GPUs, o CUDA e o OpenCL. A décima terceira pergunta questiona se os alunos utilizariam tal aplicação. Todos os alunos

responderam que sim, utilizariam a ferramenta, e a consideram importante, sendo que através dela, eles poderiam testar seus códigos, o que atualmente não é possível a eles. Por fim, a última pergunta questiona os alunos sobre a utilização do ambiente completo ou de apenas a ferramenta para execução remota de códigos. Para esta pergunta, seis alunos responderam que utilizariam o ambiente, para não ser preciso instalar e configurar nenhum tipo de ferramenta, enquanto que apenas um aluno utilizaria apenas a aplicação. Alguns alunos, mencionaram ainda, o fato de acharem importante terem ambas as ferramentas, o ambiente completo instalado em sua máquina junto com outro sistema operacional, e a aplicação para execução remota em seu outro sistema operacional.

Através das respostas obtidas com a entrevista com os alunos, fica claro que existem diferentes possibilidades em que estes utilizariam ambientes de apoio ao ensino de programação paralela de GPUs. Com base nos resultados obtidos através da entrevista, acredita-se que seria vantajoso utilizar um ambiente de apoio ao ensino de programação paralela em arquiteturas de GPU, como ferramenta a ser adotada pelo professor ao longo da disciplina.

6.2.2 Apresentação da Ferramenta GPUHelp

Após a entrevista sobre a área de programação paralela com os alunos que já cursaram tal disciplina no curso de Ciência da Computação na UFSM, foi apresentado a eles o ambiente GPUHelp. Foram apresentadas e descritas as funcionalidades e ferramentas que o compõem. Os alunos foram convidados a utilizar o ambiente.

O ambiente foi disponibilizado em uma máquina, onde os alunos puderam interagir com a ferramenta, realizando os testes que consideram importantes. Ao longo da apresentação, foram sendo coletadas as opiniões dos alunos, através dos questionários apresentados na Tabela 6.6 e Tabela 6.7 da Seção 6.2.3.

A Seção 6.2.3 apresenta as opiniões dos alunos acerca do GPUHelp. Com base nas opiniões coletadas dos alunos, entende-se que estes consideram o desenvolvimento do ambiente proposto neste trabalho importante, visto que não existem ambientes de apoio para a disciplina de programação paralela em GPUs disponíveis atualmente.

6.2.3 Considerações dos Alunos Acerca do Ambiente

Esta etapa da entrevista tem como objetivo, no primeiro momento, disponibilizar o ambiente para os alunos utilizarem, enquanto que em um segundo momento, disponibilizar apenas a ferramenta para execução remota de códigos. Foram observadas as considerações feitas pelos

alunos através das perguntas apresentadas nas Tabelas 6.6 e 6.7 e os resultados são demonstrados a seguir.

A entrevista com os alunos foi dividida em duas etapas. A primeira, que será descrita a seguir, consiste dos questionamentos apresentados na Tabela 6.6. Em seguida, foram realizadas as perguntas apresentadas na tabela 6.7.

A primeira pergunta aborda a usabilidade do ambiente GPUHelp. Os 7 alunos conseguiram utilizar o ambiente sem dificuldades. Todos os alunos já possuíam conhecimento de alguma distribuição Linux, conforme apresentado na Seção 6.2.1, embora nem todos utilizassem o Linux Ubuntu. Porém, todos conseguiram interagir com o sistema sem nenhuma dificuldade. As ferramentas para desenvolvimento de código também foram testadas. Todos os alunos conseguiram utilizar as ferramentas sem encontrar nenhuma dificuldade. Cada um, optou por utilizar a ferramenta que possuía maior afinidade: Netbeans, Eclipse ou CodeBlocks.

Na segunda pergunta, que trata da forma de utilização do GPUHelp, os alunos apresentaram sua opinião sobre maneiras de utilizar em suas máquinas. Alguns alunos preferem utilizar o GPUHelp de forma nativa, pois acreditam que a performance do sistema ficaria melhor. Acreditam também, que por já serem usuários de *software* livre, eles poderiam passar a utilizar a distribuição Ubuntu sem qualquer problema (para aqueles que utilizam outra distribuição). Um dos alunos prefere utilizar o GPUHelp de forma virtual, pois caso precise de espaço em seu *notebook*, segundo ele, pode copiar a máquina virtual para um disco de armazenamento externo e executar o ambiente através dele. Um dos alunos levantou a possibilidade de virtualizar o ambiente em um *Tablet*, com sistema Android – uma ideia que pode ser adaptada a dispositivos móveis em trabalhos futuros para o GPUHelp.

O terceiro questionamento está relacionado às ferramentas que compõem o ambiente GPUHelp. Todos os alunos concordam com as ferramentas que foram inseridas no ambiente, e alguns ainda acreditam ser interessante a ideia de adicionar as principais ferramentas para desenvolvimento de código ao ambiente, possibilitando que diferentes usuários possam ter suas ferramentas preferidas ao seu alcance.

A pergunta seguinte, de número 4, levanta um questionamento importante sobre o ambiente e sua utilização ao longo da disciplina de programação paralela e de arquiteturas de GPU. Todos os alunos acreditam que, se tivessem utilizado o ambiente ao longo da disciplina, poderiam ter tido um melhor rendimento de forma geral, além de poderem utilizar o tempo aplicado na instalação e configuração de ferramentas, em tarefas que fossem mais pertinentes ao aprendizado da

disciplina. Outras opiniões como, possibilidade de realizar testes mais concretos além de uma melhor maneira de desenvolver códigos, visto que o ambiente pode ser levado juntamente com o *notebook*, compõem as respostas dos alunos.

Na quinta pergunta, sobre se o aluno utilizaria o ambiente após a disciplina de programação paralela e de GPUs, todos os alunos responderam que sim, utilizariam o ambiente caso este fosse disponibilizado. Segundo eles, através da utilização do ambiente, é desnecessária a instalação e configuração de *software*, pois o ambiente já está pronto e configurado, permitindo que eles invistam seu tempo desenvolvendo *software* ao invés de instalar e configurar ferramentas.

As perguntas de número 6 e 7 concluem a primeira parte, que trata do ambiente GPUHelp e sua usabilidade. Na sexta pergunta, os alunos são questionados sobre ferramentas adicionais a serem incluídas no ambiente. Os 7 alunos responderam de forma similar a esta pergunta. Segundo eles, o ambiente já possui os requisitos necessários e essenciais para o propósito pela qual foi desenvolvido. Os mesmos acreditam que, caso seja necessário, eles podem adicionar as ferramentas que precisem, visto que o ambiente permite esta funcionalidade sem afetar seu funcionamento. A sétima pergunta, que trata de melhorias acerca do ambiente, também trouxe respostas similares dos alunos. Segundo eles, o ambiente cumpre o propósito que foi desenvolvido: ser uma arquitetura que possibilite o desenvolvimento e teste de códigos para arquiteturas de programação paralela em GPU. Dessa forma, neste momento os alunos acreditam que não existam considerações acerca de funcionalidades que devem ser adicionadas.

