**Aplicação de Processamento Paralelo para Games**

**Marcelo Venerato Ponciano**

Curso de Bacharelado em Ciência da Computação – Universidade Feevale – Campus II (51) 3586-8822 – Novo Hamburgo – RS– Brasil

[sorcerer.kletian@gmail.com](mailto:sorcerer.kletian@gmail.com)

**Abstract.** This paper does a brief analysis on the processes to renderize a game on the screen, and how to use parallel processing to increase it’s performance. The objective is to make an overview on how to use parallel processing to separate processes and make use of multi-core tecnology. Improvements are still being made in how today’s Engines work with processing separated models, but new studies are made on how to increase it’s model’s performances.

**Resumo.** Esse artigo faz uma breve análise dos procedimentos para se renderizar um jogo em tela, e de que maneiras possa ser possível utilizar processamento paralelo para melhorar o desempenho atual. O objetivo é estudar maneiras de se utilizar processamento paralelo para separar procedimentos e fazer uso das tecnologias atuais de múltiplos processadores. Ainda estão sendo feitas melhorias nos sistemas atuais de processamento por módulos separados usados pelas Engines atuais, mas cada vez surgem trabalhos com modelos de melhor desempenho.

**1.Introdução**

Com a tecnologia atingindo uma barreira de melhoria de hardware na última década, o que impossibilitou que os hardwares se tornassem por si sós mais rápidos do que já são, as empresas passaram a trabalhar com a opção de múltiplos núcleos em processadores, fazendo assim com que o processamento fosse dobrado, mesmo que o limite da lei de Moore fosse atingido.

Contudo, a menos que os programadores passem a aplicar os conceitos de programação paralela para fazer uso dos múltiplos cores, o desempenho geral não será atingido, especialmente no caso de aplicações que requerem maior quantidade de processamento, como é o caso dos jogos, em que é necessário que milhares de cálculos e verificações sejam feitas em apenas 1 frame.

Durante a escrita desse artigo, foram encontrados trabalhos interessantes sobre programação paralela voltados para a área de jogos, além de trabalhos específicos para o controle e verificação de colisões de grandes quantidades de dados fazendo uso de programação paralela. Um dos artigos encontrados que teve grande valor foi “*Parallelizing a Real-Time Steering Simulation for Computer Games with Open MP”*, trabalho rico em informações sobre uma implementação de programação paralela utilizando *OpenSteerDemo*, uma mesa de testes para a biblioteca open-source *OpenSteer* para C++.

No decorrer desse artigo darei farei uma breve introdução do método de processamento de instruções em um jogo, e exemplos de instruções em que a programação paralela poderia ser aplicada, de forma a melhorar o desempenho, na seção 2. Na seção 3 será apresentado um exemplo com estudo do código fonte de um jogo em que determinados processos podem ser separados para serem executados de forma concorrente. A seção 4 define alguns dos problemas a serem encontrados ao fazer a implementação de programação paralela em determinadas instruções, e por fim na seção 5 apresentará comentários acerca do assunto estudado.

**2. O funcionamento de um jogo**

Jogos são softwares de alta complexidade geralmente com diversas funções e cálculos sendo executados constantemente. Em um jogo, toda a renderização da tela é feita em uma função em loop, que repete as mesmas ações a cada frame. Caso seja necessário que muitas operações sejam feitas em um mesmo frame, o peso no processamento pode fazer com que a taxa de frames da aplicação seja reduzida para que seja feita a sincronização de todas as operações.

O exemplo abaixo faz uso de um código de um jogo feito utilizando a ferramenta *Processing 3.3*. Um sketch em *Processing* possui dois métodos principais, *setup()*, e *draw()*, sendo respectivamente responsáveis pela inicialização de variáveis e carregamento de dados, e o loop que vai desenhar a tela a cada frame.

Uma imagem contendo captura de tela

Descrição gerada automaticamente

**Figura 1. Métodos padrão *setup()* e *draw()* implementadas em um sketch Processing.**

**Fonte: O autor, 2019**

Como dito anteriormente, o método *draw()* vai atualizar a tela com todas as informações contidas a cada frame. Sendo assim, de acordo com o exemplo acima, a cada frame será executado, nessa ordem, o método *ship.Update()*, *ship.setRotate()*, a verificação da tecla pressionada para definir se um objeto *Bullet* será instanciado, e por fim, chamará os métodos *SetRotate()*, e *AsteroidMove()* para cada objeto na lista asteroids.

A forma como essas operações são executadas segue o esquema:

Uma imagem contendo equipamentos eletrônicos

Descrição gerada automaticamente

**Figura 2. Original Cocos2d-x update loop.**

**Fonte: Kim Suntae - Multi-Threaded Parallel Processing for Physics Simulation in Cocos2d-x, 2016**

Assim, a imagem do frame apenas será renderizada após a leitura da entrada, da leitura das operações lógicas do jogo, e dos cálculos da física aplicada ao jogo, para então repetir o mesmo processo para o próximo frame, e assim por diante. Nesse tipo de modelo single-threaded, caso hajam muitas operações a serem calculadas na física do jogo, será necessário esperar pela resposta de todos os cálculos para que a tela seja renderizada no frame correspondente. Isso acaba causando quedas na taxa de quadros por segundo, e gerando atrasos na leitura de dados do usuário.

