4
Visualização Distribuída

Uma das estratégias para o tratamento de problemas similares de visualização de modelos de larga escala tem sido empregar agrupamentos de PCs de prateleira (*off-the-shelf PC clusters*), onde os PCs são equipados com placas gráficas de alto desempenho e baixo custo (GPUs). O uso de um único PC equipado com múltiplas placas gráficas também mostrou ser outra opção atraente para a visualização de grandes modelos (35). Além disso, uma das tendências recentes na área de computação de alto desempenho consiste em mover computações pesadas para longe dos computadores pessoais (55, 57, 56), utilizando centros de processamento de dados bem equipados, que já são realidade na indústria de petróleo.

Neste capítulo é apresentado um sistema para a visualização distribuída de modelos massivos de reservatórios de petróleo. Nossa solução explora o poder de processamento agregado de um agrupamento de PCs, onde cada nó é equipado com múltiplas placas gráficas. O sistema foi pensado para permitir que clientes leves tirem proveito do sistema distribuído pela conexão ao agrupamento via redes corporativas convencionais. Como consequência, o usuário não precisa estar fisicamente próximo ao agrupamento de PCs, diferentemente de propostas anteriores.

O sistema distribuído proposto utiliza a estratégia de renderização paralela com ordenação no fim (37) e dá suporte a um rico conjunto de técnicas de visualização. Ao aliar o uso eficiente das GPUs disponíveis a uma implementação em forma de linha de produção (*pipeline*) e ao uso de composições parciais de imagem nos nós do cluster, nossa solução trata bem dos problemas de escalabilidade de sistemas com ordenação no fim em agrupamentos de médio a grande porte. Os resultados dos experimentos computacionais efetuados demonstram que a nossa solução permite a visualização de modelos de reservatório com dezenas de milhões de células com taxas interativas.

Os trabalhos anteriores indicam a estratégia de ordenação no fim como a estratégia mais apropriada para a renderização paralela de modelos muito grandes (37, 59, 13, 12, 35), principalmente porque o modelo é subdividido entre os processadores (nós do agrupamento). As outras duas estratégias, ordenação no início e ordenação no meio, requerem que o modelo inteiro esteja carregado em cada nó de renderização, o que não é viável para grandes modelos dadas as restrições do *hardware*.

Conforme apontado por Molnar et al. (37), apesar de escalável em termos

de tamanho de modelo, a escalabilidade de sistemas com ordenação no fim sofre pela necessidade de se lidar com uma quantidade crescente de imagens parciais geradas pelos nós de renderização. Essas imagens devem ser enviadas pela rede para a composição, seja pelo algoritmo de *Z-Buffer*, seja por composição alfa.

O restante desse capítulo apresenta a nossa proposta em detalhe: a arquitetura de *hardware* do sistema, um algoritmo simples para a partição do modelo e nossa linha de produção distribuída. É discutido como nossa proposta obtém escalabilidade, minimizando o impacto de ter que lidar com um número grande de imagens parciais.

4.1 Arquitetura de Hardware

Nosso sistema foi pensado para utilizar um agrupamento de PCs, onde cada um é equipado com um *bridge* PCI-express com múltiplas placas gráficas conectadas. Dessa forma, tiramos proveito da baixa razão de custo por banda de tais barramentos, conforme atestado por Marchesin et al. (35).

A Figura 4.1 ilustra a organização de *hardware* da nossa solução. Os usuários requisitando a visualização de grandes modelos podem conectar ao *nó mestre* do agrupamento. Esse nó coordena a renderização distribuída de quadros completos, que são então enviados ao *cliente* por meio de uma rede corporativa convencional. Os outros nós do agrupamento servem como *nós de renderização*. Os nós do agrupamento são interconectados por uma rede rápida. Cada PC servidor inicia, para cada GPU, uma linha de execução (*thread*) que controla seu *pipeline* gráfico. Os modelos de reservatório são enviados pelo cliente para o nó mestre, que o armazena em um sistema de arquivos compartilhado.

4.2 Partição do Modelo

O primeiro passo para se obter eficiência em uma visualização distribuída com ordenação no fim reside em particionar o modelo adequadamente entre os renderizadores. Assim que o modelo é armazenado no sistema de arquivos compartilhado, o nó mestre executa o algoritmo de partição.

