

Sistemas Hardware-Software

Mutirão C

Igor

2019-02-11

Apresentação

Sistemas Hardware-Software

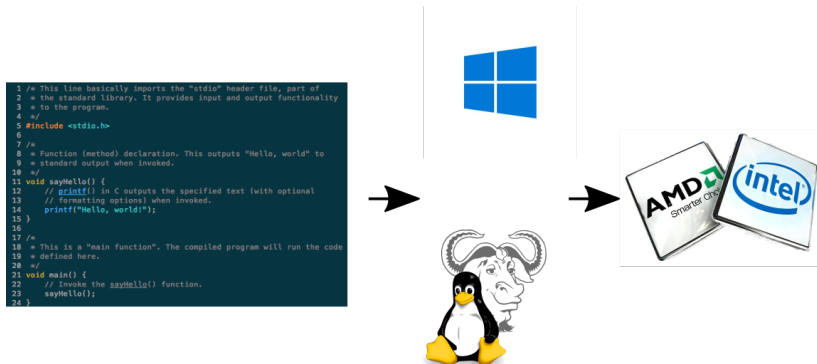


Figure 1: Estudamos a interação entre o programa, o sistema operacional e o hardware

Sistemas Hardware-Software

Objetivos de hoje:

1. entender como as capacidades do hardware alteram o desempenho de um programa;
2. entender como modificar um programa para aproveitar melhor o hardware.

Sistemas Hardware-Software

Vamos continuar com o embarcado:

1. Abram o projeto HW-SW na solução da aula
2. Rodem o projeto. O quê o botão faz?
3. Localizem a função `processImage(..)` no arquivo `main.c`.

Sistemas Hardware-Software

```
for (j=1;j<imgW-1;j++){  
    for (i=1;i<imgH-1;i++){  
        val = (int) 4 * imgIn[i][j]  
            - imgIn[i-1][j]  
            - imgIn[i+1][j]  
            - imgIn[i][j-1]  
            - imgIn[i][j+1];  
  
        imgOut[i][j] = abs(val);  
    }  
}
```

Sistemas Hardware-Software

Vamos rodar o programa agora com duas configurações diferentes:

1. loop `ji` :

2. loop `ij` :

No segundo vamos só inverter a ordem dos loops e rodar de novo.

O que está acontecendo?

As CPUs presentes em PCs (e muitos sistemas embarcados) são otimizadas para acesso **sequencial** ou **frequente** usando a memória *cache*.

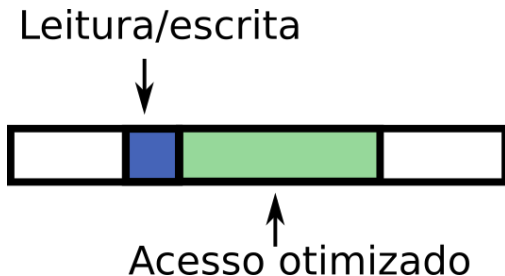
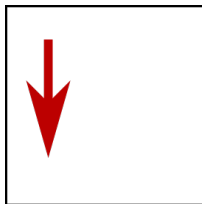


Figure 2: Acesso é otimizado quando acessamos as posições de um vetor em ordem crescente

O quê está acontecendo? - loop ji

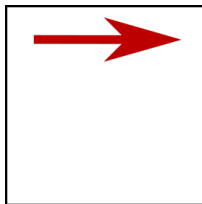
```
for (j=1;j<imgW-1;j++){  
    for (i=1;i<imgH-1;i++){  
        val = (int) 4 * imgIn[i][j]  
            - imgIn[i-1][j]  
            - imgIn[i+1][j]  
            - imgIn[i][j-1]  
            - imgIn[i][j+1];  
  
        imgOut[i][j] = abs(val);  
    }  
}
```



imgOut[i][j] = abs(val); Figure 3:

O quê está acontecendo? - loop ij

```
for (i=1;i<imgH-1;i++){  
    for (j=1;j<imgW-1;j++){  
        val = (int) 4 * imgIn[i][j]  
            - imgIn[i-1][j]  
            - imgIn[i+1][j]  
            - imgIn[i][j-1]  
            - imgIn[i][j+1];  
  
        imgOut[i][j] = abs(val);  
    }  
}
```



imgOut[i][j] = abs(val); Figure 4:

O que está acontecendo - O cache

A ordem dos loops que privilegia o acesso sequencial (por linha) tem desempenho superior!

O cache é uma memória pequena mas muito rápida usada para acelerar o desempenho de acesso à memória RAM

Cache	
Cache size	16 KB for instruction cache, 16 KB for data cache
Number of sets	256 for instruction cache, 128 for data cache
Number of ways	2 for instruction cache, 4 for data cache
Number of words per cache line	8 words (32 bytes)
ECC on Cache	Embedded

Figure 5: Configuração do cache do μC

O cache

O cache de dados possui 16kb. Cada ponto tem tamanho *1 byte*.

1. Quantos bytes cabem no cache?

O cache

O cache de dados possui 16kb. Cada ponto tem tamanho *1 byte*.

1. Quantos bytes cabem no cache? 16384

O cache

O cache de dados possui 16kb. Cada ponto tem tamanho *1 byte*.

