Uso de OpenMP na Paralelização do algoritmo Artificial Bee Colony (ABC)

Natiele Lucca[1], **Claudio Shepke**[1]

[1] Ciência da Computação– Universidade Federal do Pampa (UNIPAMPA)
Campus Alegrete – RS – Brazil

{natiele,claudio} lucca@gmail.com, schepke@unipampa.edu.br

Resumo. *Este trabalho visa paralelizar o algoritmo Artificial Bee Colony (ABC) através do uso de diretivas OpenMP. Com isso será possível melhorar o desempenho da execução de um código-fonte da aplicação. Neste artigo, testes foram realizados comparando o paralelismo com limitante de threads. Os resultados demonstraram o impacto no uso de threads no tempo de execução e na qualidade da solução do algoritmo.*

1. Introdução

Artificial Bee Colony (Colônia Artificial de Abelhas) é um algoritmo bioinspirado baseado em populações e metaheurísticas que vêm sendo usado para resolver problemas de busca e otimização (SERAPIÃO, 2009). Este algoritmo é empregado em aplicações complexas, que possuem elevado custo de execução, como problemas de engenharia, economia e ciência.

A inteligência de colônias ou inteligência coletiva, é um conjunto de técnicas baseadas no comportamento coletivo de sistemas auto-organizados, distribuídos, autônomos, flexíveis e dinâmicos (SERAPIÃO, 2009). Algoritmos inspirados em formigas, bactérias, partículas e abelhas simulam o comportamentos do agente diante da colônia, em busca de soluções ótimas.

Este trabalho visa analisar o desempenho do algoritmo Artificial Bee Colony (ABC), explorando a eficiência no tempo de execução quando submetido a programação paralela de memória compartilhada.

2. Metodologia

Para a paralelização deste algoritmo foram utilizadas instruções OpenMP. A API (Application Programming Interface) e um conjunto de diretivas que permitem a criação de programas paralelos com compartilhamento de memória através da implementação automática e otimizada de um conjunto de threads (NETO; COSTA; SILVEIRA, 2010).

Inicialmente o programa possui uma thread. Quando seções paralelas são identificadas mais threads são disparadas de acordo com a especificação. A área paralelizada só é finalizada quando todas as threads completam a execução e são encerradas, em seguida a aplicação volta a possuir somente uma thread.

OpenMP é baseado em diretivas de compilação. Isso significa que são utilizadas palavras-chave (representadas por pragmas) que são interpretadas pelo compilador no momento da compilação. Essas palavras são instruções ao compilador e caso este não

tenha suporte a esse nível de paralelização, essas palavras serão ignoradas, fazendo com que o código seja serial (NETO; COSTA; SILVEIRA, 2010).

O algoritmo Artificial Bee Colony é composto por funções que representam as operações das abelhas campeiras, seguidoras e escudeiras. As abelhas campeiras são responsáveis por verificar a qualidade da resposta encontrada na equação em questão. As abelhas seguidoras são designadas quando as campeiras encontram boas fontes, ou seja, encontram melhores soluções para o problema. Já as escudeiras são abelhas campeiras que não obtiveram bons resultados e iniciam uma nova busca. O processo repete por um limite estabelecido.

Em sequência o pseudocódigo do algoritmo Artificial Bee Colony segundo SERAPIÃO:

1. Determine o tamanho da colônia de abelhas (COL), o número inicial de abelhas campeiras (BN); o número de fontes de alimento (SN), que é igual a BN; o número inicial de abelhas seguidoras (BC), que é igual à diferença entre COL e BN; o número de abelhas escudeiras (BE); e o número de tentativas de liberar uma fonte de alimento (lim).

2. Envie aleatoriamente as abelhas campeiras para as fontes de alimento iniciais.

3. Envie as abelhas seguidoras para as melhores fontes de alimento encontradas pelas campeiras e determine as quantidades de néctar coletadas por cada uma.

4. Calcule o valor de probabilidade das fontes que serão escolhidas pelas abelhas campeiras.

5. Interrompa o processo de exploração das fontes abandonadas pelas abelhas (piores fontes).

6. Envie as escudeiras, aleatoriamente, para a área de busca para descobrir novas fontes.

7. Memorize a melhor fonte de alimento encontrada até o momento.

8. Se o número de tentativas de descobrir novas fontes de alimento fracassar (nt ¿ lim), ou seja, se durante lim tentativas as fontes de alimento não melhorarem então as abelhas escudeiras devem abandonar suas fontes estagnadas e buscar aleatoriamente novas fontes de alimento.

9. Se condição de término não for alcançada, retorne ao passo 3.

Seguem os trechos de códigos que foram paralelizados:

1. O método que calcula as probabilidades de escolha de uma fonte de alimento (proporcional a qualidade da fonte) e a atualização da qualidade da mesma.

