Otimização do algoritmo da colônia de formigas utilizando CUDA e algoritmos genéticos

Luís G. P. Lauck Marcus V. da Silva Luiz Gonzaga Jr. João Ricardo Bittencourt

Universidade do Vale do Rio dos Sinos - UNISINOS, Curso de Jogos Digitais, Brasil

Resumo

Este artigo apresenta uma comparação, em termos de performance e de implementação, do algoritmo da colônia de formigas rodando em CPU e uma versão adaptada rodando em uma GPU através da tecnologia CUDA. O objetivo é a exploração de paralelização de soluções no contexto de jogos digitais com ênfase no ganho de desempenho. Para isso, foram realizadas as duas implementações e comparados os resultados de suas respectivas execuções e aplicação da técnica de inteligência artificial de algoritmos genéticos para a solução de um dado problema.

Palavras-chave: computação paralela, inteligência artificial, implementação em GPU, algoritmos genéticos

1. Introdução

O algoritmo da colônia de formigas já é muito conhecido por sua capacidade de solucionar problemas de caminhamentos [Dorigo 2004]. Entretanto, quando aliado com a necessidade de múltiplas variáveis de entrada, em problemas de maior complexidade, torna-se difícil a parametrização manual. Para solucionar esta questão surge o uso dos algoritmos genéticos, que permitem uma evolução computacional da própria parametrização, na busca pela solução ideal [Scwhab 2004]. Este processo tende a ser demasiadamente custoso, portanto a utilização de paralelismo computacional torna-se bastante vantajosa.

Com o grande potencial de processamento paralelo disponibilizado pelas atuais GPU's e suas tecnologias, torna-se evidente a utilização deste recurso na solução de problemas paralelizáveis como no caso abordado neste artigo, além de ainda revelar uma enorme gama de novas aplicações principalmente no contexto de jogos digitais, para fins de processamento de imagens, cálculos de transformação e cálculos de detecção de colisões, entre outros [Mitchell 1997].

Neste artigo, apresentaremos um estudo inicial sobre duas abordagens algorítmicas da colônia de formigas utilizando algoritmos genéticos para busca de uma solução ótima, visando a diferenciação de desempenho e de implementação entre CPU e GPU, a fim de observar as vantagens e desvantagens de se trabalhar com paralelização e ainda demonstrar os possíveis ganhos de performance com uma implementação adaptada ao paradigma de paralelismo do CUDA [nVidia 2013]. O artigo está organizado da seguinte maneira. Na seção 2 identificamos o atual contexto da utilização de GPU's em desenvolvimento de jogos digitais. Na Seção 3, damos uma visão geral do algoritmo da colônia de formigas e de seus componentes. Na Seção 4 e na Seção 5, detalhamos, respectivamente, os parâmetros de implementação específicos para CPU e para GPU. A Seção 6 apresenta uma discussão sobre os testes realizados e resultados obtidos. A Seção 7 conclui o artigo.

2. O cenário atual

Atualmente no cenário de desenvolvimento de jogos, apesar da grande utilização de GPU para renderização, há um sub-aproveitamento dos múltiplos núcleos disponíveis para processamento. Dos muitos outros cálculos de grande complexidade necessários para a completa simulação de um frame de um jogo atual, apenas a física já é realmente explorada dentro de um ambiente paralelizável, através da tecnologia PhysX, assim como o CUDA, também da nVidia. Entretanto, itens como a inteligência artificial ainda são muito restritos ao uso de CPU, o que pode ocasionar um certo gargalo em relação as demais etapas de processamento. Sendo esta uma tarefa de grande impacto computacional e de uma necessidade de utilização de muitos agentes simultâneos, é visível que soluções de IA utilizando GPU têm muito a agregar ao desenvolvimento de jogos, devendo ser melhor exploradas.

