# Ferramentas de Programação Paralela para Arquiteturas Multicore

**Maycon Viana Bordin, Helton Eduardo Ritter**

[1]Bacharelado em Sistemas de Informação – Sociedade Educacional Três de Maio
Caixa Postal 153 – CEP 98.910-000 – Três de Maio – RS – Brazil

{heltonritter,mayconbordin}@gmail.com

***Resumo.*** *A programação paralela tem sido frequentemente adotada para o desenvolvimento de aplicações que demandam alto desempenho. Com o advento das arquiteturas* multicore *e a existência de diversos níveis de paralelismo, é importante definir recursos de software adequados para a programação paralela nestas arquiteturas. Neste sentido, este artigo apresenta o estado da arte das principais ferramentas de programação paralela desenvolvidas ou que podem ser utilizadas em ambientes* multicore*, mostrando que há uma diversidade de interfaces e modelos de programação adaptáveis a nova arquitetura.*

## 1. Introdução

No transcorrer dos últimos anos tem-se visto um grande número de sistemas computacionais à disposição. Embora a evolução tecnológica do hardware tenha avançado, as tecnologias de software nem sempre são capazes de explorar ao máximo o desempenho de novas arquiteturas. Por isso, é importante conhecer os recursos de software que podem ser aplicados nestas arquiteturas, especialmente as paralelas. Neste sentido, este trabalho apresenta uma análise dos principais modelos de programação paralela existentes e as ferramentas associadas a estes modelos que podem ser adotadas em arquiteturas *multicore*.

## 2. Paradigmas e Recursos de Programação

Buscando prover o uso das instruções de hardware foram desenvolvidas diversas interfaces de software paralelo, conforme os paradigmas e ferramentas apresentados a seguir.

### 2.1. Memória Compartilhada

No modelo de programação paralela com memória compartilhada usa-se um único endereçamento de memória. Isso possibilita que os processos compartilhem informações através de variáveis armazenadas em um espaço comum. Alguns exemplos de bibliotecas com memória compartilhada são: *Cilk*, *Pthreads* e *Threading Building Blocks*.

#### 2.1.1. Cilk

*Cilk* é uma linguagem de programação de propósito geral desenvolvida no *Massachusetts Institute of Technology* (MIT) [Frigo 2007]. A interface baseia-se na linguagem C ANSI, possuindo versões para as plataformas GNU/Linux, MacOS X e Windows. O sistema de execução de *Cilk* é encarregado de fazer o balanceamento de carga e de escalonar as tarefas criadas para executar em paralelo entre os processadores, garantindo melhor desempenho e eficiência. O escalonamento das tarefas em *Cilk* pode ser feito tanto pelo

compartilhamento de tarefas como pelo roubo de tarefas. No primeiro caso, uma *thread* é escalada para executar em paralelo a cada chamada de função. Isso maximiza o processamento em paralelo, mas é penalizado pelo custo de criação de um nova *thread*. No segundo caso, quando uma *thread* termina sua tarefa ela busca mais trabalho. A vantagem desta abordagem é minimizar o tempo de criação das *threads* e priorizar a eficiência.

### 2.1.2. PThreads

*PThreads* POSIX é uma interface para *threads* em linguagem C/C++ [Andrews 2001]. Pthreads permite a criação de *threads*. Para cada *thread* criada pode-se atribuir uma função, bem como os respectivos argumentos da mesma. O controle das *threads* é feito pelo programador, que precisa saber gerenciar o código em seções críticas, uma vez que implementações incorretas podem gerar *deadlocks*. Além disso, a biblioteca oferece recursos de sincronização e controle das *threads*.

### 2.1.3. Threading Building Blocks

*Threading Building Blocks* (TBB) é uma biblioteca para C++ desenvolvida pela Intel para o desenvolvimento de softwares que utilizem a tecnologia de processadores *multicore* [Pheatt 2008]. A versão 1.0 foi lançada em 2006 para os primeiros processadores dual-core x86, Pentium D. Sua versão comercial suporta os sistemas operacionais Windows (XP ou superior), Mac OS X (versão 10.4.4 ou superior) e Linux. A divisão das tarefas em *threads* bem como o gerenciamento das mesmas foi implementado de modo a utilizar da melhor forma possível os recursos de processador. Isso torna a programação mais fácil, pois não é necessário entender como as *threads* trabalham em baixo nível.

## 2.2. Auto-Paralelização

Auto-paralelização é uma forma simples de se extrair o paralelismo de uma aplicação, onde o compilador identifica candidatos fáceis à paralelização. Não é necessário nenhuma indicativa ou modificação do programa sequencial. Um exemplo de auto-paralelização é encontrado na opção `-parallel` do compilador Intel (ifort/icc) [Intel 2009], que busca identificar laços e outras instruções facilmente paralelizáveis. Não há muitas ferramentas que fazem uso desta abordagem, uma vez que aplicações extensas são difíceis de analisar.

## 2.3. Pragmas de Compilador

Pragma é meio pelo qual informações especiais são repassadas ao compilador através do código-fonte, sendo que estas não fazem parte do código propriamente dito. Pragmas são utilizados na programação paralela como uma alternativa simples de se indicar determinados trechos de código que podem ser executados concorrentemente. Geralmente são explorados laços e algumas instruções paralelas. Pragmas são específicos para cada compilador, uma vez que estes precisam decidir como a execução será distribuída. Alguns interfaces baseadas em pragmas são *OpenMP*, *HPF* e *CUDA*.

