CONTEXTO DA PROGRAMAÇÃO MULTITHREAD EM C++

ANDRÉ DESESSARDS JARDIM[1]; EDEVALDO SANTOS[2]; GERSON GERALDO H. CAVALHEIRO[3]

[1]*Universidade Federal de Pelotas – andre.jardim@inf.ufpel.edu.br*
[2]*Universidade Federal de Pelotas – edevaldo.santos@inf.ufpel.edu.br*
[3]*Universidade Federal de Pelotas – gerson.cavalheiro@inf.ufpel.edu.br*

1. INTRODUÇÃO

O advento das arquiteturas multiprocessadas, promoveu a exposição do paralelismo das arquiteturas aos programadores. Com as limitações impostas pelas tecnologias atuais, a Lei de Moore aparenta estar atigindo o seu limite, o que permite entender que o paralelismo passa a ser um recurso fundamental e permanente na busca por desempenho de execução de programas, uma vez que "O almoço grátis acabou." (*The free lunch is over*) SUTTER (2005). Se o software deve tirar proveito desse aumento de poder computacional, ele deve ser projetado para executar várias tarefas concorrentemente, ou seja, não pode-se esperar que o aumento de desempenho de programas seja obtido artificialmente pelo aumento da frequência dos processadores. Sutter afirma que a velocidade de processamento serial do microprocessador está chegando a um limite físico, o que leva a duas consequências principais: fabricantes de processadores irão se concentrar em produtos que suportam melhor *multithreading* (como processadores multicore), e desenvolvedores de software serão forçados a desenvolver programas maciçamente *multithreaded*, como forma de melhor utilizar esses processadores (WILLIAMS, 2012).

Com aumento crescente da necessidade de índices superiores de desempenho na execução de programas, a utilização da concorrência em processos executando de forma concorrente vem sendo substituída pelo multithreading (multiprogramação leve) (CAVALHEIRO et al., 2014), que explora o mesmo modelo de programação oferecido pela utilização de processos concorrentes.

Computadores com múltiplos processadores têm sido usados para servidores e tarefas de computação de alto desempenho há vários anos, e agora os computadores com processadores multicore são cada vez mais comuns, mesmo para usuários domésticos e em smartphones. Se eles possuem múltiplos processadores ou vários núcleos em um processador (ou ambos), esses computadores são capazes de genuinamente executar mais de uma tarefa em paralelo. Isto se chama concorrência de hardware (SROUDTRUP, 2012).

Praticamente todas as arquiteturas atuais embutem paralelismo. A exploração efetiva destas arquiteturas requer, portanto, algum nível de exploração de concorrência da aplicação. Embora soluções para desenvolvimento de programas paralelos e/ou concorrentes há um bom tempo estejam disponíveis, a necessidade de produção de software fez emergir a concepção de novas ferramentas mais confiáveis que permitissem uma maior capacidade de produção e ao mesmo tempo fossem capaz de tirar proveito do hardware paralelo.

Uma das linguagens mais usadas é C++ (IEEE SPECTRUM, 2016). O objetivo do trabalho é documentar a evolução da manipulação da concorrência

dessa linguagem através dos tempos, com vista a propor uma interface de programação adaptada às necessidades de programação das aplicações atuais.

2. METODOLOGIA

A linguagem de programação C foi concebida em grande parte entre os anos 1969-1973 como uma linguagem de implementação de sistemas para o sistema operacional Unix. Derivada da linguagem BCPL, tornou-se uma das linguagens dominantes hoje. A antecessora da linguagem C, BCPL, foi concebida por Martin Richards em meados da década de 1960. Descendentes mais recentes de C incluem Concurrent C, Objective C, C*, e C++. (RITCHIE, 1993). C++ é uma linguagem de programação de propósito geral, com uma tendência à programação de sistemas, que: é um C melhor; suporta abstração de dados; suporta programação orientada a objetos; suporta programação genérica.

