



INF01113

Organização De Computadores B

## Segundo Trabalho

*Benchmarks de Organizações de Computadores*

Bruno Alexandre Hofstetter Bourscheid (00550177)

Fernando Longhi de Andrade (00580366)

Luiz Augusto Ponzoni Schmidt (00580108)

Miguel Dutra Fontes Guerra (00342573)

Pedro Lubaszewski Lima (00341810)

Turma B

## ***Bubble Sort***

- Tamanho da entrada: 1.400;
- Complexidade:  $O(n^2)$ ;
- Algoritmo de ordenação simples;
- Muitas comparações e trocas;
- Baixo uso de memória;
- Pouco paralelismo (dependência entre as instruções).

## ***Fast Fourier Transform (FFT)***

- Tamanho da entrada: 15.000;
- Complexidade:  $O(n \log(n))$ ;
- Operações matemáticas complexas;
- Várias instruções de ponto flutuante;
- Acesso regular à memória;
- Alto paralelismo.

## ***Binary Search***

- Tamanho da entrada: 1.200.000;
- Complexidade:  $O(\log(n))$  (vetor ordenado);
- Poucas instruções;
- Bastante acesso à memória (depende da *cache*);
- Depende da latência de acesso à memória.

1

## Tamanho da *Cache* L1

Representa a quantidade de dados mais próximos da *CPU*. Isso pode afetar o desempenho de programas que dependam de dados com localidade temporal e espacial, afetando o número de *cache misses*.

2

## Associatividade da *Cache* L1

É o número de blocos de cada conjunto de memória *cache*. Também deve afetar a questão de programas com dados que apresentam localidade espacial, através de mais ou menos *cache misses*.

3

## Tamanho do *Fetch Buffer*

Um maior tamanho de *fetch buffer* implica em mais instruções sendo processadas simultaneamente. Ou seja, deve afetar algoritmos que apresentem maior paralelismo de instruções.

# Configuração Fixa

# Resultados mudando o tamanho de *cache* L1

# Análise dos Resultados

Para testar a solução, foram criadas as seguintes entradas:

- $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix};$
- $B = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix};$
- $B_{NULL} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix};$
- $C = \begin{bmatrix} 121 & 10 \\ 50 & 3 \end{bmatrix}.$

Esperando as seguintes saídas:

- $R_1 = \begin{bmatrix} 128 & 15 \\ 70 & 16 \end{bmatrix},$  com  $A$ ,  $B$  e  $C$  como entradas;
- $R_2 = \begin{bmatrix} 121 & 10 \\ 50 & 3 \end{bmatrix},$  com  $A$ ,  $B_{NULL}$  e  $C$  como entradas.

# Resultados mudando a associatividade da cache L1

Como a saída  $doutr = [128, 15, 70, 16]$ , no modo da memória *read first*, então foi validado que  $R_1 = \begin{bmatrix} 128 & 15 \\ 70 & 16 \end{bmatrix}$ .

# Análise dos Resultados

Como a saída  $doutr = [121, 10, 50, 3]$ , no modo da memória *read first*, então foi validado que  $R_2 = \begin{bmatrix} 121 & 10 \\ 50 & 3 \end{bmatrix}$ .



# Resultados mudando o tamanho do *fetch buffer*

# Análise dos Resultados

# Algoritmo em C