**Capítulo 3**

**3.1 – Introdução**

CUDA foi especialmente criada para aproveitar o mecanismo de processamento paralelo das GPUs NVIDIA a fim resolver problemas computacionais complexos. Para programar segundo a arquitetura CUDA™, os desenvolvedores hoje em dia podem usar a linguagem C [28]. O desenvolvimento de várias técnicas de simulação para análise de desempenho, é necessário o projeto de uma arquitetura de software extremamente flexível e extensível, para isso a utilização de vários padrões de projeto de software [29], é crucial para efetivação de tais requisitos.

O desenvolvimento do sistema utilizada a API CUDA da Nvidia, sendo necessário observar alguns aspectos do problema a resolver, tais como se o problema e paralelizável, conhecimento prévio da arquitetura física da placa na qual se deseja programar, a placa de vídeo deve suportar CUDA.

A noção de aceleração foi estabelecida pela lei de Amdahl [30], que foi particularmente focada no processamento paralelo. Na arquitetura de computadores, o aumento de velocidade é um número que mede o desempenho relativo de dois sistemas que processam o mesmo problema. Mais tecnicamente, é a melhoria na velocidade de execução de uma tarefa executada em duas arquiteturas similares com recursos diferentes, no entanto, o aumento de velocidade pode ser usado de forma mais geral para mostrar o efeito no desempenho após qualquer aprimoramento de recursos. Lei de Gustafson [31] dá uma avaliação mais realista do desempenho paralelo.

A programação paralela consiste em executar simultaneamente várias partes de uma mesma aplicação, tornando possível a partir do desenvolvimento de sistemas operacionais multi-tarefa, multi-thread e paralelos. As aplicações são executadas paralelamente em um mesmo processador, em uma máquina com um multiprocessador ou em um grupo de máquinas interligadas que se comporta como uma só máquina visando o ganho de velocidade.

**3.2 – Speed-up**

Ganho de velocidade (*speed-up*) consiste na comparação entre o tempo de execução do programa em um único processador e o tempo de execução utilizando vários processadores. Serve como parâmetro para a avaliação do desempenho de algoritmos paralelos. O fator de *speedup* de um sistema de computação paralela com vários processadores pode ser definido como a razão do tempo necessário para a execução serial pelo tempo gasto por um sistema paralelo na solução do mesmo problema [32]. O valor do *speedup* mostra o quanto mais rápido um algoritmo paralelo é em relação ao seu correspondente sequencial, ou seja, o quanto ele é escalável.

O *speedup* de um programa utilizando a estrutura de múltiplos processadores em computação paralela tem limitações impostas pelo tempo dispensado para a execução de uma fração sequencial de um programa. Como exemplo, se o programa gasta o equivalente a 40 horas de alocação de um único núcleo de processamento, e a parte específica de uma rotina do programa gasta duas horas para executar não podendo ser paralelizada, enquanto as 38 horas restantes (que é equivalente a 95%) do tempo de processamento de execução pode ser paralelizado, independente de quantos processadores são dedicados a execução paralela deste programa, o tempo de execução mínima não pode ser menor que a seu maior processamento. Sendo assim temos que o aumento de velocidade é limitado geralmente ao maior tempo de processo paralelizado [30].

**3.3 – Lei de Amdahl**

A Lei de Amdahl [30] é um dos métodos que tenta explicar o ganho obtido em uma aplicação ao adicionar mais processadores. O norueguês Gene Amdahl deu início em suas pesquisas seguindo a ideia de que um trecho do programa é puramente sequencial. Sua apresentação inicial ocorreu na Conferência Conjunta de Informática unificada (*Conference Proceedings*, AFIPS) no ano de 1967 [31].

