1
Introdução

Com o advento dos processadores multi-núcleo como uma alternativa para superar as limitações de aumento de velocidade dos processadores, a programação paralela torna-se uma necessidade real, uma vez que será a maneira de prover o aumento no desempenho das aplicações, sejam elas científicas, campo que já faz uso dessa tecnologia, ou comerciais, onde o uso de paralelismo é ainda pouco expressivo.

No entanto, escrever programas paralelos ainda é uma tarefa não trivial. Quando paralelizamos um algoritmo, devemos primeiro identificar as partes que podem ser executadas em paralelo e quais precisam ser executadas de forma sequencial. Além do algoritmo, que precisa ser desenvolvido para um ambiente paralelo, a comunicação entre as partes do programa – um aspecto não-funcional – deve ser considerada em relação à topologia de rede, localidade de memória, distância entre processadores, entre outros fatores. Todos esses fatores, quando considerados de forma conjunta, aumentam significativamente a complexidade do desenvolvimento de aplicações paralelas.

Uma parte dessa complexidade faz parte do próprio problema que se quer resolver e está diretamente associada ao algoritmo de solução, porém uma outra parte está relacionada às ferramentas que são usadas, bem como à crença de que precisamos atingir o melhor desempenho possível para que compense o uso de técnicas de paralelismo. No entanto, se considerarmos que o uso de paralelismo nos próximos anos deixou de ser uma opção para o desenvolvimento de sistemas, torna-se evidente a necessidade de buscarmos técnicas que nos permitam um ganho de desempenho suficientemente satisfatório sem que seja necessário ao programador tratar todos os detalhes de forma manual.

As ferramentas para suporte ao desenvolvimento de aplicações paralelas são desenvolvidas com foco no desempenho, não proporcionando facilidades para os desenvolvedores em termos de simplificar a quantidade de detalhes que precisam ser tratados, como discutido em (Karwande2003, Ke2004, Alind2008). Outro aspecto importante é que quanto mais sofisticada é a arquitetura do processador, maior é a complexidade envolvida no processo de construção da aplicação. Na grande maioria das vezes são utilizadas lingua-

gens compiladas por apresentarem reconhecidamente um desempenho maior, embora isso dificulte o processo de criação de camadas de infraestrutura mais dinâmicas. Conforme argumentado em (Skillicorn1998), um modelo que apresente maior abstração pode proporcionar aplicações que sejam mais fáceis de desenvolver e manter, porém não se sabe o quanto de desempenho seria perdido por cada escolha e o quanto valeria a pena abrir mão de uma parte do desempenho para ter uma ferramenta de maior abstração.

Considerando todo o conhecimento que a Engenharia de Software acumulou nas áreas de gestão da complexidade, métodos de desenvolvimento e testes de software para o domínio dos sistemas de informação e aplicações de negócios (Pressman1997, Sommerville2001, Peters1998), é razoável considerar a aplicação dessas técnicas na evolução da programação paralela. Embora o desempenho continue a ser um dos principais motores da arena de programação paralela (Foster1995), devemos começar a considerar que, talvez, alguma perda seja aceitável diante de outros fatores importantes e difíceis de gerir, tais como a complexidade e tamanho do software, controle de versão de manutenção, confiabilidade, desenvolvimento mais rápido, mais expressividade de modelagem e assim por diante.

Para entender melhor essa visão mais ampla do paralelismo, podemos considerar o ciclo de evolução de tecnologias para desenvolvimento de sistemas apresentado em (Carvalho2005). Este ciclo é composto por três fases sendo a primeira fase concentrada em questões de eficiência e desempenho; a segunda fase tem foco em atingir portabilidade para que seja possível utilizar aquela nova tecnologia em diferentes arquiteturas de hardware com características distintas; finalmente a terceira fase tem como objetivo buscar abstrações de mais alto nível que permitam aos desenvolvedores resolver aplicações em grande escala.

Podemos definir o problema a ser tratado neste trabalho como sendo a grande variedade de informações não relacionadas ao problema da aplicação, e que precisam ser gerenciadas pelo desenvolvedor de aplicações paralelas, principalmente no que tange detalhes de desempenho por conta de haver uma expectativa de que, dessa forma, seja utilizada toda a capacidade de processamento que a máquina é capaz de proporcionar.

Neste sentido, nossa motivação está em disponibilizar um modelo em camadas baseado em uma linguagem interpretada para abstrair, progressivamente, os detalhes envolvidos no desenvolvimento de aplicações paralelas, tais como detalhes de arquitetura, detalhes de topologia e infraestrutura de comunicação, algoritmos de comunicação, padrões de paralelismo e gerenciamento de acesso a memória entre outros. As linguagens de pro-

gramação interpretadas têm sido utilizadas em ambientes de paralelismo por permitir maiores facilidades para o desenvolvedor de aplicações, como podemos ver em (Ururahy1999, Miller2002, Ong2002, Hinsen2006, Luszczek2007, Williams2008, McIlroy2010, Li2011). Entre essas facilidades podemos citar o controle de memória, a flexibilidade de incorporar novos códigos baseados em texto fornecido pelo usuário durante a execução e o ciclo de desenvolvimento reduzido por executar diretamente o código sem etapas de compilação. Além dessas facilidades, o uso de linguagens interpretadas traz mais um atrativo para os ambientes de paralelismo: como o código é interpretado a aplicação ganha independência de arquitetura de *hardware* na qual é executada. Qualquer recurso do *hardware* pode ser exposto à linguagem através de módulos específicos desenvolvidos por um programador que tenha experiência com a arquitetura em questão sem que isso interfira diretamente no código da aplicação.