**3. Separando processos para execução paralela**

Como mostrado na seção 2, o problema de jogos que fazem uso de single-thread ocorre quando muitas operações devem ser executadas, gerando assim atraso na renderização, e queda na taxa de quadros. Para evitar esse tipo de quadro, pode-se desmembrar o jogo em passos menores, e quando possível, executá-los de forma paralela, enviando assim apenas o resultado final para a main thread, que poderá renderizar o quadro assim que obtiver todos os dados necessários.

Por exemplo, no código apresentado acima, podemos separar os métodos *ship.SetRotate()* e *asteroid.AsteroidMove()* para ser executado em uma thread separada. Não apenas isso, comumente é visto em jogos a programação em paralelo de toda a física de movimentação e controle de colisão feita em threads separadas da thread principal, esta apenas recebendo o retorno quando pronto. Estudando o caso apresentado de maneira simplista, uma forma de se adquirir o paralelismo desejado seria fazer uso de uma classe que criasse Threads recebendo os objetos e seus métodos como parâmetro, tal qual o exemplo apresentado na figura 3.

Uma imagem contendo captura de tela

Descrição gerada automaticamente

**Figura 3. Classe ThreadGenerator para executar métodos de objetos de classe.**

**Fonte: O autor, 2019.**

Através da criação do método gerador de threads por objeto, seria possível, em tese, criar instâncias de threads separadas da thread principal para que os cálculos e outras operações referentes a movimentação ou controle de colisão, ou mesmo de carregamento de recursos, fosse feita em separado. Contudo, problemas podem ser identificados no modelo citado, os quais podem ser vistos na próxima seção.

**4. Problemas encontrados ao fazer uso de programação paralela**

Como visto na seção 3, a criação de uma maneira de gerenciar a criação de threads de processos poderia ser utilizada para atender a necessidade do paralelismo. Contudo, problemas podem surgir usando uma aproximação tão simplista. Não bastando apenas a dificuldade de implementação de um sistema que funcione utilizando paralelismo, a grande parte dos problemas se encontra na dificuldade de manter a sincronização com a thread principal, uma vez que, por mais que os processos sejam feitos de forma paralela para melhor desempenho, o frame na tela não pode ser renderizado até que todos os dados pertinentes sejam retornados.

Conforme já mostrado na Figura 1, a ordem com que os passos devem ser realizados é, obrigatoriamente, a entrada de dados, a lógica do jogo, a física do jogo, e por fim a renderização. Não há como renderizar uma cena sem que todos esses passos tenham sido feitos. O processamento em threads resolveria a questão de desempenho, mas causaria o problema de perda da ordem cronológica de operações.

**5. Soluções para problema de sincronização**

Uma vez que a sincronização é um problema encontrado ao se trabalhar com paralelismo, e que jogos precisam, obrigatoriamente ser sincronizados para que a tela seja renderizada da maneira correta, Engines de jogos costumam fazer uso de uma técnica de utilizar um campo de simulações para realizar os cálculos necessários, e com todos os resultados, retornar os valores para a thread principal.

Esse comportamento é visto na *Cocos Engine*, em que uma thread com um simulador permanece sendo executada o tempo todo calculando os dados necessários baseando-se no estado atual dos elementos do jogo. A Figura 4 mostra um esboço de como funciona a *Cocos Engine* ao fazer uso de uma thread de simulação de física.

Uma imagem contendo captura de tela

Descrição gerada automaticamente

**Figura 4. Original Cocos2d-x update loop.**

**Fonte: Kim Suntae - Multi-Threaded Parallel Processing for Physics Simulation in Cocos2d-x, 2016**

Esse método de um campo de simulação também foi proposto por *Ali MOHEBALI* e *Thiam Kian CHIEW*, no tabalho *“Redefining Game Engine Architecture through Concurrency”*, em 2014. Na arquitetura proposta por Mohebali e Chiew, um modelo de separação entre dados e operações foi utilizado. Tal modelo foi desenhado baseado no conceito de redundância de dados/servidor, onde os módulos da Engine vão manter uma cópia local dos dados compartilhados, ao invés de fazer acesso direto a memória, removendo assim o uso de métodos de lock.

**6. Considerações**

Embora a capacidade de processamento atual de cada core de um processador seja muito maior se comparados aos da última década, a menos que a arquitetura de desenvolvimento seja alterada para fazer uso de toda a capacidade proporcionada por múltiplos núcleos, através de processamento paralelo, ainda teremos enormes barreiras nos impedindo de seguir adiante na qualidade de processamento de dados.

Métodos como os propostos por Mohebali e Chiew, e o utilizado na *Cocos Engine* surgem para tentar resolver a questão de sincronização sem, ou com perda menor de desempenho, mas ainda existe um longo caminho até que seja possível haver uma separação completa entre todos os módulos e operações para que o desenvolvimento de jogos possa ser feito com paralelismo o suficiente para que seja possível usar toda a capacidade proporcionada por processadores com múltiplos núcleos.

**Referências**

Ali MOHEBALI, Thiam Kian CHIEW (2014) “Redefining Game Engine Architecture through Concurrency”.

Ting-Hsuan Chien, Jhe-Wei Lin, Rong-Guey Chang (2018) “Parallel Collision Detection with OpenMP”.

Figueiredo, M. e Fernando, T. (2004) “An Efficient Parallel Collision Detection Algorithm for Virtual Prototype Environments”.

Knafla, B. e Leopold, C. (2007) “Parallelizing a Real-Time Steering Simulation for Computer Games with OpenMP”.

Suntae, K. (2016), “Multi-Threaded Parallel Processing for Physics Simulation in Cocos2d-x”, https://engineering.linecorp.com/en/blog/multi-threaded-parallel-processing-for-physics-simulation-in-cocos2d-x/