UNIVERSIDADE FEDERAL DE LAVRAS
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação
Disciplina: **Arquitetura de Computadores II - GCC123**
Trabalho Prático 2 – 1º Semestre de 2022
Professor: **Luiz Henrique A. Correia**
Data de entrega: 09/09/2022 – Valor: 15 pontos

1 Introdução

Este trabalho prático tem como objetivo estudar o paralelismo no nível de dados por meio de programação vetorial. Os atuais processadores possuem extensões dos seus conjuntos de instruções SIMD, capazes de lidar com grandes cadeias de dados (*chuncks*) em alta velocidade. A partir de 2008, os processadores x86 das fabricantes Intel e AMD incluíram nos seus conjuntos de instruções o AVX (*Advanced Vector Extensions*). O AVX inicialmente trabalhava com vetores de até 128 bits para operar funções intrínsecas de inicialização, aritméticas, permuta e embaralhamento. Em 2011 foi lançado o AVX-2, no qual o tamanho do vetores foi expandido para 256 bits, permitindo a execução de operações de multiplicação acumulada fundidas (FMA). Em 2013 o tamanho do vetor foi novamente incrementado para suportar instruções SIMD de 512 bits usando uma nova codificação de prefixo EVEX. Os processadores mais recentes da Intel que suportam vetores de 512 bits são Knights Landing, Knights Mill, Skylake-SP, Skylake-X, Cannon Lake e Ice Lake, e os da AMD são Excavator e Zen.

Os sistema operacionais que suportam as instruções SIMD do AVX são:

- Apple OS X: Suporte para AVX adicionado a partir da versão 10.6.8 (Snow Leopard).
- O DragonFly BSD adicionou suporte a partir de 2013.
- FreeBSD a partir da versão 9.1.
- Linux suporte desde a versão do kernel 2.6.30.
- Windows: suporte a partir do Windows 7 SP1, Windows Server 2008 R2 SP1, Windows 8 e no atual Windows 10.

O AVX foi desenvolvido para cálculos de ponto flutuante intensivo em aplicações multimídia, científicas e financeiras. O objetivo do AVX é aumentar o paralelismo e o rendimento em cálculos SIMD de ponto flutuante, reduzindo a carga do registrador devido às instruções não destrutivas e melhorando o desempenho do software Linux RAID (AVX2).

A descrição das operações e exemplos de programas das instruções SIMD para o AVX podem ser encontrados no livro texto [2], na Aula 21 dos slides do Capítulo 5 [3] e no tutorial de programação do AVX [4].

2 Descrição do Trabalho

Este trabalho propõe a solução de exercícios de programação usando o AVX e AVX-2, para tanto é necessário inicialmente verificar se seu computador é compatível com este conjunto de instruções. Para as distribuições Linux, basta ir no shell e digitar o comando: cat /proc/cpuinfo e nos flags verificar se contém avx e avx2.

Já para os sistemas Windows é necessário instalar um aplicativo externo ao sistema operacional. Um exemplo de aplicativo é o CoreInfo (https://docs.microsoft.com/en-us/sysinternals/downloads/coreinfo) que basta ser executado no shell para listar todas as informações do processador.

3 Avaliação

As questões do trabalho a ser desenvolvido serão avaliadas somente se o código for inédito, estiver funcionando e contiver a resolução dos exercícios com programas documentados e comentados. O código deve escrito usando programação para AVX e AVX2, por limitações de recursos computacionais, não serão aceitos códigos escritos para SIMD de 512 bits EVEX. O trabalho deverá ser realizado em grupos de 5 alunos.

Data de entrega: este trabalho deve ser postado no Campus Virtual da disciplina GCC123 até o dia **09/09/2022** e **não haverá adiamento**.

4 Exercícios

1. Escreva um programa que suporte a entrada de três vetores A, B e C. Uma instrução do AVX deve realizar uma operação de F*used Multiply Add*, conforme apresenta o código abaixo :

```
FOR j := 0 to 7
i := j*32
        IF ((j & 1) == 0)
            dst[i+31:i] := (a[i+31:i] * b[i+31:i]) - c[i+31:i]
        ELSE
            dst[i+31:i] := (a[i+31:i] * b[i+31:i]) + c[i+31:i]
        FI
ENDFOR
dst[MAX:256] := 0
```

   Usando uma instrução SIMD do AVX, escreva um código que realize a operação entre os vetores, considerando que a entrada deve ser independente e selecionada de acordo com os tipos de dados dos vetores:

   a) *double* (4 x 64bits),

   b) *float* (8 x 32bits)

   c) *integer* (8 x 32 bits).

   Apresente os resultados convenientemente na tela para cada tipo de entrada.

2. Escreva um programa que calcule o produto de duas matrizes usando funções intrínsecas. O programa deve possuir entrada para as dimensões das matrizes e verificar se o produto das matrizes é possível. Os elementos das matrizes serão números reais de precisão dupla (dimensão máxima da matriz 4x4 para 256 bits).

3. A cifra de César é um dos métodos mais simples de criptografia [1]. É um tipo de cifra de substituição na qual cada letra do texto é substituída por outra, por um deslocamento fixo de posições das letras do alfabeto. Dado um arquivo texto, realize a cifragem de César usando instruções de permutação e apresente o arquivo de saída cifrado. Considere que o processo possa ser revertido e que o arquivo original possa ser recuperado e apresentado.

4. Escreva um programa que calcule a operação do loop DAXPY. Os vetores X e Y devem ser lidos da memória usando a instrução _mm256_load_ps [5] e o valor do escalar deve ser lido do console. O resultado da operação deve ser armazenado na memória usando a instrução _mm256_store_ps e apresentado convenientemente na tela.

```
for (int i=0; i<8; i++)
   res[i] = a*X[i]+Y[i]
```

5. Ao adicionar ou subtrair vetores inteiros, é importante verificar a diferença entre as funções _add_ e _sub_ e as funções _adds_ _subs_. O **s** extra significa saturação, que é produzida quando o resultado requer mais memória do que o o vetor pode armazenar. Crie um programa que exemplifique esta situação ao somarmos dois vetores A e B, no qual seus elementos são definidos como _mm256_add_epi8 e _mm256_subs_epi16. Apresente o resultado convenientemente na tela.

6. Os valores dos vetores A, B e C são do tipo *double* e devem ser carregado na memória via terminal. Desenvolva um programa que dado que os vetores A, B e C foram carregados, e usando uma instrução FMA (*Fused Multiply and Add*) realize a operação:

```
for (int i=0; i<4; i++) {
 d[i] = a[i]*b[i]
 d[i] = d[i]+c[i]
}
```

References

[1] Cifra de César.
`https://pt.wikipedia.org/wiki/Cifra_de_Cesar`.
Acessado em Agosto de 2022.

[2] Hennessy, John L. and Patterson, David A.. Computer Architecture, Fifith Edition: A Quantitative Approach. Morgan Kaufmann Publishers Inc., 2012.

[3] Correia, Luiz H. A.. Notas de aula da disciplina Arquitetura de Computadores 2. Capítulo 4. Universidade Federal de Lavras. Acessado em Agosto de 2022.

[4] SCARPINO, Matt. Crunching Numbers with AVX and AVX2. Code Project official page.
`https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX`.
Acessado em Agosto de 2022.

[5] Intel Corporation. Intrinsic Guide.
`https://software.intel.com/sites/landingpage/IntrinsicsGuide/`.
Acessado em Agosto de 2022.