UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES - CAMPUS DE ERECHIM
DEPARTAMENTO DE ENGENHARIAS E CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MARCOS VINICIUS DE MOURA LIMA

PESQUISA E IMPLEMENTAÇÃO DE UMA APLICAÇÃO PARALELA UTILIZANDO O PARADIGMA DE GRAFOS DE DEPENDÊNCIAS DE TAREFAS

ERECHIM - RS
2018

MARCOS VINICIUS DE MOURA LIMA

PESQUISA E IMPLEMENTAÇÃO DE UMA APLICAÇÃO PARALELA UTILIZANDO O PARADIGMA DE GRAFOS DE DEPENDÊNCIAS DE TAREFAS

Projeto de Conclusão de Curso elaborado e apresentado na disciplina de Projeto de Conclusão de Curso, Curso de Curso de Ciência da Computação, Departamento de Engenharias e Ciência da Computação da Universidade Regional Integrada do Alto Uruguai e das Missões - Campus de Erechim.

Professor. Fábio A. Zanin

ERECHIM - RS
2018

RESUMO

O desenvolvimento de aplicações paralelas voltadas a arquiteturas heterogêneas pode vir a ser algo desafiador, pois nessas arquiteturas, os componentes são formados por processadores *multi-core* (CPUs) e placas gráficas (GPUs). O programador então, deverá distribuir as instruções que devem ser executadas em alguns destes componentes, para tentar alcançar o melhor desempenho da aplicação. Existem atualmente ambientes que se encarregam desta distribuição das instruções para os componentes da arquitetura, como por exemplo, o *StarPU*. Tendo em vista isto, este trabalho explorará o paradigma de Grafo de Dependência de Tarefa, aplicando-o a uma aplicação paralela de simulação de Decomposição Cartesiana sobre o ambiente de execução *StarPU*. Ao final deste trabalho, serão coletados dados sobre esta aplicação buscando avaliar seu desempenho neste ambiente, comparando com a versão sequencial da aplicação.

Palavras-chave: Aplicação Paralela. Arquiteturas Heterogêneas. Computação de Alto Desempenho.

ABSTRACT

The development of parallel applications for heterogeneous architectures can be somewhat challenging because, in these architectures, the components are made up of processors multi-core and accelerators such as the GPU. The developer should then distribute the instructions that must be performed on some of these components, to gain the best performance of the application. There are currently runtime environments that are responsible for distributing the instructions to the components of the architecture, for example, the StarPU. from this perspective this paper explores the Task Dependency Graph paradigm, applying it to a parallel application of Cartesian decomposition simulation on the StarPU environment. At the end of this work, data will be collected on this application, searching to evaluate their performance in this environment, comparing with the sequential version of the application.

Keywords: Parallel Application. Heterogeneous Architectures. High Performance Computing.

SUMÁRIO

1 INTRODUÇÃO

Arquiteturas Heterogêneas são compostas pelo uso de diferentes tipos de componentes em um mesmo nó computacional. Geralmente essas arquiteturas são compostas por CPUs (*Central Processing Unit*) e aceleradores, como por exemplo, a GPU (*Graphics Processing Unit*). Atualmente, muitos fabricantes de chips estão integrando os aceleradores junto com a CPU em um mesmo chip, um exemplo disso é o *Intel Graphics*.

Utilizar aceleradores como a GPU, junto com a CPU aumenta o desempenho e diminui o gasto energético, porém, o desenvolvimento de aplicações paralelas para arquiteturas heterogêneas é de certa forma visto como algo desafiador. Para o desenvolvedor conseguir aproveitar totalmente os componentes e recursos, o mesmo deverá identificar e distribuir as instruções mais adequadas em alguns destes componentes, para então tentar alcançar o melhor desempenho da aplicação. Atualmente existem ambientes que se encarregam desta distribuição das instruções para os componentes da arquitetura, como por exemplo, o *StarPU*.

Neste contexto, o trabalho proposto tentará avaliar o desempenho de uma aplicação quando executada em um destes ambientes. Para conseguir explorar os ambientes de execução em arquiteturas heterogêneas, será necessário desenvolver uma aplicação. Essa aplicação será a de uma simulação de transferência de calor em uma placa metálica bidimensional, utilizando o paradigma de Grafo de Dependência de Tarefas.

