Resolução de Sistemas Lineares com Eliminação de Gauss em CUDA

Aluno: Adrian Cerbaro - 178304@upf.br

Implementação

O algoritmo foi paralelizado adotando uma estratégia de decomposição de domínio onde o laço externo (pivô i) permanece no *Host* (CPU) e o processamento das linhas da submatriz ativa ($j > i$) é executado no Device (GPU). Para cada iteração do pivô, um kernel é lançado. A distribuição de carga utiliza o padrão *Grid-Stride Loop*. Isso desacopla o tamanho do problema (N) dos recursos de hardware: cada bloco processa um conjunto de linhas iterativamente, e dentro do bloco, as threads processam colunas. Isso permite que a solução escale para matrizes maiores que a capacidade física da grid sem alterações no código. Para a etapa de *Back-Substitution*, implementou-se uma abordagem híbrida comparativa: (A) execução serializada na GPU com 1 bloco/1 thread e (B) cópia dos dados para o Host e execução sequencial na CPU.

Ambiente e Metodologia

Os testes foram realizados em um processador **Intel Core i7-2600 @ 3.40GHz** com **8GB** de RAM e uma GPU **NVIDIA GeForce GTX 1660** (22 SMs, limite de 1024 threads/SM e 16 blocos/SM). O kernel utiliza 32 registradores/thread, sem uso de memória local (spill). Cada configuração foi executada 7 vezes, utilizando a **mediana** dos tempos para a análise.

Recursos Utilizados

Utilizou-se memória global com acessos coalescidos para maximizar a largura de banda. A sincronização intra-bloco (*__syncthreads*) foi empregada para garantir a consistência na leitura do multiplicador (ratio) antes das escritas.

Dificuldades encontradas

Tentou-se utilizar Cooperative Groups e Atomic Operations para sincronização global e evitar o *overhead* de lançamento de múltiplos kernels. Contudo, essa abordagem limitou a quantidade máxima de blocos ao número de SMs disponíveis (22 no hardware testado), inviabilizando a ocupação total para N grande e apresentando riscos de deadlock. Além disso, identificou-se que o uso máximo de threads (1024) ou configurações específicas (como 256 threads) podem sofrer limitações de agendamento não triviais, onde o número real de blocos residentes é inferior ao teórico, causando quedas de ocupação.

Análise dos Resultados

Os testes revelaram que o speedup não é linear em relação ao número de recursos, sendo limitado pela saturação dos SMs e pelo *Tail Effect*.

1. **Saturação e Tail Effect**: Enquanto a GPU não está saturada, o aumento de blocos eleva o desempenho quase linearmente. Após a saturação, observou-se um comportamento de "dente de serra" (Fig. 1). Quedas bruscas de desempenho ocorrem quando a quantidade de blocos não é divisível pela capacidade simultânea da GPU, gerando uma "onda final" de execução ineficiente.

2. **Configuração Ideal**: O melhor *speedup* foi de 24.97x (Tabela 1) com 704 blocos e 512 threads. Conforme observado no mapa de calor da Figura 2, esta região representa o ponto ótimo global. Isso se deve ao alinhamento perfeito de quantização: 704 é múltiplo exato de 22 SMs e preenche a capacidade de slots, maximizando a ocupação sem gerar sobras.

3. **Back-Substitution**: A execução no Host (CPU) mostrou-se superior à execução no Device (Tabela 2). O *overhead* da transferência de dados ($N = 7500$) foi inferior ao custo de executar um algoritmo inerentemente sequencial na GPU, que possui *clock* inferior à CPU.

Considerações Finais

O trabalho evidenciou que maximizar o número de threads e blocos indiscriminadamente não garante o melhor speedup. O desempenho é ditado pelo equilíbrio entre a granularidade da tarefa e os limites físicos da arquitetura (slots por SM e divisibilidade pelos SMs). Conclui-se que o ajuste fino dos parâmetros de lançamento para evitar o tail effect e o uso de estratégias híbridas (CPU para serial, GPU para paralelo) são fundamentais para atingir o teto de desempenho do hardware.

