

**UNIVERSIDADE FEDERAL DE SÃO PAULO
INSTITUTO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA E TECNOLOGIA**

Laboratório 01

Calculando distância euclidiana com coordenadas polares usando Threads

Nomes: Ana Júlia de Oliveira Bellini
Willian Dihanster Gomes de Oliveira

RA: 111774
RA: 112269

**SÃO JOSÉ DOS CAMPOS
2018**

Introdução

Nesta atividade, é feito o cálculo da distância euclidiana com diversos pontos gerados aleatoriamente, representados em coordenadas polares. Posteriormente, com os valores obtidos, é feita uma busca pela maior distância calculada pelo algoritmo.

Para a implementação, foram utilizadas threads, a fim de analisar o desempenho do código feito com programação concorrente, em diversas condições.

Metodologia

O algoritmo possui duas versões de implementação, sendo elas nas linguagens C e Java, utilizando PThreads e JavaThreads, respectivamente. Foram feitos testes com 10^5 e 10^7 pontos aleatórios, utilizando 1, 2, 4 ou 8 threads.

Experimentos

Para os experimentos realizados com os programas em C, as especificações da máquina são descritas a seguir:

Processador: Intel (R) Core (TM) i5-7200U CPU @ 2.50GHz

Núcleos físicos: 2

Memória Cache: 3Mb L3

Memória RAM: 8 GB

Threads: 4

Hyperthreading: Sim

Sistema Operacional: Ubuntu 17.0 64 bits

Compilador: gcc 7.2.0

Para os experimentos realizados com os programas em Java, as especificações da máquina são descritas a seguir:

Processador: Intel (R) Core (TM) i5-7200U CPU @ 2.50GHz

Núcleos físicos: 2

Memória Cache: 3Mb L3

Memória RAM: 8 GB

Threads: 4

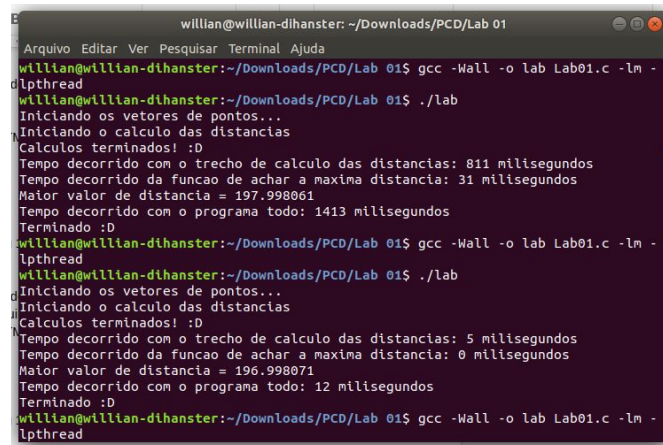
Hyperthreading: Sim

Sistema Operacional: Ubuntu 18.04.1 64 bits

Compilador: javac 10.0.2

Resultados

Na Imagem I, temos exemplos de simulações feitas no terminal do Ubuntu.



```
willian@willian-dihanster: ~/Downloads/PCD/Lab 01
Arquivo Editar Ver Pesquisar Terminal Ajuda
willian@willian-dihanster:~/Downloads/PCD/Lab 01$ gcc -Wall -o lab Lab01.c -lm -lpthread
willian@willian-dihanster:~/Downloads/PCD/Lab 01$ ./lab
Iniciando os vetores de pontos...
Iniciando o calculo das distancias
Calculos terminados! :D
Tempo decorrido com o trecho de calculo das distancias: 811 milisegundos
Tempo decorrido da funcao de achar a maxima distancia: 31 milisegundos
Maior valor de distancia = 197.998061
Tempo decorrido com o programa todo: 1413 milisegundos
Terminado :D
willian@willian-dihanster:~/Downloads/PCD/Lab 01$ gcc -Wall -o lab Lab01.c -lm -lpthread
willian@willian-dihanster:~/Downloads/PCD/Lab 01$ ./lab
Iniciando os vetores de pontos...
Iniciando o calculo das distancias
Calculos terminados! :D
Tempo decorrido com o trecho de calculo das distancias: 5 milisegundos
Tempo decorrido da funcao de achar a maxima distancia: 0 milisegundos
Maior valor de distancia = 196.998071
Tempo decorrido com o programa todo: 12 milisegundos
Terminado :D
willian@willian-dihanster:~/Downloads/PCD/Lab 01$ gcc -Wall -o lab Lab01.c -lm -lpthread
```