Nesse paradigma, o código da aplicação registra a criação de tarefas e a dependência de dados que existe entre elas. Esse registro de criação de tarefas é direcionado a um ambiente de execução, que se encarrega de distribuir as tarefas de maneira igualitária entre os recursos computacionais.

Em um primeiro momento será implementado uma versão sequencial da aplicação e posteriormente, será implementado uma versão paralela, para ser executada no ambiente *StarPU*. Ao final desse trabalho, será coletado dados sobre as duas implementações e realizado uma análise de desempenho, com o objetivo de avaliar se a aplicação utilizando o paradigma de grafo de dependência de tarefas, terá maior desempenho quando executada no ambiente *StarPU*.

O presente trabalho está estruturado da seguinte maneira: a seção 2 apresenta os objetivos gerais e específicos, a sessão 3 a justificativa do projeto de pesquisa e a seção 4 o referencial teórico. Em seguida a seção 5 apresenta a metodologia e a sessão 6 o cronograma de atividades. Por fim a seção 7 apresenta os resultados esperados.

2 OBJETIVOS

O presente trabalho tem como principal objetivo, analisar o desempenho de uma aplicação paralela utilizando o paradigma de Grafo de Dependência de Tarefas no ambiente de execução StarPU.

2.1 Objetivos Específicos

A seguir são listados os objetivos específicos para este projeto:

- Implementar a aplicação sequencial da simulação de transferência de calor;
- Configurar o ambiente StarPU;
- Implementar a aplicação paralela da simulação de transferência de calor com o paradigma de grafo de dependências de tarefas;
- Verificar qual é melhor técnica de análise de desempenho sobre os dados das execuções das aplicações.

3 JUSTIFICATIVA

A maioria dos sistemas de processamento de alto desempenho tem atualmente a sua arquitetura heterogênea, composta por CPUs (*Central Processing Unit*) *multicore* e aceleradores como a GPU (*Graphics Processing Unit*). O desenvolvimento de aplicações para essas arquiteturas é feito através de APIs (*Application Programming Interface*), ou bibliotecas. Essas APIs são específicas para cada tipo de componente de uma arquitetura heterogênea. Por exemplo, nas aplicações que exploram os recursos das CPUs *multicore* pode se usar a API OpenMP (*Open Multi-Processing*) (OPENMP, 2018) ou a biblioteca Intel TBB (INTEL, 2018) (*Threading Building Blocks*). Já nas aplicações que visam utilizar a GPU pode se usar OpenCL(*Open Computing Language*) (NVIDIA, 2018b) ou CUDA (*Compute Unified Device Architecture*) (NVIDIA, 2018a).

De acordo com (STRINGHINI; GONÇALVES; GOLDMAN, 2012), caso o desenvolvedor queira extrair o melhor desempenho possível e utilizar ambos recursos de uma arquitetura heterogênea, é necessário que o mesmo saiba identificar quais são as tarefas mais adequadas para a execução em um acelerador e aquelas mais adequadas à execução em uma CPU *multicore*.

Outra questão que é observada é sobre a portabilidade de um código para uma arquitetura heterogênea. Segundo (PHOTHILIMTHANA et al., 2013), um dos grandes desafios para utilizar os recursos heterogêneos é a diversidade de dispositivos em diferentes máquinas. Pois um programa que tenha sido otimizado para um determinado acelerador, pode não funcionar tão bem na próxima geração de processadores ou em um dispositivo de um outro fornecedor. No caso da GPU a programação é feita em um nível próximo ao do *hardware*, fazendo com que muitas vezes o código fique restrito a um modelo e ou fabricante (PINTO, 2011).

Vale salientar que o desempenho das CPUs e GPUs também variam entre as máquinas. Uma execução específica pode ter um melhor desempenho na CPU, enquanto que em outra máquina o desempenho pode ser melhor na GPU. Existe também alguns problemas no escalonamento das tarefas de uma aplicação para os recursos da máquina, que podem ser resolvidas com balanceamento de carga. Como por exemplo, redistribuir a carga para uma CPU ociosa, quando a GPU estiver sobrecarregada. (PHOTHILIMTHANA et al., 2013).