Segundo Amdahl, apenas o desempenho do trecho que não é sequencial pode ser aumentado, sendo assim para medir os benefícios que uma arquitetura paralela é capaz de oferecer, é necessário algum tipo de métrica que forneça em números o ganho obtido ao executar paralelamente uma determinada aplicação. Em outras palavras, obtém-se a razão entre o tempo de execução da aplicação rodando de forma puramente sequencial e o tempo de execução da mesma aplicação rodando de forma paralela. Na figura 3 têm-se a gráfico para a justificativa da argumentação de *speedup* ligado ao aumento de processadores segundo a Lei de Amdahl [32].

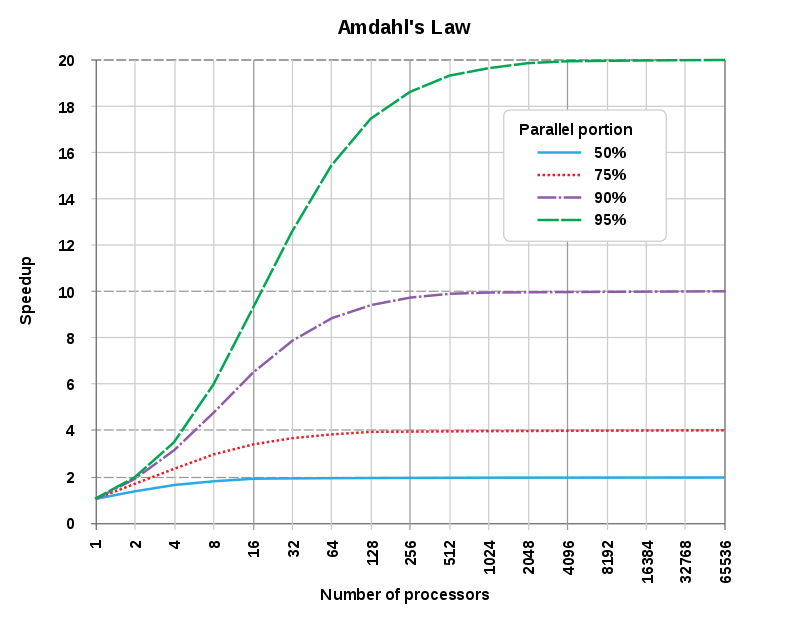
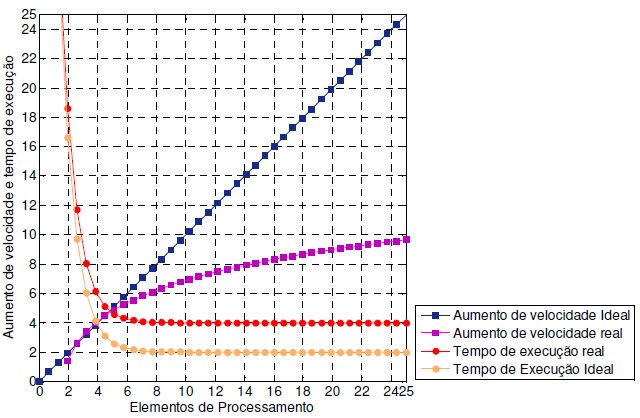


Figura 3. Argumento de Amdahl [32].

A Lei de Amdahl, também conhecida como argumento de Amdahl [32] é usada para encontrar a máxima melhora esperada para um sistema em geral quando apenas uma única parte do mesmo é melhorada. Isto é frequentemente usado em computação paralela para prever o máximo *speedup* teórico usando múltiplos processadores [32].

A figura 4 mostra o gráfico do tempo de execução em relação ao *speedup* de programa paralelizável. Onde a linha azul demonstra um cenário ideal, a linha lilás indica o aumento real, a linha amarela indica o tempo de execução no caso ideal e a vermelha o tempo real.

Figura 4. O tempo de execução e *speedup*, adaptado de Amdahl [32].