Tabela 6.6: Considerações acerca do ambiente GPUHelp

Número	**Questão**
1	Conseguiu utilizar o ambiente completo?
2	De que forma utilizaria o ambiente? De forma nativa ou virtual em sua máquina?
3	Qual sua opinião acerca das ferramentas que compõem o ambiente?
4	Acredita que com a disponibilização do ambiente aos alunos da disciplina, esta iria tornar-se mais simples?
5	Utilizaria o ambiente após a disciplina, para o teste de códigos de sua autoria?
6	Quais ferramentas devem ser incluídas ou removidas no ambiente?
7	Quais melhorias devem ser aplicadas no ambiente?

A segunda etapa da entrevista relacionada ao GPUHelp trata em específico da aplicação para execução remota de códigos do ambiente. Os questionamentos realizados aos alunos podem ser visualizados na Tabela 6.7.

A primeira pergunta trata da usabilidade da aplicação. Os alunos foram convidados a utilizar a ferramenta que compõe o GPUHelp. Todos os alunos conseguiram utilizar a aplicação sozinhos, sem precisarem de apresentação ou ajuda sobre suas funcionalidades.

O questionamento seguinte, tratava dos testes da ferramenta. Os alunos utilizaram a ferramenta em seus dois modos: usuário iniciante e usuário experiente. Em ambos os modos, os alunos conseguiram realizar testes e executar códigos no ambiente servidor.

Na terceira pergunta, sobre a utilização da ferramenta, todos os alunos concluíram que a ferramenta é simples e intuitiva. Segundo eles, suas funcionalidades são importantes e essenciais à área, porém, sem abrir mão de uma simplicidade que agrada os utilizadores. Outra resposta obtida envolve o projeto da aplicação. Segundo um dos alunos, o fato de a aplicação possuir dois modos de utilização (iniciante e experiente), instiga o utilizador a tornar-se um usuário experiente, caso seja um usuário iniciante.

Na pergunta seguinte, todos os alunos consideram a aplicação simples e sua usabilidade boa. Não houveram críticas acerca da interface e do projeto desta. Os alunos consideram interessante a maneira como a aplicação foi projetada, em 3 interfaces: usuário decide qual seu nível; usuário iniciante e usuário experiente. Segundo eles, dessa forma, a aplicação tem sua interface limpa, sem precisar de diversas funcionalidades desnecessárias em uma única tela, visto que ela possui diferentes níveis de utilização.

Para a quinta pergunta, todos os alunos responderam que utilizariam a ferramenta para execução remota de códigos. Um dos alunos respondeu que a considera interessante, uma vez que ela traz exemplos que demonstram o funcionamento das arquiteturas CUDA e OpenCL, possibilitando que o utilizador compreenda ambas. Outras respostas obtidas envolvem a necessidade e importância da ferramenta, uma vez que a maioria dos alunos entrevistados não possui placa gráfica, não sendo possível o teste dos códigos, enquanto que outro aluno respondeu que a aplicação possibilita que o utilizador possa comparar o código desenvolvido na máquina local com o desempenho obtido no ambiente servidor.

A última pergunta, de número 6, questiona os alunos sobre quais melhorias deveriam ser aplicadas a ferramenta. Os alunos, em sua maioria não fizeram considerações, pois acreditam que a ferramenta cumpre o objetivo principal. Segundo eles, a praticidade da aplicação é boa,

aliando funcionalidade e simplicidade. Um dos alunos gostaria que a aplicação fosse portável, para qualquer sistema operacional – requisito este que já está disponível na aplicação. Outro aluno gostaria que a ferramenta pudesse ser executada em dispositivos móveis, *tablets* e *smartphones*.

Tabela 6.7: Considerações acerca da ferramenta para execução remota de códigos

Número	**Questão**
1	Conseguiu utilizar a ferramenta sem auxílio?
2	Conseguiste realizar testes na ferramenta?
3	Em sua opinião, a aplicação é de fácil utilização?
4	Como considera a interface e a usabilidade da aplicação?
5	Utilizaria o ambiente após a disciplina, para o teste de códigos de sua autoria?
6	Utilizaria a ferramenta para o teste de códigos em arquiteturas de GPU? Por quê?
7	Quais melhorias devem ser aplicadas na aplicação?

6.3 Considerações

O ambiente GPUHelp trouxe resultados positivos quando apresentado aos alunos. Segundo eles, com a utilização de uma ferramenta como esta, eles podem ter acesso a um recurso que antes não possuíam, além de outras considerações positivas, como por exemplo, a possibilidade de dedicarem mais tempo a tarefas mais pertinentes do que instalar e configurar *software*, bem como o fato de terem todas as ferramentas instaladas e configuradas a seu alcance.

Quanto a aplicação para execução remota de códigos, acredita se ter atingido o resultado esperado com o desenvolvimento da ferramenta, uma vez que esta teve como características principais a funcionalidade, usabilidade e simplicidade. Desse modo, o resultado esperado no início deste trabalho foi alcançado uma vez que os alunos conseguiram utilizá-la sem precisarem de ajuda, demonstrando assim, sua simplicidade de utilização.

7 CONCLUSÃO

7.1 Conclusões

Existem diferentes ferramentas de apoio ao ensino de programação paralela atualmente (PARK et al., 2000) (IVICA; RILEY; SHUBERT, 2009) (DELISTAVROU; MARGARITIS, 2011). Tais ambientes possuem sua contribuição no ensino comprovada através de diferentes pesquisas (FREITAS, 2012) (GIACAMAN, 2012) (IPARRAGUIRRE; FRIEDRICH; COPPO, 2012) (LIU; WU; MARSAGLIA, 2012) (DELISTAVROU; MARGARITIS, 2011) (IVICA; RILEY; SHUBERT, 2009). Tal característica está relacionada, principalmente, as dificuldades acerca do ensino da área de programação paralela. Com relação ao ensino da área de programação paralela em GPU, esta complexidade torna-se ainda maior.

Este trabalho apresentou o ambiente desenvolvido para atuar como uma ferramenta de apoio ao ensino da área de programação paralela em GPU, o GPUHelp. O GPUHelp suporta o desenvolvimento e teste de códigos nas principais arquiteturas de GPU. A ferramenta é uma arquitetura composta de um lado cliente, que conta com ferramentas para o desenvolvimento de códigos, *drivers* para programação paralela em GPU para as arquiteturas CUDA e OpenCL, sistema operacional Linux e uma aplicação para execução remota de códigos em um ambiente servidor com placa gráfica com suporte aos principais *frameworks* de programação paralela em GPU, o CUDA e o OpenCL. Através do desenvolvimento do GPUHelp, torna-se possível aos usuários que não possuem placas gráficas em seus *desktops* ou *notebooks* desenvolverem e testarem códigos escritos para as arquiteturas de GPU. A ferramenta para execução remota de códigos está disponível *online*, no Google Code (https://code.google.com/p/gpuhelp/). Através disso, os utilizadores podem efetuar o *download* do código fonte completo da aplicação e implementá-la em suas Universidades.

O estado da arte não apresenta ambientes de apoio ao ensino para a área de programação paralela em arquiteturas de GPUs. Tal fato tem impacto negativo de diferentes maneiras, sendo que a ausência de tais ambientes contribui para uma menor eficiência no ensino da área, consequentemente criando um menor interesse por parte dos alunos por esta. Os ambientes de apoio ao ensino apresentados na Seção 3.2 são direcionados apenas para a área de programação paralela em arquiteturas de CPU.