Atualmente existem ambientes de execução que dão suporte a CPUs e GPU simultaneamente, como por exemplo, o StarPU (AUGONNET, 2011). Esses ambientes buscam distribuir as tarefas de maneira igualitária entre os recursos computacionais CPUs e GPUs. De acordo com (KUMAR, 2017), a utilização destes ambientes libera o desenvolvedor da necessidade de adaptar a aplicação especificamente à máquina destino e unidades de processamento.

4 REFERÊNCIAL TEÓRICO

4.1 Programação Paralela

Um programa de processamento paralelo é um único programa que é executado em vários processadores simultaneamente. A programação paralela pode ser usada para o reduzir o tempo de execução de um programa, que busca encontrar uma solução para um problema complexo, como por exemplo, problemas voltados as áreas científicas (HENNESSY; PATTERSON, 2014).

A diferença de um programa sequencial para um programa paralelo, é que um programa sequencial é visto como uma série de instruções sequenciais que devem ser executadas em um único processador. Já um programa paralelo, é visto como um conjunto de partes que podem ser resolvidos concorrentemente, essas partes, são constituídas de instruções sequenciais (HENNESSY; PATTERSON, 2014; TANENBAUM; MODERNOS, 2010)

4.2 Arquiteturas Voltadas a Computação Heterogênea

Muitas das arquiteturas que estão atualmente nos supercomputadores é voltada a computação heterogênea (MEUER et al., 2014). O termo heterogênea, descreve que diferentes arquiteturas possam ser usadas em um mesmo nó computacional, como por exemplo, processadores multicore e aceleradores como a GPU e a FPGAs (STRINGHINI; GONÇALVES; GOLDMAN, 2012).

4.2.1 Processadores *Multicore*

Os processadores *multicore* ou de múltiplos núcleos, são um único componente de computação (um único chip), com duas ou mais unidade de processamento que são chamadas de núcleos. (BLAKE; DRESLINSKI; MUDGE, 2009) “logo um microprocessador quadcore é um chip que contém quatro processadores, ou quatro núcleos” (HENNESSY; PATTERSON, 2014, p. 31)

Os multiprocessadores com múltiplos núcleos ganharam destaque nos últimos anos, “a partir de 2006 todas as empresas de desktop e servidor estavam usando microprocessadores com múltiplos processadores por chip” (HENNESSY; PATTERSON, 2014, p. 31) O motivo de se tornarem tão populares foi a barreira física encontra no aumento de desempenho dos processadores sequenciais, que era baseado no aumento da frequência de *clock* (HENNESSY; PATTERSON, 2014).

Para o sistema operacional cada núcleo é visto como um processador lógico independente.

> Cada CPU tem sua própria memória e sua própria cópia privada do sistema operacional. Em consequência, as *n* CPUs operam então como *n* computadores independentes. Uma otimização obvia consiste em permitir que todas as CPUs compartilhem o código do sistema operacional e façam cópias privadas somente dos dados. (TANENBAUM; MODERNOS, 2010, p. 331)

Para programar processadores *multicore* pode-se optar por utilizar ferramentas como OpenMP e Intel TBB. OpenMP é uma API que suporta programação de processadores *multicore* em C/C++ e Fortran. A API consiste na adição de diretivas de compilação para indicar blocos que podem paralelizados. Intel TBB é uma biblioteca desenvolvida pela Intel, onde são usados *templates* para programação paralela em linguagem C++, neste caso não são necessários a utilização de linguagens ou compiladores especiais

4.2.2 Aceleradores

Um acelerador é um *hardware* que tem a função de executar algumas instruções mais eficientes, quando comparada com a CPU. São exemplos de aceleradores as GPUs e as FPGAs (KINDRATENKO et al., 2010).

4.2.2.1 GPUs

As GPUs, em português, unidade de processamento gráfico, inicialmente foram desenvolvidas para ser um processador otimizado para gráficos 2D e 3D, vídeo, computação visual e exibição de dados. As GPUs modernas são um multiprocessador altamente paralelo e *multithread*, que podem ser utilizadas para resolver problemas complexos com programas paralelos (HENNESSY; PATTERSON, 2014). No contexto das arquiteturas heterogêneas podemos encontrar as GPUs trabalhando com as CPUs em nosso dia a dia, como por exemplo. "PCs e consoles de jogo combinam uma GPU com uma CPU para formar sistemas heterogêneos" (HENNESSY; PATTERSON, 2014, p. A-569).