### 2.3.1. OpenMP

*OpenMP* permite a paralelização de algoritmos escritos de forma sequencial de um modo bastante simples [Chandra 2001]. Com *OpenMP*, *threads* são criadas para a execução concorrente de instruções que foram sinalizadas no código fonte para serem executadas de forma paralela. Para tanto, basta que o compilador tenha suporte a pragmas. A simplicidade faz com que *OpenMP* seja frequentemente utilizado especialmente em laços iterativos.

### 2.3.2. High Performance Fortran

*High Performance Fortran* (HPF) é uma extensão de Fortran 90 com construções que suportam a computação paralela [Koebel et al. 1994]. HPF está baseado na programação usando paralelismo de dados, provendo portabilidade para diferentes arquiteturas, especialmente multicomputadores. HPF também oferece interfaces abertas e interoperabilidade com outras linguagens, particularmente C, e paradigmas de programação, como MPI. Diversas melhorias propostas por HPF foram incluídas na versão 95 de Fortran.

### 2.3.3. CUDA

*CUDA* é uma arquitetura de computação paralela criada pela nVidia [Nickolls et al. 2008]. Ela possibilita a utilização de *Graphic Processing Unit* (GPU), integradas as placas de vídeo, para o desenvolvimento de softwares capazes de utilizar a arquitetura para a programação de alto desempenho. As vantagens de se usar *CUDA* são principalmente pelo uso da memória compartilhada de acesso rápido e a leitura de endereços arbitrários na memória, dentre outras. Por outro lado, funções recursivas não são suportadas e a precisão de dados do tipo *Double* não segue o padrão IEEE 754.

## 3. Avaliação

A Tabela 1 apresenta uma comparação entre as diferentes interfaces de programação vistas anteriormente, destacando aspectos positivos e negativos de cada uma delas.

Com base na Tabela 1, pode-se perceber que ferramentas de programação baseadas em pragmas para o compilador oferecem formas simples de se gerar um programa paralelo a partir de um código sequencial. Esta abordagem é interessante quando o código é facilmente paralelizável e não há interesse em efetuar grandes alterações no código. O mesmo vale para compiladores que permitem a auto-paralelização. Embora *CUDA* tenha surgido recentemente, se enquadrando nesta abordagem, a mesma ainda está em fase de desenvolvimento. Assim, OpenMP e HPF continuam sendo as principais alternativas.

Por outro lado, recursos de programação desenvolvidas para arquiteturas com memória compartilhada também voltaram a ganhar ênfase com o surgimento dos *multicores*. Padrões clássicos como *PThreads* e *Cilk* foram constantemente adotados no desenvolvimento de programas com múltiplas *threads*. Por sua vez, *TBB* surgiu especificamente focado para as arquiteturas *multicore*.

**Tabela 1. Avaliação das Interfaces**

| **Interface** | Aspecto Positivo | Aspecto Negativo |
|---|---|---|
| PThreads | Bom desempenho | Balanceamento de carga e sincronismo de acesso manuais |
| Cilk | Baseado em C, simples, balanceamento de carga automático | Não adotado em larga escala |
| TBB | Lida com tarefas (não *threads*). Eficiente, escalável e simples | Número fixo de *threads* no código. *Locks* em dados causam serialização |
| Autoparalelização | Paralelismo automático | Limitado ao paralelismo de laços |
| OpenMP | Implementação simples | Escalabilidade limitada pela arquitetura da memória |
| HPF | Paralelismo automático de laços | Limitado a Fortran 90 |
| CUDA | Memória compartilhada rápida e leitura alealória da memória | Não suporta funções recursisva. Gargalo na comunicação CPU/GPU |

## 4. Conclusões

Neste artigo foram avaliadas algumas interfaces de programação paralela que podem ser usadas para em ambientes *multicore*. As interfaces contemplam recursos de programação clássicos e alguns mais recentemente desenvolvidos para a arquitetura em questão. Tais modelos permitem a exploração implícita (auto-paralelização), inserção de primitivas em código sequencial (pragmas) e a programação propriamente dita em ambientes com memória compartilhada. Assim, a escolha da ferramenta vai depender da aplicação em questão, uma vez que não há nenhuma interface padrão para arquiteturas *multicores*.

## Referências

Andrews, G. R. (2001). *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, Reading.

Chandra, R. (2001). *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, USA.

Frigo, M. (2007). Multithreaded Programming in Cilk. In *PASCO '07: Proceedings of the 2007 International Workshop on Parallel Symbolic Computation*, pages 13–14, New York, NY, USA. ACM.

Intel (2009). Intel Fortran Compiler Professional Edition for Linux. Disponível em: <http://software.intel.com/sites/products/collateral/hpc/co mpilers/flin_brief.pdf>. Acesso em jul. 2009.

Koebel, C., Loveman, D., Schreiber, R., Jr., G. S., and Zosel, M. (1994). *The High Performance Fortran Handbook*. MIT Press, Cambridge.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable Parallel Programming with CUDA. *Queue*, 6(2):40–53.

Pheatt, C. (2008). Intel® Threading Building Blocks. *Journal of Computing Sciences in Colleges*, 23(4):298–298.