Nós utilizamos uma estrutura de *KD-tree* para particionar recursivamente a caixa envolvente alinhada do modelo inteiro. Considerando que nc denota o número de células no (sub)modelo corrente e que n é o número de nós de renderização, as nc células são ordenadas de acordo com as coordenadas dos seus centróides ao longo do maior eixo da caixa envolvente. O (sub)modelo é então particionado em duas partes, onde a primeira parte é designada para

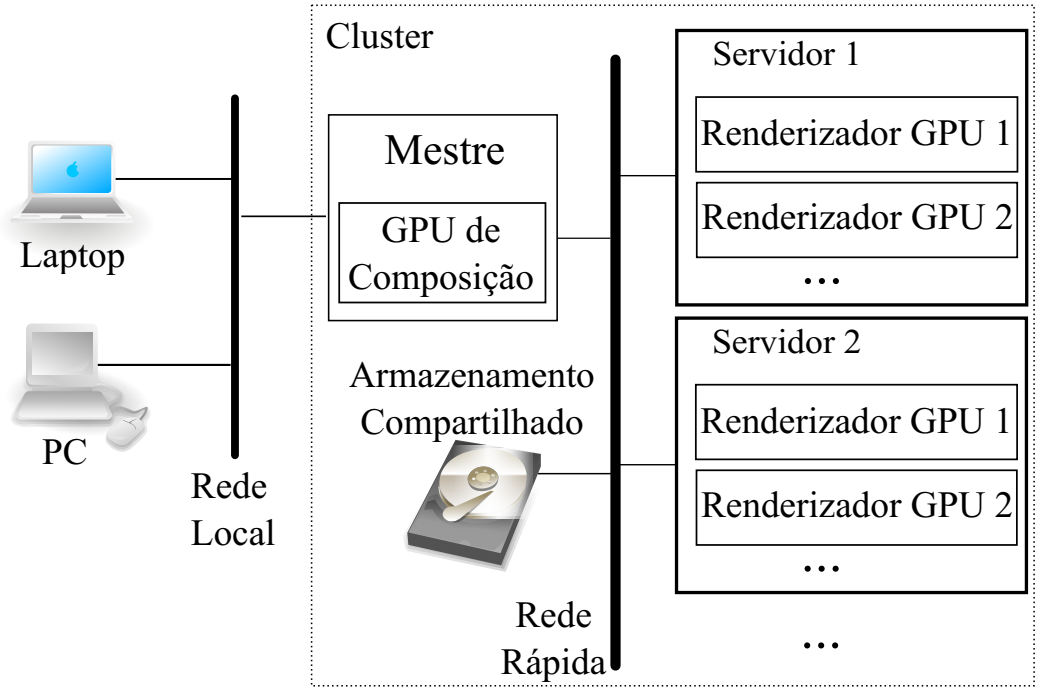


Figura 4.1: Organização de *hardware* da nossa solução.

os primeiros $\lfloor \frac{n}{2} \rfloor$ nós renderizadores e a segunda parte para os $n - \lfloor \frac{n}{2} \rfloor$ nós renderizadores restantes. Os números de GPUs no primeiro e segundo grupos são utilizados como pesos para posicionar o plano de partição: se o primeiro e segundo grupos possuem respectivamente g_1 e g_2 GPUs, então o plano de partição é posicionado sobre o centróide da célula de índice $nc * \frac{g1}{g1+g2}$ do *array* ordenado de células. As células cruzando o plano de partição são designadas para os dois grupos e planos de *clipping* são adicionados para restringir o desenho às caixas envolventes subdivididas.

Quando a recursão chega em um grupo com apenas um nó, as células a ele designadas são novamente subdivididas, dessa vez entre as GPUs do nó em questão. O mesmo procedimento se aplica, onde a primeira e a segunda parte são alocadas para as primeiras $\lfloor \frac{g}{2} \rfloor$ GPUs e as $g - \lfloor \frac{g}{2} \rfloor$ GPUs restantes respectivamente.

Esse procedimento resulta em uma partição bem balanceada em termos de números de células. A cobertura média de tela de cada partição também é reduzida, dadas as propriedades espaciais da subdivisão KD-tree. Isso reduz ambos o uso de largura de banda da rede e o custo de composição de imagens. O procedimento é simples e eficiente; no entanto, requer o armazenamento de todos os centróides na memória do nó mestre. Isso pode ser um fator impeditivo para modelos muito grandes, o que não é tratado neste trabalho.