Para ser possível programar aplicações de computação paralela em GPUs, opta-se geralmente por modelos de programação, como CUDA e OpenCL. CUDA, é uma interface para programação paralela escalável baseada em linguagem C/C++, para GPUs fabricadas pela NVIDIA (HENNESSY; PATTERSON, 2014).

OpenCL, é uma interface voltada para programação paralela em ambientes heterogêneos, possibilitando que o desenvolvimento da aplicação seja executado em diferentes dispositivos instalados em uma máquina (CPU-GPU). Diferente da interface CUDA, a OpenCL não está vinculada a uma fabricante de GPU, portanto o desenvolvimento de aplicações com essa interface não fica limitado a um fabricante específico (NVIDIA, 2018b).

4.3 Ambientes de Tempo de Execução Baseados em Tarefas

Os ambientes de execução baseados em tarefas tem o objetivo de facilitar o desenvolvimento e a execução de aplicações paralelas em sistemas heterogêneos. Esses

ambientes permitem que os desenvolvedores programem aplicações paralelas em alto nível com APIs simples, retirando a obrigação do mesmo de lidar com detalhes de baixo nível, como transferência de dados, agendamento de tarefas e sincronizações (KUMAR, 2017).

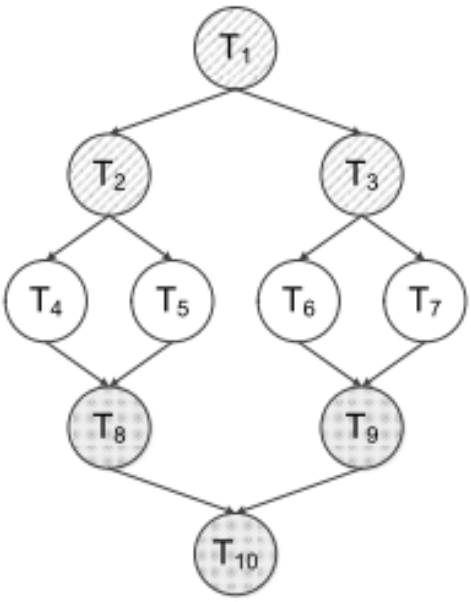
Nesses ambientes é possível desenvolver aplicações paralelas orientadas ao paradigma de Grafo de Dependências de Tarefas. Nesse paradigma, o código da aplicação registra a criação de tarefas e a dependência de dados que existem entre elas. Esse registro é expressado como um grafo acíclico direcionado - *Directed Acyclic Graph* (DAG), onde os vértices do grafo representam as tarefas a serem executadas e as arestas as dependências de dados que existem entre essas tarefas (KUMAR, 2017; THOMAN et al., 2018).

4.3.1 Paradigma de Paralelismo de Tarefas

> O paralelismo de tarefas (*task parallelism*) é o tipo de paralelismo que é naturalmente expresso por meio de tarefas paralelas em um grafo de dependência de tarefas. Algoritmos como o quicksort paralelo, a fatoração de matrizes esparsas e os algoritmos derivados pela decomposição por divisão e conquista são exemplos de algoritmos que podem ser descritos nesse paradigma (GRAMA et al., 2003 apud PINTO, 2013)

Um dos motivos que justifica utilizar esse paradigma é que muitas tarefas em um problema maior não levam a mesma quantidade de tempo para serem executadas. Colocar essas tarefas para executar paralelamente pode reduzir o tempo necessário para resolver o problema em si. Outro ponto é que, “as tarefas podem ter dependências entre si ou podem ser completamente independentes (PINTO, 2013)".

Figura 1 – Exemplo de um grafo de dependência de tarefas para o algoritmo mergesort.



Fonte: Pinto (2013)

A Figura 1 demonstra um grafo de dependência de tarefas de um algoritmo de mergesort, nesse exemplo, as tarefas *T1*,*T2* e *T3* dividem a entrada inicial, cada uma com metade do tamanho (*n/2*). Cada metade é uma nova tarefa, sendo que são elas são, *T4*, *T5*, *T6* E *T7*, essas tarefas são responsáveis por ordenar os dados. Apos isso, as tarefas *T8*, *T9* e *T10* fazem a combinação das partes separadas.

