7 Processamento Paralelo

Yes, of course, who has time? Who has time? But then if we do not ever take time, how can we ever have time? (The Matrix)

7.1 Introdução – Classificação de Sistemas Paralelos

Diversas aplicações procuram lançar mão de multi-processamento para obter melhor performance, a começar pelos próprios sistemas operacionais ou serviços de busca para Internet. Os objetivos do paralelismo podem ser categorizados como:

- Permitir que sistemas munidos de mais de um processador possam acelerar a resolução de um problema de natureza paralelizável;
- Poder executar vários processos simultaneamente (define-se neste caso, segundo Carissimi (2001), que processo é um programa em execução)
- Otimizar o tempo ocioso (*idle time*) das CPU's;

A divisão de processos pode ocorrer tanto ao nível de várias aplicações independentes sendo executadas simultaneamente, bem como uma mesma aplicação dividida em vários *threads* (*thread level paralelism* -TLP) (Marr, 2002).

De acordo com Andrews (2000), toda aplicação paralela pode ser categorizada como sendo de uma das seguintes formas:

- Sistema *Multi-Threaded*: um programa que contém mais processos do que o número de processadores disponíveis. Estes sistemas têm como objetivo gerenciar múltiplas tarefas independentes. Um sistema operacional se encontra dentro desta categoria;
- Sistema Distribuído: vários processos são executados em várias máquinas, conectadas entre si de alguma forma. Estes processos se comunicam mutuamente através de mensagens;

- Computação Paralela: Um programa se divide em vários processos para resolver o mesmo problema num tempo menor. Geralmente haverá o mesmo número de processos quanto o de processadores.

Também pode-se categorizar uma aplicação paralela de acordo com o modelo de programação adotado (Andrews, 2000):

- Variáveis compartilhadas: os processos trocam informações entre si através de uma memória compartilhada, numa área comum a todos;
- Troca de mensagens: Não há disponibilidade de uma área comum de memória, ou isto não convém para a aplicação (são máquinas diferentes e separadas, por exemplo). Para tanto, a comunicação entre os processos é feita através de envio e recebimento de mensagens;
- Dados Paralelos: cada processo executa as mesmas operações sobre diferentes partes do dado distribuído. Este modelo também é conhecido como *Single Instruction Multiple Data* (*SIMD*).

Em aplicações de visualização para tempo real, são inúmeros os trabalhos que paralelizam as tarefas. Merecem destaque (Kempf 1998, Igehy 1998).

7.2 *Multi-threading* e *Hyper-threading*

Um *thread* pode ser entendido também como um processo, que pode pertencer a uma aplicação ou a aplicações completamente independentes. Sistemas com apenas um processador podem gerenciar vários *threads* simultaneamente através de métodos de *Time Slice*, concedendo para cada processo uma fatia de tempo da CPU, sendo que este tempo não precisa ser necessariamente o mesmo para cada um. Além disso, cada processo pode conter um nível de prioridade: *threads* com alta prioridade podem interromper *threads* com menor prioridade. Entretanto, por melhor que seja o gerenciamento dos *threads* por parte da aplicação ou do sistema operacional, o *Time Slice* sempre pode trazer latências, devido a compartilhamento de *cache* ou dificuldade de gerenciamento entre o trânsito de processos.

Há muitos anos têm-se adotado paralelismo para gerenciamento de vários processos. Entretanto, esta solução costuma ser cara e requer equipamentos

especiais, como *clusters* de CPU's, por exemplo, não sendo adequados para aplicações de usuários padrões, tais como jogos.

O *hyper-threading* é uma tecnologia que permite que um único processador possa ser visto por uma aplicação ou sistema operacional como dois processadores separados. Isto é possível, devido à arquitetura interna da CPU, que é composta por dois processadores lógicos. Do ponto de vista do *software* equivale a poder desenvolver uma aplicação para um sistema composto por mais de um processador e do ponto de vista de *hardware*, significa que instruções são alocadas para partes diferentes da CPU e são executadas individual e simultaneamente. Cada um destes pode usufruir das suas próprias interrupções, pausas, finalizações, etc. A nível de *hardware*, no entanto, a CPU consiste em dois processadores lógicos, mas que diferentemente do que processadores duais, compartilham uma série de componentes: os recursos do *core* do processador, *cache*, barramento de interface, *firmware*, etc. A arquitetura interna de *hyper-threading* permite que haja um ganho de até 30% em relação a sistemas de multi-processamento padrões, de acordo com INTEL (2001).