1. Quantos bytes cabem no cache? 16384
2. Quantas linhas da nossa imagem (320×320) cabem no cache?

O cache

O cache de dados possui 16kb. Cada ponto tem tamanho *1 byte*.

1. Quantos bytes cabem no cache? 16384
2. Quantas linhas da nossa imagem (320×320) cabem no cache?
51

O cache

O cache de dados possui 16kb. Cada ponto tem tamanho *1 byte*.

1. Quantos bytes cabem no cache? 16384
2. Quantas linhas da nossa imagem (320×320) cabem no cache?
51
3. Qual é o tamanho da maior imagem quadrada que cabe inteira no cache?

O cache

O cache de dados possui 16kb. Cada ponto tem tamanho *1 byte*.

1. Quantos bytes cabem no cache? 16384
2. Quantas linhas da nossa imagem (320×320) cabem no cache?
51
3. Qual é o tamanho da maior imagem quadrada que cabe inteira no cache? 128×128 .

O cache

O cache de dados possui 16kb. Cada ponto tem tamanho *1 byte*.

1. Quantos bytes cabem no cache? 16384
2. Quantas linhas da nossa imagem (320×320) cabem no cache?
51
3. Qual é o tamanho da maior imagem quadrada que cabe inteira no cache? 128×128 .

Nossa imagem possui $320 \times 320 = 100kb$. Ela é *pequena* perto do cache do μC (16kb)

O que está acontecendo - O cache

Vamos agora testar nossa hipótese: podemos desligar o cache do μC .

Abra o arquivo `conf_board.h` e comente a linha abaixo:

```
#define CONF_BOARD_ENABLE_DCACHE
```

Rode de novo os experimentos:

1. loop `ji` :
2. loop `ij` :

Sistemas Hardware-Software

Vamos mais longe: podemos habilitar também o *cache de instruções*. Ao invés de buscar cada instrução na memória uma a uma podemos buscar um conjunto grande delas de uma vez só.

Abra o arquivo `conf_board.h` e comente a linha abaixo:

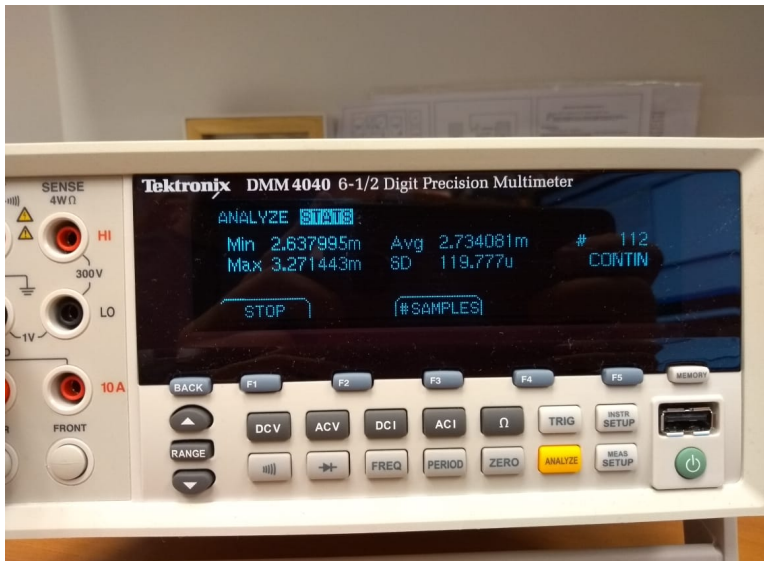
```
#define CONF_BOARD_ENABLE_ICACHE
```

1. loop `ji` :

2. loop `ij` :

Sistemas Embarcados (de novo)

Vamos medir de novo o consumo de energia, agora com todos os caches ligados.



Sistemas Hardware-Software

Por que há diferença entre os loops j_i e i_j ?

► Processo de compilação: Otimizações

- Não é linha a linha
- Extremamente sensível ao código e à sequência de comandos
- Instruções são escolhidas conforme seu desempenho
- Pequenas mudanças podem gerar grandes diferenças

Sistemas de Hardware e Software

```
val = (int) 4 * imgIn[i][j]
        - imgIn[i-1][j]
        - imgIn[i+1][j]
        - imgIn[i][j-1]
        - imgIn[i][j+1];
```

ji

```
ldrb.w  r8, [r2, #320]
ldrb     r3, [r2]
rsb r3, r3, r8, lsl #2
ldrb.w  r8, [r2, #640]
sub.w   r3, r3, r8
ldrb.w  r8, [r2, #319]
sub.w   r3, r3, r8
ldrb.w  r8, [r2, #321]
sub.w   r3, r3, r8
```

ij

```
ldrb     r7, [r2, #1]
ldrb     r3, [r5, #1]!
rsb r3, r3, r7, lsl #2
ldrb     r7, [r4, #1]!
subs     r3, r3, r7
ldrb     r2, [r2]
subs     r3, r3, r2
ldrb     r2, [r1, #1]
subs     r3, r3, r2
```

Sistemas de Hardware e Software

- ▶ Representação de dados na CPU
- ▶ Representação de programas na CPU
 - ▶ Assembly Intel x64
- ▶ Conceitos de Sistemas Operacionais
 - ▶ Processos e Sinais
 - ▶ Hierarquia de memória
 - ▶ Entrada e saída
 - ▶ Programação concorrente