Nesse exemplo é possível notar as dependências de dados que existem entre as tarefas, pois as tarefas de combinação não podem ser executadas antes que as tarefas de divisão e ordenação tenham sido concluídas;

Aplicações baseadas no paradigma de paralelismo de tarefas podem explorar mais os recursos computacionais quando direcionadas a um ambiente de execução. Após o desenvolvedor descrever as tarefas e as suas dependências, o ambiente é responsável pelo agendamento de tarefas, transferência de dados e gerenciamento das dependências (PINTO et al., 2017).

Para executar essas atividades descritas acima, o sistema de execução infere características sobre o código da aplicação, como a quantidade de tarefas implementadas, unidade de processamento disponível (quantidade de núcleos de CPU e de dispositivos de GPUs), duração estimada da tarefa, localidade dos dados e largura de interconexão. Apos isso, o tempo de execução pode usar heurísticas de agendamento apropriadas e executar otimizações para obter um melhor desempenho (PINTO et al., 2017).

4.3.2 StarPU

StarPU é um sistema de tempo de execução que oferece suporte a arquiteturas multicore heterogêneas, oferecendo uma visão unificada dos recursos computacionais (CPUs e aceleradores ao mesmo tempo). O ambiente implementa mecanismos para escalonar de forma eficiente as tarefas em uma arquitetura heterogênea (AUGONNET, 2011).

Esta ferramenta tem como objetivo permitir que os programadores explorem o poder de computação das CPUs e GPUs disponíveis, ao mesmo tempo que em que os libera da necessidade de adaptar especialmente seus programas à máquina de destino e unidades de processamento. StarPU é uma extensão para linguagens da família C (C Extensions), ela fornece uma API para descrever as tarefas e as suas dependências que podem existir em uma aplicação (AUGONNET, 2011).

> O modelo de execução do StarPU propõe uma abordagem de tarefas independente da arquitetura base. São definidos codelets como uma abstração de uma tarefa que pode ser executada em um núcleo de uma CPU multicore ou submetido a um acelerador. Cada codelet pode ter múltiplas implementações, uma para cada arquitetura em que o codelet pode ser executado. Cada implementação utiliza as linguagens de programação ou bibliotecas específicas para a arquitetura alvo. Um codelet contém uma descrição dos dados e o tipo de acesso (leitura, escrita ou ambos). Codelets são lançados de forma assíncrona. Com isso o escalonador pode reordenar as tarefas para melhorar o desempenho respeitando as dependências entre elas. Uma aplicação StarPU é descrita como

> um conjunto de codelets com suas dependências de dados. (PINTO, 2011, p. 26)

4.4 Transferência de Calor em Placas Metálicas

5 METODOLOGIA

Para conseguir explorar os ambientes de execução em arquiteturas heterogêneas será necessário desenvolver uma aplicação. Essa aplicação que será desenvolvida será a de uma simulação de transferência de calor em uma placa metálica bidimensional, utilizando a decomposição cartesiana.

Inicialmente, será feita uma pesquisa sobre arquiteturas heterogêneas e como funcionam os ambientes de execução para estas arquiteturas, mais especificamente o ambiente StarPU. Também serão estudados alguns trabalhos relacionados, que utilizam este ambiente e o paradigma de grafo de dependências de tarefas.

Posterior a isso, será realizado a implementação sequencial da simulação. Este passo é interessante para entender como funciona a transferência de calor, utilizando a decomposição cartesiana e as dependências de dados que existem na aplicação.

Apos a implementação sequencial, será necessário estudar e configurar o ambiente StarPU. Para a configuração do mesmo, será estudado a documentação do ambiente e implementado alguns scripts do próprio tutorial. O tutorial do StarPU é importante tanto para entender como funciona o ambiente de execução, como para configurar de forma correta o ambiente.

Em seguida, será realizado a implementação paralela da simulação de transferência de calor utilizando o paradigma de grafos de dependências de tarefas. Apos isso, a aplicação será executada no ambiente StarPU.

Por fim, será verificado qual é a melhor técnica de análise de desempenho para os dados obtidos nas execuções, para então, ser concluído se aplicação paralela teve melhor desempenho, ou não, quando comparada com a aplicação sequencial.

6 CRONOGRAMA

O Quadro 1 apresenta o cronograma de atividades para este trabalho proposto.

Quadro 1 – Cronograma de Atividades.

