Módulo 7

Super Escalaridade

Copie para a sua directoria local o ficheiro `/share/acomp/P7-PROD.zip`. Faça o `unzip` e verifique a directoria `P7-PROD`. Construa o executável.

Note que o algoritmo a executar calcula o produto de todos os elementos de um vector com 256 M valores representados em vírgula flutuante, precisão simples, conforme reproduzido abaixo (ver ficheiro `prod.c`):

```
float prod1  (float *arr, int n) {
        int i;
        float prod=1.f;

        for (i = 0; i < n; ++i) {
                prod *= arr[i];
        }
        return prod;
}
```

A micro-arquitectura Ivy Bridge da Intel, usada nos microprocessadores utilizados nesta Unidade Curricular, inclui 6 unidades funcionais, conforme ilustrado na figura do anexo 1. No limite estes processadores podem terminar 6 instruções por ciclo do relógio.

Exercício 1 – Execute a versão 1 de `prod()` e preencha a respectiva linha na Tabela 1. O comando a usar é:

```
> sbatch prod.sh 1
```

Cada iteração do ciclo implica a realização de 5 operações: leitura (*load*) do valor do vector, multiplicação FP, adição inteira do contador do ciclo, comparação com `n` e salto condicional.

O valor do CPI reportado parece-lhe adequado para os recursos disponibilizados pelo *hardware*? Qual será o principal obstáculo à utilização simultânea de mais unidades funcionais (paralelismo)?

A técnica de *loop unrolling* consiste em fazer várias cópias do núcleo de um ciclo (*loop*).

Um exemplo de *unrolling* de grau 2, é apresentado na tabela abaixo.

no unroll	***unroll-2***

`for (k=0; k<n ; k++) {`
`  a[k] = a[k] * 2.0;`
`}`
	`for (k=0; k<n ; k+=2) {`
	`  a[k] = a[k] * 2.0;`
	`  a[k+1] = a[k+1] * 2.0; }`

O *unrolling* nem sempre resulta num aumento de desempenho, uma vez que o código gerado será mais longo. Mas exibe potencial para aumentar o desempenho por duas vias:

1. redução do número de instruções executadas, uma vez que o ciclo é iterado menos vezes;

2. disponibilização de mais instruções para lançamento simultâneo (*issue*) nos diferentes *pipelines* (unidades funcionais).

Exercício 2 – Altere a função `prod2()` para explorar *loop unrolling* nível 2, conforme apresentado a seguir:

```
float prod2  (float *arr, int n) {
        int i;
        float prod=1.f;

        for (i = 0; i < n; i+=2) {
                prod *= arr[i];
                prod *= arr[i+1];
        }
        return prod;
}
```

Execute a versão 2 e preencha a respectiva linha na Tabela 1. O comando a usar é:

```
> sbatch prod.sh 2
```

Analise cuidadosamente as variações (relativamente a `prod1()`) no `Texec`, `#I` e `CPI`. O que aconteceu? Porque aumentou o CPI? Terá aumentado o paralelismo? Porque diminuíram o número de instruções?

A associatividade da multiplicação (que não é verdadeira quando os valores são representados com precisão finita, aspecto que ignoraremos neste módulo), permite agrupar subconjuntos de multiplicações, acumular o produto em variáveis diferentes (contornando assim a dependência) e calculando o produto final depois do ciclo.

Exercício 3 – Altera a função `prod3()` para explorar *loop unrolling* nível 2, mas separando o cálculo dos produtos, conforme apresentado a seguir:

```
float prod3  (float *arr, int n) {
        int i;
        float prod[2]={1.f, 1.f}, prodf;

        for (i = 0; i < n; i+=2) {
                prod[0] *= arr[i];
                prod[1] *= arr[i+1];
        }
        prodf = prod[0] * prod[1];
        return prodf;
}
```

Execute a versão 3 e preencha a respectiva linha na Tabela 1. O comando a usar é:

```
> sbatch prod.sh 3
```

Analise cuidadosamente as variações (relativamente a `prod2()`) no `Texec`, `#I` e `CPI`. O que justifica o ganho no CPI?

Exercício 4 – Altera a função `prod4()` para explorar *loop unrolling*, tal como `prod3()`, mas agora realizando 4 produtos por iteração, mantendo a independência entre o cálculo destes produtos.

Execute a versão 4 e preencha a respectiva linha na Tabela 1. O comando a usar é:

```
> sbatch prod.sh 4
```

O compilador pode ser instruído para fazer *loop unrolling* se tal for previsto como vantajoso pelo modelo de execução mantido pelo GCC. A opção de *loop unrolling* pode ser activada na linha de comando (caso em que se aplica a todo o código) ou dentro do código, permitindo seleccionar a que secções do código se aplica. `#pragma GCC optimize("unroll-loops")` activa esta opção e `#pragma GCC reset_options` desactiva-a.

Exercício 5 – A função `prod5()` activa a opção de *loop unrolling* e inclui um ciclo aninhado para facilitar a tarefa ao compilador no sentido de descobrir qual o ciclo a desenrolar, conforme apresentado a seguir:

```
#define UNROLL 4
#pragma GCC optimize("unroll-loops")

float prod5  (float *arr, int n) {
        int i, u;
        float prod[UNROLL], prodf=1.f;

        for (u=0 ; u< UNROLL ; u++) prod[u] = 1.f;
        for (i = 0; i < n; i+=UNROLL) {
                for (u=0 ; u< UNROLL ; u++)
                        prod[u] *= arr[i+u];
        }
        for (u=0 ; u< UNROLL ; u++) prodf *= prod[u];
        return prodf;
}
```

Execute a versão 5 e preencha a respectiva linha na Tabela 1. O comando a usar é:

```
> sbatch prod.sh 5
```

A que atribui o ganho no tempo de execução relativamente a `prod4()`? E relativamente a `prod2()`?

	T_{exec} (ms)	#I (M)	CPI
`prod1()`			
`prod2()`			
`prod3()`			
`prod4()`			
`prod5()`			

Tabela 1 -Métricas para produto de vector

GEMM

Copie o ficheiro `/share/acomp/GEMM-Lab2.zip` para a sua directoria, extraia o seu conteúdo e construa o executável na directoria `Lab2`. Verifique que a função `gemm4()` faz o *unroll* explícito do ciclo *for* mais aninhado, enquanto na função `gemm5()` a responsabilidade do *unroll* é passada ao compilador (com uma pequena ajuda do programador).

Exercício 6 – Preencha a Tabela 2. Comparando os resultados de `gemm4()` e `gemm5()` com `gemm3()`, comente a que se deve o ganho obtido. Em particular, indique se se deve a uma melhor exploração da super-escalaridade **e/ou** redução do número de acessos à memória **e/ou** melhor exploração da hierarquia da memória **e/ou** redução do número total de instruções.

n		T (ms)	#I (G)	CPI	LD_INS + SR_INS (G)	L1_DCM (M)
512	`gemm3 ()`					
	`gemm4 ()`					
	`gemm5 ()`					

Tabela 2 - GEMM e *loop unrolling*

Anexo 1 – Micro-Arquitectura Intel Sandy Bridge (~Ivy Bridge)