Imagem I: Simulação feita para a aplicação desenvolvida em linguagem C no terminal do Ubuntu.

Já na Tabela I e Tabela II temos os resultados obtidos para $N = 10^5$ e $N = 10^7$, respectivamente, para os programas desenvolvidos na linguagem C.

Número de Threads	Trecho Cálculo dos Pontos (ms)	Trecho Cálculo do Máximo (ms)	Programa Completo (ms)
1	5	0	12
2	7	0	14
4	13	0	42
8	17	0	44

Tabela I: Resultados obtidos com $N = 10^5$ com o programa em C.

Número de Threads	Trecho Cálculo dos Pontos (ms)	Trecho Cálculo do Máximo (ms)	Programa Completo (ms)
1	484	24	1045
2	483	24	1022
4	660	39	1221
8	1008	51	1606

Tabela II: Resultados obtidos com $N = 10^7$ com o programa em C.

Na Imagem II, temos um exemplo de simulação feita no software IntelliJ IDEA:

```

Run: Lab01 x
/usr/lib/jvm/java-10-oracle/bin/java -javaagent:/home.
INÍCIO DO PROGRAMA
Distâncias Finalizadas
Tempo para Distâncias: 395 milissegundos.
Máximos Finalizados
Máximo Global = 199.6085556838962
Tempo para Máximos: 37 milissegundos.
Tempo Geral: 1447 milissegundos.
FIM DO PROGRAMA

Process finished with exit code 0
  
```

Imagem II: Simulação feita para a aplicação desenvolvida em linguagem Java no software IntelliJ IDEA.

Na Tabela III e Tabela IV temos os resultados obtidos para $N = 10^5$ e $N = 10^7$, respectivamente, para os programas desenvolvidos na linguagem Java.

Número de Threads	Trecho Cálculo dos Pontos (ms)	Trecho Cálculo do Máximo (ms)	Programa Completo (ms)
1	16	2	84
2	20	3	98
4	34	10	113
8	33	13	110

Tabela III: Resultados obtidos com $N = 10^5$ com o programa em Java.

Número de Threads	Trecho Cálculo dos Pontos (ms)	Trecho Cálculo do Máximo (ms)	Programa Completo (ms)
1	200	12	1242
2	204	13	1255
4	274	27	1326
8	395	37	1447

Tabela IV: Resultados obtidos com $N = 10^7$ com o programa em Java.

Conclusões

Pode-se perceber que para os programas desenvolvidos em linguagem C e $N = 10^5$, um número mais baixo de threads se saiu melhor em relação ao desempenho e tempo. Já para $N = 10^7$, houve um melhor resultado com 2 threads, mas com mais de 2 threads, o desempenho caiu.

Enquanto isso, para a linguagem Java, em $N = 10^5$, com 4 threads, houve um desempenho pior, apresentando um desempenho levemente melhor para 8 threads. Para $N = 10^7$, com 1 e 2 threads, o resultado foi bastante semelhante, enquanto com 4 e 8 threads, o desempenho foi inferior.

Como o uso de threads é um procedimento não estocástico, é possível também notar uma variação no tempo de execução das operações para cada execução e com isso, podem ocorrer alterações.

No entanto, podemos concluir que o desempenho em ambas as linguagens é semelhante, e que a programação paralela pode melhorar o desempenho de programas, dependendo do hardware utilizado e da quantidade de threads criadas, devendo ser utilizada sempre que possível.