Atividades	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
1. Revisão bibliográfica									
2. Redação da Monografia									
3. Estudo sobre as ferramentas									
4. Trabalhos relacionados									
5. Implementação sequencial da aplicação									
6. Configuração do ambiente StarPU									
7. Implementação paralela da aplicação									
8. Execução no ambiente StarPU									
8. Testes e coleta de dados									
9. Procurar a melhor técnica para a análise de desempenho									
10. Revisão final e entrega da monografia									
11. Elaboração da apresentação									
12. Defesa da monografia									
13. Entrega da versão final									

7 RESULTADOS ESPERADOS

Ao término deste trabalho espera-se saber se a aplicação paralela implementada com o paradigma de grafo de dependências de tarefas e executada no ambiente StarPU, terá melhor desempenho quando comparada com a implementação sequencial da aplicação.

REFERÊNCIAS

AUGONNET, C. **Scheduling Tasks over Multicore machines enhanced with accelerators: a Runtime System's Perspective**. Tese (Doutorado) — Université Bordeaux 1, 2011. Citado 2 vezes nas páginas 3 e 7.

BLAKE, G.; DRESLINSKI, R. G.; MUDGE, T. A survey of multicore processors. **IEEE Signal Processing Magazine**, IEEE, v. 26, n. 6, 2009. Citado na página 4.

GRAMA, A. et al. **Introduction to parallel computing**. [S.l.]: Pearson Education, 2003. Citado na página 6.

HENNESSY, J. L.; PATTERSON, D. A. **Organização e projeto de computadores: a interface hardware/software**. [S.l.]: Elsevier Brasil, 2014. v. 4. Citado 2 vezes nas páginas 4 e 5.

INTEL. **Threading Building Blocks**. 2018. Disponível em: <https://www.threadingbuildingblocks.org/>. Acesso em: 18 de maio de 2018. Citado na página 3.

KINDRATENKO, V. et al. High-performance computing with accelerators. **Computing in Science & Engineering**, IEEE, v. 12, n. 4, p. 12–16, 2010. Citado na página 5.

KUMAR, S. **Scheduling of Dense Linear Algebra Kernels on Heterogeneous Resources**. Tese (Doutorado) — Université de Bordeaux, abr. 2017. Disponível em: <https://tel.archives-ouvertes.fr/tel-01538516>. Citado 2 vezes nas páginas 3 e 6.

MEUER, H. W. et al. The top500: History, trends, and future directions in high performance computing. Chapman & Hall/CRC, 2014. Citado na página 4.

NVIDIA. **Nvidia Accelerated Computing CUDA**. 2018. Disponível em: <https://developer.nvidia.com/cuda-zone>. Acesso em: 18 de maio de 2018. Citado na página 3.

NVIDIA. **Nvidia Accelerated Computing OpenCL**. 2018. Disponível em: <https://developer.nvidia.com/opencl>. Acesso em: 18 de maio de 2018. Citado 2 vezes nas páginas 3 e 5.

OPENMP. **OpenMP The OpenMP API specification for parallel programming**. 2018. Disponível em: <https://www.openmp.org/>. Acesso em: 18 de maio de 2018. Citado na página 3.

PHOTHILIMTHANA, P. M. et al. Portable performance on heterogeneous architectures. In: ACM. **ACM SIGARCH Computer Architecture News**. [S.l.], 2013. v. 41, n. 1, p. 431–444. Citado na página 3.

PINTO, V. et al. A visual performance analysis framework for task-based parallel applications running on hybrid clusters. 2017. Citado na página 7.

PINTO, V. G. Ambientes de programação paralela híbrida. **Porto Alegre**, 2011. Citado 2 vezes nas páginas 3 e 8.

PINTO, V. G. Escalonamento por roubo de tarefas em sistemas multi-cpu e multi-gpu. 2013. Citado na página 6.

STRINGHINI, D.; GONÇALVES, R. A.; GOLDMAN, A. Introdução à computação heterogênea. **XXXI Jornada de atualizaça em Informática (JAI)**, 2012. Citado 2 vezes nas páginas 3 e 4.

TANENBAUM, A. S.; MODERNOS, S. O. **3a. Edição**. [S.l.]: Editora Prentice-Hall, 2010. Citado 2 vezes nas páginas 4 e 5.

THOMAN, P. et al. A taxonomy of task-based parallel programming technologies for high-performance computing. **The Journal of Supercomputing**, Springer, v. 74, n. 4, p. 1422–1434, 2018. Citado na página 6.