Resultados das Métricas de Desempenho: Programação Paralela e Distribuída

Daniel F. Costa

Departamento de Ciência da Computação
Universidade do Estado do Rio Grande do Norte (UERN) – Natal, RN – Brasil

daniel.ferreira.costa@hotmail.com

Abstract. *For the purpose of any circumstance-related computing, aimed at performance, this report aims to make a comparative of results of use of sequential programming, with topics and with MPI, to solve or problem of traveling hairless using genetic algorithm. Explain the expected results, Results were found based on concrete results.*
Keywords: *Comparison; Genetic Algorithm; Cacheiro traveler. Comparison; Genetic Algorithm; Cacheiro traveler.*

Resumo. O intuito de qualquer circunstância relacionada a computação, visa desempenho, esse relatório tem como objetivo fazer um comparativo dos resultados do uso de programação sequência, com threads e com MPI, para resolver o problema do cacheiro viajando usando algoritmo genético. Explicar os resultados esperados, comparar com os obtidos e esclarecer como a forma que foi feita a implementação influenciou nos resultados.
Palavras-chave: Comparação; Algoritmo genético; Cacheiro viajante.

1. Introdução

Para comparar os resultados entre a programação paralela e distribuída, antes, foi necessário, fazer as métricas da sequencial. Foram usados dois critérios de avaliação: o menor caminho encontrado (resultado); e o *speedup* (tempo da execução sequencial dividido pelo tempo de execução com X processadores).

Os códigos foram implementados com amplas opções de paralelização, entretanto, apenas a quantia de gerações e a quantidade de processadores foram alterados e levado em consideração nessa avaliação. Se tratando de quantia de gerações, os valores avaliados foram: 1000, 5000 e 10000. Se tratando de quantidade, no caso de threads é a quantidade de threads e para MPI é a quantidade de maquinas associadas, os valores avaliados foram: 1, 2 e 3.

Os códigos foram todos avaliados nos mesmos computadores, para que não houvesse interferência de hardware nos resultados. Foi utilizado um cluster no laboratório de CC da UERN. Para os códigos de RMI existia um computador principal, e dois "lacaios" por esse motivo a quantidade máxima de processadores avaliado foi 3. Os mesmos parâmetros foram igualados para os três tipos de código durante todos testes: tamanho da população (não confundir com nós), quantidade de melhores e quantidade de melhores que passarão para a próxima geração.

2. brazil68

Na Tabela 1, é avaliado a entrada “brazil58”, que possui 58 nós. É possível notar em qualquer perspectiva que o algoritmo de thread obteve os piores resultados, possuindo em todas as situações o tempo mais alto e o maior caminho. O tempo mais alto é devido a forma que foi implementado, todas as threads trabalham com seu próprio conjunto de população, não ajudando umas às outras, acresce a isso tem o tempo de sincronização, é fácil notar essa característica ao olhar que a média do *speedup* diminui conforme o aumento da quantidade de threads. Um fator incomum (em threads), é que os resultados de caminho obtidos conforme se aumentam as gerações não teve um padrão coerente quando comparado com as outras duas formas de implementação, que melhoraram seus resultados.

Já em relação ao MPI, pode-se dizer que foi o que apresentou os melhores resultados (menor caminho), ficando atrás do sequencial apenas quando avaliado com um único processador, é interessante notar que conforme a quantidade de gerações aumenta o resultado tende a melhorar significativamente. O tempo médio do sequencial foi o melhor, uma vez que se faz uso de troca de mensagens no MPI, porem em um dos casos o MPI apresentou o melhor *speedup* da tabela.

