Computação Paralela

Trabalho – OpenMP : paradigma de memória partilhada

João Luís Sobral

14-Novembro-2018

Trabalho – OpenMP

- Objetivo:
  - Avaliar a aprendizagem do paradigma de computação paralela baseada em memória partilhada (OpenMP)

- Parâmetros da avaliação

	Peso
Qualidade da implementação sequencial (opt, SIMD)	20%
Desenho e implementação da versão paralela (+otimização)	20%
Qualidade (e quantidade) da experimentação (dados de entrada, métricas, medições)	20%
Análise dos resultados (justificação para valores obtidos)	20%
Relatório	20%

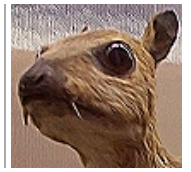
Trabalho – OpenMP

- Grupos:
  - 2 elementos (ou 1)
- Entrega:
  - Envio por email até às 24h do dia 11(?)-Dez-2018
    - Relatório (**máx 6 páginas, pdf**) + código
    - Email (jls@di.uminho.pt)
    - Apresentação (pdf)
  - Apresentação do trabalho dia 12-Dez-2018
- Escolha dos trabalhos
  - Livre entre os vários disponíveis
  - Sujeito a arbitragem!!!!

Trabalho – OpenMP

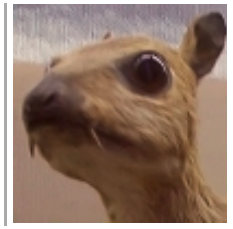
- 1ª Opção: Convolução para melhorar a qualidade de uma imagem

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

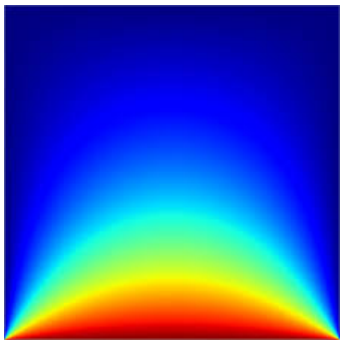


- Algoritmo:

```
for each image row in input image:
   for each pixel in image row:

      set accumulator to zero

      for each kernel row in kernel:
         for each element in kernel row:

            if element position  corresponding* to pixel position then
               multiply element value  corresponding* to pixel value
               add result to accumulator
            endif

      set output image pixel to accumulator
```

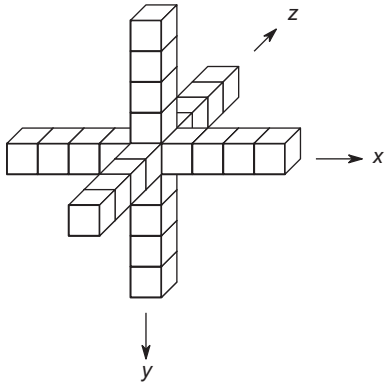
Trabalho – OpenMP

- 2ª Opção: Simulação do processo de propagação de ondas sonoras usando um “stencil” para implementar o operador Laplaciano



- Algoritmo:

```
One_iteration() {

  for(int i=1; i<N-1; i++) {

    for(int j=1; j<N-1; j++)

      G1[i][j]=  C[0]*G2[i][j]+

                 C[1]*G2[i][j+1]+
                 C[2]*G2[i][j+2]+
                 C[3]*G2[i][j+3]+
                 C[4]*G2[i][j+4]+

                 …
   }
}
... // copia G1 para G2
```

