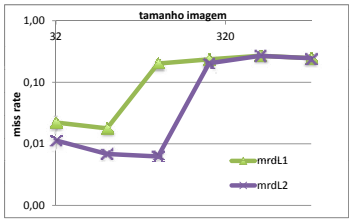
a) Avaliar a execução da função **convolve3x1()**

Objetivo: perceber a influência da utilização das caches no CPI e CPE

Tamanho	#I	#CC	L1_DCA (*)	L1_DCM	L2_DCM	CPI	CPE	%Mem	mrdL1	mrdL2	Mem KB (dados)
32	19.263	34.260	12.759	283	145	1,78	33,46	66%	2%	1%	8
64	67.841	55.927	46.141	823	316	0,82	13,65	68%	2%	1%	32
128	263.306	320.112	180.493	36.365	1.125	1,22	19,54	69%	20%	1%	128
256	1.047.489	2.925.720	719.607	169.824	146.592	2,79	44,64	69%	24%	20%	512
512	4.188.806	11.262.346	2.879.342	775.475	773.163	2,69	42,96	69%	27%	27%	2.048
1024	16.763.304	86.145.974	11.525.047	2.841.951	2.834.859	5,14	82,16	69%	25%	25%	8.192

(1) dados esgotam a L1 (32KB)
(2) dados esgotam a L2 (256KB?)
(3) dados esgotam a L3 (3MB?)



CPI=#CC/#I
CPE=#CC/(Tamanho^2)

%Mem=L1_DCA/#I

mrdL1=L1_DCM/L1_DCA
mrdL2=L2_DCM/L1_DCA

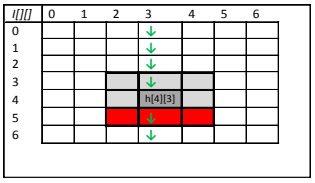
(*) Não existindo **PAPI_L1_DCA** pode usar-se a aproximação :
PAPI_L1_DCA ≈ PAPI_LD_INS+PAPI_SR_INS ↔ #acessosL1 ≈ #acessosMem

- **Conclusão**: quando os dados da convolução esgotam a capacidade da cache (casos (1), (2) e (3)) o CPI e o CPE aumentam significativamente, contrariando a tendência para diminuirem com o aumento do tamanho da imagem → isto deve-se ao aumento significativo das *miss rates*

b) Avaliar a execução da função **convolve()** *(na aula fazer apenas o caso 128x128)*

Objetivo: perceber a influência da localidade temporal no desempenho dum programa.

Tamanho	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%Mem	mrdL1	mrdL2	Mem KB (dados)
32	76.725	63.466	59.770	270	133	0,83	62	78%	0%	0%	8
64	295.351	190.987	232.286	5.071	537	0,65	47	79%	2%	0%	32
128	1.168.833	1.200.527	921.394	32.847	1.136	1,03	73	79%	4%	0%	128
256	4.660.728	4.162.289	3.675.933	131.701	131.116	0,89	64	79%	4%	4%	512
512	18.624.192	19.937.216	14.690.189	527.055	525.048	1,07	76	79%	4%	4%	2.048
1024	74.469.898	176.912.665	58.739.649	2.108.820	2.101.280	2,38	169	79%	4%	4%	8.192



- Apesar de termos mais operações por pixel (**CPE**↑), e implicitamente mais instruções (**#I** ↑), o total de misses é ligeiramente menor em (b) do que em (a) (L1_DCM↓ e L2_DCM ↓) .
- **Conclusão**: (b) faz mais cálculos para o mesmo número de acessos à memória, logo (b) é mais "amigável" da hierarquia de memória do que (a).
- Isto também justifica que **CPI**, **mrdL1** e **mrdL2** sejam mais baixos em (b).

A imagem é processada por colunas, logo reutiliza os valores das linhas anteriores.
Na figura, o cálculo do pixel [4][3] reutiliza os valores a cinzento usados anteriormente no cálculo do pixel [3][3] → **localidade temporal**.

c) Avaliar a execução da função **convolve3x1()** trocando a ordem dos ciclos relativos a X e a Y

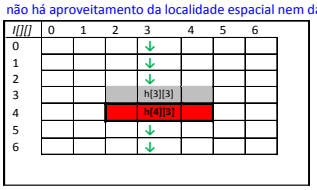
Objetivo: perceber a influência da localidade espacial no desempenho dum programa.