Após o modelo ser particionado, as caixas envolventes de cada partição devem ser enviadas para os nó renderizadores relevantes. O nó carrega os

submodelos associados a cada das suas GPUs e então inicia as linhas de execução de renderização com a técnica de visualização desejada. As técnicas de visualização suportadas foram mencionadas no Capítulo 3.

Nosso sistema opera todas as composições de imagem nas GPUs. Como todas as caixas envolventes dos submodelos são convexas e adjacentes, é bem simples ordenar as partições de acordo com a posição do observador. Isso nos permite utilizar o algoritmo do pintor para resolver a visibilidade corretamente no estágio de composição: ao invés de ler e enviar o *buffer* de profundidades de cada região de tela coberta e fazer composição utilizando o *z-buffer*, é suficiente ler e enviar a componente alfa de cada *pixel*. Um valor de alfa igual a 0 sinaliza que o *pixel* não foi escrito ao longo da renderização. O valor de alfa também é suficiente para compor sub-imagens com transparência, necessária pela técnica de visualização volumétrica. A Tabela 4.1 lista as componentes que devem ser lidas e indica os operadores de composição. O operador OVER foi definido por Porter e Duff (44).

Algoritmo de visualização	Componentes lidas	Operador de composição
Iso-contorno/iso-linha	RGBA	Teste de alfa
Visualização volumétrica	RGBA	OVER

Tabela 4.1: Componentes lidas e operadores de composição para cada algoritmo de visualização.

4.3 Linha de Produção Distribuída

A Figura 4.2 detalha a nossa implementação. A execução feita em cada entidade (cliente, mestre e renderizadores) segue uma linha de produção, onde cada estágio executa em uma linha de execução separada.

4.3.1 Renderizadores

Os nós renderizadores são responsáveis por carregar e efetivamente renderizar as partes do modelo designadas às suas GPUs. Cada nó executa um número de linhas de produção de GPU em paralelo. Como resultado, cada nó de renderização gera um conjunto de imagens parciais. No entanto, ao invés de enviar todas as imagens parciais para o nó mestre, nós optamos por introduzir um estágio de *composição parcial* no nó de renderização.

Com base na partição do modelo descrita na Seção 4.2, cada nó de renderização compõe as imagens parciais geradas pelas suas GPUs de trás

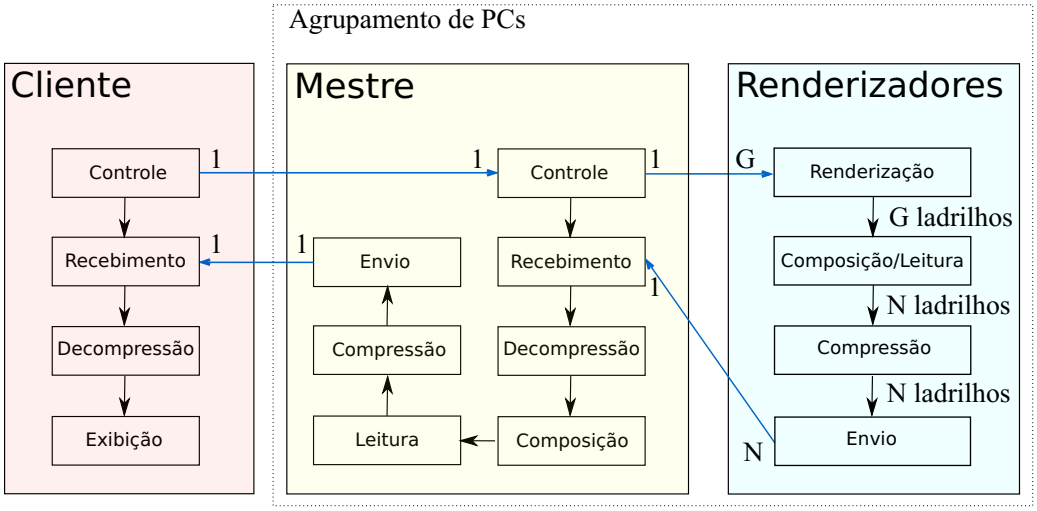


Figura 4.2: Nossa arquitetura de linha de produção com ordenação no fim: G é o número agregado de GPUs; N é o número de nós renderizadores.

para a frente. A primeira GPU na ordem de trás para a frente será responsável por operar a composição parcial. A linha de execução de cada outra GPU deve ler a região da tela coberta pela caixa envolvente de seu submodelo e passar o resultado para a linha de execução da primeira GPU, que fará a composição na ordem correta. Após a composição, a união de todas as coberturas de tela é lida e inserida na fila de compressão.

