O JOGO RESTA UM

Comparação entre algoritmo Serial e Paralelo escrito em C, Java e Python

Leandro Loffi 1, Taylan Branco Meurer2, Rodrigo Curvêllo3

1Acadêmico Ciência da Computação – Instituto Federal Catarinense Campus Rio do Sul

2Acadêmico Ciência da Computação – Instituto Federal Catarinense Campus Rio do Sul

3Professor Ciência da Computação – Instituto Federal Catarinense Campus Rio do Sul

1leandroloffi3@gmail.com, 2tbmeurer@gmail.com,

3rodrigo.curvello@ifc-riodosul.edu.br

**Abstract.** There are few academic papers on games made with parallelism, especially with C ++, Java and Python, trying new applications. This work compared serial and parallel algorithms, which show the actual effectiveness of using multiple cores to run an application in parallel in case the Peg Solitaire game with side quest in order to highlight exactly the moment of parallelism. This work also refers the efficiency of the different languages used, along with the OpenMP specifications for each language.

***Keywords:*** *Game, OpenMP, One Left, Parallelism.*

**Resumo:** Há poucos trabalhos acadêmicos sobre jogos feitos com paralelismo, ainda mais com C++, Java e Python, tratando de novas aplicações. Neste trabalho é comparado algoritmos seriais e paralelos, onde mostram a eficiência real do uso de vários núcleos para executar uma aplicação em paralelo, no caso o jogo Resta Um, com busca lateral, para poder destacar exatamente o momento de paralelismo. Esse trabalho também remete a eficiência das diferentes linguagens utilizadas, juntamente com as especificações do OpenMP para cada linguagem.

***Palavras-chave:*** *Jogo, OpenMP, Resta Um, Paralelismo.*

**1 INTRODUÇÃO**

O trabalho trata sobre serialização e paralelismo de algoritmos. Os algoritmos em série são aqueles executados linha após linha por um único processador e algoritmos paralelos são aqueles executados por dois ou mais processadores ao mesmo tempo.

As linguagens de programação paralela surgiram como um novo recurso para facilitar a tarefa de construir programas que utilizem os recursos de paralelismo de equipamentos com arquitetura de computação paralela (JUNIOR, MATTES, 2004). Para utilizar plenamente os recursos de um equipamento multiprocessado, existem basicamente duas opções, construir programas com controle sobre os trechos de paralelismo, utilizando recursos de uma Linguagem de Programação para a criação das threads e processos necessários, ou utilizar-se de uma linguagem de programação paralela, a qual, normalmente, é composta por diretivas de compilador para uma linguagem de programação pré-existente, como C.

Desde a criação dessas arquiteturas, surgiram diversas linguagens de programação paralela, entre elas a OpenMP, que será utilizada nesse trabalho. O desenvolvimento desta linguagem ou API foi realizado através de um trabalho colaborativo entre os diversos parceiros interessados no projeto, entre eles: Compaq/Digital, Hewlett-Packard, Intel, IBM, KAI, Silicon Graphics, Sun entre outros.(JUNIOR, MATTES, 2004).

O trabalho busca analisar e comparar o desempenho entre as linguagens Java, Python e C para execução de um algoritmo de multiplicação entre matrizes. As matrizes serão quadradas e terão ordem 500, 1500 e 2000. O desempenho será avaliado com emprego da lei de Amdhal e com o comando time do shell GNU/Linux em bash.

Conforme Albuquerque (2004), a lei de amdhal é uma maneira de expressar o speedup máximo como uma função da quantidade de paralelismo e da fração da computação que é inerentemente sequencial. O speedup máximo (S) alcançado por um computador paralelo com p processadores executando a computação é:

A linguagem C fará uso da API openMP, a Java do Jomp e a Python do multiprocessing. O Cython, que é um Python otimizado, também será empregado para paralelização.

**2 OPENMP**

A API OpenMP oferece um conjunto de diretivas para a programação paralela em sistemas multiprocessados com memória compartilhada, realizando para tal a criação e o controle de Threads (JUNIOR, MATTES, 2004). As diretivas da OpenMP podem ser incluídas em programas escritos nas linguagens “C” e FORTRAN para especificar pedaços de programas que devem ser executados em paralelo.

Para a implementação dos laços paralelos a OpenMP utiliza a biblioteca "Pthreads", o que que lhe proporciona um bom desempenho e um alto grau de portabilidade, uma vez que "Pthreads" possui desempenho otimizado e já foi implementada em uma inúmeras de plataformas. O Open speciﬁcation for Multi-Processing, ou simplesmente OpenMP, é um modelo de programação em memória compartilhada. Este surgiu a partir da cooperação de grandes fabricantes de hardware e software como a Sun, Intel, SGI, AMD, HP, IBM e outras (OpenMP). Projetada para ser utilizada com C/C++ e Fortran, as especificações são desenvolvidas e mantidas pelo grupo OpenMP ARB (Architecture Review Board). Trata-se de um padrão (não é uma linguagem de programação) que define como os compiladores devem gerar códigos paralelos através de diretivas de compilação, rotinas de biblioteca e variáveis de sistema. Por este motivo o resultado depende, em grande medida, do compilador que foi utilizado para gerar o aplicativo. Atualmente a especificação encontra-se na versão 4.5, lançada em novembro de 2015.