Quantidade de Nós: 58					
Tipo	**Gerações**	**Quantidade**	**Tempo / s**	**Resultado**	**Speedup**
Sequencial	1000	Ø	4.603304	34342	1,000000000
Sequencial	5000	Ø	22.841405	33160	1,000000000
Sequencial	10000	Ø	45.728962	32903	1,000000000
MPI	1000	1	4.637393	37640	0,992649103
MPI	5000	1	22.802446	38211	**1,001708545**
MPI	10000	1	46.134214	38347	0,991215804
MPI	1000	2	4.606709	32215	0,999260861
MPI	5000	2	23.227286	29537	0,983386737
MPI	10000	2	46.349618	**26905**	0,986609253
MPI	1000	3	4.628077	31025	0,994647237
MPI	5000	3	23.124022	31755	0,987778207
MPI	10000	3	46.161855	28717	0,990622279
Thread	1000	1	4.708647	35915	0,977627756
Thread	5000	1	23.327615	28588	0,979157321
Thread	10000	1	46.878805	34109	0,975472007
Thread	1000	2	4.876277	45412	0,944020202
Thread	5000	2	24.486904	34796	0,932800856
Thread	10000	2	48.369365	28416	0,945411667
Thread	1000	3	5.013027	47569	0,918268344
Thread	5000	3	24.234801	33368	0,942504335
Thread	10000	3	49.276080	40869	0,928015418

Tabela 1. brazil58; os nomes em verde são os melhores resultados de sua respectiva coluna

3. si175

Na Tabela 2, é avaliado a entrada “si175”, que possui 175 nós. Mais uma vez o MPI obteve os melhores resultados de caminho, porem o *speedup* não superou em nenhum momento o do sequencial, entretanto a diferença entre o tempo de execução dos dois métodos é ínfima, comparado com a melhoria do resultado obtido em RMI.

Novamente thread ficou com os piores resultados, apresentando um péssimo *speedup* quando analisado com 10.000 gerações. Porem essa grande diferença de desempenho pode se dever ao momento da análise, o computador também estar sendo o usado por outros alunos analisando seus respectivos códigos. Contudo, sabe-se que mesmo sem essa interferência threads continuaria a apresentar os piores resultados, devido a forma no qual foi implementado, mas, talvez o tempo de execução pudesse ser melhorado. Diferente da tabela passada, houve uma lógica de melhora dos resultados de threads ao aumentar a quantidade de população.

Quantidade de Nós: 175					
Tipo	**Gerações**	**Quantidade**	**Tempo / s**	**Resultado**	**Speedup**
Sequencial	1000	Ø	35.107102	29262	**1,000000000**
Sequencial	5000	Ø	173.447849	25435	**1,000000000**
Sequencial	10000	Ø	342.970102	23961	**1,000000000**
MPI	1000	1	36.463284	30144	0,962806916
MPI	5000	1	179.388031	26925	0,966886408
MPI	10000	1	350.170112	24714	0,979438536
MPI	1000	2	36.515617	28613	0,961427052
MPI	5000	2	179.229178	25505	0,967743372
MPI	10000	2	350.236730	22621	0,979252239
MPI	1000	3	36.591933	29167	0,959421903
MPI	5000	3	179.398248	24315	0,966831343
MPI	10000	3	350.323974	**21407**	0,979008368
Thread	1000	1	36.532388	36536	0,960985688
Thread	5000	1	259.652808	28659	0,667999127
Thread	10000	1	2122.662572	23317	0,161575423
Thread	1000	2	37.878321	37433	0,926838917
Thread	5000	2	258.622108	27356	0,670661338
Thread	10000	2	2122.566371	25194	0,161582746
Thread	1000	3	39.171328	38678	0,896244876
Thread	5000	3	261.95678	29269	0,621239160
Thread	10000	3	2123.011584	30192	0,161548860

Tabela 2. si175; os nomes em verde são os melhores resultados de sua respectiva coluna

4. si535

Na Tabela 3, é avaliado a entrada “si535”, que possui 535 nós. Dentre as três tabelas avaliadas até o momento, o *speedup* médio do MPI finalmente conseguiu superar o do sequencial, obtendo também, de forma unanime, os melhores resultados de caminho. Até o momento, a cada acréscimo na quantidade de nós e acréscimo na quantidade de gerações o MPI vem se mostrando a melhor escolha.

Já a forma como foi implementada threads vem obtendo resultados bastante inferiores. Entretanto, é notório comparar os resultados de 10.000 gerações dessa tabela, com a tabela passada que obteve um *speedup* incomum, isso leva a crer que o momento testado da tabela Tabela 2 (ao mesmo tempo que outras pessoas) influenciou negativamente nos resultados, uma vez que o *speedup* da Tabela 3 foi semelhante ao da Tabela 1.