A exploração de concorrência em sistemas de computação com a execução simultânea de diversos fluxos de execução sobre os recursos de uma mesma arquitetura monoprocessadora. Nesta arquitetura, os fluxos de execução foram concebidos para, além de compartilhar no tempo, o acesso ao processador, e também os demais recursos da arquitetura, a cooperar entre si por meio de dados compartilhados em memória (GONÇALVES et al., 2002).

Quando relacionado ao desenvolvimento de aplicações, o termo concorrência é associado a aspectos ligados à descrição de atividades concorrentes de uma aplicação. As modernas ferramentas de programação multithread oferecem recursos para descrever a concorrência da aplicação, delegando a um núcleo de escalonamento, realizado em nível aplicativo, a responsabilidade de explorar efetivamente os recursos de processamento disponíveis (CAVALHEIRO et al., 2014).

A concorrência foi introduzida nos sistemas de computação como um meio para aumentar o desempenho, minimizando a ociosidade da(s) CPU(s). Um novo aumento do desempenho foi obtido ao arranjar as operações de CPU e de entrada e saída em paralelo. Isto imediatamente impulsionou a programação concorrente e o desenvolvimento de mecanismos e sistemas operacionais que suportassem a concorrência e o paralelismo de operações.

As principais motivações para exploração de programação concorrente são (WILLIAMS, 2012): (1) a separação de interesses**,** onde agrupando pedaços de código relacionados entre si e mantendo pedaços independentes de código separados, os programas tornam-se mais fáceis de entender e testar. E (2) para aumento de desempenho. Existem dois modelos básicos para construção de programas concorrentes. O primeiro consiste no paralelismo de tarefas, onde o algoritmo é decomposto em tarefas independentes, podendo cada uma ser executada concorrentemente. O segundo consiste no paralelismo de dados, onde um espaço de dados é decomposto e atribuído à computações independentes.

Em um programa, cada processo possui, no mínimo, um fluxo de execução (thread). Processos podem ser executados com múltiplos threads, realizando mais de uma tarefa de cada vez (*multithread*). Um programa *multithread* é aquele que pode ser descrito como efetuando diversas atividades ao mesmo tempo.

A prevalência de APIs C para *multithreading* levou muitos fornecedores de compiladores C++ a suportar *multithreading* com várias extensões específicas de plataforma. Este suporte de compilador é geralmente limitado para permitir a

utilização da API C correspondente para a plataforma e garantir que a Biblioteca *Runtime* C++ trabalhe na presença de múltiplos threads.

O Padrão C++11 é uma linguagem orientada a objetos, com recursos para *multithreading* disponibilizados na forma de classes e objetos, onde o programador não necessita conhecer características da arquitetura para realizar operações atômicas, pois ele incorpora na própria linguagem recursos para este tipo de operação. A interface para programação *multithread* estende o poder de expressão em relação ao convencional uso de uma biblioteca Pthreads em um programa C++.

C++11 não é acompanhado de um ambiente de execução. Sua implementação é realizada sobre os recursos de multiprogramação leve nativos dos diferentes sistemas operacionais. É implementado sobre Pthreads, como é o caso dos compiladores GNU C++ e ICPC (da Intel). A popularização das arquiteturas multicore popularizou o acesso a arquiteturas paralelas. É importante saber escolher a linguagem/ferramenta em função de suas características e adequação ao problema a ser implementado considerando critérios de desempenho. Na Tabela 1 são contrastadas características de várias ferramentas de programação. As primeiras cinco linhas dizem respeito a interface de programação, as demais, ao suporte de execução. (CAVALHEIRO et al., 2014).

TABELA 1 – Comparativo entre as ferramentas (CAVALHEIRO et al., 2014)

Características	Ferramentas				
	Pthreads	C++11	OpenMP	Cilk Plus	TBB
Complexidade da interface	Média	Média	Baixa	Baixa	Alta
Aderência a linguagem base	Baixa	Alta	Alta	Média	Alta
Suporte à orientação a objetos	Não	Sim	Não	Não	Sim
Abstrações de alto nível	Não	Não	Algum	Algum	Sim
Descrição da concorrência	Explícita	Explícita	Explícita	Explícita	Explícita
Escalonamento em nível aplicativo	Não	Não	Sim	Sim	Sim
Decomposição do paralelismo	Explícita	Explícita	Implícita	Implícita	Implícita
Tamanho do executável	Pequeno	Grande	Pequeno	Médio	Grande

Vantagem de C++11: os threads estão embutidos na linguagem de programação. Todos os recursos para programação orientada a objetos estão disponíveis. Desvantagem: curva de aprendizagem da ferramenta. A capacitação no uso das facilidades para programação multithread em C++11 requer que o programador domine a linguagem propriamente dita, de forma a poder separar o que é C++ e o que é programação *multithread* em C++.