A INTEL implementou esta tecnologia de *hyper-threading* para servidores da série XEON e mais recentemente para a nova linha de processadores Pentium IV, sendo que já pode ser considerada uma tecnologia barata e acessível. Como este trabalho visa sobretudo aplicações de jogos 3D, que requer tipicamente recursos populares, escolheu-se esta arquitetura para realizar a implementação do paralelismo.

7.3 Paralelismo em pipelines de visualização tempo real

De acordo com Akenine-Möller (2002), um sistema paralelo de visualização em tempo real pode ser dividido em dois métodos: paralelismo temporal e paralelismo espacial.

O paralelismo temporal consiste em modelar o problema semelhantemente a uma linha de produção: cada processo possui a capacidade de resolver uma parte do problema. Uma vez terminada a sua parte, o processo envia os resultados para o processo seguinte e fica disponível para receber novos dados. Neste modelo, pode-se denominar cada processo como sendo um estágio do problema. A resolução completa do problema consiste em percorrer por todos os estágios. O

gargalo, neste caso, consiste no estágio que leva mais tempo para ser processado. A configuração ideal para este sistema é ter um processador para cada estágio.

Já o modelo de paralelismo espacial consiste em criar vários processos capazes de fazer a mesma coisa: ganha-se performance distribuindo partes menores do problema para cada processo. Deve-se observar que neste modelo apenas se podem implementar problemas que sejam de natureza paralelizável. Surge neste sistema um problema extra: o programa deve implementar uma função que se encarregua de juntar todos os dados, após terem sido devolvidos por cada processo. Em muitos casos, esta etapa pode inviabilizar a abordagem paralela, pois faz com que se gaste um tempo adicional que antes não era necessário.

7.4 Paralelismo e os Impostores com relevo

Neste trabalho, ressalta-se que a etapa de *pre-warping* envolvida num sistema de *ibr* é apropriada para um estágio que é processado pela CPU e que a etapa de texturização é um estágio para a GPU. Tendo em conta que os impostores com relevo constituem apenas parte da modelagem da cena e que a outra parte é descrita através de modelagem geométrica convencional, desenvolveu-se um *framework* que encara a CPU e a GPU como dois processadores separados e paralelos. Além disso, lançando-se mão de um processador com *hyper-threading*, vê-se a CPU como uma máquina paralela por si só.

Inicialmente, o sistema modelado é do tipo variáveis compartilhadas: a CPU e a GPU disputam os mesmos recursos. A CPU cuida do estágio de aplicação (*framework*) e a GPU utiliza estes resultados para realizar os estágios de geometria e rasterização. Há uma dependência temporal entre ambos, pois a GPU necessita dos resultados da CPU para seguir adiante, como se pode ver na figura 7.1. Desta maneira, diz-se que o sistema segue o modelo de paralelismo temporal.

Para realizar o *pre-warping* cria-se um *thread*. Este processo pode correr em paralelo ao estágio de aplicação da CPU, pois não há dependência de dados. Assim, o *thread* de *pre-warping* é visto como um sistema de paralelismo espacial em relação ao processo do *framework*.

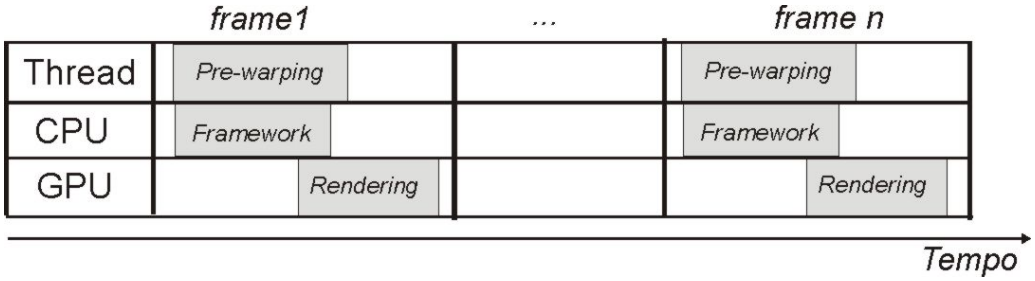

Figura 7.1 – Enquanto o processo de rendering da GPU é do tipo de paralelismo temporal em relação ao *framework*, o *thread* de *pre-warping* é do tipo de paralelismo espacial em relação ao mesmo.

