Avaliação de implementações do Algoritmo Genético Paralelo para solução do Problema do Caixeiro Viajante usando OpenMP e Pthreads

Harison Herman Silva, Carlos Augusto Paiva Silva Martins (orientador)
Pontifícia Universidade Católica de Minas Gerais
harison.silva@sga.pucminas.br, capsm@pucminas.br

Resumo

Algoritmos Genéticos Paralelos são heurísticas que podem ser utilizadas para obter resultados em problemas Não-Polinomiais, onde o resultado ótimo só é garantido por algoritmo força bruta, o que pode ser inviável devido ao grande tempo para sua execução. Neste trabalho é apresentado um algoritmo utilizando uma função evolucionária alternativa com implementações em OpenMP e Pthread, as quais foram comparadas sobre diferentes arquiteturas. Os resultados obtidos demonstram a necessidade de análise da arquitetura antes da implementação, onde pode haver maior benefício no uso do OpenMP ou da Pthread.

1. Introdução

O Problema do Caixeiro Viajante (PCV) [1] é definido como o problema de um caixeiro para determinar a menor distância para se percorrer N cidades, sem repetição de cidade, retornando à cidade inicial, ou seja, este problema pode ser modelado através de um grafo hamiltoniano de N vértices. Sua solução ótima só é garantida através de algoritmo força bruta, o que torna impraticável o seu uso sem nenhuma forma de otimização para um número muito grande de cidades [1].

Este problema possui complexidade fatorial, e a adição de cada cidade na carga de trabalho pode aumentar fatorialmente o tempo de execução, inviabilizando seu uso. Por isto o PCV é frequentemente utilizado para validação de heurísticas, como o Algoritmo Genético (AG), que podem tornar viável o uso de grandes cargas de trabalho, porém não possuindo garantia de obter o valor ótimo.

As heurísticas têm o objetivo de apenas gerar uma solução dentro de uma faixa de erro determinada em pequeno tempo de execução, em relação ao algoritmo ótimo. Seus resultados podem ser ainda mais relevantes através do paralelismo, porém exigindo melhor análise da arquitetura e ferramentas utilizadas.

O objetivo deste trabalho é comparar o desempenho das ferramentas quando são utilizadas sem otimizações das quais dispõem, apenas foi utilizado o essencial ao seu funcionamento, sem levar em consideração a arquitetura utilizada, apresentando através dos resultados obtidos que o OpenMP pode obter melhor desempenho em uma arquitetura e o Pthread em outra arquitetura.

2. Trabalhos correlatos

O AG é uma heurística, originalmente proposta por J. H. Holland como apresentado em [2], que se baseia metaforicamente na natureza, onde indivíduos mais bem adaptados permanecem para as próximas gerações e seus descendentes tendem a melhorar sua aptidão ainda mais através de cruzamentos e mutações.

Os Algoritmos Genéticos Paralelos (AGPs) podem utilizar a arquitetura paralela dos processadores comercializados atualmente com diversos núcleos, e a natureza paralela da aplicação para obter ganho de desempenho sobre a versão sequencial. Diversas propostas foram apresentadas para possibilitar o paralelismo, das quais destaca-se o modelo em ilhas, como utilizado por [3], onde cada ilha representa uma unidade computacional independente que pode ser executada concorrentemente com parâmetros para as funções evolucionárias também independentes para cada ilha, e cada ilha possui sua própria população.

Há diversos trabalhos que utilizam o AGP para resolução do PCV relatados na literatura. Alguns destes alcançaram seus objetivos através de variações do AGP, como [3] e [4]. Os AGPs podem possuir diversas variações na forma de realizar cruzamentos e mutações, que pode promover ganho de desempenho.

Em [5] é realizada uma comparação entre OpenMP e Pthread. Em [3] e [4] a implementação foi realizada através do MPI, porém não há comparação com outras implementações. Não foi encontrado na revisão da literatura trabalho que avaliasse o PCV solucionado

pelo AGP com implementações em OpenMP e Pthreads que analisasse os resultados em diferentes computadores, como é feito neste trabalho.

3. Algoritmo proposto

Para o PCV cada cidade foi modelada como um gene, que compõe um cromossomo. Um caminho entre todas as cidades como um indivíduo e um conjunto de indivíduos como uma população. Uma população com muitos indivíduos possui maior probabilidade de encontrar o ótimo global em relação a uma população relativamente menor, porém com mais indivíduos o tempo de execução de cada geração é maior.