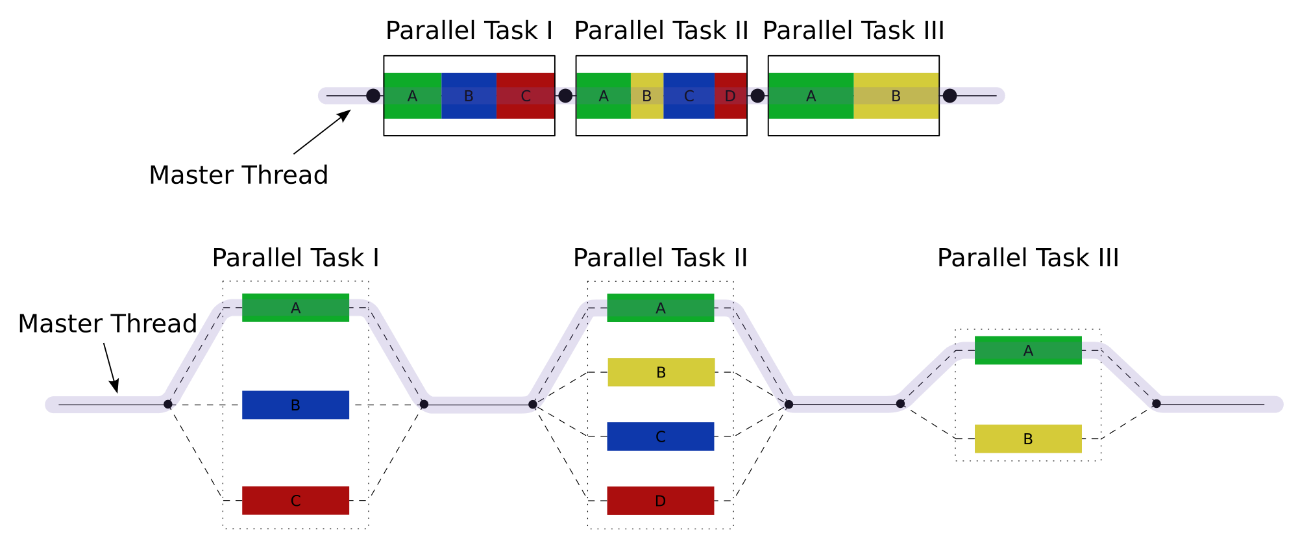


Figura 1 - Paralelismo

Conforme figura 1, pode-se visualizar a forma como pode ser dividida e o funcionamento de um programa que tem tarefas paralelas. Esta imagem foi brevemente dividida uma primeira parte em uma tarefa de 3 núcleos, na segunda parte em uma tarefa de 4 núcleos, e por último uma tarefa em 2 núcleos. Entre as tarefas, tem uma área comum serial, estas podem ser momentos em que uma tarefa pode estar ocupando um recurso de outra tarefa. Por fim mesmo que algum momento uma tarefa ficar dependendo de recurso de outra tarefa, ao final do programa a execução deverá ser mais rápida.

Segundo a revista programar (edição 46 – setembro de 2014), a OpenMP usa threads para obter paralelismo. Portanto todas as Threads usam o mesmo espaço de endereçamento de memória, sendo que os dados podem ser partilhados por todas as Threads, ou privados de cada thread (cada thread tem uma cópia da variável à qual só ela acede). De notar que o OpenMP utiliza um modelo de memória de consistência relaxada, na medida em que cada Threads pode ter uma cópia local temporária (por exemplo, em registos ou cache) cujo valor não necessita de ser constantemente consistente com o valor global visto por todas as outras Threads (vários mecanismos são disponibilizados para que o programador possa forçar a consistência de memória em determinados pontos da aplicação).

Como o enfoque do trabalho é aplicações paralelas, o OpenMP se encaixa exatamente no que é esperado para a finalidade. Portanto o OpenMP tem a possibilidade de distribuição de instruções de um programa entre vários fluxos de execuções, ou seja, Threads, as quais compartilham parte da sua memória. Algumas definições que o OpenMP permite é a forma como os cálculos serão distribuídos entre os fluxos de execução, como também a forma como os dados serão replicados ou compartilhados entre os fluxos de execução, e por último as sincronizações entre os fluxos de execução.

