6
Conclusão e Trabalhos Futuros

Graças às facilidades disponibilizadas pela linguagem Lua, foi possível escrever bibliotecas para organizar o processamento interno da API na forma de estruturas de dados dinâmicas, assim implementamos uma camada da biblioteca em Lua que permite expor para o programador de aplicações uma visão mais confortável do que trabalhar diretamente com bibliotecas para controle de *threads* com linguagens compiladas. Já a base do modelo foi construída em C e compilada tanto para Intel/AMD quanto para Cell/BE.

As abstrações do modelo são fornecidas na forma de camadas construídas no topo das anteriores, para fornecer serviços que permitam expressar o problema de paralelismo de forma mais simples. Nesse contexto, a abstração Core é o principal diferencial do modelo, pois permite que o desenvolvedor de aplicações paralelas pense em termos de execuções de chamadas às suas funções ao invés de pensar em termos de *threads* ou tarefas que executam continuamente.

Por serem interpretados, programas Numina são necessariamente mais lentos que o código compilado C. No entanto o código Numina pode ser escrito com aproximadamente metade das linhas de código além de não ter detalhes referentes a sincronização espalhados pelo código. Nosso principal objetivo é proporcionar meios para tornar mais fácil para o programador usar os processadores multi-core. Como vimos o impacto pelo uso de camadas não é significativo, mas a implementação através de uma linguagem interpretada aumenta o tempo de execução.

O modelo fornece ao usuário uma alternativa de mais alto nível e que tem um custo de desempenho que depende do tipo de problema abordado. Observamos que com o uso de camadas podemos aplicar otimizações ao próprio ambiente revertendo a melhora em melhor desempenho para a aplicação do usuário sem que seja necessário modificá-la. Em especial o uso de técnicas JIT mostrou-se promissor por atingir um desempenho bem próximo ao de linguagens compiladas, mas utilizando todas as facilidades providas pelo modelo.

Trabalhamos com duas arquiteturas diferentes, em especial o processador

Cell/BE apresenta características muito peculiares o que torna o desenvolvimento de aplicações para essa arquitetura uma tarefa bastante difícil. Através do uso do modelo Numina o desenvolvedor pode escrever e testar sua aplicação em ambiente Intel/AMD e executar em um processador Cell/BE sem maiores problemas. Por não existir suporte a JIT nesta plataforma não foi possível tirar proveito dessa alternativa para acelerar as aplicações.

Uma característica importante do modelo Numina é expor para o programador uma API para o uso de um cluster muito semelhante à API utilizada para a programação de uma máquina paralela única. Em especial, isso permite tratar o uso de clusters de máquinas heterogêneos de forma mais simples, principalmente no que diz respeito a utilizar os recursos de cada tipo de máquina. Quando comparado a MPI, nosso modelo permite a incorporação dinâmica de novos nós de processamento mesmo que a aplicação já esteja em andamento. No entanto um problema que pode surgir é a necessidade de controle de balanceamento de carga já que as máquinas podem ter capacidades de processamento diferentes. Se a aplicação não tiver como tratar essa condição a solução do problema pode ser atrasada com o tempo de máquina sendo desperdiçado.

Ainda em relação à abstração de cluster fornecida, podemos discutir as questões relativas à configuração do ambiente. Do ponto de vista do programador o impacto é bem menor porque seu programa não precisa ser copiado para os diferentes nós de execução do ambiente. Isso é realizado durante a execução e essa transferência é feita na forma de trechos de código em formato texto. Do ponto de vista da equipe de infraestrutura o trabalho não é muito diferente do que é feito para manter um cluster com MPI já que torna-se necessário configurar as regras de visibilidade entre máquinas e configurar o *daemon* Numina para iniciar toda a vez que a máquina for reiniciada.

6.1 Trabalhos Futuros

O modelo Numina permite maior facilidade no desenvolvimento de aplicações paralelas, mas ainda são necessárias algumas evoluções para oferecer suporte a mais facilidades e arquiteturas. Em especial, o suporte a GPUs através do uso de uma linguagem de domínio específico parece promissor, sendo que podemos usar uma linguagem criada em Lua para embutir em Numina.

Uma avaliação com um grupo de controle também poderá ajudar a identificar quais das facilidades oferecidas são mais úteis para os usuários de forma a promover novas evoluções seguindo essa linha de interesse.

Uma decisão de implementação foi deixar o código que configura a distribuição de dados centralizado. É interessante estudar se delegar essa

decisão para a hierarquia de nós não poderia trazer melhorias no desempenho. Além disso o emprego das políticas de distribuição poderia acontecer de uma forma padrão automatizada, sendo possível ao programador especificar explicitamente caso precise de uma opção diferente. Isso poderia simplificar o desenvolvimento do suporte a comunicação inter-núcleo que ainda não é fornecido pelo modelo e pode ter um impacto no desempenho de algumas aplicações. A investigação das características de outras aplicações paralelas vai permitir expandir o modelo para um suporte mais completo às abstrações necessárias a outros domínios de aplicação.

Um outro aspecto relativo à distribuição de dados é que também precisamos tratar os casos quando o conjunto de dados não corresponde exatamente ao número de núcleos para processamento.

Graças ao uso de camadas, o usuário pode prover suas próprias políticas de particionamento de dados, no entanto isso precisa ser feito nas próprias bibliotecas Numina. O ideal seria ter um ponto de extensão que permitisse ao usuário fornecer uma função Lua que fosse incorporada ao ambiente de forma transparente.