As dificuldades encontradas ao longo da construção deste trabalho consistem no complexo desenvolvimento da aplicação para execução remota de códigos, bem como na configuração

envolvendo o ambiente servidor. Para que os códigos desenvolvidos pelos usuários, para as arquiteturas CUDA e OpenCL, pudessem ser corretamente compilados e executados no ambiente servidor, optou-se por deixar a cargo do utilizador selecionar e enviar todas as bibliotecas que são referenciadas em seu código. Caso contrário, a configuração do ambiente servidor seria ainda mais complexa, visto que este precisaria contar com praticamente todas as *libraries* disponíveis atualmente na área de programação paralela em arquiteturas de GPU, caso contrário, os códigos seriam compilados com erros no momento em que fizessem determinada referência a tais bibliotecas. Quanto a aplicação para execução remota de códigos, esta teve de ser projetada com base em uma interface simples e intuitiva, que permitisse a utilização por diversos tipos de usuários. O processo de envio de arquivos de forma segura ao servidor encontrou algumas dificuldades iniciais ao ser testado na rede da UFSM, por esta bloquear a porta 21, padrão do protocolo FTP. Dessa forma, adotou-se uma nova metodologia de envio dos arquivos, convertendo-os em *strings* para serem enviados pela rede, de forma transparente, aumentando assim, a eficiência da aplicação. Outra dificuldade está relacionada à avaliação do ambiente. Devido ao fato de não existirem ambientes de apoio ao ensino de programação paralela em arquiteturas de GPU, teve-se de realizar uma avaliação com ambientes de apoio voltados para as arquiteturas de CPU.

Os testes e avaliações realizados demonstram que a proposta desenvolvida neste trabalho é uma alternativa viável para atuar como ferramenta de apoio ao ensino na área de programação paralela em arquiteturas de GPU. Os resultados apresentam a aceitação positiva dos alunos entrevistados, que foram convidados a testar o GPUHelp. O ambiente desenvolvido neste trabalho pode ser aplicado em diferentes cenários: como um ambiente de apoio ao ensino de programação paralela em arquiteturas de GPU; ferramenta para desenvolvimento de *software*; ambiente de treinamento; utilização em laboratórios da UFSM; ambiente para oferta de cursos e mini cursos na área, entre outros. Essa vasta gama de possíveis aplicações demonstra a flexibilidade e usabilidade proporcionada pelo GPUHelp.

Com a utilização do GPUHelp como ferramenta de apoio na disciplina de programação paralela em arquiteturas de GPU, os professores podem focar esforços no ensino da área, além de melhores técnicas para apresentação dos exemplos aos alunos, pois o ambiente desenvolvido possui todas as configurações necessárias para comportar o desenvolvimento e teste de códigos para as arquiteturas de GPU, o CUDA e o OpenCL.

7.2 Trabalhos Futuros

A concepção da solução e os resultados obtidos abriram diversas possibilidades de trabalhos futuros.

Uma possibilidade a ser implantada em um trabalho futuro, seria a migração do ambiente cliente para a nuvem. Ao hospedar o GPUHelp em uma arquitetura de nuvem, seria possível ao utilizador ter acesso a ferramenta de qualquer lugar, a qualquer momento, com seus arquivos sincronizados e disponíveis, desde que este disponha de uma conexão à *Internet*.

Outras possibilidades podem vir a ser implementadas. Uma delas seria a de gerar as imagens de máquinas de forma automatizada, com informações dos usuários que utilizam a ferramenta através de um formulário que sincronize as respostas com a aplicação para gerar as VMs. É possível ainda, pensar-se na criação de um Cluster de GPUs. Através da criação de tal Cluster, acredita-se que a carga de desempenho suportada pela aplicação tenha um aumento considerável. Pode ser implementada ainda, uma maneira automatizada para o gerenciamento das contas de usuários da aplicação.

Por fim, outra possibilidade a ser implementada é a criação de um servidor de arquivos que disponibilize um espaço para os usuários manterem seus códigos salvos e acessíveis de outros lugares.

REFERÊNCIAS

AMAZON. **Amazon**. Disponível em: http://www.amazon.com. Acesso em: Setembro de 2013.

AMAZON. **Amazon Elastic Compute Cloud (Amazon EC2)**. Disponível em: http://aws.amazon.com/pt/ec2/. Acesso em: Setembro de 2013.

AMD. **AMD Graphics**. Disponível em: http://www.amd.com/us/products/desktop/graphics/pages/radeon.aspx. Acesso em: Julho de 2013.

AMD. **The open standard for parallel programming of heterogeneous systems**. Disponível em: http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/. Acesso em: Agosto de 2013.

BARNEY, B.; LIVERMORE, L. **Introduction to Parallel Computing**. Disponível em: https://computing.llnl.gov/tutorials/parallel_comp/. Acesso em: Junho de 2013.

BERILLO, A. **NVIDIA CUDA**. Disponível em: http://ixbtlabs.com/articles3/video/cuda-1-p1.html. Acesso em: Julho de 2013.

BOYER, M.; MENG, J.; KUMARAN, K. Improving GPU Performance Prediction with Data Transfer Modeling. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS PHD FORUM (IPDPSW), 2013 IEEE 27TH INTERNATIONAL. **Anais...** [S.l.: s.n.], 2013. p.1097–1106.

BUYYA, R. et al. High Performance Cluster Computing: architectures and systems (volume 1). **Prentice Hall, Upper SaddleRiver, NJ, USA**, [S.l.], v.1, p.999, 1999.

CANONICAL. **Ubuntu**. Disponível em: http://www.ubuntu.com. Acesso em: Julho de 2013.

CANONICAL. **O que é o Ubuntu?** Disponível em: http://www.ubuntu-br.org. Acesso em: Julho de 2013.

CHARMM. **CHARMM (Chemistry at HARvard Macromolecular Mechanics)**. Disponível em: http://www.charmm.org. Acesso em: Julho de 2013.

CODEBLOCKS. **Code**::blocks. Disponível em: http://www.codeblocks.org. Acesso em: Outubro de 2013.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas distribuídos**: conceitos e projeto. [S.l.]: Grupo A, 2007.

CROIX, J.; KHATRI, S. Introduction to GPU programming for EDA. In: COMPUTER-AIDED DESIGN - DIGEST OF TECHNICAL PAPERS, 2009. ICCAD 2009. IEEE/ACM INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. p.276–280.

CUDA, N. **CUDA C Programming Guide**. [S.l.: s.n.], 2013. Disponível em: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Acesso em: Outubro de 2013.

CUDA, N. **What is CUDA**. Disponível em: https://developer.nvidia.com/what-cuda. Acesso em: Abril de 2013.

CUDA, N. **WHAT IS GPU ACCELERATED COMPUTING?** Disponível em: http://www.nvidia.com/object/what-is-gpu-computing.html. Acesso em: Julho de 2013.

CUDA, N. **CUDA ZONE**. Disponível em: http://www.nvidia.com.br/object/cuda_home_new_br.html. Acesso em: Julho de 2013.

