

## Módulo 9 Auto-Vectorização

Copie o ficheiro `/share/acomp/LOOP-P09.zip` para a sua directoria, faça o unzip e gere os executáveis com nível de optimização O2 e 2 opções diferentes: `novect-kerne1` não inclui vectorização, enquanto `vec-kerne1` inclui a opção de auto-vectorização pelo compilador. Para esse efeito, basta escrever `./mymake.sh`

Este comando gerará também os ficheiros `vec-kerne1.s` e `novect-kerne1.s` contendo o *assembly* correspondente aos diferentes níveis de optimização.

Ao longo deste módulo iremos concentrar a nossa atenção nas funções `loop1()` a `loop6()`, `loop_AoS()` e `loop_SoA()` que se encontram em `vec-kerne1.c`.

**Exercício 1** - Para cada uma das funções `loop1()` a `loop6()` e `loop_AoS()` examine o respectivo código e indique justificando, na coluna “Obs.” da Tabela 1, se pensa tratar-se, ou não, do código vectorizável pelo compilador. As linhas correspondentes a `loop3-V2()` e `loop_SoA()` serão preenchidas nos exercícios posteriores.

Preencha a para ambas as versões, cuja principal diferença será a vectorização. Note que pode verificar a vectorização consultando os ficheiros `.s`. Para executar cada versão da função `loop()` use o comando `qsub` conforme indicado abaixo para a versão `loop1()`. Este comando executará ambos os executáveis, isto é, `novect-kerne1` e `vec-kerne1`. O argumento usado para seleccionar a versão pode ser consultado na coluna V da Tabela 1:

```
qsub -F “1” loop.sh
```

**Exercício 2** - Analize a função `loop3()`. Porque é que não vectoriza? Consegue sugerir uma alteração ao código desta função de forma que vectorize? Se sim, altere-a e preencha a linha `loop3-V2()` acima.

(Resposta LPS: usar `(?:)` em vez de `if`:

```
c[i] = a[i]*a[i]*a[i] + (a[i]<100.f ? 10.0f:20.0f ) /a[i] + 100.f/(a[i]*a[i]);
```

**Exercício 3** - Escreva a função `loop_SoA()`, para que realize os mesmos cálculos que `loop_AoS()` mas usando uma estrutura de *arrays*.

Note que:

- o tipo de dados `My_SoA` já está definido em `vec-kerne1.h`:  
`typedef struct {float *a, *c} MY_SoA;`
- a variável `SoA` é declarada em `main.h` e os respectivos vectores (`a` e `c`) devidamente inicializados.

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

### Exercício Complementar

Considere a função `loop5()`, que não vectoriza devido a uma dependência RAW. Consegue reorganizar as computações de forma a que esta possa vectorizar parcialmente?

Sugestão: Pode separar o ciclo em dois ciclos, um deles sem dependências!

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

func	v	opt	T <sub>exec</sub> (ms)	#I (M)	CPI	#VEC_SP (M)	Obs .
loop1 ()	1	no	283	550	1.3	0	Vectoriza!
		vec	72	81	2.3	350	
loop2 ()	2	no	142	275	1.3	0	Não vectoriza! stride=2 , logo acessos à memória não são consecutivos
		vec					
loop3 ()	3	no	273	650	1.1	0	Não vectoriza devido à estrutura condicional
		vec					
loop3-V2 ()	3	no					Vectoriza usando (?) em vez de if ()
		vec	71	93	2.0	400	
loop4 ()	4	no	5391	17229	0.8	0	Não vectoriza devido à invocação de uma função.
		vec					
loop5 ()	5	no	290	600	1.3	0	Não vectoriza pois existe uma dependência RAW (Read After Write) entre c[i] de iterações consecutivas
		vec					
loop6 ()	6	no	281	600	1.2	0	Vectoriza pois a dependência entre c[i]'s consecutivos é WAR (Write After Read)
		vec	71	100	1.8	400	
loop_AoS ()	7	no	282	550	1.3	0	Não vectoriza, devido ao facto de os AoS.a[i] e os c[i] não serem consecutivos
		vec					
loop_SoA ()	8	no	282	550	1.3	0	Vectoriza
		vec	72	81	2.3	350	

Tabela 1