2. Função responsável por inicializar as fontes de alimento, onde acontecem os sorteios aleatórios das posições para os alimentos e as inicializações do vetor de fontes.

3. Método que compara as soluções encontradas pelas threads e em caso de um resultado melhor atualiza os valores.

4. Determinar para as fontes de baixa qualidade (soluções ruins) um novo inicio.

5. Múltiplas execuções concorrentes (runtime).

O algoritmo utilizado para a pesquisa é disponibilizado pelo Intelligent Systems Research Group (ABC, 2009), na linguagem de programação C e para a paralelização do mesmo são empregadas instruções OpenMP.

3. Resultados

Foram realizadas 30 execuções sequenciais do algoritmo ABC. Como ele possui variáveis determinadas por valores aleatórios há a necessidade de repetir o processo para que os dados não sejam equivocados. Em média o tempo gasto pela aplicação foi de 6,254s, com a solução em média de 0,36e-6 próximo ao ideal (zero).

O computador utilizado nos testes possui um processador Intel (R) Core (TM) i5-5200U, 4 cores, CPU de 2.20GHz e 4GB de memória.

Em seguida o algoritmo foi analisado e métodos paralelizáveis foram identificados e paralelizados através do pragma ?parallel for?, limitando o número máximo de threads em 2. Sendo registrado o tempo de execução e a qualidade da solução obtida para cada função exposto na Tabela 1.

Função	Tempo de Execução (s)	Solução
1 - CalculateProbabilities	6,22	$7.57e^{-08}$
2 - init	6,18	$4.80e^{-10}$
3 - SendEmployedBees	6,30	$3.98e^{-11}$
4 - SendScoutBees	6,20	$7.21e^{-9}$
5 - Main	6,21	$1.08e^{-11}$

Figure 1. Tabela 1. Tempo de execução para as funções paralelizadas com 2 threads

Como podemos observar o tempo de execução dos métodos foi semelhante ao sequencial e em apenas um caso superior, uma vez que o uso fixo de threads minimiza o tempo de espera entre os blocos de execução. Enquanto a solução apresentou melhora para todos os testes efetuados.

Um segundo teste foi realizado limitando o número de threads em 10 para cada uma das funções anteriormente citadas. Em busca de melhor desempenho como demonstrado na Tabela 2.

Como podemos observar na Tabela 2, o tempo de execução foi superior ao teste anterior (Tabela 1)e ao sequencial. Para os casos 2, 3 e 5 o algoritmo teve a qualidade da sua solução afetada.

As vezes instruções OpenMP não apresentam o resultado esperado, uma vez que o fluxo de execução de uma área paralelizada é encerrado apenas quando a sincronização das threads acontece.

Função	Tempo de Execução (s)	Solução
1 - CalculateProbabilities	9,51	$1.17e^{-8}$
2 - init	8,32	$1.04e^{+1}$
3 - SendEmployedBees	8,24	$1.43e^{+1}$
4 - SendScoutBees	9,4	$1.10e^{-12}$
5 - main	3,97	$4.52e^{+2}$

Figure 2. Tabela 2. Tempo de execução para as funções paralelizadas com limite de 10 threads

4. Considerações Finais

Os testes realizados e descritos nesse trabalho exploram a paralelização do algoritmo Artificial Bee Colony através de diretrizes OpenMp.

O algoritmo de otimização ABC apresentou melhora no tempo de execução e na solução do problema quando foram utilizadas 2 threads, mas quando submetido a 10 threads o tempo de execução e a solução tiveram um aumento.

Os resultados mostram a influência da quantidade de threads no desempenho de um algoritmo, apesar da análise que várias threads executam determinada tarefa em menos tempo, detalhes como capacidade de hardware e tempo de sincronização entre as threads, podem interferir na execução do algoritmo.

Em sequência, serão realizados outros testes com as funções que apresentaram bons tempos de execução e/ou qualidade de solução. Em busca de minimizar o tempo de execução do algoritmo Artificial Bee Colony, mantendo uma boa solução.

5. Referências

SERAPIÃO, A. B. S. Fundamentos de otimização por inteligência de enxames: uma visão geral. Revista Controle e Automação, v. 20, n. 3, p. 271-304, 2009.

NETO, H. P. B.; COSTA, J. A.; SILVEIRA, E. S. Processamento paralelo com openmp em um simulador dinâmico de linhas de ancoragem e risers, parte ii. vol, v. 29, p. 4, 2010.

ABC, Intelligent Systems Research Group, Department of Computer Engineering, Erciyes University, Turkiye, 2009.

References