
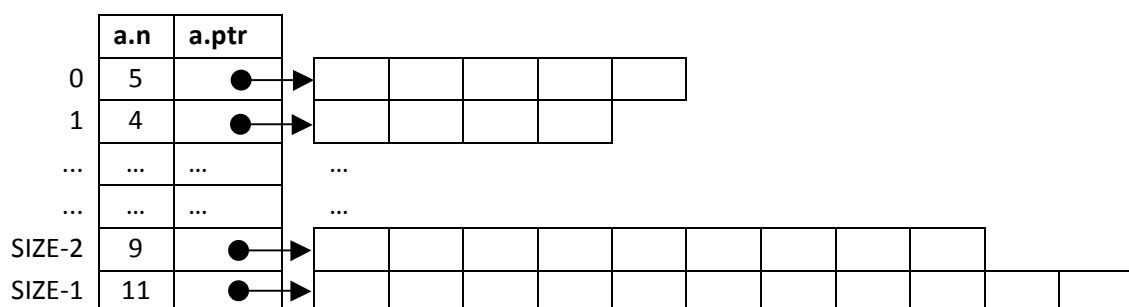
 Universidade do Minho	<b>Módulo 13</b> <b>OpenMP e Vectorização</b> <b>Desafio</b>	
------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------	-------------------------------------------------------------------------------------

Descarregue o código disponibilizado na plataforma de eLearning para este módulo.

Note que a função principal invoca 3 funções:

- *initialize()* que cria e inicializa a estrutura de dados descrita abaixo;
- *process()* que processa os dados e sobre a qual se pretende que aplique as otimizações de vectorização e *multithreading*;
- *release()* que liberta a memória alocada na fase de inicialização.

A estrutura de dados utilizada é um vector de tamanho fixo (dado por *SIZE*). Cada elemento deste vector tem um campo (*ptr*) que aponta para um outro vector com elementos do tipo de dados *float*. Uma vez que o número de elementos de cada um dos vectores de *floats* é variável, cada elemento do vector principal guarda também o número de elementos do vector de *floats* que lhe está associado (campo *n*). A figura abaixo ilustra essa estrutura de dados.



Note que embora o tamanho dos vectores de *floats* varie ao longo do vector principal, e que esse tamanho seja mesmo determinado aleatoriamente, há uma tendência para que o tamanho (dos vectores de *floats*) cresça com o índice do vector principal. Isto é, os elementos de maior índice do vector principal apontarão, com maior probabilidade, para vectores com mais *floats*.

A função *process()* processa cada um destes *floats*. Pretende-se que aplique optimizações relacionadas com vectorização e *multithreading* para reduzir o tempo de execução desta função. Abaixo apresentam-se os tempos de execução obtidos nas máquinas do Laboratório e compilando com:

```
gcc -O3 -fopenmp -ftree-vectorizer-verbose=2 -march=native PL13.c -o PL13
```

O tempo de execução com o código original foi de 3870 ms (milissegundos). O tempo de execução com o código optimizado (vectorização e *multithreading* usando OpenMP) foi de 1042 ms. Consegue desenvolver uma versão que exiba um desempenho semelhante ou ainda melhor?