Os estágios de compressão comprimem as imagens parciais sem perda utilizando a biblioteca LZO (40), passando o resultado para o estágio de envio. Este por sua vez é responsável por enviar as imagens comprimidas para o estágio de recebimento do nó mestre.

4.3.2 Mestre

O nó mestre é responsável por controlar a renderização distribuída. Esse controle é implementado no estágio de controle da linha de produção. Suas tarefas incluem:

- manter os parâmetros de câmera atualizados em todos os renderizadores;
- calcular e enviar a região de tela coberta por cada caixa envolvente; isso é feito utilizando o algoritmo descrito por Blinn (9);
- repassar todos os parâmetros específicos da aplicação para os renderizadores;
- requisitar a renderização de um novo quadro.

Esse estágio também implementa o controle de quadros. Enquanto um quadro está sendo recebido e composto, outro quadro pode ser requisitado

em todos os renderizadores, como feito em (2). Apesar disso resultar em um quadro adicional de latência, o paralelismo entre todos os estágios é aumentado, resultando em um melhor desempenho em termos de taxas de renderização.

O estágio de recebimento espera por imagens parciais comprimidas enviadas pelos nós renderizadores. Cada imagem recebida é repassada para o estágio de descompressão, que as descomprime e as repassa para o estágio de composição. As imagens parciais são finalmente compostas de trás para a frente na GPU do nó mestre.

Assim que a imagem final está pronta, suas componentes RGB são lidas de volta para a memória principal e são passadas para a compressão e envio para o nó cliente. Com o objetivo de melhorar o desempenho, a leitura, compressão e envio são implementados em paralelo. A imagem final é dividida em ladrilhos horizontais. Esses ladrilhos são lidos para a memória principal utilizando a versão assíncrona da chamada `glReadPixels` oferecida pela extensão ARB_pixel_buffer_object do OpenGL. Dessa forma, a compressão e envio podem ser feitas em paralelo com a leitura da placa: enquanto um ladrilho está sendo lido, o ladrilho lido anteriormente pode ser posto na fila para a compressão e envio para o nó cliente.

Note que o estágio de composição parcial feito em cada nó renderizador resulta em uma grande redução no número de imagens parciais comprimidas, transmitidas e descomprimidas no nó mestre. O estágio de composição feito no nó mestre, que pode facilmente se tornar o gargalo do sistema, também tem a sua quantidade de trabalho reduzida a cada quadro. Isso é especialmente o caso quando um número grande de GPUs é utilizado.

A descompressão de ladrilhos também pode facilmente se tornar o gargalo do sistema. Se isso acontecer, é possível utilizar múltiplas linhas de execução de descompressão. Nós mestres com múltiplos núcleos de processamento podem utilizar duas ou mais linhas de execução de descompressão sem degradar o desempenho da linha de produção, pois esse é o único estágio pesado em termos de CPU neste nó.

4.3.3 Cliente

O nó cliente recebe, descomprime e exibe a imagem RGB final. Esses três estágios executam em paralelo com os estágios de leitura, compressão e envio do nó mestre. Conforme iremos demonstrar, esse paralelismo ao transferir a imagem final para o cliente resulta em um ganho significativo de desempenho quando a largura de banda da rede entre o cliente e o mestre aparenta ser o gargalo do sistema.

O nó cliente também executa um estágio de controle. Ele é responsável por tratar a entrada do usuário e controlar o motor de renderização implementado no cluster.