Na modelagem de objetos por impostores com relevo torna-se necessário mais um processo, que é o responsável por gerar dinamicamente novas texturas com relevo. Como o processo de *pre-warping* precisa destas texturas para ser inicializado, o seu *thread* correspondente não pode ser disparado enquanto não receber o resultado da nova textura, já pronta. Surge assim um tempo de espera do sistema todo, uma vez que os estágios de geometria e de rasterização da GPU não podem dar inicio à visualização da textura enquanto este não lhe for enviado. A figura 7.2 ilustra esta situação. Vale ressaltar que este tempo de espera da GPU se dá apenas para a visualização do impostor com relevo, uma vez que os objetos geométricos da cena estão sendo tratados independentemente pelo *framework* e pela GPU.

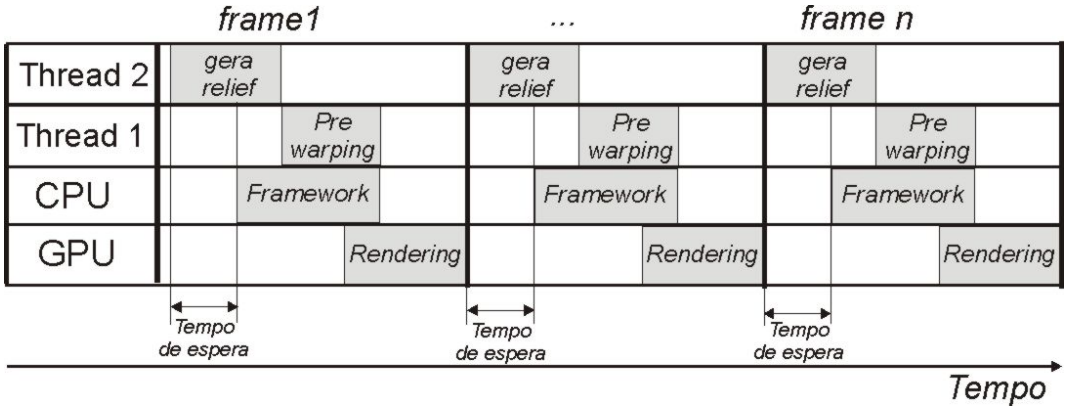

figura 7.2 – O processo responsável por gerar uma nova textura com relevo faz com que o tempo necessário para a visualização do impostor com relevo seja aumentado, devido à cadeia de dependências que se cria.

Para solucionar este problema, sugere-se fazer um sistema de previsão de texturas com relevo. A idéia é que, fazendo-se isto, quando a métrica de erro

indicar que uma textura com relevo caducou e uma nova textura se tornou necessária, há uma grande probabilidade desta já se encontrar pronta, evitando-se assim o tempo de espera que antes existia. Esta previsão pode ser feita através de uma inferência no vetor velocidade referente ao observador. Também se pode desenvolver uma estrutura de dados que armazena as texturas invalidadas num *cache*, para poderem ser utilizadas novamente, evitando-se assim um novo cálculo das mesmas. No capítulo 10, esta possibilidade é descrita com mais detalhes.

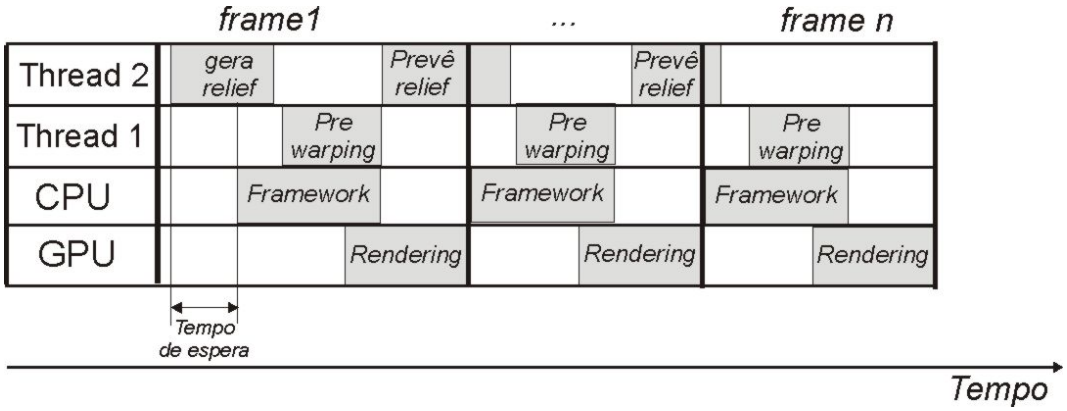

figura 7.3 - Um sistema de previsão é capaz de minimizar o tempo de espera de todo o *pipeline* para a visualização do impostor com relevo.

No capítulo seguinte discute-se com mais detalhes a arquitetura destes sistemas paralelos, bem como resultados obtidos por meio da sua implementação.