3. Implementação do algoritmo

A implementação consiste basicamente de uma estrutura onde existem colônias de formigas e um ambiente, representado por um grafo que é gerado apenas uma vez e reutilizado por cada colônia. A cada geração, um número de colônias com determinado número de formigas é criado, e então é feita a simulação deste grupo de indivíduos, onde cada um busca individualmente por um caminho até a comida. Caso o indivíduo encontre a comida, irá gerar uma trilha de feromônio no caminho percorrido, caso contrário ela chegará a uma solução inútil e não deixará rastros para que outros indivíduos não sejam induzidos a seguir este mesmo caminho. Após simular todas as colônias de uma geração, é feita a etapa da evolução, que consiste na avaliação das colônias e posterior escolha das mais eficientes para serem mantidas na próxima geração (em geral 10%), algumas descartadas (normalmente 40%), outras serão cruzadas para gerar novas formigas (em geral 70% das melhores restantes), além da possibilidade de mutações (menos de 1% de chance por atributo). Com a junção sequencial destas etapas será criada uma nova geração

de colônias, que então serão simuladas. O número de gerações é determinado por um parâmetro, permitindo definir quantas vezes será executado o cálculo de seleção natural. Ao final da simulação, gera-se um arquivo contendo todos os dados gerados durante o processo, permitindo uma clara avaliação do resultado.

3.1 Colônia

Contém todos os dados da simulação, sendo composta por uma referencia ao grafo e um conjunto de formigas de mesma carga genética.

3.2 Grafo

O grafo é o ambiente pelo qual as formigas andam, ele é composto basicamente por nodos e arestas, sendo as arestas os elos entre os nodos. Além disso, possui uma posição inicial definida e múltiplos objetivos. Como ele é gerado a partir de um gerador de números pseudorrandômicos uniformes, há uma boa variação na distribuição dos nodos dentro do espaço e garante-se que ele seja gerado sempre de maneira coerente dado uma mesma semente inicial. A Figura 1 demonstra um grafo gerado de forma randômica.

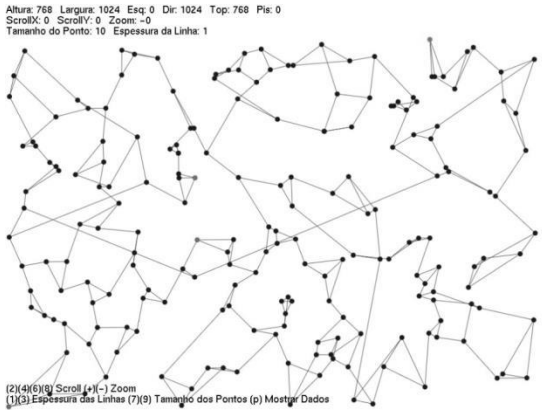

Figura 1: Grafo gerado de forma uniformemente randômica

3.3 Formiga

A formiga é um dos principais componentes da simulação, ela irá evoluir na busca do cromossomo mais eficiente. Dentre suas atribuições estão a decisão de qual aresta seguir, baseada em suas características de exploração, e a atribuição de feromônio aos nodos do caminho percorrido caso encontre um objetivo, sem nunca repetir um nodo durante o seu caminhamento.

3.3.1 Feromônio

O feromônio, em termos algoritmícos, nada mais é do que um valor que irá definir o grau de interesse de uma formiga sobre determinada aresta. Quanto maior o grau de feromônio, maior a atração da formiga por este caminho. Quando uma formiga encontra seu objetivo, será calculado o caminho percorrido por ela e, então, o feromônio que ela possui será distribuído uniformemente pela distância percorrida. Desta forma, arestas maiores irão possuir uma maior quantidade de feromônio que as menores. Cada formiga possui um valor de 0 a 1 para indicar sua capacidade de feromônio, mas ao longo da execução, haverá o acúmulo causado por conta das diversas formigas que irão percorrer o caminho.

3.4 Gerações

Uma geração corresponde a um conjunto de colônias que irão coexistir no mesmo período de tempo.