Em [3] e [4] foram utilizadas funções de mutação com diferentes taxas por ilha. Em tais trabalhos, o maior número de ilhas, proporcionado pelo *cluster* de computadores com processadores *multicore,* possibilitava resultados mais próximos do ótimo global. Entretanto, neste trabalho não foi utilizado um *cluster,* devido ao seu grande custo e consumo energético, optou-se por utilizar um menor número de ilhas disponibilizadas por um computador com processador *multicore*, de menor valor e consumo.

Neste algoritmo, para que haja evolução nos indivíduos e estes busquem reduzir o seu custo de seu trajeto, cada indivíduo fora da elite deve passar pela operação *Gene String Moves* (GSM) que move trechos do trajeto dentro do próprio indivíduo, em substituição à mutação e *crossover* tradicionais [6]. Esta função é executada paralelamente para cada ilha, ou seja, cada população pode realizar a evolução de seus indivíduos independente das demais populações. Para as implementações não foi necessário o uso da barreira, disponibilizada pelas bibliotecas, uma vez que cada *thread* acessa apenas sua região da população global.

Nas implementações realizadas em linguagem C, as coordenadas das cidades são armazenadas em um vetor estático de ponto flutuante, cujo tamanho sempre é o número total de cidades. Cada indivíduo é representado através de um vetor de inteiros, onde o tamanho também coincide com o total de cidades, e os seus valores representam as posições relativas das cidades. Já uma população é um vetor de indivíduos. As gerações são incrementadas através de um contador de escopo global. O número total de gerações, que foi o único critério utilizado para convergência do algoritmo, é o máximo de gerações dividido pelo número de *threads*.

O algoritmo proposto foi implementado com uso da biblioteca OpenMP, apenas com a inserção da função **omp_get_thread_num()** e da diretiva **#pragma omp parallel for**. O OpenMP é largamente utilizado por programadores com pouca, ou nenhuma, experiência em paralelismo, pois ele abstrai a criação de *threads* e necessita que o programador apenas mostre o que deve ser paralelizado. Existem diversas otimizações disponibilizadas por esta biblioteca que podem gerar ganhos significativos, como função para desenrolar e aninhar laços de repetição, dentre outras. Estas otimizações não foram exploradas nesta implementação, pois este não é o objetivo do trabalho.

Por ocultar muitos detalhes de implementações sobre as *threads*, buscando simplicidade para o desenvolvedor, entende-se que o OpenMP tende a não ser tão eficiente quanto as Pthreads [6]. Porém, isto deve ser analisado especificamente sobre o caso de estudo e a arquitetura utilizada. Por exemplo, caso o desenvolvedor busque apresentar um programa paralelizado que tenha portabilidade, o OpenMP pode lhe atender bem.

A Pthread tende a utilizar melhor do *hardware* disponível devido à implementação manual das *threads*, dentre outros motivos, bem como otimizações disponibilizadas pela biblioteca. Entretanto, a Pthread exige um maior nível de experiência com programação paralela, e o tempo para seu aprendizado pode ser maior do que para o OpenMP.

A implementação com Pthread utilizou um **struct** para cada *thread,* e neste há uma população, ou seja, um vetor de indivíduos. Desta forma, cada ilha acessa apenas os dados de sua população durante a evolução de seus indivíduos, o que permite evoluir os indivíduos livremente sem a implementação de barreiras.

4. Resultados experimentais

Os códigos foram executados em dois computadores com processadores *multicore* cujas configurações são apresentadas na Tabela 1. É importante observar que o "Computador 1" suporta 4 *threads* simultaneamente via *Simultaneous multi-threading* (SMT), ou seja, cada núcleo real emula 2 núcleos que irão concorrer pelas unidades funcionais da arquitetura superescalar, como barramento de memória e memória *cache*, enquanto o "Computador 2" possui 4 núcleos reais, o que evita a concorrência pelos recursos. Ambos utilizaram sistema operacional Linux e compilador GNU GCC 4.6.

	Computador1	**Computador2**
Processador Intel	Core i3 M380	Core i5-2400
Clock	2.53 GHz	3.10 GHz
Núcleos	2	4
Threads suportadas	4	4
Cache	3 MB	6 MB

Tabela 1 – Configurações dos computadores utilizados

Foram realizados testes onde foi fixado o número de gerações, produzindo diferentes tempos de execução e taxas de erro, em função do número de *threads*. Os resultados apresentados são a média aritmética de 3 execuções para cada carga de trabalho, pois os resultados podem sofrer variações no tempo de execução devido a outros processos executados simultaneamente pelo Sistema Operacional e possível melhor uso da memória *cache* em função de pontos escolhidos para função evolutiva.