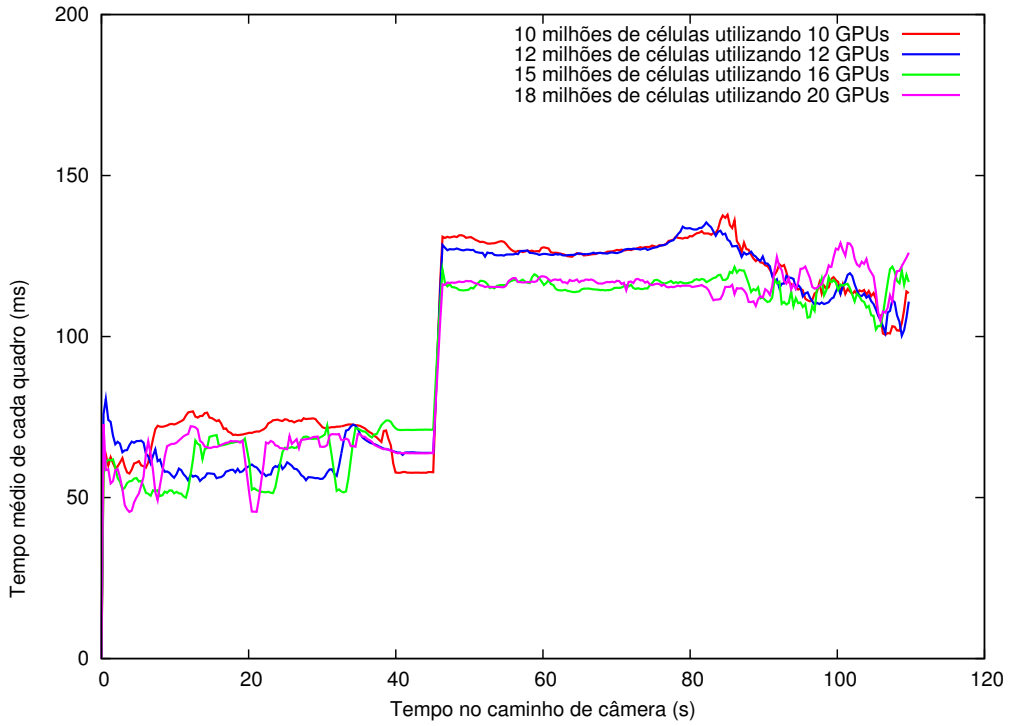
4.4 Resultados Experimentais

O sistema proposto foi testado utilizando um agrupamento de PCs com 16 nós, onde cada nó é equipado com dois processadores *dual-core* AMD Opteron 2,4 Ghz, 8 gigabytes de memória RAM e conectados por duas redes ao mesmo tempo: uma rede Ethernet num comutador (*switch*) de 1 gigabit por segundo e uma rede Infiniband 4x. Isso nos permitiu testar nosso sistema em agrupamentos de PCs para a visualização com componentes *de prateleira* e agrupamentos com conexões mais caras. O nó mestre é equipado com uma placa gráfica NVIDIA Geforce GTX 280. Cada um dos 15 nós renderizadores é equipado com 4 placas gráficas NVIDIA Quadro FX 5600, cada uma com 1,5 gigabytes de memória de vídeo. Essas placas gráficas foram ligadas a dois barramentos PCI-express x16. Uma descrição detalhada do *hardware* do agrupamento de PCs utilizado pode ser encontrado em (50). O sistema do nó cliente foi um *laptop* conectado ao agrupamento de PCs por meio de uma rede local Ethernet de 1 gigabit por segundo.