3.5 Seleção natural

A seleção natural é o procedimento que permite que as formigas evoluam e se tornem mais eficientes. O processo consiste primeiramente em selecionar aquelas colônias que tiveram o melhor desempenho na geração, por meio da função de fitness. A segunda etapa é a do acasalamento entre colônias, criando duas colônias para cada casal ou mesmo clonando-as de acordo com a probabilidade de cross-over (cruzamento entre colônias). Por fim, é possível que ocorram mutações, assim como na teoria evolucionista, isto permite que haja uma maior diversidade de indivíduos.

3.5.1 Função de fitness

A função de fitness consiste do processo de rankear os indivíduos que foram mais bem sucedidos na busca pelo objetivo, tornando-os mais indicados para participarem do processo evolutivo [Bourg 2004]. No algoritmo implementado o grau de sucesso de uma colônia é definido pela quantidade de objetivos encontrados no grafo, seguido do número de passos necessários para isso, resultando na eficiência.

4. Implementação para simulação em CPU

O algoritmo para a CPU é implementado de forma que cada colônia seja executada serialmente. Após uma geração ter sido processada, cria-se uma nova geração que irá também ser processada de maneira serial. A estrutura básica para esta operação é um loop comum com uma condição de parada, conforme a quantidade de gerações a serem simuladas. A implementação para CPU difere bastante da feita em GPU no que diz respeito ao tipo de estrutura para armazenamento, já que a GPU não oferece suporte para certas estruturas de dados comumente utilizadas no contexto de CPU. Desconsiderando a forma sequencial ou paralela da simulação, o algoritmo é essencialmente o mesmo, apenas com algumas adaptações.

5. Implementação para simulação em GPU

A implementação do algoritmo para a GPU é pensada de forma a paralelizar a simulação das colônias, onde

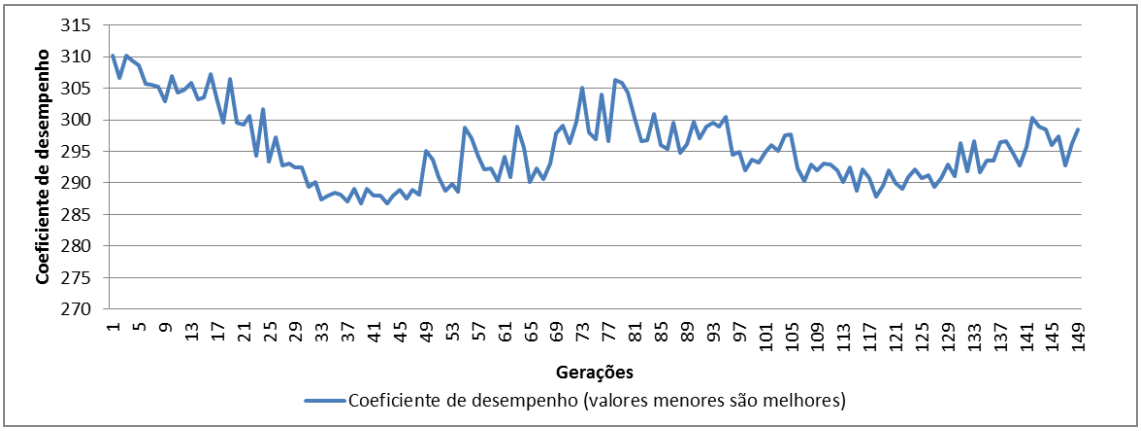

Figura 2: Gráfico de desempenho de várias gerações sucessivas de colônias

se espera ganhar em desempenho sobre o algoritmo para CPU. Existem outras abordagens, como no caso da paralelização do caminhamento de cada formiga da colônia, entretanto para este caso, que também se utiliza de algoritmos genéticos, passa a ser mais interessante paralelizar diferentes colônias [Tantawy 2011]. Entretanto, as gerações não podem ser paralelizadas, pois são dependentes das anteriores, assim como as próprias formigas. A lógica de cada colônia é executada de forma sequencial dentro do kernel (trecho de código executado pela GPU), de forma que é distribuída entre diferentes blocos e threads para processamento. O modelo de algoritmo adotado para simulação em GPU segue, basicamente, a seguinte estrutura: geração de colônias, envio dos dados para GPU, processamento paralelo dos dados em GPU, captação e análise dos dados gerados pela GPU, e finalmente a geração de colônias novamente, repetindo este ciclo até a condição de parada definida pelo número de gerações. A plataforma utilizada para implementação do algoritmo de processamento paralelo em GPU foi o CUDA (Compute Unified Device Architecture), disponibilizado por um dispositivo de hardware gráfico da NVIDIA.