Tamanho	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%Mem	mrdL1	mrdL2	Mem KB (dados)
32	17.342	18.054	8.075	256	116	1,04	17,63	47%	3%	1%	8
64	59.872	49.680	26.513	561	189	0,83	12,13	44%	2%	1%	32
128	230.953	181.547	100.257	1.530	366	0,79	11,08	43%	2%	0%	128
256	917.214	635.246	395.259	4.382	732	0,69	9,69	43%	1%	0%	512
512	3.666.080	2.616.424	1.575.293	16.782	1.112	0,71	9,98	43%	1%	0%	2.048
1024	14.669.220	10.062.701	6.295.436	66.335	2.425	0,69	9,60	43%	1%	0%	8.192

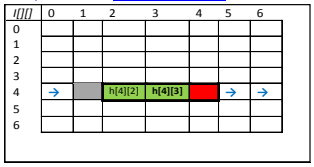
- Os valores dos pixeis são, agora, acedidos pela ordem que estão armazenados em memória → isto melhora a **localidade espacial**, logo temos muito menos misses na cache L1 e L2

```
for (y=0 ; y<H ; y++) { // for each row of I
  for (x=1 ; x<(W-1) ; x++) { // for each column of I
        kernel (h, I, y*W+x);
```

- em (a) como a imagem é processada por colunas e a máscara é horizontal, não há aproveitamento da localidade espacial nem da temporal.



- em (c) como a imagem é processada por linhas e a máscara é horizontal, para calcular um pixel (por ex. **h[4][3]**) aproveitam-se 2 valores próximos em memória, utilizados no cálculo do pixel anterior (h[4][2]) → aproveita-se a **localidade espacial**.



d) Avaliar a execução da função **convolve()** usando uma imagem 512x512 e variando o tamanho da máscara

Máscara	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%Mem	mrdL1	mrdL2	
1	18.624.193	24.637.829	14.690.205	528.313	525.356	1,32	376	79%	4%	4%	
3	68.872.427	62.640.284	53.419.457	799.542	798.647	0,91	956	78%	1%	1%	
5	158.729.674	123.048.420	121.314.986	835.761	828.663	0,78	1.878	76%	1%	1%	
7	288.042.613	200.041.376	218.251.221	865.986	854.388	0,69	3.052	76%	0%	0%	
9	456.658.088	309.365.119	344.102.615	1.989.948	858.179	0,68	4.721	75%	1%	0%	

- Quando se aumenta o tamanho da máscara aplicada a cada pixel o número de misses é praticamente constante, apesar de aumentar o #I (mais cálculos por pixel).
- Conclusão: faz-se mais cálculos para os mesmos acessos à memória → usar uma máscara maior torna o programa mais "amigável" da hierarquia da memória.

Avaliar a execução da função **convolve()** usando uma imagem 256x256 e variando o tamanho da máscara

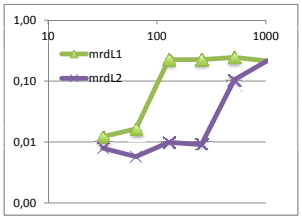
Mask	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%M	mrdL1	mrdL2	
1	4.600.000	4.000.000	4.000.000	131.000	5.000	0,9	61	87%	3%	0%	
3	17.000.000	11.000.000	13.000.000	183.000	7.000	0,6	168	76%	1%	0%	
5	40.000.000	25.000.000	31.000.000	209.000	6.000	0,6	381	78%	1%	0%	
7	71.000.000	46.000.000	58.000.000	217.000	11.000	0,6	702	82%	0%	0%	
9	113.000.000	73.000.000	91.000.000	232.000	6.000	0,6	1.114	81%	0%	0%	

a) Convolve3x1

Tamanho	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%M	mrdL1	mrdL2	Mem KB
32	17.000	20.000	11.000	135	87	1,2	19,5	65%	1%	1%	8
64	66.000	66.000	44.000	721	253	1,0	16,1	67%	2%	1%	32
128	261.000	344.000	173.000	39.000	1.706	1,3	21,0	66%	23%	1%	128
256	1.000.000	1.400.000	721.000	162.000	6.559	1,4	21,4	72%	22%	1%	512
512	4.000.000	25.000.000	3.000.000	735.000	315.000	6,3	95,4	75%	25%	11%	2.048
1024	16.000.000	20.000.000	14.000.000	3.000.000	3.000.000	1,3	19,1	88%	21%	21%	8.192

1) dados esgotam a L1 (32KB)

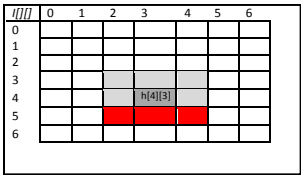
2) dados esgotam a L2 (2MB?)