**3.4 – Lei de Gustafson**

A análise de Amdahl parte do princípio de que a fração serial (não-paralelizável) do algoritmo é independente do número de processadores. Segundo Gustafson do ponto de vista prático tal consideração não seria razoável em alguns casos, pois as tarefas aumentam com o aumento da capacidade computacional e a fração de código serial, porém em sua grande maioria, não existe crescimento de mesma proporção. Isto levaria muitos algoritmos paralelos a terem sua fração serial reduzida com o aumento do número de processadores. A partir da observação de resultados práticos, Gustafson sugere uma métrica alternativa a Lei de Amdahl, onde um só processador é no pior dos casos poucas vezes mais lento que a execução com alguns processadores, não tendo assim grandes vantagens aparentes.

**3.5 – Vantagens da programação paralela**

Algumas das vantagens da programação paralela consiste em sistemas paralelos fortemente acoplados, compartilhando hardware ou se comunicando através de um barramento de alta velocidade, previsibilidade em virtude do comportamento de sistemas paralelos ser bastante previsível em comparação com outros tipos de programação e o controle de todos os recursos computacionais. Os ganhos com o uso desse tipo de programação melhoraram o poder de processamento, apresentam um melhor desempenho, permitir compartilhar dados e recursos, podendo apresentar maior confiabilidade, permitem reutilizar serviços já disponíveis e atender um maior número de usuários.

Existem pesquisas que avaliam o desempenho das principais APIs de programação paralela, através da execução de algoritmos em CPU e GPU. Para que a avaliação de desempenho fosse possível, primeiramente foram instaladas em um computador as APIs CUDA (GPU), OpenCL (GPU), OpenACC (GPU) e OpenMP (CPU) [33]. Os speedups dos testes mostraram que a GPU obteve um desempenho superior em relação a CPU na maioria dos algoritmos. Acredita-se que esse resultado ocorreu pelo fato da GPU possuir mais núcleos se comparada a CPU, apesar da frequência dos núcleos ser inferior [33]. Os resultados do artigo mostraram que os processos paralelos têm maior ganho computacional conforme a complexidade dos algoritmos ou mesmo do processo computacional.

**3.6 – Principais dificuldades**

As principais dificuldades consistem em desenvolver, gerenciar e manter o sistema em virtude da alta complexidade encontrada neste tipo de cenário, controlar o acesso concorrente a dados e a recursos compartilhados, evitar que falhas de máquinas ou da rede comprometam o funcionamento do sistema, garantir a segurança do sistema e o sigilo dos dados trocados entre máquinas e lidar com a heterogeneidade do ambiente. Outro fator limitante seria pouco trecho de código paralelizável do algoritmo sendo assim muito importante identificar se o programa é *Embarrassingly Parallel*, isto é, um problema em que sua solução pode ser dividida em várias partes independentes, que tarefas sejam executadas simultaneamente.

**3.7 – Sistema operacional multitarefa**

Um Sistema operacional multitarefa permite simular o paralelismo em um único processador, alternando a execução de processos, um processador com núcleo múltiplo permite paralelismo real entre processos, executando múltiplas instruções por ciclo, uma placa mãe multiprocessador permite que cada processador execute um processo, um cluster é uma solução de baixo custo para processamento de alto desempenho, uma computação distribuída é possível em redes, como numa *Intranet* e na *Internet* [11].

O Suporte computacional para computação paralela são compostos por sistemas operacionais multi-tarefa e linguagens multi-tarefa como exemplo o CUDA. Neste contexto composto por mecanismos para controle de concorrência, mecanismos para comunicação entre processos, *threads* paralelos e distribuídos, ferramentas e mecanismos para desenvolvimento [11].

A Figura 5 mostra que neste cenário um programa que possui um conjunto de instruções em uma linguagem de alto nível ou de máquina, um processo que resulta da execução do programa e *threads* que são processos menores que compartilham o mesmo "espaço de endereçamento“ (memória). Cada um dos multiprocessadores tem várias ALUs que em qualquer ciclo de *clock* executam as mesmas instruções, tendo assim uma quantidade enorme de capacidade de processamento.