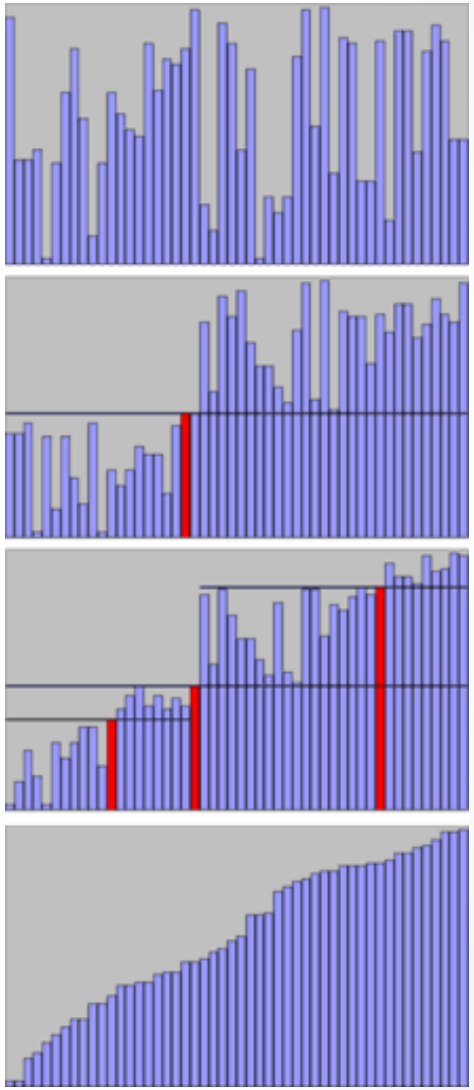


Trabalho – OpenMP

- 3ª Opção: Quick-sort

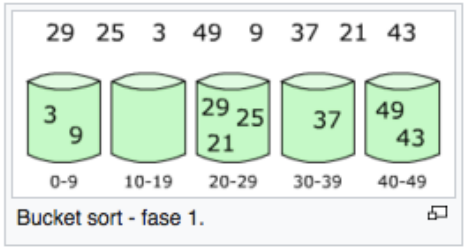
- Código:

```
private void quicksort(int lo, int hi)
 {
   int i=lo,j=hi,h;
   int x=array[(lo+hi)/2];

   //partition
   do
   {
     while(array[i]<x) i++;
     while(array[j]>x) j--;
     if(i<=j)
     {
       h=array[i]; array[i]=array[j]; array[j]=h;
       i++; j--;
     }
   } while(i<=j);

   //recursion
   if(lo<j) quicksort(lo,j);
   if(i<hi) quicksort(i,hi);
 }
```

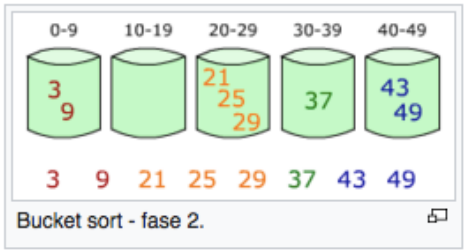


Trabalho – OpenMP

- 4ª Opção: Bucket-sort



Bucket sort - fase 1.



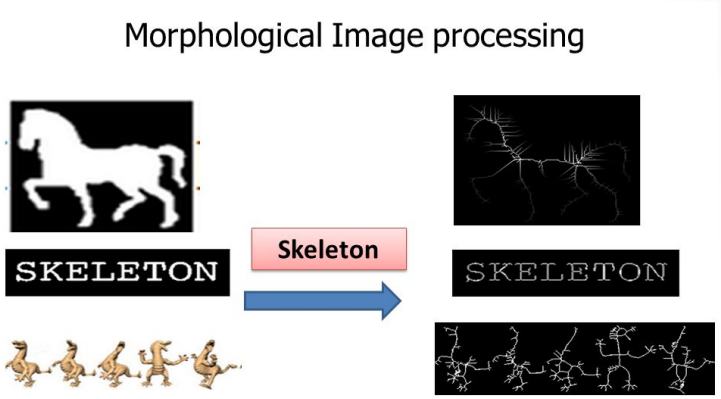
Bucket sort - fase 2.

- Algoritmo:

1. Cria um vetor de “buckets”, inicialmente vazios
2. Coloca cada número do vetor original no “bucket” correspondente
3. Ordena os “buckets” não vazios (pode ser usado qq. outro algoritmo de ordenação)
4. Coloca os elementos no vetor original

Trabalho – OpenMP

- 5ª Opção: Esqueletização de uma imagem binária



- Algoritmo (sugestão):

Repetir

- Numerar vizinhança

	P_9	P_2	P_3
	P_8	P_1	P_4
	P_7	P_6	P_5

- **1ª Passagem - Remover P_1 se**

  i) $2 <= N(P_1) <= 6$, $N(P_1)$ - número de vizinhos a '1'

  ii) $S(P_1) = 1$, $S(P_1)$ - nº de transições 0-1 na seq. $P_2, P_3, ..., P_9$

  iii) $\overline{P_4} + \overline{P_6} + \overline{P_2}\,\overline{P_8} = 1$

- **2ª Passagem - Remover P_1 se**

  Substituir iii) por

  iv) $\overline{P_2} + \overline{P_8} + \overline{P_4}\,\overline{P_6} = 1$

Até não remover mais pixels

Trabalho – OpenMP

- 6ª Opção: K-Means clustering
- Algoritmo (sugestão):
  - Atribuir um “cluster” inicial a cada dado
  - Enquanto existirem alterações aos “clusters”
    - Calcular o centróide de cada “cluster”
    - Atribuir cada dado ao “cluster” mais próximo