Universidade do Minho

PL07 : Princípios básicos de vectorização

Descarregue o código disponibilizado no BB para este módulo. Note que a função `main()` utiliza o PAPI para fazer um conjunto de medições do desempenho da execução da função `loop()`:

```
void loop () {  int i;

 for (i=0 ; i<SIZE; i++) {    c[i] = a[i] + b[i]; } }
```

Nota ainda que os vectores de inteiros **a**, **b** e **c** são globais a todas as funções do módulo `vec_main.cpp`.

Exercício 1

1. Compile este código usando a `Makefile` disponibilizada e assegurando que o nível de optimização é O2 (`CCFLAGS = -O2 -march=native -ftree-vectorizer-verbose=2 –fno-inline).`
2. Com o nível de optimização usado o compilador não gera código vectorial. Confirme gerando o `assembly` do módulo `vec_main.cpp` e localizando o código correspondente a este ciclo. Use o comando:
   `g++ -S -O2 –march=native –ftree-vectorizer-verbose=2 –fno-inline vec_main.cpp`
3. Execute o programa `vec_main` e preencha na tabela abaixo as métricas de desempenho correspondentes.
4. Altere o tipo de dados dos vectores `a`, `b` e `c` para `float` e recompile o código.
5. Repita o ponto 3) acima para este tipo de dados, preenchendo as células apropriadas na tabela.
6. Recompile este código alterando a `Makefile` para que o nível de optimização seja O3 (isto é, use a linha `CCFLAGS = -O3 -march=native -ftree-vectorizer-verbose=2 –fno-inline).`
7. O compilador terá agora gerado código vectorial (os comentários impressos na consola pela opção `verbose` do vectorizador assim o demonstram). Confirme gerando o `assembly` do módulo `vec_main.cpp` e localizando o código correspondente a este ciclo. Use o comando:
   `g++ -S –O3 –march=native –ftree-vectorizer-verbose=2 –fno-inline vec_main.cpp`
8. Execute o programa `vec_main` e preencha na tabela abaixo as métricas de desempenho correspondentes. Lembre-se que o tipo de dados dos vectores é agora `float`.
9. Mude o tipo de dados para `int` e repita as alíneas 7) e 8).
10. Analise os resultados da tabela e comente:
    a. os valores reportados para o número de instruções;
    b. os valores do CPI e como é que isto tem impacto no tempo de execução.

	-O2			**-O3**		
	Texec	**#I**	**CPI**	**Texec**	**#I**	**CPI**
int						
float						

Exercício 2

A programação estruturada não se compagina com a utilização de variáveis globais e funções definidas no mesmo módulo. Note que o código disponibilizado inclui o ficheiro `loop.cpp` com a definição da função `loop()` com parâmetros. Altere o código em `vec_main.cpp` e a `Makefile` de forma a que seja esta função `loop()`, contida em `loop.cpp`, a ser utilizada.

1. Compile o código usando a opção `-O3`. Qual o comentário produzido pelo vectorizador? Esclareça o que entende por *versioning* e por *aliasing*.
2. a *keyword* `__restrict__` do C é usada em declaração de apontadores para garantir ao compilador que durante a existência desse apontador apenas ele será usado para aceder o objecto para o qual aponta. O compilador tem assim uma garantia de que não existe *aliasing*. Modifique a assinatura da função `loop()` incluindo o qualificador `__restrict__` e verifique que já não são geradas múltiplas versões do código do ciclo.
3. Usando o comando
   `g++ -S -O3 -march=native -ftree-vectorizer-verbose=2 -fno-inline vec_main.cpp`
   verifique o código *assembly* gerado.