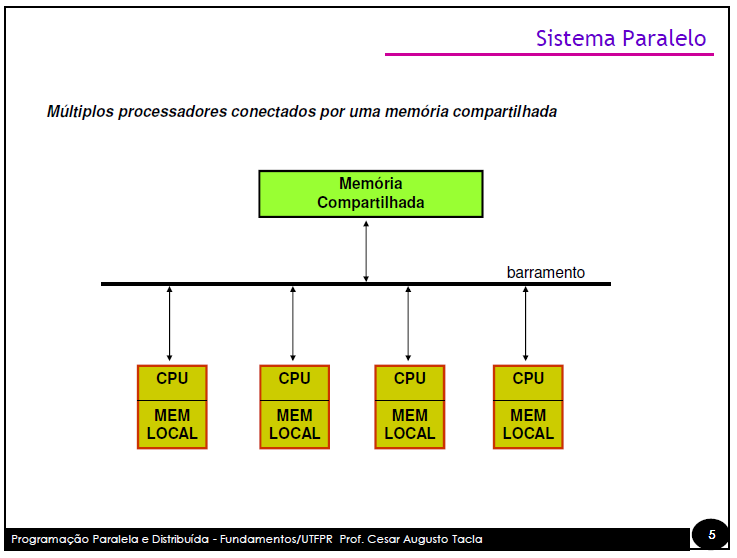


Figura 5. Sistema paralelo, adaptado de Correa [33].

**3.8 – Flops**

*FLOPS* (flops, FLOP/s ou flop/s) é um acrônimo na computação que significa operações de ponto flutuante por segundo (*Floating-point Operations Per Second)*. Isto é usado para determinar o desempenho de um computador, especificamente no campo de cálculos científicos, que fazem grande uso de cálculos com ponto flutuante; similar a instruções por segundo. O "S" no final do termo não denota plural, mas significa exatamente o que o acrônimo apresenta. Refere-se apenas à quantidade das operações de ponto flutuante sem delimitar tempo. É um valor utilizado para indicar a taxa de velocidade dos microprocessadores (em computadores). Expressa a performance do microprocessador.

Os dispositivos de computação têm enorme capacidade de processamento, convém utilizar unidades maiores que *FLOPS*, seus múltiplos. Os múltiplos mais utilizados são: megaflop/s (Mflop/s), gigaflop/s (Gflop/s), teraflop/s (Tflop/s), petaflop/s (Pflop/s) e exaflop/s (Eflop/s).

As representações de valores são expressas para facilitar a demonstração numérica do desempenho de um sistema com supercomputadores, é necessário escolher uma métrica de performance que possa representá-lo sem introduzir distorções, sem influenciar os resultados e sem favorecer um determinado escalonador do computador [34].

Na Tabela 1 é mostrado as representações de valores de cada um dos múltiplos *FLOPS*.

|  |  |
| --- | --- |
| **Desempenho Computacional** | |
| Ordem de grandeza | Quantidade (flop/s) |
| megaflop/s | 106 |
| gigaflop/s | 109 |
| teraflop/s | 1012 |
| petaflop/s | 1015 |
| exaflop/s | 1018 |
| zettaflop/s | 1021 |
| yottaflop/s | 1024 |

**Tabela 1:** Valores para ilustrar a quantificação do desempenho computacional [35]

**3.9 – As principais diferenças entre o acesso as memorias da GPU e CPU.**

As principais memórias de uma GPU são compostas pela memória global, memória compartilhada e memória local. A memória global é a principal memória da GPU, podendo ser acessada por os *threads/cores,* porém possui alta latência e baixo *throughput.* A Memória compartilhada é a memória dedicada de cada SM (*Stream Multiprocessors*), que possui baixa latência. Somente *threads* de um mesmo bloco pode acessá-la. Já a memória local possui este nome pois é a memória específica de um thread [36]. Das memórias descritas acima a CPU tem apenas o acesso a memória global. A distribuição de threads e hierarquia de memória é ilustrada pela Figura 6.

Figura 6. Memória compartilhada, adaptado de Chapman [36].