O AGP pode ser ajustado para obter resultado em poucos segundos de execução, através da redução do total de gerações, ou ajustado para obter um resultado ainda melhor através de maior número de gerações, o que provoca aumento no tempo de execução. Além disso, diversos parâmetros como tamanho de elite e forma de evolução, podem ser alterados.

4.1. Computador 1

Executando 4 *threads* no Computador 1, por 100.000 gerações, 64 indivíduos na população e 15% de elite, os tempos de execução paralela cresceram linearmente em função do tamanho da carga de trabalho, enquanto a execução sequencial cresceu exponencialmente o tempo de execução, como pode ser observado na Figura 1.

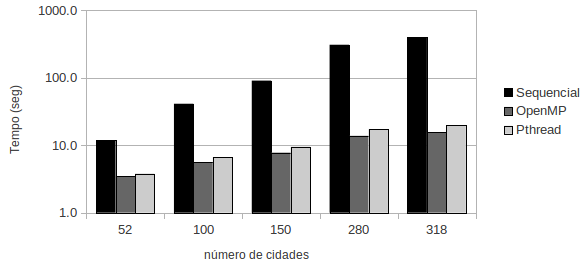

Figura 1 – Tempo de execução no Computador 1

A Tabela 2 apresenta a taxa de erro em cada implementação e o *speedup* obtido pelas implementações paralelas sobre a sequencial.

Cidades	*Speedup*		Erro (%)		
	OpenMP	Pthread	Sequencial	OpenMP	Pthread
52	3,38	3,09	63,93	31,95	19,49
100	7,39	6,17	84,55	65,25	33,90
150	11,66	9,50	85,90	73,07	45,54
280	22,33	17,81	8,50	16,48	12,36
318	25,69	20,37	64,94	62,53	50,45

Tabela 2 – *Speedup* e erro em relação ao ótimo apresentado na TSPLIB

Nestes resultados pode ser observado que a implementação em OpenMP obteve maiores valores de *Speedup,* entretanto, os resultados obtidos pela Pthread são mais próximos dos valores ótimos, apresentados pela *Traveling Salesman Problem Library* (TSPLIB) [7].

Como pode ser observado, em todas as cargas de trabalho testadas o Pthread obteve melhor qualidade que o OpenMP e para a maioria das cargas de trabalho testadas as implementações paralelas obtiveram melhores resultados que a sequencial, com exceção da carga com 280 cidades, o que será explicado na análise comparativa entre os computadores 1 e 2.

Assim, caso seja necessário obter resultado mais próximo ao ótimo neste computador o indicado é utilizar a implementação em Pthread, porém caso seja necessário obter o resultado em menor tempo o indicado é utilizar a implementação em OpenMP.

4.2. Computador 2

Realizando execução semelhante para o Computador 2 pode ser observado que a implementação com Pthread se manteve obtendo menores taxas de erro que as obtidas pelo OpenMP.

Porém, no Computador 2 a implementação com Pthreads obtém resultados em menos tempo, como pode ser observado na Figura 2.

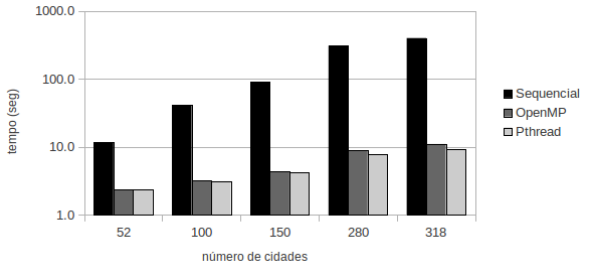

Figura 2 – Tempo de execução no Computador 2

Estes valores podem ser mais facilmente analisados através da Tabela 3.

Cidades	*Speedup*		Erro (%)		
	OpenMP	Pthread	Sequencial	OpenMP	Pthread
52	4,98	5,00	63,93	21,80	14,11
100	12,85	13,14	84,55	55,29	38,54
150	20,71	21,55	85,90	66,73	43,73
280	34,57	39,66	8,50	9,87	9,74
318	36,16	43,05	64,94	59,92	49,17

Tabela 3 – *Speedup* e erro em relação ao ótimo apresentado na TSPLIB

Diversos fatores podem promover essa melhora, dentre os quais podem-se destacar melhor utilização da memória *cache* e a existência de 4 núcleos reais, enquanto o Computador 1 possui 4 núcleos por SMT.