6. Resultados

Para a realização dos testes foi utilizado um computador equipado com um processador Intel Core i5-750 a 2.66GHz, 2x4GB de memórias RAM a 1333MHz em dual-channel e uma GPU Nvidia GTX560Ti com 1024MB de RAM. Todos os testes foram realizados três vezes seguidas, tanto para CPU quanto para GPU, e o resultado final foi definido pela média dos três valores resultantes de tempo gasto para a simulação de todo o processo e em seguida somente considerando o tempo gasto para a simulação das colônias.

Os testes de desempenho foram realizados com um número de 50 formigas por colônia e rodados com apenas 1 geração por vez, pois tanto o número de formigas quanto o número de gerações afetam o tempo de simulação total e das colônias de forma linear por não serem paralelizáveis, e portanto a variação destes parâmetros não é visada para fins de testes comparativos. O ambiente utilizado foi composto por um grafo de 200 nodos interligados entre si por no mínimo 3 arestas por nodo, onde um dos nodos era o ninho e 5 nodos eram comidas. Os resultados obtidos foram satisfatórios no que diz respeito à simulação das colônias, pois em ambos os casos as simulações foram bem sucedidas, porém o desempenho geral (tempo total) da simulação da GPU em relação à CPU não ficou dentro do esperado, como pode ser observado na Tabela 1. O desempenho geral da GPU acabou ficando muito prejudicado por conta da constante alocação e desalocação de memória, isto porque a cada nova geração era necessário resgatar os dados de dentro da GPU para que se fizesse a análise sobre eles, e depois de geradas novas colônias se fazia necessário o reenviode vários dados. O processamento de cada tarefa, depois de já estar na GPU, foi muito positivo à medida que o número de colônias por geração foi aumentando, pois teve uma grande vantagem em relação à CPU por tratar os dados paralelamente, como se pode ver no gráfico representado na Figura 3.

Tabela 1: Comparação de tempo de simulação total e das colônias entre GPU e CPU

Colônias por geração	Simulação em CPU		Simulação em GPU	
	Total (ms)	Colônias (ms)	Total (ms)	Colônias (ms)
500	175,4	162,3	23378,2	327,1
1000	343,2	323,9	45807,0	555,9
1500	511,8	486,6	67677,5	572,5
2000	680,8	648,2	90431,3	573,1
2500	851,3	811,8	111816,3	590,6
3000	1021,0	974,1	142644,0	717,2

No caso da CPU tudo correu dentro do que era esperado e até um pouco além, pois acabou superando a GPU no geral. Certamente isso se deve à implementação da parte do algoritmo relacionado à manipulação da memória, como já foi destacado. O maior problema relacionado à GPU foi a necessidade de realizar a alocação dinâmica dos dados, o que implicou realizar múltiplas chamadas para que se alocasse o espaço necessário.

