

# Comparação entre algoritmos serial e paralelo escrito em C, Java e Python

Taylan Branco Meurer  
Leandro Loffi

Programação Paralela e Multicore  
7º fase BCC

24 de junho de 2016, v-1.9.6

# Sumário

- 1 Introdução
- 2 OpenMP
- 3 Multiprocessing
- 4 Cython
- 5 Metodologia
- 6 Resultados
- 7 Conclusão
- 8 Referências

# Introdução

Neste trabalho compara-se o desempenho de algoritmos seriais e paralelos. Os algoritmos realizam uma multiplicação entre matrizes quadradas e serão escritos em C, Java e Python. O paralelismo foi efetuado através da biblioteca OpenMP, do Jomp, do Multiprocessing e da linguagem Cython. Os tempos de execução foram obtidos por meio do comando `time` de um terminal `bash`. A métrica utilizada é baseada na Lei de Amdhal, portanto a linguagem com melhor desempenho em paralelo tem o maior speedup. Os algoritmos foram executados em uma máquina com Intel Core I7-3630QM, com SSD (mSATA) e com Sistema Operacional Debian stretch e Gnome 3.20.

# Diretivas OpenMP

Diretivas utilizadas nos códigos paralelos

## For

```
#pragma omp for  
#pragma omp for shared(A, B, C, size) private(i,j, k) num_threads(8)
```

- O primeiro for foi empregado na geração da matriz para o código C e Java

# Diretivas OpenMP

Diretivas utilizadas nos códigos paralelos

## For

```
#pragma omp for  
#pragma omp for shared(A, B, C, size) private(i,j, k) num_threads(8)
```

- O primeiro for foi empregado na geração da matriz para o código C e Java
- O segundo for foi empregado no procedimento que efetua a multiplicação entre matrizes

# Diretivas OpenMP

Diretivas utilizadas nos códigos paralelos

## For

```
#pragma omp for  
#pragma omp for shared(A, B, C, size) private(i,j, k) num_threads(8)
```

- O primeiro for foi empregado na geração da matriz para o código C e Java
- O segundo for foi empregado no procedimento que efetua a multiplicação entre matrizes
- Para mais informações, consulte <<http://openmp.org/wp/>>

# Multiprocessing

Visite <<https://docs.python.org/2/library/multiprocessing.html>>.

Use-a como um guia de orientações gerais.

Multiprocessing é um módulo Python para processamento paralelo. Ele faz uso de uma API semelhante ao threading. O pacote multiprocessing oferece concorrência local e remota, contornando o Global Interpreter Lock através de subprocessos no lugar de threads. Por essa razão, esse módulo permite ao programador tirar proveito total dos múltiplos processadores em uma determinada máquina.

Visualizar código no repositório:

- 1 <[https://github.com/targueriano/MxM/blob/master/PyMxM\\_multiprocessing/pymxm\\_multiprocessing.py](https://github.com/targueriano/MxM/blob/master/PyMxM_multiprocessing/pymxm_multiprocessing.py)>

Visite <<http://docs.cython.org/src/userguide/parallelism.html>>.  
Use-a como um guia de orientações gerais.

O Cython pode ser definido como uma linguagem com tipos de dados em C. Quase todo código em Python é válido em Cython. 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.

Visualizar código no repositório:

① <[https://github.com/targueriano/MxM/blob/master/PyMxM\\_paralelo/pymxm\\_p.pyx](https://github.com/targueriano/MxM/blob/master/PyMxM_paralelo/pymxm_p.pyx)>



## 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.

Os testes foram executados em uma máquina 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 e Gnome.

Tabela: Tempo de execução – Java

Matriz(ordem)	Serial(s)	Paralelo(s) – Jomp
500	0.228	0.189
1000	0.845	0.413
1500	2.590	1.000
2000	6.274	2.157

# Tabela



Tabela: Tempo de execução – C

Matriz(ordem)	Serial(s)	Paralelo(s)
500	0.446	0.127
1000	3.292	0.898
1500	11.461	3.189
2000	26.847	7.235

Tabela: Tempo de execução – Cython

Matriz(ordem)	Serial(s)	Paralelo(s) – Cython
500	0.837	0.504
1000	10.186	3.841
1500	30.911	12.314
2000	1min41.794	29.442

Tabela: Tempo de execução – Python2.7

Matriz(ordem)	Serial(s)	Paralelo(s) – Multiprocessing
500	1.021	0.912
1000	17.073	9.797
1500	1min7.847	31.116
2000	2min27.214	1min17.700

# Gráficos

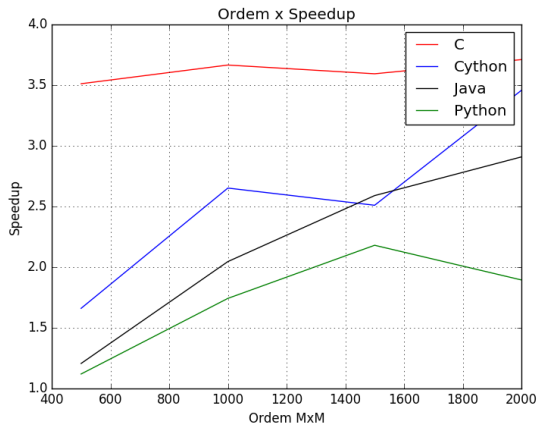


Figura: Speedup Fonte: do autor

# Gráficos

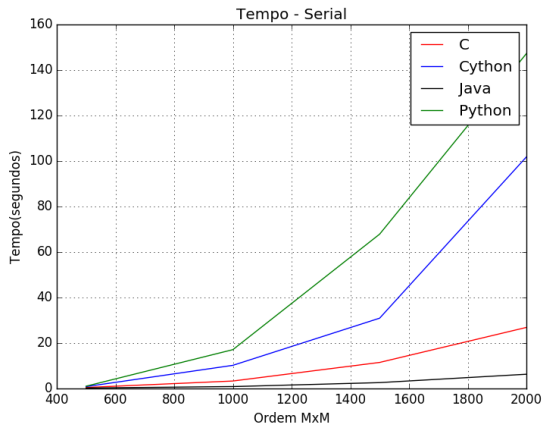


Figura: Tempo serial Fonte: do autor

# Gráficos

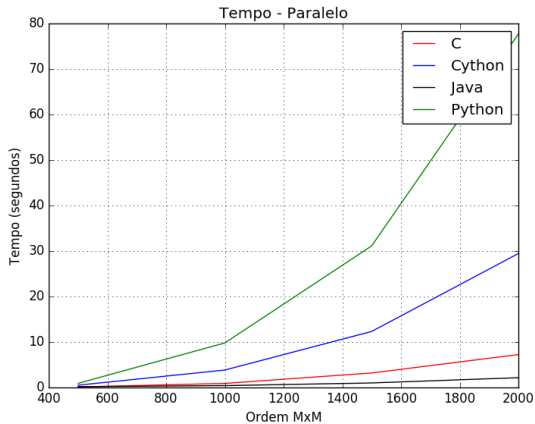



Figura: Tempo paralelo Fonte: do autor



## Conclusão

A partir dos resultados, que são bem limitados, nota-se que o emprego do OpenMP é mais vantajoso para a linguagem nativa, a saber, a C. Embora haja diversos projetos, bibliotecas ou módulos eficientes para as linguagens apresentadas nesse artigo, a interação entre API e linguagem ainda não é madura suficiente.

# Referências I

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