Quantidade de Nós: 535					
Tipo	**Gerações**	**Quantidade**	**Tempo / s**	**Resultado**	**Speedup**
Sequencial	1000	Ø	271.025104	100979	1,000000000
Sequencial	5000	Ø	1338.210218	77290	1,000000000
Sequencial	10000	Ø	2612.936454	60147	1,000000000
MPI	1000	1	269.259159	99832	**1,006558533**
MPI	5000	1	1338.117226	76469	1,000069495
MPI	10000	1	2613.012648	56125	0,999970841
MPI	1000	2	269.862673	97257	1,004307491
MPI	5000	2	1338.126965	72324	1,000062216
MPI	10000	2	2613.791934	51232	0,999672705
MPI	1000	3	270.126903	98346	1,003325108
MPI	5000	3	1338.956346	72113	0,999442754
MPI	10000	3	2613.959024	**51171**	0,999608804
Thread	1000	1	283.540345	107977	0,955860811
Thread	5000	1	1441.089702	102257	0,928609937
Thread	10000	1	2853.959024	82648	0,915547992
Thread	1000	2	286.483574	98645	0,946040641
Thread	5000	2	1444.929457	98664	0,926142250
Thread	10000	2	2854.564789	99784	0,915353704
Thread	1000	3	288.858348	80645	0,938263013
Thread	5000	3	1447.256785	82082	0,924652924
Thread	10000	3	2856.095567	62345	0,914863103

Tabela 3. si535; os nomes em verde são os melhores resultados de sua respectiva coluna

5. si1032

Na Tabela 4, é avaliado a entrada “si1032”, que possui 1032 nós. Essa foi a tabela com menos variações de resultados, como esperando, levando em consideração os outros, threads obteve os piores resultados. Já MPI, estagnou em um menor caminho igual por cinco vezes, o que possivelmente é o menor caminho possível, na maioria das vezes o *speedup* dele foi o melhor. Nessa tabela também ocorreu algo que em nenhuma outra aconteceu, o melhor *speedup* também obteve o melhor resultado.

O tempo de execução dos códigos foram incrivelmente longos, tendo em vista a quantidade de nós, e a quantidade de gerações, por esse motivo foi colocado para mais de um código executar simultaneamente, objetivando ganhar tempo, isso de alguma forma pode ter influenciado negativamente no desempenho dos resultados.

Quantidade de Nós: 1032					
Tipo	**Gerações**	**Quantidade**	**Tempo / s**	**Resultado**	**Speedup**
Sequencial	1000	Ø	1349.031492	106678	1,000000000
Sequencial	5000	Ø	6701.579583	97012	1,000000000
Sequencial	10000	Ø	12061.057385	94960	1,000000000
MPI	1000	1	1347.033650	94475	1,001483142
MPI	5000	1	6658.785743	**92650**	**1,006426673**
MPI	10000	1	12061.167838	**92650**	0,999990842
MPI	1000	2	1348.187545	94256	1,000625986
MPI	5000	2	6658.983574	**92650**	1,006396773
MPI	10000	2	12061.668952	**92650**	0,999949297
MPI	1000	3	1349.283750	93932	0,999813043
MPI	5000	3	6659.019564	**92650**	1,006391334
MPI	10000	3	12061.908456	93376	0,999929441
Thread	1000	1	1359.793568	108099	0,992085507
Thread	5000	1	6751.579583	105002	0,992594326
Thread	10000	1	12091.748635	95738	0,997461802
Thread	1000	2	1361.094783	119481	0,991137068
Thread	5000	2	6755.095768	99234	0,992077657
Thread	10000	2	12101.013789	100935	0,996698094
Thread	1000	3	1362.985327	105007	0,989762300
Thread	5000	3	6759.094882	100982	0,991490680
Thread	10000	3	12103.578451	93041	0,996486901

Tabela 4. si1032; os nomes em verde são os melhores resultados de sua respectiva coluna

6. Conclusão

De acordo com os gráficos da tabela 5, o MPI possui o melhor *speedup* em grandes quantidades de nós, e se iguala ao se sequencial em pequenas quantidades de nós. Em todos os casos o menor caminho foi apresentado pelo MPI. O de thread em todas as circunstâncias foi avaliado como o pior. Já o sequencial, obteve bons resultados, principalmente no quesito tempo de execução, perdendo por pouco quando comparado ao RMI.