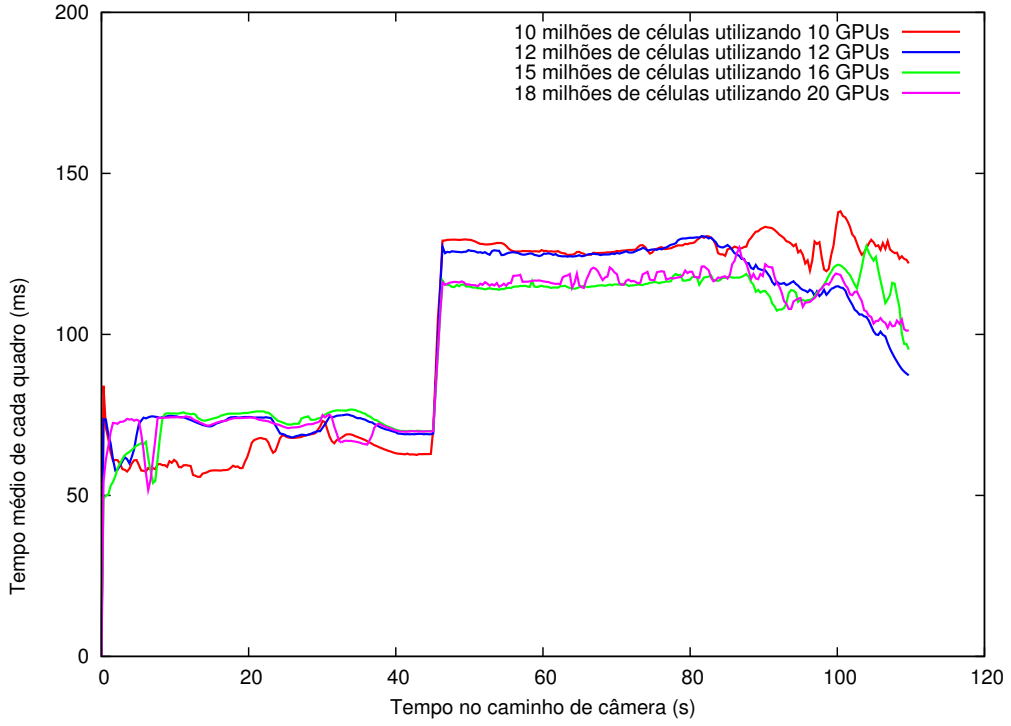
O sistema foi implementado utilizando C++, OpenGL, a biblioteca *pthreads* e soquetes TCP/IP em um sistema operacional Linux. Cada linha de execução de renderização utiliza as técnicas de descarte por volume de visão e oclusão mencionadas no Capítulo 3, reduzindo assim o número de primitivas gráficas enviadas às linhas de produção das placas gráficas a cada quadro. Todos os testes foram feitos com uma tela com resolução de 1680 x 970.

O sistema foi testado utilizando um modelo de simulação de reservatórios de petróleo com diferentes discretizações, variando de 10 a 60 milhões de células no total. Uma limitação do nosso sistema é o consumo de memória na fase de partição do modelo, necessitando de um PC com memória suficiente para carregar as coordenadas do modelo inteiro. Como nosso nó mestre possui 8 gigabytes de memória, a partição teve de ser feita em um computador separado em alguns casos.

4.4.1
Escalabilidade do Sistema



(a)



(b)

Figura 4.3: Escalabilidade do sistema utilizando (a) a rede Ethernet (b) a rede Infiniband.

Com o objetivo de testar a escalabilidade do sistema, nós variamos ambos o tamanho do modelo e o número de nós renderizadores utilizados. O teste foi feito utilizando apenas 2 GPUs de cada nó. A Figura 4.3 ilustra os resultados obtidos utilizando a rede Ethernet de 1 GBps e a rede Infiniband. Os gráficos mostram o tempo gasto na renderização de cada quadro ao longo de um caminho de câmera. Nesse caminho de câmera, o modelo original é rotacionado até $t = 45s$. Nesse ponto, as camadas do reservatório são separadas, aumentando o esforço de renderização. O modelo é a partir daí rotacionado até $t = 90s$, quando uma navegação é feita para inspecionar o modelo mais de perto. Como pode ser notado, o sistema escala razoavelmente bem em ambas as configurações de rede, apresentando um bom desempenho geral.

É importante notar que o desempenho do sistema parece não ser afetado pelo uso de uma rede mais rápida. Isso se deve ao fato da transmissão de dados não ser o gargalo do sistema nessa configuração. Apesar de não afetar a taxa de renderização, o uso de conexões mais rápidas e de compressão reduz a latência do sistema, outro aspecto importante de sistemas interativos.

4.4.2 Composição Parcial

Como cada nó de renderização teve uma parte pré-definida do modelo designada a ele, o sistema pode sofrer de desbalanceamento de carga quando se aproxima do modelo, dadas as diferentes eficácias das técnicas de aceleração utilizadas. Adicionalmente, em tais situações, alguns submodelos designados a alguns nós de renderização podem cobrir largas áreas da tela, necessitando de mais largura de banda e de esforços de composição.

Sob tais circunstâncias, a composição parcial de imagens em cada nó de renderização se mostrou bastante eficiente. Isso é demonstrado testando o sistema utilizando um caminho de câmera diferente. O caminho de câmera começa visualizando o modelo inteiro sem separação de camadas. A câmera se aproxima do modelo até $t = 30s$ e retorna para enquadrar o modelo inteiro na tela em $t = 60s$. Em $t = 65s$, as camadas do reservatório são separadas, e uma navegação para inspeccionar o modelo entre camadas é feita até $t = 115s$. Após isso, o observador retorna à sua posição original.