Figura 1: Análise de Speedup Real vs. Teórico e Pontos de Queda (Tail Effect)

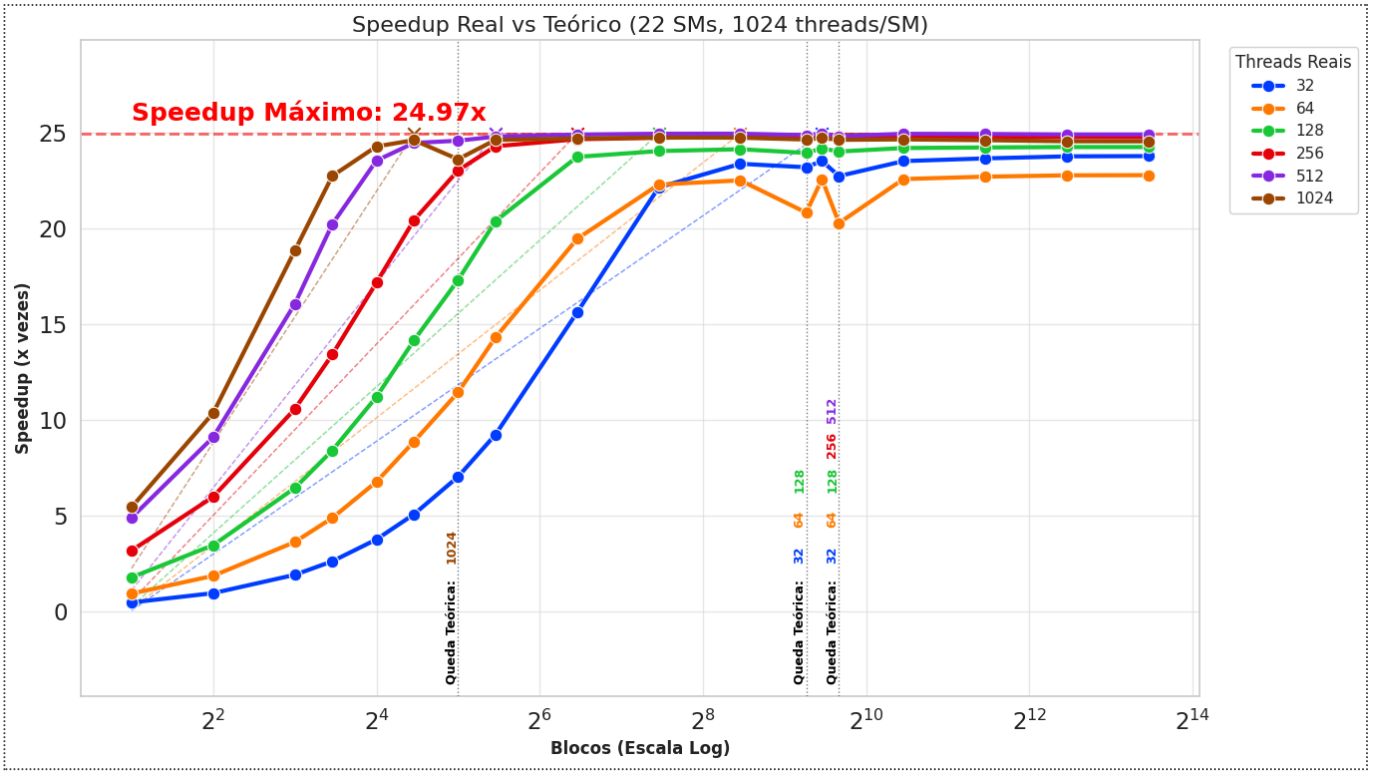

Figura 2: Mapa de Calor de Speedup (Speedup por Configuração)

Tabela 1: Dados Brutos da Melhor Configuração (N=7500, BS=Host, Blocks=704, Threads=512)

Execução	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run 7**
Tempo (s)	14.66 s	14.67 s	14.66 s	14.66 s	14.71 s	14.66 s	14.67 s
Speedup	24.97 x	24.96 x	24.97 x	24.97 x	24.88 x	24.97 x	24.97 x
						Mediana	14.66 s

Tabela 2: Comparativo de Estratégias de *Back-Substitution* (N=7500)

Estratégia	**Local de Execução**	**Tempo Total Médio (s)**	**Observação**
BS-Device	GPU (Kernel Serial)	$\approx 15.65\ s$	Lento devido ao baixo clock e serialização na GPU.
BS-Host	CPU (Sequencial)	$\approx 14.66\ s$	**Mais rápido**, apesar do *overhead* de cópia de memória.