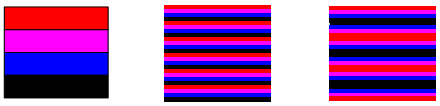
Týden 11.
Maticové výpočty a aplikace.
Úlohy ze zpracování signálů a obrazů.

Proužkové mapování matic na procesory

P procesorů s indexy 0,1, ..., P-1 (virtuální lineární pole)
matice m řádků, n sloupců; k řádků (sloupců) v bloku

1. řádkové
2. sloupcové

- 1 cykl po blocích max. velikosti $k = \lceil m/P \rceil$ nebo $k = \lceil n/P \rceil$, $k \geq 1$
- cyklicky po blocích (k mezi 1 a max)
- cyklicky po řádcích nebo sloupcích, tj.po blocích min. velikosti (k = 1)

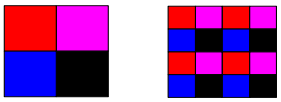


normální reverzované

Šachovnicové mapování matic na procesory

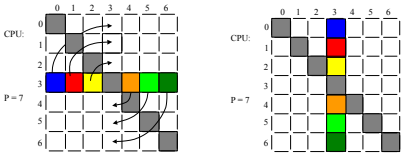
P procesorů s indexy 0,1, ..., P-1, virtuální mřížka $\sqrt{P} \times \sqrt{P}$
matice: m řádků, n sloupců

a. 1 cykl po blocích max. velikosti $(m/\sqrt{P}) \times (n/\sqrt{P})$
b. cyklicky po blocích [k_1 řádků, k_2 sloupců na blok]
c. cyklicky po blocích min. velikosti (po prvcích)



Transpozice matice n x n (MP, proužky)

- nic se nepočítá, čistě komunikační záležitost
- každá CPU má n/P řádků výchozí matice
- transponuje P submatic velikosti n/P x n/P, z nich P-1 pošle jiným CPU (komunikace AAS) a submatici na diagonále si ponechá doma

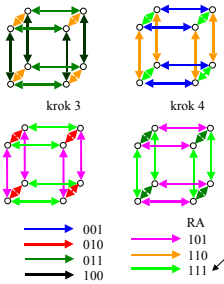


Transpozice matice na svazku (COW, proužky)

- Každá CPU má blok n/P po sobě jdoucích řádků (lze na ně pohlížet jako na P submatic x[i] velikosti n/P × n/P .
- AAS: P-1 kroků, v každém kroku permutační směrování, aby nedošlo k tlačenici v multiportovém přepínači
- SW řetězení: překryj transpozici následující submatice s odesíláním právě transponované submatice:

1. Transponuj submatici (myid+1)mod P
2. Cykl (P-1)- krát, h = 2,3,…,P
   transponuj submatici (myid+h)mod P paralelně s odesíláním předchozí submatice (myid+h-1)mod P
   (transponovaná submatice (myid) se nikam neodesílá)

Transpozice na SF hyperkostce, P/2 kroků



$T_{AAS} = (P/2)(t_s + mt_w)$

Plán přesunů: co - kdy - kam

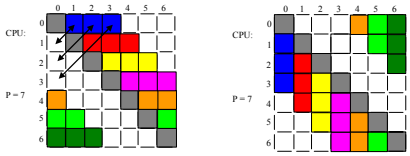
RA	dimenze		
krok	0	1	2
1	3	6	4
2	1	7	6
3	7	2	5
4	5	3	7

Relativní adresa zprávy
RA = src ⊕ dst

do směru dim se ve fázi krok posílá submatice s indexem myid XOR RA[krok][dim]

Transpozice matice n x n (proužky) ve SM

- nic se nepočítá, čistě přemisťovací záležitost
- každá CPU transponuje svoje řádkové submatice (v cache), pak je zapíše jako sloupcové submatice do paměti; po bariéře si načte nové řádkové submatice ; minimalizovat počet výpadků!
- celkem se po sběrnici přesune sekvenčně P x (P-1) submatic;



Násobení matic C = A × B na 2D-T (šachovnice)

akumulací součinů zkřivených a rotovaných matic A, B.