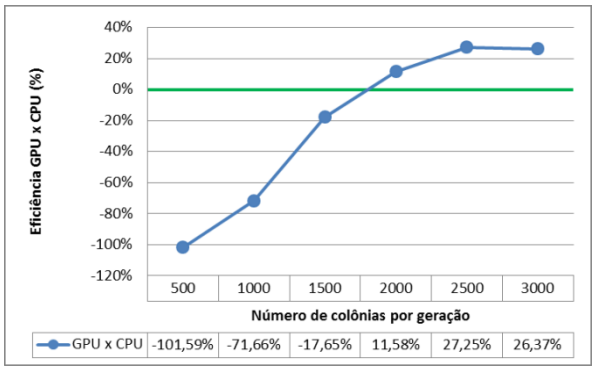

Figura 3: Gráfico de desempenho da GPU em relação à CPU

Sobre o algoritmo genético em si, pôde-se observar o correto funcionamento da implementação tanto em GPU quanto em CPU, porém não se conseguiu observar de maneira conclusiva a convergência em direção a um resultado ótimo em todos os casos testados. Conclui-se que isso se deve à grande quantidade de parâmetros configuráveis, fazendo-se necessário, neste caso, sucessivos testes de ajuste dos parâmetros até que se consiga direcionar os resultados em uma direção mais favorável. Outro empecilho foi o modelo de algoritmo adotado, que ao longo do tempo armazena os dados obtidos em memória para a posterior geração de um relatório com os dados produzidos pela simulação, fazendo com que após certa quantidade de gerações (e levando-se também em consideração o número de colônias por geração e de formigas por colônia) a memória se torne insuficiente para o armazenamento destes dados, portanto não foi possível executar testes com um número muito grande de gerações. A Figura 2 mostra um dos casos em que se obteve uma boa convergência em direção a um bom resultado por volta da geração 39 com um coeficiente de desempenho de 286.69 (quanto menor melhor), de modo que para este teste utilizou-se uma taxa de mutação de 0.8%, cross-over de 70%, elitismo de 10% e descarte 40%, para 1000 formigas por colônia, 50 colônias por geração e 150 gerações.

7. Conclusão

A utilização de GPU's para processamento de dados certamente pode trazer resultados extremamente satisfatórios em relação às CPU's, porém o maior de todos os obstáculos é a adaptação a um novo paradigma de programação. A programação sequencial é algo muito mais natural e lógico para se tratar, porém o alto poder das GPU's não pode de modo algum ser ignorado. Sem dúvidas tal atividade foi importante para servir de introdução a este novo mundo de possibilidades que o processamento em GPU disponibiliza, tal e qual também foi imposta pelo desafio de conseguir unir uma técnica de IA com programação paralela. Fica muito evidente a importância do estudo mais aprofundado da técnica de paralelização antes de aplica-la de forma prática em outras soluções, para que se possa introduzir de maneira eficiente dentro de aplicações interativas como jogos, por exemplo, visto que todo o planejamento deve ser feito com maior cuidado a fim de evitar gargalos como os experimentados. Como aprendizado e trabalhos futuros fica a necessidade de criação de uma estruturação algorítmica mais eficiente para se trabalhar com IA dentro do contexto da GPU, assim como uma reestruturação do modelo de paralelização adotado, já que o modelo escolhido para a solução do problema apresentado se mostrou falho em alguns pontos. Nas próximas revisões do algoritmo devemos evitar ao máximo o trânsito constante de informações entre GPU e CPU, visando o aproveitamento de desempenho máximo da combinação entre os dois.

Referências

MITCHELL, T.M., 1997. Machine Learning. McGraw-Hill.

SCHWAB, B., 2004. AI Game Engine Programming. Hingham: Charles River Media.

BOURG, D.M. ESEEMAN, G., 2004. AI for Game Developers. O'Reilly:

NVIDIA, 2013. *CUDA Parallel Computing* [online] nVidia. Disponível em: http://www nvidia.in/object/cuda-parallel-computing-in html [Acessado em 25 de novembro de 2013].

NVIDIA, 2013 *CUDA C Programming Guide*[online] nVidia. Disponível em: http://docs.nvidia.com/cuda/cuda-c-programming-guide [Acessado em 20 de dezembro de 2013].

COOK, S., 2013. CUDA Programming: A Developer's Guide to Parallel Computing with GPU's. Waltham: Morgan Kaufmann.

TANTAWY, K., 2011. *Ant Colony Optimization Parallel Algorithm for GPU*. Honours project, Carleton University.

DORIGO, M. ESTÜTZLE, T., 2004 Ant Colony Optimization. Cambridge: The MIT Press.