O teste foi feito utilizando 12 nós de renderização, cada um utilizando 4 GPUs e interconectados pela rede Ethernet. A Figura 4.4 ilustra o ganho de desempenho por conta de aplicar um estágio de composição parcial em cada nó, especialmente para a segunda metade do experimento, onde o desempenho é crítico por conta da separação de camadas.

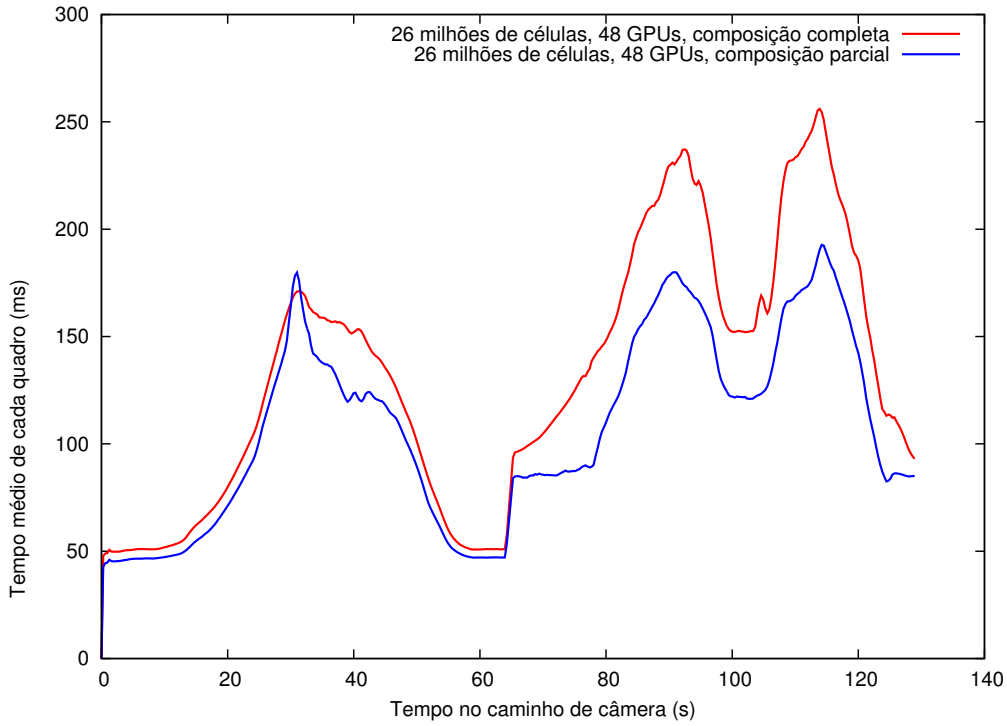


Figura 4.4: Comparação entre composição completa e composição parcial.

4.4.3 Partição da Imagem Final

Nós utilizamos o mesmo caminho de câmera do último experimento para testar os benefícios de particionar a imagem final, paralelizando os estágios de leitura, compressão e envio do nó mestre e os estágios de recebimento, descompressão e exibição no nó cliente.

O teste foi feito utilizando 11 nós de renderização com 4 GPUs cada, conectados pela rede Infiniband. Nós variamos o número de ladrilhos para a transferência da imagem final do mestre para o cliente. Conforme mostrado na Figura 4.5, nós obtivemos o melhor ganho ao particionar a imagem final em 4 ladrilhos.

4.4.4 Estressando o Sistema

Com o objetivo de estressar o sistema, nós aumentamos o tamanho do modelo para 60 milhões de células. A Figura 4.6 ilustra o desempenho obtido utilizando um total de 59 GPUs distribuídas entre 15 nós renderizadores, todos conectados pela rede Ethernet. Nesse teste, a câmera utilizou o mesmo caminho dos dois últimos experimentos. O gráfico mostra que o nosso sistema é capaz de renderizar o modelo com uma taxa interativa de renderização ao longo de todo

o caminho de câmera. No entanto, é possível notar que o sistema encontra-se próximo do limite mínimo de desempenho para a obtenção de uma visualização interativa, tornando evidente a difícil escalabilidade dessa estratégia para a visualização de modelos massivos.