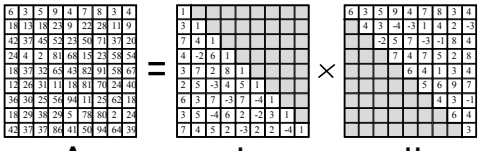
Cannonův algoritmus používá rozčlenění A a B do $\sqrt{P} \times \sqrt{P}$ čtvercových submatic a pak jejich zkřivení (skewing).

zkřivení podél +1.osy: -2. osy:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \rightarrow A^{(1)} = \begin{bmatrix} a & h & f \\ d & b & i \\ g & e & c \end{bmatrix}, \quad A^{(-2)} = \begin{bmatrix} a & b & c \\ e & f & d \\ i & g & h \end{bmatrix}$$

```
Atmp = A(-2) ; Btmp = B(1) ; C = 0;      inicializace
for (i=1;i<= √P;i++) {
  C=C+Atmp*Btmp;  maticový součin stejnolehlých  submatic!!
  Atmp = rotace(Atmp ve 2. ose);   submatice →
  Btmp = rotace(Btmp  v 1. ose);   submatice ↓
}
```

Faktorizace A = L x U



A L U

Použití: Řešení systému lin. rovnic pro mnoho pravých stran b (Jiná možnost by byla vypočítat inverzní matici A^{-1}).

- Cíl: řešit Ax = LUx = b pro x.
- To lze provést ve dvou snadných krocích:
  - řešit Ly = b pro y (dopředná substituce)
  - řešit Ux = y pro x (zpětná substituce).

lze uložit do A (místo A)

Postup faktorizace A = L x U

- Gaussova eliminace (= transformace A na matici U)
- Matice L má na hlavní diagonále jedničky, nad ní nuly a prvky pod hlavní diagonálou jsou koeficienty, kterými byl násoben pivot. řádek

  `L[i,k]=A[i,k]/A[k,k]  i=k,k+1,...,n`
- je-li pivotní prvek nula, došlo by k přetečení; je-li blízký nule, mohlo by dojít k numerické nestabilitě. Proto se vyměňují řádky matice A tak, aby se na diagonálu dostal prvek s největší abs. hodnotou ve sloupci a stal se pivotním (tzv. *partial pivoting*). V kroku k bude pivot prvek `A[ix[k],k]`. Možnosti paralelního postupu:
  - každý proces vybere maximum ve svém proužku a aktualizuje svou kopii pivotních indexů ix. Bez bariéry.
  - jeden proces vybere globální maximum a přehodí řádky; bariéra nutná.
  - každý proces vybere maximum ze své části, pak společně globální maximum; bariéra nutná, privátní kopie ix.

Sekvenční výměna řádků (partial pivoting)

```
double A[1:n,1:n],LU [1:n,1:n],pivot, pivotRow,mult;
int ix[1:n],t;
# inicializuj ix[i]=i, LU[i,j]=A[i,j];
# proveď přehození řádků před Gaussovou eliminací:
for [k=1 to n-1] {                # iteruj dolů podél hlavní diagonály
  pivot = abs(LU[ix[k],k]); #vyber prvek na diagonále ve sloupci k
  for [i=k+1 to n] {
    if (abs(LU[ix[i],k])>pivot) {  # je-li pod ním větší prvek
       pivot=abs(LU[ix[i],k]); pivotRow = i;  #vezmi ten
    }
  }
  if (pivotRow !=k) {#došlo-li k výměně, vyměň i indexy ve vektoru ix
    t = ix[k]; ix[k]=ix[pivotRow]; ix[pivotRow]=t;
  }
  pivot=LU[ix[k],k]; #skutečná hodnota pivotního prvku v kroku k
```

Schema programu SV pro dekompozici LU

```
double A[1:n,1:n],LU [1:n,1:n]; #A je již inicializována
int ix[1:n];                    # vektor indexů pivotních řádků
process worker (w=1 to P) {
  double pivot, mult;
  deklarace dalších lokálních proměnných jako např. privátní kopie ix;
  for [i=w to n by P] #řádky cyklicky procesorům kvůli stejné zátěži
     inicializuj ix a svoje proužky LU;
     barrier(w);
     # proveď Gaussovu eliminaci (řádky již přeházeny)
    for [k=1 to n-1] {            #pro všechny řádky submatice
      mult= LU[ix[i],k]/pivot; # vypočítej násobící koeficient
      LU[ix[i],k]=mult;          # a ulož jej jako prvek matice L
      for [j=k+1 to n]       # odečítej násobkek pivotního řádku:
         LU[ix[i],j]=LU[ix[i],j]-mult*LU[ix[k],j];
    }
    barrier(w);
  }
}
```

Úlohy ze zpracování signálů a obrazů

- Paralelizace umělých neuronových sítí (ANN)
- jednorozměrná rychlá Fourierova transformace 1D-FFT
- 2D - FFT
- Vyhlazování, zostření, potlačení šumu;
- Detekce hran (Sobelův a Laplaceův operátor, CNN)

Výpočetní požadavky u zpracování obrazů:

- pixmapa 1024 × 1024 pixelů, 8 bitů na pixel;
- paměťová kapacita 2^{20} byte = 1 Mbyte;
- 2^{20} operací/snímek, tj. 10 ms při trvání jedné operace 10 ns;
- provoz v RT vyžaduje 60 - 85 snímků/s, tj. 12-16 ms/snímek;
- za dobu 1 snímku musí být zpracovány všechny pixely;
- často složité operace pro každý pixel.

