Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

OAC – PRÁTICA 06 - RELATORIO

Desenrolando laços de repetição em código C a fim de demonstrar o impacto do pipeline.

Luiz Eduardo Garzon de Oliveira

Cascavel 24 de abril de 2023

Desenrolando laços de repetições pode-se melhorar o desempenho do pipeline porque reduzimos a quantidade de instruções de controle de fluxo que precisam ser processadas, permitindo que o pipeline execute mais instruções de cada vez e reduzindo o número de ciclos de clock necessários para executar a função.

Os passos de verificações e instruções adicionais são minimizados pois o programa executa as instruções em sequência reduzindo assim o número de instruções de controle de fluxo do laco até mesmo os acessos a memória principal podem ser reduzidos.

Códigos utilizados:

- prodEscOR.c (código original sem nenhuma alteração)

```c
double prodEscalar(double* v1, double* v2){
    double sum = 0.0;

    for (int i = 0; i < N; i++) {
        sum += v1[i] * v2[i];
    }

    return sum;
}
```

```c
double* inicializarVetor(){
    double *v;
    v = (double *) calloc(N, sizeof(double));
    for (int i = 0; i < N; i++) {
        v[i] = i;
    }
    return v;
}
```

- prodEscLP01.c (código com laco de repetição desenrolado para 4 níveis na função inicializarVetor)

```c
double* inicializarVetor(){
    double *v;
    v = (double *) calloc(N, sizeof(double));
    for (int i = 0; i < N; i+=4) {
        v[i] = i;
        v[i + 1] = i + 1;
        v[i + 2] = i + 2;
        v[i + 3] = i + 3;
    }
    return v;
}
```

- prodEscLP02.c (código com laco de repetição desenrolado para 4 níveis das funções inicializarVetor e prodEscalar)

```c
double *inicializarVetor()
{
    double *v;
    v = (double *)calloc(N, sizeof(double));
    for (int i = 0; i < N; i += 4)
    {
        v[i] = i;
        v[i + 1] = i + 1;
        v[i + 2] = i + 2;
        v[i + 3] = i + 3;
    }
    return v;
}
```

```c
double prodEscalar(double *v1, double *v2)
{
    double sum = 0.0;
    double sum1 = 0.0, sum2 = 0.0, sum3 = 0.0, sum4 = 0.0;
    for (int i = 0; i < N; i += 4)
    {
        sum1 += v1[i] * v2[i];
        sum2 += v1[i + 1] * v2[i + 1];
        sum3 += v1[i + 2] * v2[i + 2];
        sum4 += v1[i + 3] * v2[i + 3];
    }
    sum += (sum1 + sum2 + sum3 + sum4);

    return sum;
}
```

Os códigos são analisados, compilados e executados utilizando o comando gcc -O0 codigo.c -o codigo.x.

Logo após, os tempos de execuções de cada código são estimados pelo comando Measure-Command {.\codigo.exe} onde é feito o cálculo da média para cada código.

É gerado o arquivo gmon.out pelo gprof para analisar o desempenho do código em execução e os tempos das funções principais apontados pelo self-second que indica o tempo de execução gasto de uma determinada função. Todos os tempos são anotados na planilha demonstrando uma melhora constante no tempo de execução conforme os laços de repetições são desenrolados. Utilizando as otimizações do gproof (01 e 02) os resultados são melhores pois o compilador resolve de forma mais eficiente o que foi feito manualmente nos laços de repetições.

Testes	prodEscOR	prodEscLP01	prodEscLP02
1	6,2176086	3,5479164	3,5758781
2	5,6684286	5,4260144	3,357854
3	5,5340707	3,9301885	3,4073602
4	5,3196168	4,211229	3,4454248
5	5,2036005	3,6211295	3,4137045
Media	5,5340707	3,9301885	3,4137045

Coluna1	prodEscOR	Coluna2
Testes	**O1**	**O2**
1	2,779811	2,2949961
2	2,4130902	2,3715936
3	2,363873	2,4171352
4	2,5239643	2,3044363
5	2,3217738	2,6886692
Media	2,4130902	2,3715936

Codigos	prodEscOR	prodEscLP01	prodEscLP02
Dados	**Self Seconds**	**Self Seconds**	**Self Seconds**
inicializarVetor	4.03	3.17	2.89
prodEscalar	1.05	1.27	0.58