É importante deixar claro que o algoritmo thread obteve resultados tão inferiores em relação ao tempo devido a forma de implementação, onde a população avaliada é multiplicada pela quantidade de threads, ou seja, com exceção de uma thread, em todos os outros casos ele terá que analisar uma população maior que os outros tipos de implementação. O que não explica os resultados do caminho também serem inferiores.

Uma surpresa é o *speedup* superior do MPI em relação ao sequencial, uma vez que as trocas de mensagens deviam atrasar a finalização.

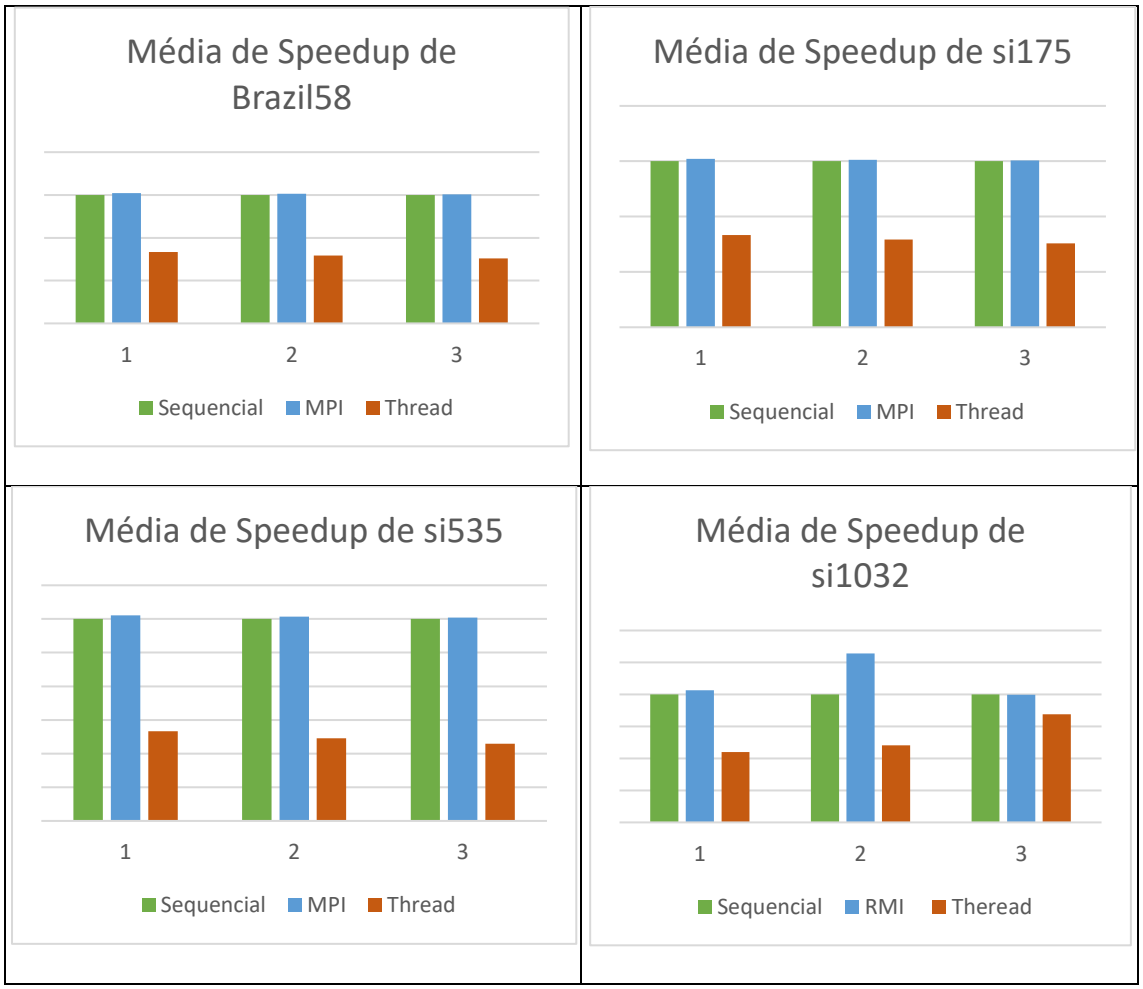

Tabela 5. Gráficos comparativos das quatro entradas avaliadas