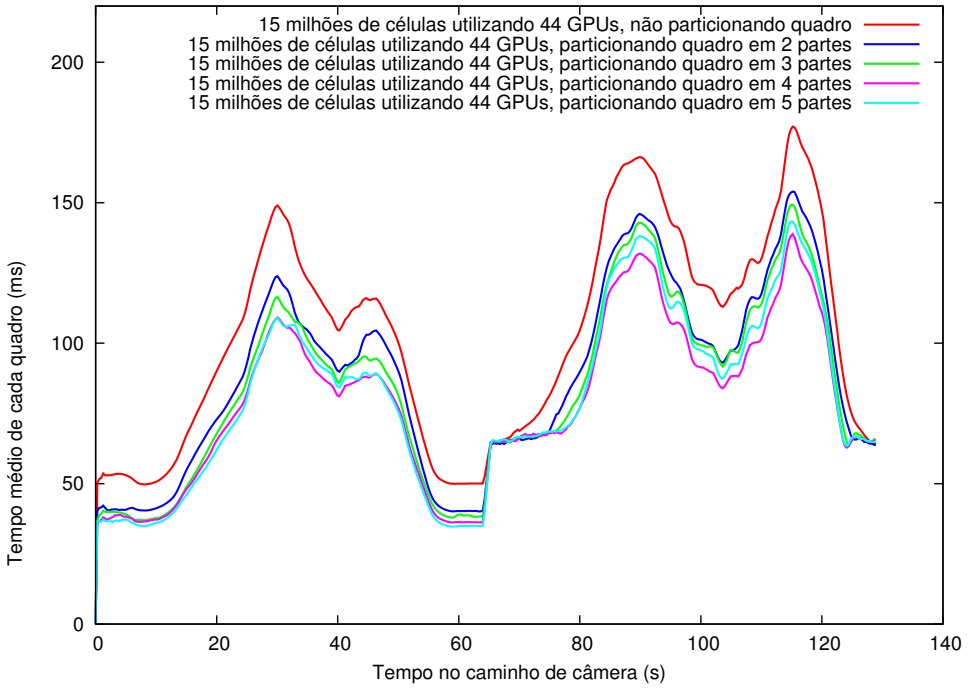


Figura 4.5: Comparação ao se particionar a imagem final.

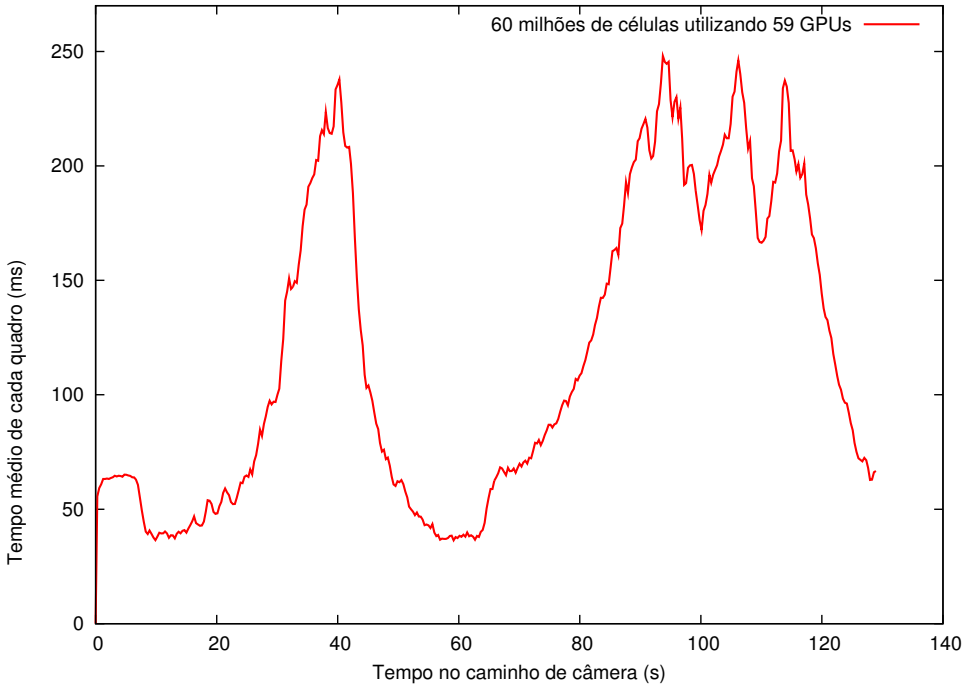


Figura 4.6: Desempenho do sistema na visualização de 60 milhões de células.