CUDA, N. **CUDA Tools Ecosystem**. Disponível em: https://developer.nvidia.com/cuda-tools-ecosystem. Acesso em: Julho de 2013.

DELISTAVROU, C.; MARGARITIS, K. Survey of Software Environments for Parallel Distributed Processing: parallel programming education on real life target systems using production oriented software tools. In: INFORMATICS (PCI), 2010 14TH PANHELLENIC CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.231–236.

DELISTAVROU, C.; MARGARITIS, K. Towards an Integrated Teaching Environment for Parallel Programming. In: INFORMATICS (PCI), 2011 15TH PANHELLENIC CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.3–7.

ECLIPSE. **Eclipse.org**. Disponível em: http://www.eclipse.org. Acesso em: Outubro de 2013.

ECLIPSE. **Eclipse Juno**. Disponível em: http://www.eclipse.org/juno/. Acesso em: Novembro de 2013.

FLYNN, M. Some Computer Organizations and Their Effectiveness. **Computers, IEEE Transactions on**, [S.l.], v.C-21, n.9, p.948–960, 1972.

FLYNN, M. J.; RUDD, K. W. Parallel architectures. **ACM Computing Surveys (CSUR)**, [S.l.], v.28, n.1, p.67–70, 1996.

FOSTER, I. **Designing and building parallel programs**. [S.l.]: Addison-Wesley Reading, 1995. v.95.

FREITAS, H. de. Introducing parallel programming to traditional undergraduate courses. In: FRONTIERS IN EDUCATION CONFERENCE (FIE), 2012. **Anais...** [S.l.: s.n.], 2012. p.1–6.

GEBALI, F. **Algorithms and Parallel Computing**. [S.l.]: Wiley. com, 2011. v.84.

GIACAMAN, N. Teaching by Example: using analogies and live coding demonstrations to teach parallel computing concepts to undergraduate students. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS PHD FORUM (IPDPSW), 2012 IEEE 26TH INTERNATIONAL. **Anais...** [S.l.: s.n.], 2012. p.1295–1298.

GPGPU. **GPGPU (General-Purpose Computation on Graphics Hardware)**. Disponível em: http://gpgpu.org. Acesso em: Julho de 2013.

HWU, W.-M. W.; KIRK, D. B. **Programando para Processadores Paralelos**: uma abordagem prática a programação de gpus. Rio de Janeiro, RJ, BR: Elsevier, 2011.

ILLINOIS, U. of. **What is VMD?** Disponível em: http://www.ks.uiuc.edu/Research/vmd/allversions/what_is_vmd.html. Acesso em: Julho de 2013.

ILLINOIS, U. of. **NAMD - Scalable Molecular Dynamics**. Disponível em: http://www.ks.uiuc.edu/Research/namd/. Acesso em: Julho de 2013.

INTEL. **Intel Graphics**. Disponível em: http://www.intel.com.br/content/www/br/pt/architecture-and-technology/hd-graphics/hd-graphics-developer.html. Acesso em: Julho de 2013.

IPARRAGUIRRE, J.; FRIEDRICH, G.; COPPO, R. Lessons Learned after the Introduction of Parallel and Distributed Computing Concepts into ECE Undergraduate Curricula at UTN-Bah x00ED;a Blanca Argentina. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS PHD FORUM (IPDPSW), 2012 IEEE 26TH INTERNATIONAL. **Anais...** [S.l.: s.n.], 2012. p.1317–1320.

IVICA, C.; RILEY, J.; SHUBERT, C. StarHPC x2014; Teaching parallel programming within elastic compute cloud. In: INFORMATION TECHNOLOGY INTERFACES, 2009. ITI '09. PROCEEDINGS OF THE ITI 2009 31ST INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. p.353–356.

JPEDAL. **Jpedal *(Java PDF Extraction and Decoding Library)***. Disponível em: `http://freecode.com/projects/jpedal`. Acesso em: Novembro de 2013.

JTATTOO. **What is JTattoo?** Disponível em: `http://www.jtattoo.net`. Acesso em: Novembro de 2013.

KABANOV, J. **Java EE Productivity Report 2011**. Disponível em: `http://zeroturnaround.com/rebellabs/java-ee-productivity-report-2011/#ides`. Acesso em: Outubro de 2013.

KHRONOS. **The Khronos Group**. Disponível em: `https://www.khronos.org`. Acesso em: Julho de 2013.

KHRONOS. **Accelerated Parallel Processing (APP) SDK**. Disponível em: `https://www.khronos.org/opencl/`. Acesso em: Julho de 2013.

LIU, J.; WU, Y.; MARSAGLIA, J. Making Learning Parallel Processing Interesting. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS PHD FORUM (IPDPSW), 2012 IEEE 26TH INTERNATIONAL. **Anais...** [S.l.: s.n.], 2012. p.1307–1310.

MA, K. et al. GreenGPU: a holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In: PARALLEL PROCESSING (ICPP), 2012 41ST INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.48–57.

MAROWKA, A. Think Parallel: teaching parallel programming today. **Distributed Systems Online, IEEE**, [S.l.], v.9, n.8, p.1–1, 2008.

MATTSON, T.; WRINN, M. Parallel programming: can we please get it right this time? In: DESIGN AUTOMATION CONFERENCE, 2008. DAC 2008. 45TH ACM/IEEE. **Anais...** [S.l.: s.n.], 2008. p.7–11.

MIT. **MIT - Massachusetts Institute of Technology**. Disponível em: `http://web.mit.edu`. Acesso em: Setembro de 2013.

MOLECULE, A. A. **Abalone**. Disponível em: `http://www.biomolecular-modeling.com/Abalone/`. Acesso em: Julho de 2013.

MORON, C.; RIBEIRO, J.; SILVA, N. da. A teaching environment for the development of parallel real-time programs. In: FRONTIERS IN EDUCATION CONFERENCE, 1998. FIE '98. 28TH ANNUAL. **Anais...** [S.l.: s.n.], 1998. v.2, p.903–908 vol.2.

NVIDIA. **NVIDIA**. Disponível em: `http://www.nvidia.com/`. Acesso em: Julho de 2013.

NVIDIA. **NVIDIA CUDA**. Disponível em: `https://developer.nvidia.com/cuda-toolkit`. Acesso em: Julho de 2013.

OLIVEIRA, R. S. d.; CARISSIMI, A. d. S.; TOSCANI, S. S. Sistemas operacionais. **Revista de informática teórica e aplicada. Porto Alegre. Vol. 8, n. 3 (dez. 2001), p. 7-39**, [S.l.], 2001.

OPENEYE. **FastROCS**. Disponível em: `http://www.eyesopen.com/fastrocs`. Acesso em: Julho de 2013.

OPENEYE. **ROCS**. Disponível em: `http://www.eyesopen.com/rocs`. Acesso em: Julho de 2013.

ORACLE. **Oracle**. Disponível em: `http://www.oracle.com/index.html`. Acesso em: Setembro de 2013.

ORACLE. **NetBeans IDE**. Disponível em: `https://netbeans.org`. Acesso em: Outubro de 2013.