4.3. Comparação

A Tabela 4 apresenta uma comparação entre os valores de *speedup n*os Computadores 1 e 2.

Cidades	Computador 1		Computador 2	
	OpenMP	**Pthread**	**OpenMP**	**Pthread**
52	3,38	3,09	4,98	5,00
100	7,39	6,17	12,85	13,14
150	11,66	9,50	20,71	21,55
280	22,33	17,81	34,57	39,66
318	25,69	20,37	36,16	43,05

Tabela 4 – Comparação do valor de s*peedup*

Pelos resultados apresentados pôde ser observado que em ambos computadores utilizados as implementações paralelas obtiveram *speedup* significante, sempre acima de 3, além de obter resultados mais próximos ao ótimo global, com exceção da carga de trabalho com 280 cidades.

A TSPLIB apresenta o melhor trajeto e seu valor já encontrados para cada carga. Analisando-se o trajeto ótimo desta carga com 280 cidades, pôde-se observar que ela obteve desempenho melhor no código sequencial devido ao seu padrão de distribuição das cidades, onde diversos trechos foram apenas invertidos em relação à sua posição inicial, o que não é otimizado pela função GSM utilizada que move trechos inteiros de um trajeto.

Em ambos computadores utilizados pode ser observado que de acordo com que a carga de trabalho aumenta, a diferença de *speedup* entre as implementações também aumenta.

Nestes resultados também pôde ser observado que a Pthread obteve melhores resultados em uma arquitetura com 4 núcleos reais, maior *clock* e maior *cache,* como apresentado na Tabela 4. Desta forma, a arquitetura disponível deve ser analisada para se determinar a melhor implementação do algoritmo, considerando-se também o esforço requerido por cada implementação.

5. Conclusão e trabalhos futuros

Para diferentes execuções foi observado ganho de tempo e redução da taxa de erro no código paralelo. Considerando-se não terem sido utilizadas otimizações nas implementações, pode-se considerar como alcançados parcialmente os objetivos deste trabalho ao terem sido obtidos resultados que demonstram a relevância da análise da arquitetura utilizada.

Considerando-se uma aplicação real onde os valores ótimos não são conhecidos, o OpenMP pode ser escolhido caso o computador disponível seja equivalente ao Computador 1, com 4 núcleos via SMT, devido ao seu melhor *speedup* em relação ao Pthread. Enquanto para computador semelhante ao Computador 2, de 4 núcleos reais, o Pthread obteve melhores resultados. Isto demonstra que nem sempre a Pthread obtém melhores resultados, dependendo muito da forma de implementação e da arquitetura utilizada.

Como trabalho futuro, este algoritmo pode ser implementado utilizando uma *Graphics Processing Unit* (GPU), possibilitando o uso de mais ilhas com diferentes taxas de mutação, como feito por [3] e [4]. Também podem ser utilizados novos métodos de evolução em conjunto com outras heurísticas.

6. Referências

[1] APPLEGATE, D. L. et al. The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics). Princeton, NJ, USA: Princeton University Press, 2007. ISBN 0691129932, 9780691129938.

[2] HOLLAND, J. H. Genetic algorithms and classifier systems: foundations and future directions. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987. p. 82–89. ISBN 0-8058-0158-8. Disponível em: <http://dl.acm.org/citation.cfm?id=42512.42524>.

[3] SILVEIRA, L. G. H. D.; CARVALHO, M. B.; MARTINS, C. A. P. S. Algoritmo genético paralelo com taxas distintas de mutação. In: I Concurso de Trabalhos de Iniciação Científica em Arquitetura de Computadores e Computação de Alto Desempenho, WSCAD-CTIC 2007. [S.l.: s.n.], 2007.

[4] VIANA, I. Eveli M. et al. Algoritmo Genético Paralelo com uma Ilha Semi-Aleatória. In: WSCAD-WIC (Ed.). Anais X Simpósio em Sistemas Computacionais. [S.l.]: SBC, 2009. X.

[5] LORENZON, A. F. et al. Análise de Interfaces de Programação Paralela na Determinação de Similaridade de Histogramas . In: WSCAD-WIC (Ed.). Anais XII Simpósio em Sistemas Computacionais. [S.l.]: SBC, 2011. XII.

[6] SILVA, H. H. Algoritmo Genético Paralelo para resolução do Problema do Caixeiro Viajante. [S.l.], Trabalho de Diplomação, Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte. Junho 2012.

[7] REINELT, G. Tsplib - a traveling salesman problem library. ORSA Journal on Computing, v. 3, n. 4, p. 376–384, 1991.