O funcionamento do paralelismo de programas é bem simples de declarar, através do OpenMP. Durante o desenvolvimento do programa o programador deve declarar através da diretiva #pragma, mais qual opção de paralelismo deseja efetuar. Tem-se várias opções, uma das mais usadas é #pragma omp parallel {Instruções}, que dá a várias outras variações à programas C, C++ e Fortran e de rotinas definidas pela biblioteca OpenMP. Caso o compilador não tenha suporte às diretivas OpenMP, no momento da compilação do código é simplesmente ignorado as diretivas e é compilado de forma sequencial (SENA e COSTA, 2008).

Por fim, o OpenMP é de grande valia para ser desenvolvido a aplicação deste trabalho para cada tipo de linguagem. A forma mais simples de se extrair o paralelismo usando o OPENMP é através da identificação das instruções de laços que podem ser executadas concorrentemente por threads diferentes. Assim

2.1 MULTIPROCESSING

Multiprocessing é um pacote Python que suporta processamento paralelo por meio de uma API semelhante ao módulo threading. Esse pacote oferece concorrência local e remota com uso de subprocessos em vez de threads[[1]](#footnote-2). Por essa razão, o multiprocessing permite aproveitamento dos múltiplos processadores em determinada máquina.

O módulo Multiprocessing também apresenta APIs que não têm análogos no módulo threading. Um exemplo disso é o objeto Pool, o qual oferece um meio conveniente de paralelização de uma função em vários valores de entrada, distribuindo essas entradas em processos distintos (paralelismo de dados).

2.2 CYTHON

O Cython pode ser definido como uma linguagem Python com tipos de dados em C. Quase todo código em Python é válido em Cython[[2]](#footnote-3). Ele é uma linguagem que permite a geração de código C compilável e a geração de módulos para Python, tudo a partir de um código em Python.

Por que gerar módulos para Python em Cython? Por causa do desempenho, pois garante-se, no mínimo, 25% de ganhos com relação ao Python puro. A virtude do Cython é a sua capacidade de usar váriáveis e parâmetros como tipos de dados em C. As vaviáveis e parâmetros podem ser misturados sem afetar a execução. Além disso, o Cython suporta paralelismo nativo através do módulo cython.parallel, que emprega a API openMP.

**3 METODOLOGIA**

A metodologia faz uso da linguagem C, Java e Python (versão 2.7). Cada linguagem é composta por dois algoritmos, um serial e outro paralelo. O código paralelo em C é implementado com uso da API openMP. Em Java, o paralelismo é feito com Jomp e em Python com multiprocessing e Cython.

O algoritmo implementa uma multiplicação entre matrizes. Essa implementação é bastante conhecida e generalista. Além disso, exige grande esforço computacional.

Esboço em pseudocodigo

Os testes foram executados em uma máquina com 4 processadores Intel Core I7 3630QM, CPU 3.4 GHz. A máquina possui 8GB de memória RAM e faz uso de um SSD mSATA. O Sistema Operacional instalado é o Debian stretch com kernel 4.6.0-1-amd64.

**4 RESULTADOS E DISCUSSÕES**

A tabela a seguir demonstra os resultados obtidos nos testes com uso do comando time. Os tempos estão em segundos.

O speedup obtido para cada matriz pode ser visualizado nas ilustrações abaixo:

**5 CONCLUSÃO**

O trabalho comparou o desempenho de algoritmos seriais com paralelos. Os algoritmos implementaram uma multiplicação entre matrizes quadradas de ordem 500, 1000, 1500 e 2000.

Referências

CHANDRA, R. et al. “Parallel programming in openmp”. Morgan Kaufmann, 2001.

DIVERTUDO, “Resta Um”, http://www.divertudo.com.br/restaum/restaum.html, acessado em 02/06/16.

CORMEN, Thomas H. et al. “Introduction to Algorithms. The MIT Press”. Cambridge, Massachusetts, 2 Edição, 2001.

DANTAS, Sérgio Carrazedo, “Produções de significados para o jogo resta zero”, 2013.

SOURCEFORGE, “Solitário”, http://peg-solitaire.sourceforge.net/, 2015, acessado em 02/06/16.

OpenMP, “OpenMP Application Programming Interface”, http://www.openmp.org/mp-documents/openmp-4.5.pdf, 2015, acessado em: 02/06/2016

M. FELDGEN and O. CLUA, "Games as a motivation for freshman students learn programming". Frontiers in Education, 2004. FIE 2004. 34th Annual, 2004, pp. S1H/11-S1H/16 Vol. 3.

DEITEL, Harvey M.; DEITEL, Paul J. “Java como programar”. 8. ed. São Paulo: Pearson Prentice Hall, 2010. XL, 804-854 p.

1. Em: <<https://docs.python.org/2/library/multiprocessing.html>> Acesso em: 20 junho 2016. [↑](#footnote-ref-2)
2. Em: <<http://docs.cython.org/src/tutorial/cython_tutorial.html>> Acesso em: 20 junho 2016. [↑](#footnote-ref-3)