PACHECO, P. **An introduction to parallel programming**. [S.l.]: Access Online via Elsevier, 2011.

PARK, I. et al. Towards an Integrated, Web-executable Parallel Programming Tool Environment. In: SUPERCOMPUTING, ACM/IEEE 2000 CONFERENCE. **Anais...** [S.l.: s.n.], 2000. p.9–9.

PLIMPTON, S.; THOMPSON, A.; CROZIER, P. **LAMMPS Molecular Dynamics Simulator**. Disponível em: `http://lammps.sandia.gov`. Acesso em: Julho de 2013.

SANTOS, A. D. dos. **VirtCUDA**: possibilitando a execução de aplicações cuda em máquinas virtuais. Disponível em: `http://hdl.handle.net/10183/27970`. Acesso em: Janeiro de 2013.

SEBESTA, R. W. **Concepts of programming languages**. [S.l.]: Addison Wesley, 2005. v.4.

SILVEIRA, C. L. B.; SILVEIRA JR., L. G. da; CAVALHEIRO, G. G. H. **Programacao Paralela em Ambientes Computacionais Heterogeneos com OpenCL**. Disponível em: `http://labs.v3d.com.br/wp-content/uploads/2010/11/wscad2010-opencl.pdf`. Acesso em: Abril de 2013.

SILVEIRA, C. L. B.; SILVEIRA JR., L. G. da; CAVALHEIRO, G. G. H. **Programacao em OpenCL**: uma introducao pratica. Disponível em: `http://labs.v3d.com.br/wp-content/uploads/2010/11/sbgames2010-opencl.pdf`. Acesso em: Abril de 2013.

STALLINGS, W. **Computer Organization and Architecture**: designing for performance. [S.l.]: Prentice Hall, 2005. v.7.

STALLINGS, W. **Arquitetura e Organizacao de Computadores**. [S.l.]: Prentice Hall, 2010. v.8.

TANENBAUM, A. S. **Structured Computer Organization**. [S.l.]: Prentice Hall, 2010. v.5.

VALIEV, M. et al. NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations. **Computer Physics Communications**, [S.l.], v.181, n.9, p.1477 – 1489, 2010.

VMWARE. **VMWare Virtualization**. Disponível em: `http://www.vmware.com`. Acesso em: Julho de 2013.

WHITE, O. **Developer Productivity Report 2012**: java tools, tech, devs data. Disponível em: http://zeroturnaround.com/rebellabs/developer-productivity-report-2012-java-tools-tech-devs-and-data/. Acesso em: Outubro de 2013.

YONGCHAO, L. **CUSHAW - fast short read alignment for CUDA-enabled GPUs**. Disponível em: http://cushaw.sourceforge.net/homepage.htm#latest. Acesso em: Julho de 2013.

APÊNDICE A MANUAL DE UTILIZAÇÃO DA APLICAÇÃO PARA EXECUÇÃO REMOTA DE CÓDIGOS

Introdução

O GPUHelp é um ambiente de apoio ao ensino de programação paralela em GPU com suporte as arquiteturas CUDA e OpenCL. Tal ambiente foi desenvolvido com o objetivo de atuar como uma ferramenta a ser utilizada por professores e alunos da Universidade Federal de Santa Maria – UFSM. O ambiente pode ser utilizado em diferentes situações: ferramenta de apoio em sala de aula, desenvolvimento de códigos pelos alunos, ofertas de cursos e mini cursos, treinamento em geral, entre outros.

Este manual descreve, de forma sucinta, como utilizar a ferramenta para execução remota de códigos. A aplicação é dividida em três interfaces: interface inicial, usuário iniciante e usuário experiente. A seguir, serão descritas cada uma das interfaces e seu processo de utilização.

Interface Inicial

A interface inicial apresenta ao usuário os dois níveis disponíveis na ferramenta. A figura A.1 apresenta a interface inicial da aplicação.

Figura A.1: Interface inicial da aplicação para execução remota de códigos

Conforme pode ser visualizado na figura A.1, o usuário pode optar entre dois modos de utilização: usuário iniciante e usuário experiente. Neste trabalho, entende-se como usuário iniciante aquele que não possui experiência com a área de Programação Paralela em GPUs. Já para

o usuário experiente, entende-se que este possua algum conhecimento na área de Programação Paralela em arquiteturas de GPU, seja em CUDA ou OpenCL.

As figuras A.2 e A.3 apresentam uma descrição, disponível na opção *"Veja Aqui"* na interface inicial, para usuários que não sabem qual nível escolher: usuário iniciante ou usuário experiente.