Notar os aumentos correspondentes no CPI e CPE nos casos 1) e 2)

b) Convolve

Tamanho	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%M	mrdL1	mrdL2	Mem KB
32	75.000	56.000	61.000	247	150	0,7	55	81%	0%	0%	8
64	293.000	200.000	240.000	969	500	0,7	49	82%	0%	0%	32
128	1.000.000	800.000	900.000	32.000	2.000	0,8	49	90%	4%	0%	128
256	4.600.000	4.000.000	4.000.000	131.000	5.000	0,9	61	87%	3%	0%	512
512	18.000.000	58.000.000	15.000.000	500.000	300.000	3,2	221	83%	3%	2%	2.048
1024	74.000.000	340.000.000	62.000.000	2.000.000	3.000.000	4,6	324	84%	3%	5%	8.192

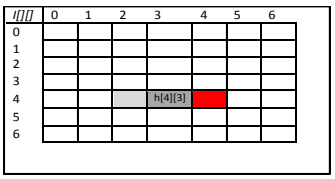


Apesar de termos mais operações por pixel (e implicitamente #I) o total de misses é ligeiramente menor
ou seja, faz mais cálculos para os mesmos acessos à memória logo é mais "amigável" da hierarquia da memória
isso também justifica o CPI, mrdL1 e mrdL2 mais baixos

A imagem é processada por colunas, logo reutiliza os valores das linhas anteriores
Na figura os valores a cinzento são reutilizados do cálculo do pixel [3][3]

c) Convolve3x1

	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%M	mrdL1	mrdL2	Mem KB
32											8
64											32
128	220.000	310.000	147.000	2.000	2.000	1,4	18,9	67%	1%	1%	128
256	900.000	1.300.000	460.000	8.000	8.000	1,4	19,8	51%	2%	2%	512
512	4.000.000	5.000.000	2.000.000	32.000	32.000	1,3	19,1	50%	2%	2%	2.048
1024	15.000.000	21.000.000	7.000.000	131.000	131.000	1,4	20,0	47%	2%	2%	8.192



Os valores dos pixels são, agora, acedidos pela ordem que estão armazenados (melhora a localidade espacial, logo temos mt. menos misses na L1 e L2)

```
for (y=0 ; y<H ; y++) { // for each row of I
  for (x=1 ; x<(W-1) ; x++) { // for each column of I
      kernel (h, I, y*W+x);
```

d) Convolve 256x256

Mask	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%M	mrdL1	mrdL2	
1	4.600.000	4.000.000	4.000.000	131.000	5.000	0,9	61	87%	3%	0%	
3	17.000.000	11.000.000	13.000.000	183.000	7.000	0,6	168	76%	1%	0%	
5	40.000.000	25.000.000	31.000.000	209.000	6.000	0,6	381	78%	1%	0%	
7	71.000.000	46.000.000	58.000.000	217.000	11.000	0,6	702	82%	0%	0%	
9	113.000.000	73.000.000	91.000.000	232.000	6.000	0,6	1.114	81%	0%	0%	

Convolve 512x512

Mask	#I	#CC	L1_DCA	L1_DCM	L2_DCM	CPI	CPE	%M	mrdL1	mrdL2	
1	18.000.000	50.000.000	15.000.000	500.000	200.000	2,8	763	83%	3%	1%	
3	69.000.000	100.000.000	55.000.000	700.000	600.000	1,4	1.526	80%	1%	1%	
5	158.000.000	153.000.000	127.000.000	800.000	622.000	1,0	2.335	80%	1%	0%	
7	288.000.000	232.000.000	231.000.000	878.000	795.000	0,8	3.540	80%	0%	0%	
9	1.150.000.000	1.187.000.000	931.000.000	825.000	829.000	1,0	18.112	81%	0%	0%	

O número de misses é praticamente constante, apesar de aumentar o #I (mais cálculos por pixel)
ou seja, faz-se mais cálculos para os mesmos acessos à memória logo é mais "amigável" da hierarquia da memória