A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era - Resenha

Aurelio Grott Neto[1]

[1]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC) – Joinville, SC – Brasil

aurelio@grott.me

1. Parecer

Atualmente não é possível confiar em unidades de processamento mais eficientes para aumentar a performance de uma aplicação, o certo é obter o uso apropriado e coordenado de múltiplas unidades de processamento. o dimensionamento da aplicação não tem acompanhado o de velocidade, e o tamanho do problema continua sendo fator crucial. A responsabilidade de atingir um paralelismo escalável é do desenvolvedor.

Existem duas formas principais para se abordar a paralelização de aplicações: a autoparalelização e a programação paralela, as quais diferem na performance alcançada e facilidade de paralelização.

A autoparalelização pode ser alcançada simplesmente pelo uso de compiladores específicos, porém a complexidade do processo de transformação de um programa sequencial para um paralelo é alta e acarreta em um baixo nível de paralelismo alcançado. Por outro lado a programação paralela garante uma aplicação planejada para o paralelismo, envolvendo quesitos de particionamento, mapeamento das tarefas e outras etapas que garantem uma performance mais alta que programas autoparalelizados.

Na programação paralela é interessante destacar padrões utilizados em solução de problemas. Existem quatro padrões que podem ser relacionados a diferentes modelos de programação paralela: *Single Program Multiple Data* SPMD, Mestre/Escravo, loop e fork/join.

Sistemas paralelizados podem ser classificados em duas categorias: memoria compartilhada e memória distribuída. Na memória compartilhada, tem-se uma memória única acessível a todos os processadores[1]. Já na distribuída, cada processador tem sua própria memória.

Atualmente existem computadores que partilham das duas arquiteturas de memória, provendo maior flexibilidade no paralelismo e fazendo com que o programa seja mais eficiente. A recente disponibilidade de GPUs em um sistema *multi-core* vem guiando o desenvolvimento de aplicações paralelas para um ambiente heterogêneo, o que acarreta em um maior uso do modelo HPP (*Heterogeneous Parallel Programming*), que visa explorar ao máximo os recursos disponíveis.

Dos modelos de programação paralela:

- **POSIX Threads:** Extremamente flexível, porém também de extremo baixo nível. Usualmente associado à memória compartilhada e sistemas operacionais. Apro-

[1]Por processador entenda unidade de processamento

priado para o padrão fork/join de progamação. Geralmente não é o modelo recomendado, a falta de estruturação de sua natureza torna difícil o desenvolvimento de programas ”ajustáveis”a longo prazo, seus programas também não são facilmente escaláveis para um grande número de processadores.

- **Shared Memory OpenMP:** Associado ao modelo de programação com memória compartilhada e orientado à tarefa, este modelo funciona com um nível maior de abstração do que *threads*. Com o uso da API OpenMP, o uso de *threads* ganha uma nova estrutura, pois todo o modelo foi projetado especificamente para aplicações em paralelo. Por se tratar de *threads* também segue o padrão fork/join de programação.
- **Message Passing:** É o modelo de programação paralela onde toda a comunicação entre processos é feita pela troca de mensagens, atuando sobre uma arquitetura com memória distribuída. A interface padrão ficou definida como sendo MPI. MPI favore SPMD e segue o padrão de programação Mestre/Escravo. Neste modelo, os processos executados em paralelo possuem espaços de endereçamento de memória separados.
- **Heterogeneous Parallel Programming:** Conforme falado anteriormente, ambientes heterogêneos vem surgindo, um exemplo é um ambiente que possui múltiplos processadores juntamente de GPU e APUs. Com isso, várias táticas e formas distintas de fazer uso desse ambiente vem surgindo, dentre elas:
  - CUDA: é um modelo de programação paralela desenvolvido pela NVIDIA que utiliza da linguagem C como linguagem de alto nível para programação. Altamente recomendada para uma implementação SPMD.
  - OpenCL: padrão para programação paralela de propósito geral entre CPUs, GPUs e outros processadores. Pode ser usada também em ambientes sem GPU, somente com *multi-core*
  - DirectCompute: Método da Microsoft para programação em GPU. Infelizmente só funciona em plataforma Windows.
  - Array Building Blocks: solução para programação paralela vetorial visando computação matemática com entrada massiva de dados.

Para fazer uso tanto de ambientes com memória compartilhada e distribuída, a programação híbrida surge com o objetivo de explorar as vantagens de cada modelo: a eficiência, economia de memória e facilidade de programação de um ambiente com memória compartilhada e a escalabilidade do modelo de memória distribuída. A melhor solução quando se trata de programação híbrida depende da característica de cada aplicação.