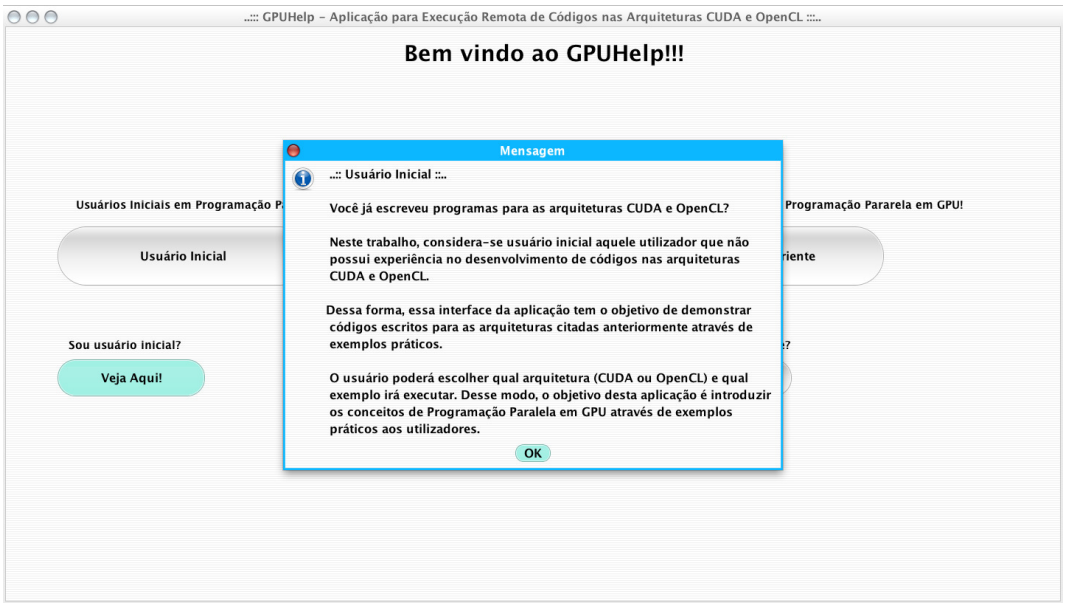


Figura A.2: Conteúdo da opção *"veja aqui"* para usuários iniciantes

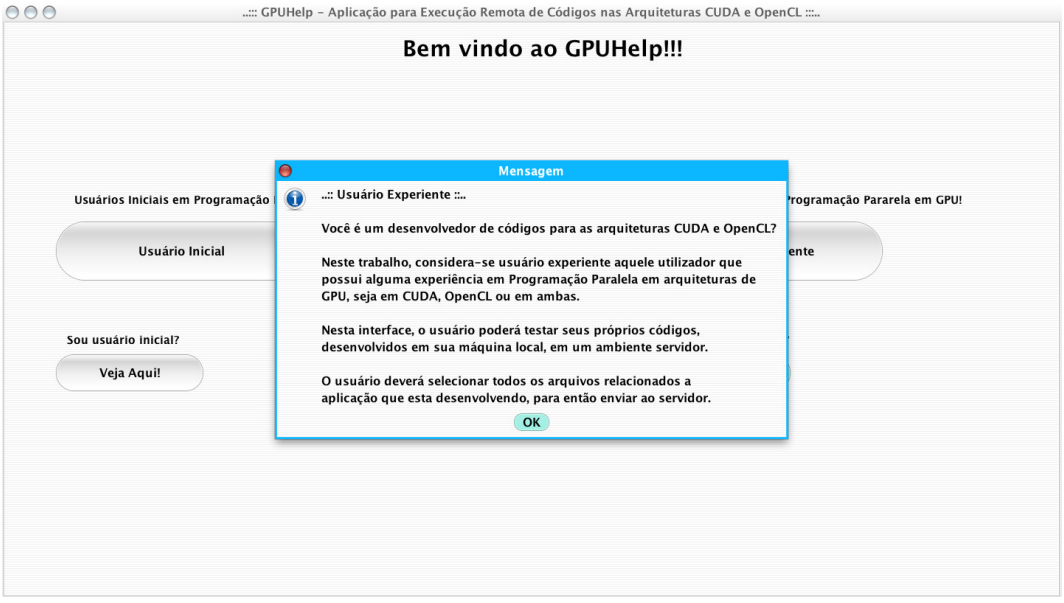


Figura A.3: Conteúdo da opção *"veja aqui"* para usuários experientes

A seguir, serão descritas as interfaces para usuário iniciante e usuário experiente.

Interface Usuário Iniciante

Nesta interface, o usuário poderá testar os exemplos disponíveis para as arquiteturas CUDA e OpenCL. Os códigos são originalmente pertencentes aos repositórios da NVIDIA e do grupo Khronos. A figura A.4 apresenta a interface do usuário iniciante.

Figura A.4: Interface usuário inicial

A figura A.5 apresenta o editor de códigos disponíveis na aplicação. O usuário precisa apenas selecionar qual código deseja editar e clicar na opção *"Editar Código"*. Logo após, será aberta a aplicação para edição dos códigos, conforme ilustrado pela figura A.6. Através desta opção, é possível ao usuário alterar os valores das variáveis dos algoritmos, realizar alterações em sua estrutura principal e testar novamente, verificando os novos resultados obtidos.

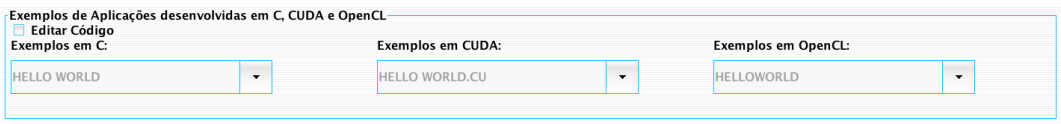


Figura A.5: Opção para edição de códigos

Para executar um código, o usuário precisa apenas selecionar um dos exemplos e clicar na opção *"Executar"*, conforme ilustrado pela figura A.7. Ao realizar este procedimento, o código será executado no ambiente servidor, uma máquina com placa gráfica compatível com as arquiteturas CUDA e OpenCL. O resultado da execução do código retornará na tela do usuário, conforme apresentado na figura A.8.

Interface Usuário Experiente

Através desta aplicação, torna-se possível ao usuário escrever os códigos em sua própria

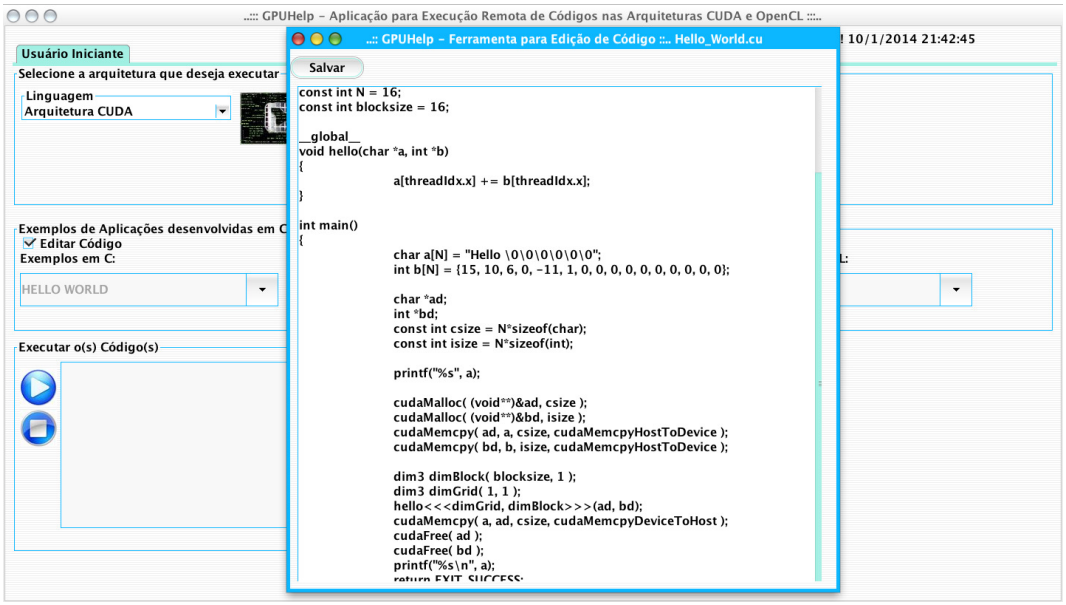


Figura A.6: Ferramenta para edição de códigos disponível na aplicação usuário inicial

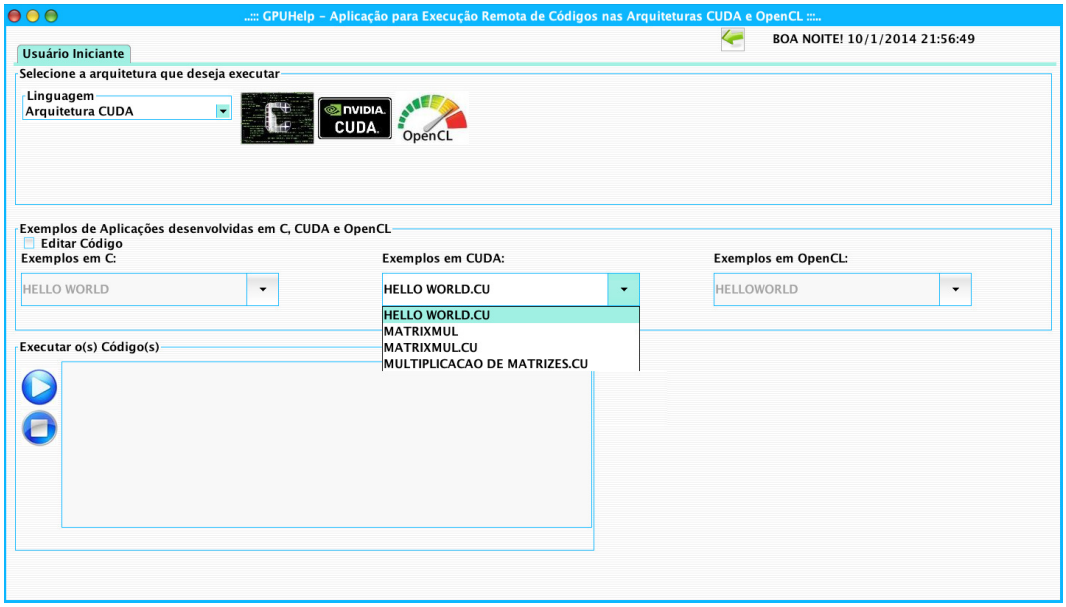


Figura A.7: Usuário selecionando código para executar no ambiente servidor

máquina e testar no ambiente servidor. A figura A.9 apresenta a interface para usuários experientes.

