Relatório da Tarefa 18: Comparação de Execução entre CPU e GPU com OpenMP

Aluno: Cristovão Lacerda Cronje

---

1. Introdução

Este relatório analisa o desempenho de uma operação de adição vetorial implementada em C com paralelismo via OpenMP, comparando a execução na CPU e na GPU (offloading). Embora a GPU possua maior capacidade de paralelismo, observou-se que sua performance foi inferior à da CPU neste caso. Este documento discute os motivos dessa diferença e os fatores que impactam a eficiência computacional em cada arquitetura.

---

2. Objetivos

- Implementar a adição de vetores utilizando OpenMP.
- Realizar o offloading para GPU com diretivas OpenMP.
- Medir e comparar os tempos de execução entre CPU e GPU.
- Verificar a corretude dos resultados em ambas as execuções.

---

3. Metodologia

3.1 Estrutura do Código

O código segue estas etapas:

- **Inicialização:** Preenchimento dos vetores a, b e res (esperado).
- **Cálculo:** Soma elemento a elemento dos vetores a e b, armazenando em c.
- **Verificação:** Comparação entre c[i] e res[i] com contagem de erros.
- **Medição de tempo:** Uso de omp_get_wtime() para medir a duração da computação.

```
// 1. Inicialização dos vetores (paralelizada)
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    a[i] = (float)i;           // Preenche a com valores 0, 1, 2, ..., N-1
    b[i] = 2.0f * (float)i;    // Preenche b com valores 0, 2, 4, ..., 2(N-1)
    c[i] = 0.0f;               // Inicializa c com zeros
    res[i] = i + 2 * i;        // Calcula o resultado esperado (a[i] + b[i])
}

// 2. Operação de soma vetorial (paralelizada)
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];       // Soma os elementos de a e b em c
}

// 3. Verificação dos resultados (paralelizada com redução)
#pragma omp parallel for reduction(+:err)
for (int i = 0; i < N; i++) {
    float val = c[i] - res[i]; // Diferença entre valor calculado e esperado
    val = val * val;           // Quadrado da diferença (para eliminar sinais)
    if (val > TOL) err++;      // Incrementa contador se diferença > tolerância
}

printf("Vetores somados com %d erros\n", err);
return 0;
}
```

3.2 Diretivas OpenMP Utilizadas

- #pragma omp parallel for:
    - Utilizada nos três laços principais.
    - Distribui iterações entre as threads.

    - Implementação básica sem especificação de escalonamento.
- reduction(+:err):
    - Utilizada na verificação dos resultados.
    - Garante incremento seguro da variável err em paralelo, evitando condições de corrida.

---

4. Resultados e Análise

4.1 Dados de Execução

--- OpenMP Vector Add Test ---
Vector size: 10000000

CPU computation time: 0.030648 seconds
GPU offloading time: 0.639364 seconds

Verification CPU: SUCCESS
Verification GPU: SUCCESS

Sample results (first 5 elements):
c_cpu[0] = 0.0, c_gpu[0] = 0.0
c_cpu[1] = 3.0, c_gpu[1] = 3.0
c_cpu[2] = 6.0, c_gpu[2] = 6.0
c_cpu[3] = 9.0, c_gpu[3] = 9.0
c_cpu[4] = 12.0, c_gpu[4] = 12.0

4.2 Análise de Desempenho

Apesar da arquitetura massivamente paralela da GPU (V100, no cluster NPAD), a execução com offloading foi significativamente **mais lenta** do que na CPU. Fatores determinantes:

- **Overhead de inicialização:**
  Configurar o ambiente para offload na GPU consome tempo considerável.
- **Latência de comunicação:**
  A cópia dos vetores a, b e c entre a memória do host e da GPU impõe custo fixo elevado.
- **Baixa intensidade computacional:**
  A operação de soma vetorial é muito simples (1 FLOP por elemento), o que não explora a capacidade computacional da GPU.

Conclusão: O custo fixo da comunicação e configuração superou os ganhos do paralelismo, tornando a GPU **~21x mais lenta** que a CPU para este problema.

4.3 Explicação Técnica

- **Natureza da Operação (Memory-Bound):**
  A adição vetorial depende mais da largura de banda de memória do que da

capacidade de cálculo.
→ Compute-to-memory ratio muito baixo (~1 FLOP para cada 12 bytes acessados).

- **Overhead da Paralelização com OpenMP:**
  - Criação e sincronização de threads consome tempo.
  - Para $N = 10^7$, esse overhead já é perceptível mesmo na CPU.
- **Implementação Básica:**
  - Ausência de otimizações como vetorização explícita (#pragma omp simd).
  - Escalonamento padrão pode não ser o ideal para cargas leves.
  - Nenhuma tentativa de sobreposição de comunicação e computação.
- **Verificação de Corretude:**
  - Os resultados da CPU e da GPU coincidem para todos os elementos testados.
  - O mecanismo de redução (reduction(+:err)) funcionou corretamente.
  - Diferenças decimais pequenas são numericamente insignificantes.

---

5. Conclusão

O experimento demonstrou que:

5.1 Execução em CPU com OpenMP:

- Funcionou corretamente, com overhead moderado.
- Produziu resultados precisos.
- Melhor desempenho para tarefas simples e memory-bound.

5.2 Execução em GPU com Offloading:

- Verificação de corretude foi bem-sucedida.
- Performance inferior devido à baixa complexidade computacional da tarefa.
- Não justificou o custo de comunicação e inicialização do offload.

5.3 Recomendações de Melhoria:

- Para vetores menores ($N < 10^6$), preferir implementação serial.
- Explorar diferentes estratégias de escalonamento (schedule).
- Incluir vetorização explícita com #pragma omp simd.
- Considerar alinhamento de dados e uso de memória unificada.
- Testar operações com maior intensidade computacional para aproveitar o paralelismo da GPU.