Relatório: Pipelining e Vetorização

Aluno: Cristovão Lacerda Cronje

1. **Introdução**

Este relatório investiga como técnicas de paralelismo ao nível de instrução (ILP), incluindo pipelining e vetorização (SIMD), aceleram operações em laços de repetição.. O pipelining permite a execução sobreposta de múltiplas instruções, dividindo-as em estágios sequenciais (fetch, decode, execute, memory access, writeback), enquanto a vetorização (via instruções SIMD - Single Instruction Multiple Data) possibilita processar vários elementos de dados simultaneamente em uma única operação. A análise foi realizada considerando três cenários em C:

- Inicialização de vetor (sem dependências).
- Soma cumulativa (com dependência sequencial).
- Soma com múltiplas variáveis (quebra de dependências).

Foram comparados os tempos de execução com diferentes níveis de otimização do GCC (-O0, -O2, -O3). Estes ativam progressivamente técnicas de paralelismo. Em -O0, nenhuma otimização é aplicada: o código executa sequencialmente, sem pipelining ou vetorização. Com -O2, o compilador habilita pipelining básico (reordenação de instruções e loop unrolling) para reduzir stalls, mas não aplica vetorização SIMD. Já -O3 ativa pipelining agressivo e vetorização (usando instruções AVX/SSE), transformando operações escalares em operações paralelas (ex.: processando 4 inteiros por ciclo com AVX). Além disso, -O3 inclui otimizações arriscadas, como fusão de loops. Seus resultados demonstram isso claramente: a diferença entre -O2 e -O3 em laços vetorizáveis (como a soma cumulativa) deve-se diretamente ao uso de SIMD.

ILP (Paralelismo ao Nível de Instrução): Capacidade de executar múltiplas instruções simultaneamente em um único fluxo de execução. Duas formas principais:

- Pipelining: Divide a execução de instruções em estágios (fetch, decode, execute, etc.), permitindo que diferentes estágios de múltiplas instruções sejam processados ao mesmo tempo (como uma linha de montagem). É uma forma implícita de ILP: otimiza a temporalidade da execução (instruções em estágios diferentes).
- Vetorização (SIMD): Uma única instrução opera sobre vários dados simultaneamente (ex.: somar 8 inteiros de uma vez com AVX2). É é ILP explícito: otimiza a capacidade de processamento (mais dados por ciclo).

2. **Metodologia**

Código Implementado

```c
#include <stdio.h>
#include <sys/time.h>

#define SIZE 100000000

// Alocação estática dos vetores
int a[SIZE];
int b[SIZE];

double get_time() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec + tv.tv_usec * 1e-6;
}

int main() {
    double start, end;
    int sum;

    // 1. Inicialização com cálculo simples
    start = get_time();
    for (int i = 0; i < SIZE; i++) {
        a[i] = i+1;
    }
    end = get_time();
    printf("----------TESTE----------\n");
    printf("Tempo inicializacao: %.6f segundos\n", end - start);

    // 2. Soma cumulativa com dependência
    sum = 0;
    start = get_time();
    for (int i = 0; i < SIZE; i++) {
        sum += a[i];
```

```c
int main() {
    for (int i = 0; i < SIZE; i++) {
        sum += a[i];
    }
    end = get_time();
    printf("Tempo soma cumulativa (1 var): %.6f segundos\n", end - start);
    printf("Valor da Soma: %d\n", sum);

    // 3. Soma com múltiplas variáveis (quebrando dependências)
    int sum1 = 0, sum2 = 0;
    start = get_time();
    for (int i = 0; i < SIZE; i += 2) {
        sum1 += a[i];
        sum2 += a[i+1];

    }
    sum = sum1 + sum2;
    end = get_time();
    printf("Tempo soma com 2 vars: %.6f segundos\n", end - start);
    printf("Valor da Soma: %d\n", sum);
    printf("------------------------------------------------\n");

    return 0;
}
```

Amazon Q: Edit (Ctrl+I)

Compilação:

- gcc -o ilp_experiment_o0 -O0 ilp_experimento.c
- gcc -o ilp_experiment_o2 -O2 ilp_experimento.c
- gcc -o ilp_experiment_o3 -O3 ilp_experimento.c

Execução:

-O0: Sem otimizações (baseline).	-O2: Otimizações padrão (incluindo ILP e unrolling).	-O3: Máxima otimização (incluindo vetorização).
----------TESTE---------- Tempo inicializacao: 0.405091 segundos Tempo soma cumulativa (1 var): 0.282418 segundos Valor da Soma: 987459712 Tempo soma com 2 vars: 0.153819 segundos Valor da Soma: 987459712 ----------	----------TESTE---------- Tempo inicializacao: 0.195454 segundos Tempo soma cumulativa (1 var): 0.065114 segundos Valor da Soma: 987459712 Tempo soma com 2 vars: 0.047759 segundos Valor da Soma: 987459712 ----------	----------TESTE---------- Tempo inicializacao: 0.166560 segundos Tempo soma cumulativa (1 var): 0.030187 segundos Valor da Soma: 987459712 Tempo soma com 2 vars: 0.030461 segundos Valor da Soma: 987459712 ----------

3. **Resultados**

Operação	**-O0 (s)**	**-O2 (s)**	**-O3 (s)**	**Ganho (O3 vs O0)**
Inicialização (a[i] = i+1)	0.405091	0.195454	0.166560	**~2.43× mais rápido**
Soma cumulativa (sum += a[i])	0.282418	0.065114	0.030187	**~9.3× mais rápido**
Soma com 2 vars (sum1 + sum2)	0.153819	0.047759	0.030461	**~5.05× mais rápido**

"O ganho foi calculado como a razão entre os tempos de execução (T_O0 / T_O3)."

Observações

1.Inicialização (a[i] = i+1)

- •Melhoria significativa em -O2 e -O3 devido a pipelining e vetorização.
- •Sem dependências, permitindo paralelismo total.

2.Soma cumulativa (sum += a[i])

•Dependência sequencial limita otimizações em -O0.
•Em -O2 e -O3, o compilador aplica ILP e reordenação de instruções.
•Vetorização (SIMD) em -O3 acelera ainda mais.

3.Soma com múltiplas variáveis (sum1 + sum2)

•Quebra de dependências permite maior paralelismo.
•Em -O0, já é 43% mais rápido que a versão sequencial.
•Em -O3, o ganho é menor porque a vetorização já maximiza o desempenho.

4. **Análise de ILP e Vetorização**

- **Efeitos do Pipelining**

•Laço de inicialização: Alto paralelismo, pois não há dependências.
•Soma cumulativa: Stall de pipeline devido à dependência em “sum”

- **Quebra de Dependências**

•Usar sum1 e sum2 permite que duas operações sejam executadas em paralelo.
•Redução de stalls e melhor aproveitamento do ILP.

- **Impacto da Vetorização (-O3)**

•Instruções SIMD (e.g., AVX) processam múltiplos dados por ciclo.
•Exemplo: Em -O3, o laço de soma pode ser vetorizado como:

5. Conclusões

- Otimizações do compilador (-O2/-O3) aceleram significativamente laços sem dependências.
- Dependências sequenciais(Laços como sum += a[i]) limitam o ILP, mas técnicas como acumuladores múltiplos melhoram o desempenho.
- Vetorização (-O3) é mais eficaz em operações paralelizáveis.
- Recomendações:
  •Prefira -O3 para código numérico.
  •Quebre dependências manualmente em laços críticos.
  •Utilize diretivas de vetorização (e.g., #pragma omp simd, para forçar vetorização em laços complexos) quando possível.