CPUs e GPUs: Arquiteturas e Programação

CPUs e GPUs: Arquiteturas e Programação

- Introdução às arquiteturas de processadores
- Visão geral sobre CPUs e evolução das arquiteturas
- Arquitetura de GPUs: conceitos e evolução
- Comparação detalhada entre CPUs e GPUs
- Modelo de programação CUDA para GPUs
- Modelos de programação OpenMP para paralelismo SIMD em GPUs

Introdução às arquiteturas de processadores

- **Histórico e evolução tecnológica:** O aumento exponencial do poder computacional impulsionou a necessidade por arquiteturas avançadas.
- **Funções e aplicações das CPUs:** CPUs operam tarefas gerais, focando controle sofisticado e processamento sequencial eficiente em sistemas computacionais.
- **GPU: arquitetura e finalidade:** GPUs otimizam processamento paralelo massivo para gráficos, simulações e aprendizado de máquina, diferindo das CPUs.

Visão geral sobre CPUs e evolução das arquiteturas

- **Arquitetura básica das CPUs:** Unidade de controle, ULA e registradores cooperam para execução sequencial e controle preciso de instruções.
- **Evolução das famílias de processadores:** Intel Core, AMD Ryzen e ARM diferem em desempenho, consumo e aplicações comerciais variadas significativas.
- **Lei de Moore e multicore:** Limitações térmicas motivaram transição para múltiplos núcleos, exigindo software adaptado para paralelismo real.

Arquitetura de GPUs: conceitos e evolução

- **Paralelismo massivo e design simplificado:** GPUs possuem milhares de núcleos simples para executar muitas threads simultaneamente com eficiência SIMD/SIMT.
- **Evolução histórica das GPUs:** De unidades fixas nos anos 1990 a shaders programáveis nos 2000, permitindo flexibilidade e maior controle gráfico.
- **Avanços tecnológicos recentes:** Arquiteturas como NVIDIA Ampere e AMD RDNA 2+ incorporam ray tracing, tensor cores e memórias de alta largura de banda.

Comparação detalhada entre CPUs e GPUs

- **Complexidade e número de núcleos:** CPUs possuem poucos núcleos complexos, GPUs contam com milhares de núcleos simples para paralelismo massivo.
- **Controle de fluxo e caches:** CPUs gerenciam controle sofisticado e caches hierárquicos para baixa latência, GPUs simplificam para throughput alto.
- **Paralelismo SIMT e aplicações:** GPUs usam modelo SIMT para execução eficiente em IA, simulações complexas e processamento gráfico avançado.

Modelo de programação CUDA para GPUs

- **Hierarquia de threads em CUDA:** Threads organizam-se em blocos e grids tridimensionais para escalabilidade e paralelismo eficiente.
- **Variáveis especiais CUDA:** threadIdx, blockIdx e blockDim identificam posição das threads, facilitando mapeamento de processamento paralelo.
- **Exemplo básico de kernel:** Kernel soma vetores ilustra execução independente de threads, mostrando estrutura típica do código CUDA.

Modelos de programação OpenMP para paralelismo SIMD em GPUs

- **Diretivas-chave do OpenMP para GPUs:** OpenMP 4.0 introduziu offloading; #pragma omp simd vectoriza loops para execução SIMD eficiente em GPUs.
- **Combinação de paralelismo de threads e SIMD:** #pragma omp parallel for simd une paralelismo entre threads com vetorização, otimizando recursos computacionais.
- **Exemplo e benefícios em comparação ao CUDA:** Exemplo em C soma vetorial com #pragma omp target mostra portabilidade e simplicidade versus programação CUDA complexa.