3. RESULTADOS E DISCUSSÃO

Segundo WILLIAMS (2012), o Comitê de Padrões C++ e da Biblioteca Padrão de Threads C++ em particular; tem como um dos objetivos do projeto o fato de que deve haver pouco ou nenhum benefício ao utilizar diretamente as APIs de nível inferior, onde a mesma facilidade está a ser prestada. A biblioteca foi, portanto, concebida para permitir a implementação eficiente (com baixo nível de abstração) na maioria das principais plataformas.

A Biblioteca Padrão C++ fornece abstrações e facilidades de alto nível que tornam a escrita de código *multithreaded* mais fácil e menos propenso a erros. O

uso destas facilidades vem com um custo de desempenho devido ao código adicional que deve ser executado. Mas este custo desempenho não implica necessariamente uma penalidade maior de abstração; em geral, o custo não é maior do que seria incorrido pela escrita da funcionalidade equivalente à mão.

4. CONCLUSÕES

C++ não deve perder sua posição de destaque entre as linguagens de programação mais utilizadas e é de se esperar que recursos de programação concorrente sejam desenvolvidos para ela. Neste trabalho o que se propõe é o desenvolvimento de uma interface de programação com tais recursos.

Uma das novas funcionalidades mais importantes do padrão C++11 é suporte de programas *multithread*. Pela primeira vez, o padrão C++ reconhece a existência de aplicações *multithread* na linguagem e fornecer componentes na biblioteca para escrever aplicações *multithread*. Isto torna possível escrever programas C++ *multithread* sem depender de extensões específicas de plataforma e, assim, permite escrever código *multithreaded* portável com o comportamento garantido. Ele também vem em um momento em que os programadores estão cada vez mais em busca de concorrência em geral, e de programação *multithread* em particular, para melhorar o desempenho da aplicação.

5. REFERÊNCIAS BIBLIOGRÁFICAS

CAVALHEIRO, G. G. H.; DU BOIS, A. R. **Ferramentas modernas para programação multicore**. In: Salgado, Ana Carolina; Lóscio, Bernadette Farias; Alchieri, Eduardo; Barreto, Priscila Solís. (Org.). Atualizações em Informática 2014. 1ed.Porto Alegre: SBC, 2014, v. 1, p. 41-83.

GONÇALVES, R.G., PADILHA, T. P. P. **Conceitos e Implementação de um Sistema Concorrente Orientado a Objeto**. Congresso de Computação e Sistemas de Informação – EncoInfo. Anais. 2002. Centro universitário Luternao de Palmas.

IEEE SPECTRUM. **Interactive: The Top Programming Languages 2016**. USA, 16 jul. 2016. Acessado em 30 jul. Online. Disponível em: http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016

RITCHIE, D. **The Development os the C Lannguage**. HOPL-II The Second ACM SIGPLAN conference on History of programing languages. ACM, Ney York, NY, USA, 1993. P. 201-208

STROUSTRUP, B. **Princípios e Práticas de Programação com C++.** Bookman Editora Ltda. Porto Alegre, RS, Brasil. 2012

SUTTER, H. **The free lunch is over: A fundamental turn toward concurrency in software**. Dr. Dobb's Journal. USA. Mar 2005. Acessado em 30 jul. Online. Disponível em: http://www.gotw.ca/publications/concurrency-ddj.htm

WILLIAMS, A. **C++ Concurrency in Action**. Manning Publications Co. Development. Shelter Island, NY, USA. 2012.