O usuário deve selecionar a pasta com todos os arquivos relacionados ao seu projeto para enviar ao ambiente servidor. A figura A.10 ilustra o processo de seleção dos arquivos a serem enviados ao ambiente servidor.

O passo seguinte consiste em informar uma *flag* para compilar o(s) código(s) selecionado(s). A aplicação para execução remota de códigos não possui um analisador de *flags*, dessa forma, o

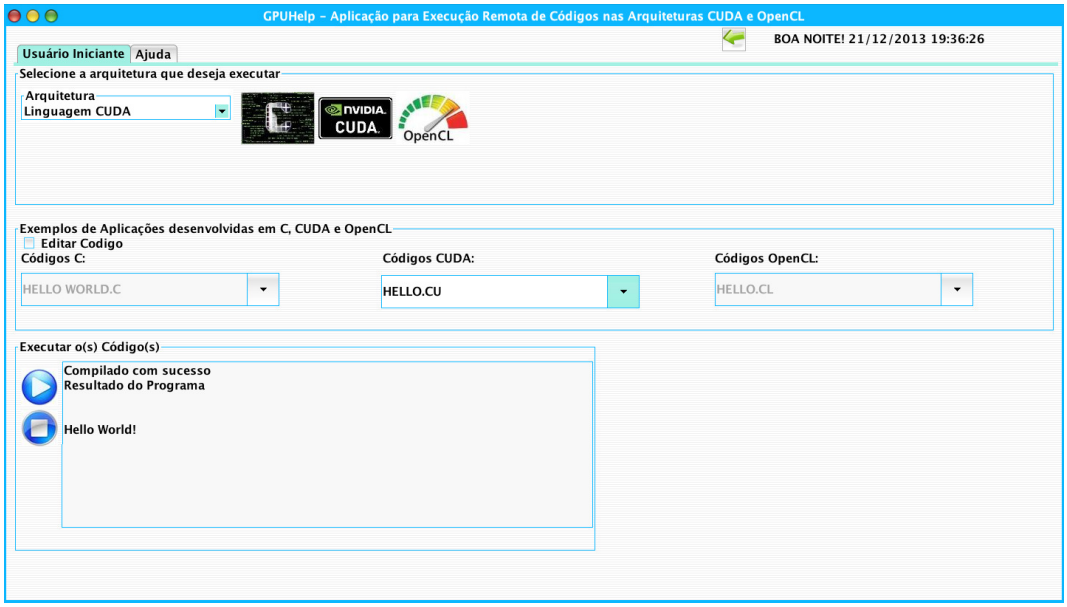


Figura A.8: Resultado da execução de um código no ambiente servidor na tela do usuário

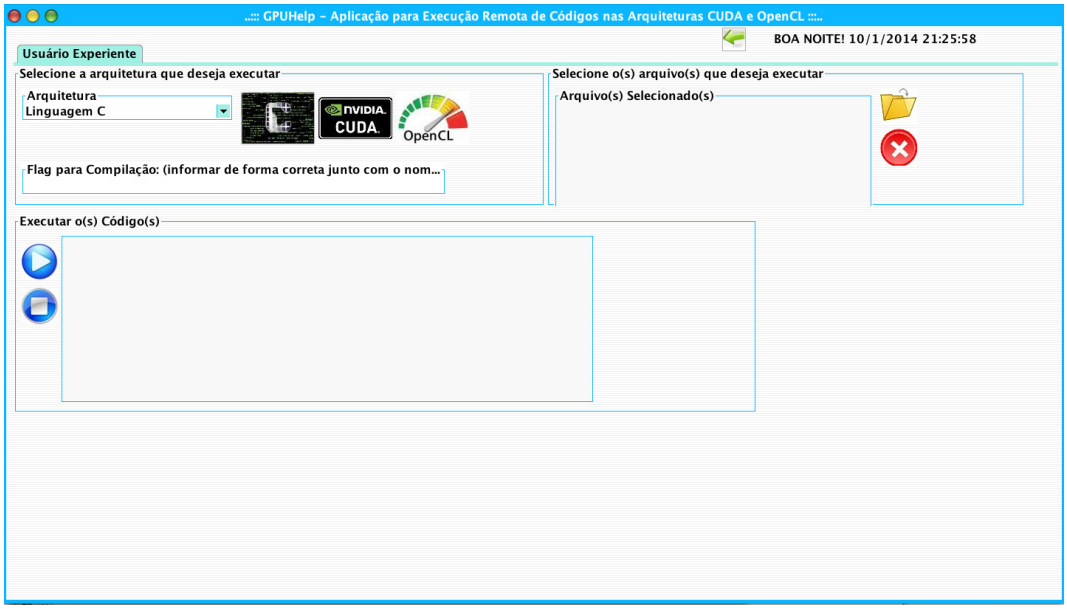


Figura A.9: Interface usuário experiente

usuário deve informar de forma correta os parâmetros para compilar seu código. As figuras A.11 e A.12 apresentam um exemplo de passagem de *flag* para CUDA e OpenCL, respectivamente. Cabe salientar, que o ambiente servidor é composto por um sistema operacional Linux Ubuntu, dessa forma, as *flags* para compilação devem ser compatíveis com tal sistema.