Torna-se necessário formalizar dois papéis fundamentais no desenvolvimento de aplicações:

1. Desenvolvedor de Aplicações – é o desenvolvedor que está envolvido com o domínio da aplicação que está sendo criada para resolver um problema de alguma área de conhecimento. Trabalha junto do usuário final da aplicação (em muitos casos pode ser o próprio usuário) e está interessado no resultado da aplicação. Tem como necessidade o uso do paralelismo para atingir seu resultado mais rapidamente. Tem maior conhecimento da área de conhecimento da qual o problema se originou.

2. Desenvolvedor de Sistemas – é o desenvolvedor que está envolvido com as bibliotecas que dão suporte ao desenvolvimento de aplicações paralelas. Trabalha desenvolvendo e evoluindo as bibliotecas que são necessárias para o paralelismo e está interessado em melhorar as características (normalmente de desempenho) das funcionalidades providas por sua biblioteca. Tem maior conhecimento dos detalhes técnicos da execução e comportamento de sua biblioteca nas (possivelmente) diversas plataformas em que é suportada incluindo os detalhes de cada arquitetura.

1.1 Objetivo

Nossa contribuição com este trabalho é a implementação de um modelo para paralelismo que permita simplificar o desenvolvimento de aplicações, discutindo como o uso de camadas pode isolar os problemas relacionados ao conhecimento detalhado de diferentes arquiteturas de máquinas paralelas. Para essa discussão, torna-se necessário avaliar o custo em termos de desempenho

em relação ao benefício atingindo com a simplificação da programação e com a melhoria em termos de manutenção e legibilidade de código. Uma discussão semelhante pode ser encontrada em (Chamberlain2007), onde os autores propõem que as partes de código que não precisam de maior eficiência sejam mais simples de expressar, e nos casos em que o desenvolvedor de aplicação precise de maior desempenho, seja possível atingir isso através de construções mais explícitas.

Temos então uma inversão do enfoque no suporte ao paralelismo, onde, ao invés de disponibilizar um ambiente que provê alguma facilidade e tenta atingir o máximo de desempenho fornecido pelo *hardware* utilizado, propomos fornecer um maior conjunto de facilidades para o programador, tentando atingir um desempenho satisfatório. A diferença nessa abordagem é que podemos investir em melhorias no desempenho com um impacto menor na forma como as aplicações são desenvolvidas, adaptando a busca por desempenho sem deixar para o programador uma grande quantidade de informação a ser tratada.

Como questões quantitativas temos que observar o quanto de desempenho se perde por usar um modelo de abstração em camadas baseado em uma linguagem interpretada. Ainda como forma de quantificar as diferenças consideraremos o tamanho do código de aplicação em termos de linhas de código.

Como vantagens e benefícios a serem observados resultantes deste modelo, temos a melhoria qualitativa do desenvolvimento de aplicações paralelas, tanto do ponto de vista da facilidade de desenvolvimento, quanto da manutenção e tempo de vida de uma aplicação, já que o código fica livre de dependências de arquiteturas e mecanismos de comunicação. Uma aplicação ganha vida mais longa a medida que se torna mais portátil, consequentemente pronta para executar em novas arquiteturas que tenham o modelo implementado para si.

Como estudo de caso vamos implementar o modelo proposto para o desenvolvimento de aplicações paralelas para a plataforma *Cell Broadband Engine* (Kahle2005), em seguida expandindo para uma arquitetura Intel/AMD de 32 e 64 bits para finalmente implementar o suporte a *clusters* compostos tanto de máquinas Cell BE quanto Intel/AMD. Embora o processador Cell BE tenha sido descontinuado pela IBM para novas atualizações, ele ainda está disponível para uso e muitas pesquisas ainda tem sido realizadas para essa plataforma tais como (McIlroy2010, Khoury2011, Arandi2011). Além disso, este processador é um bom exemplo de arquitetura complexa para a qual o modelo proposto pode ajudar a diminuir a dificuldade no desenvolvimento de aplicações.

1.2 Estrutura do Texto

O Capítulo 2 discute os conceitos relacionados ao desenvolvimento de sistemas paralelos, considerando as questões de quantificação. O Capítulo 3 apresenta o modelo Numina para abstrair os detalhes referentes ao desenvolvimento de aplicações paralelas, discutindo o uso de camadas para cada nível de abstração proposto e como essas camadas foram implementadas para as arquiteturas destino deste trabalho. O Capítulo 4 apresenta os resultados das medições de tempos de execução oferecendo uma perspectiva quantitativa, em seguida avaliando as facilidades oferecidas pelo modelo sob uma perspectiva qualitativa. O Capítulo 5 apresenta uma análise dos trabalhos relacionados que ajudaram a definir nossa linha de ação. Por fim, o Capítulo 6 apresenta as contribuições desta tese e alguns trabalhos futuros.