Universidade do Minho

PL08 : Auto vectorização

Descarregue o código disponibilizado para o módulo PL08.

Note que para gerar os executáveis com níveis de optimização O2 e O3, basta escrever:

`./mymake.sh`

Este comando gerará também o ficheiro **`vec_main.s`** contendo o assembly correspondente ao nível de optimização O3.

Ao longo deste módulo iremos concentrar a nossa atenção na função **`loop()`** que se encontra em **`vec_main.cpp`**.

Exercício 1

1. Inspeccione a função **`loop()`** e indique se pensa tratar-se de código vectorizável.
2. Construa os executáveis para os dois níveis de optimização seguindo as instruções acima. Execute e verifique os tempos de execução. Calcule o ganho da versão O3 relativamente à versão O2.
3. Dos 3 factores que influenciam o tempo de execução de um programa (CPI, #I e frequência) onde é que as se ganhou com a vectorização?
4. Leia cuidadosamente o código da função **`loop()`** em **`vec_main.s`**, verificando que a sequencia de operações especificadas em C é de facto realizada pela sequência de instruções vectoriais.

Para cada um dos seguintes exercícios:

a) dê o seu parecer sobre se o código apresentado é vectorizável, justificando;
b) verifique o seu parecer inspeccionando o código de **`loop ()`** gerado em **`vec_main.s`**
c) nos casos em que o código vectoriza compare o desempenho das versões escalar e vectorial, calculando o ganho e comparando o CPI e #I.

Exercício 2

Altere a função **`loop()`** para:

```
for (i=0 ; i<SIZE ; i+=2) {
  c[i] = a[i]* a[i]* a[i] + 10.0f / a[i] + 100.f / (a[i]*a[i]); }
```

Exercício 3

Altere a função **`loop()`** para:

```
for (i=0 ; i<SIZE ; i++) {
  if (a[i] <100.f) {
    c[i] = a[i]* a[i]* a[i] + 10.0f / a[i] + 100.f / (a[i]*a[i]);
  } else {
    c[i] = a[i]* a[i]* a[i] + 20.0f / a[i] + 100.f / (a[i]*a[i]);
```

Exercício 4

Altere a função **`loop()`** para:

```
for (i=0 ; i<SIZE ; i++) {
  c[i] = powf(a[i], 3.0f) + 10.0f / a[i] + 100.f / (a[i]*a[i]); }
```

Exercício 5

Altere a função **loop()** para:

```
for (i=1 ; i<SIZE ; i++) {
  c[i] = a[i]*a[i]*a[i] + 10.0f / c[i-1] + 100.f / (a[i]*a[i]); }
```

Exercício 6

Altere a função **loop()** para:

```
for (i=0 ; i<SIZE-1 ; i++) {
  c[i] = a[i]*a[i]*a[i] + 10.0f / c[i+1] + 100.f / (a[i]*a[i]); }
```

Exercício 7

Considere as duas versões apresentadas abaixo da função **loop()**. A versão a) usa uma estrutura de dados do tipo AoS, enquanto a versão b) usa uma SOA. Qual delas vectoriza e porquê?

```
   a)
typedef struct { float a, b ,c, pad} MYDATA;

MYDATA d[SIZE] __attribute__ ((aligned(16)));


for (i=0 ; i<SIZE ; i++) {
 d[i].c = d[i].a * d[i].a * d[i].a + 10.0f / d[i].a + 100.f / (d[i].a * d[i].a);
}

   b)
typedef struct {
  float a[SIZE] __attribute__ ((aligned(16)));
  float b[SIZE] __attribute__ ((aligned(16)));
  float c[SIZE] __attribute__ ((aligned(16)));
} MYDATA;

MYDATA d;

for (i=0 ; i<SIZE ; i++) {
 d.c[i] = d.a[i] * d.a[i] * d.a[i] + 10.0f / d.a[i] + 100.f / (d.a[i] * d.a[i]);
}
```

Exercício 8

Indique e verifique se cada uma das versões seguintes da função **loop()** vectoriza:

```
   a) for (i=1 ; i<SIZE-1 ; i++) {
        a[i-1] = 100.f / (a[i]*a[i]); }

   b) for (i=1 ; i<SIZE-1 ; i++) {
        a[i-1] = 100.f / (a[i-1]*a[i-1]); }

   c) for (i=0 ; i<SIZE-1 ; i++) {
        b[i] = 100.f / (a[i]*a[i]);
        c[i] = b[i-1] + a[i];
      }
```