Logo após o usuário ter certeza de que selecionou todos os arquivos referentes a seu projeto, informou a *flag* para compilação correta, juntamente com o nome dos arquivos, este precisa apenas clicar na opção *"Executar"*, conforme ilustrado pela figura A.13. Após este passo, a

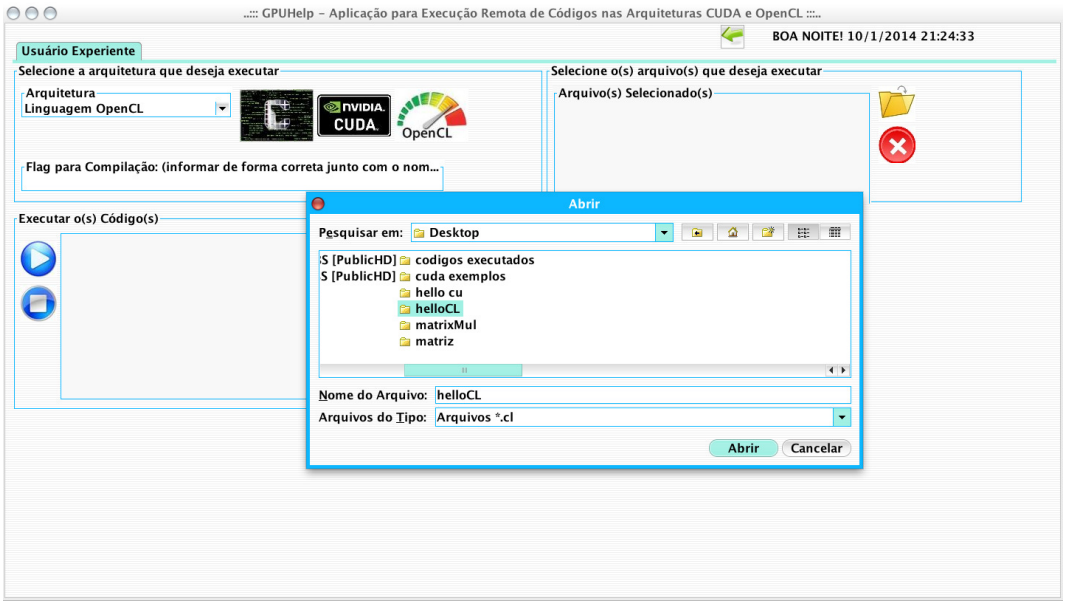


Figura A.10: Seleção de arquivos pelo usuário a serem executados no ambiente servidor

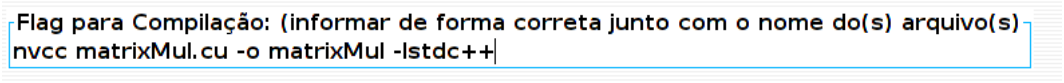


Figura A.11: Exemplo de passagem de *flag* pelo usuário para arquitetura CUDA

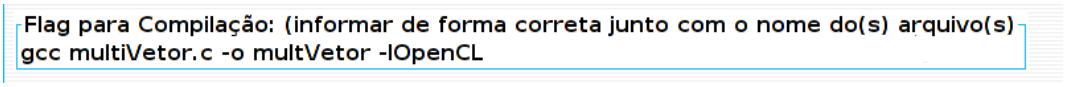


Figura A.12: Exemplo de passagem de *flag* pelo usuário para arquitetura OpenCL

aplicação irá enviar os arquivos ao servidor, compilar, executar e retornar o resultado na tela do usuário conforme ilustrado na figura A.14.

Considerações Finais

Este manual descreveu de forma sucinta a utilização da aplicação para execução remota de códigos.

Caso o utilizador tenha encontrado algum problema, possua alguma dúvida ou sugestão, este pode enviar um e-mail para dborges@inf.ufsm.br solicitando maiores esclarecimentos.

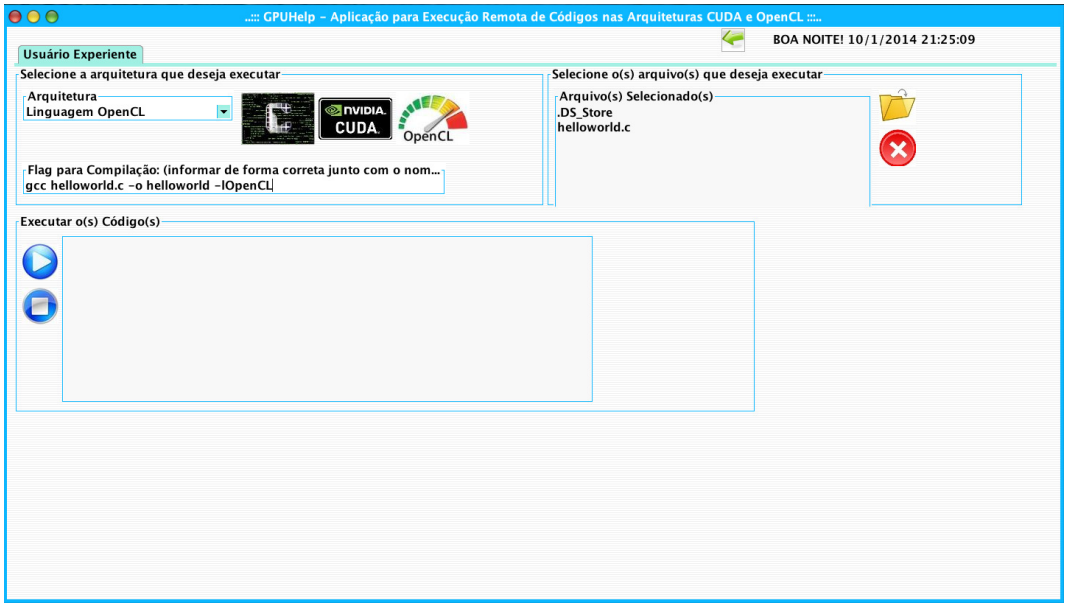


Figura A.13: Usuário executando os arquivos selecionados

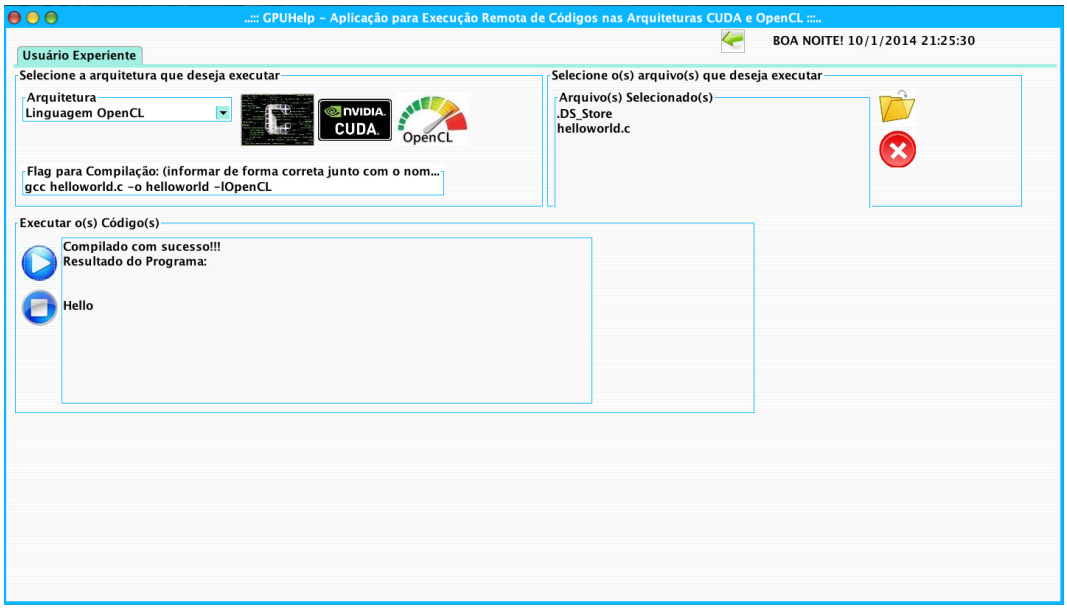


Figura A.14: Resultado da execução