Paralelní implementace ANN - případová studie

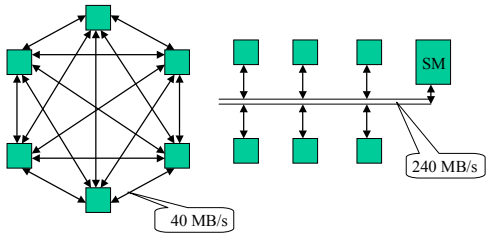
Zadání: Paralelní implementace umělých neuronových sítí pro vestavěné aplikace a práci v RT

Řešení by mělo zodpovědět otázky:

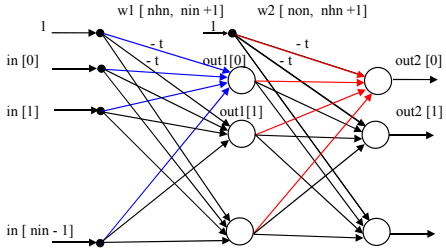
- Jako alternativa k neuronovým čipům, je paralelní implementace levnější? snadněji aplikovatelná? výkonnější?
- Když paralelní implementace, tak která architektura, DM-MP nebo SM se sběrnicí?
  DSP SHARC CPU je vhodný pro obě řešení.

Paralelní implementace ANN

ADSP-21062 SHARC by AMD



Rozčlenění třívrstvé ANN



Implementace MP - jen fáze vybavování

WHILE TRUE
SEQ

1. Rozhlášení (OAB) vstupního vektoru z uzlu id = 0
2. Vypočti výstupní hodnoty přidělených skrytých neuronů (násobení matice vah vektorem aktivací,
   $$x = a_1 w_1 + a_2 w_2 + \ldots + a_n w_n + a_0 w_0$$
   potom funkce exp a dělení) $$f(x) = \frac{1}{1+\exp(-x)}$$
3. AAB, všichni dostanou všechny hodnoty výstupů skrytých neuronů
4. Vypočti přidělené výstupy
5. Posbírej výstupní vektor (AOG) v uzlu id = 0.

SW řetězení

čas → k-1 k k+1

OAB						
HID	OAB					
AAB	HID	OAB				
OUT	AAB	HID	OAB			
OAG	OUT	AAB	HID	OAB		
	OAG	OUT	AAB	HID	OAB	
		OAG	OUT	AAB	HID	OAB
			OAG	OUT	AAB	HID
				OAG	OUT	AAB
					OAG	OUT
						OAG

Paralelní implementace se SM

WHILE TRUE
SEQ

1. Pouze řídicí CPU: získej nový vstupní vektor ze vstupního buferu do SM; bariéra;
2. Všichni: načti vstupní hodnoty ze SM; vypočti hodnoty na výstupech přidělených skrytých neuronů; bariéra;
3. Všichni: čti hodnoty výstupů skrytých neuronů ze SM; vypočti přidělené výstupy; bariéra;
4. Pouze řídicí CPU: dej výstupní vektor ze SM do výstupního buferu; bariéra;

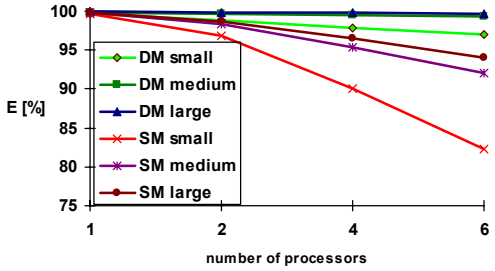
Paralelní implementace se SM - pokrač.

Předpoklady:

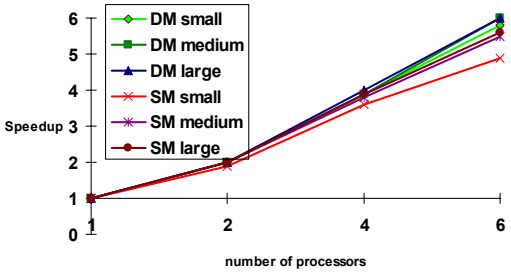
- velikost bloku cache 32 byte, data: stačí přesnost REAL32
- výpadek čtení každých 8 načtených čísel u prvků
  - vstupního vektoru (všechny CPU),
  - skrytého vektoru (všechny CPUs) nebo
  - výstupního vektoru (pouze řídicí CPU)

  pokud není podporován předvýběr dat v HW (IA-64) nebo SW
- operace write back není potřeba, zapisuje se stále do stejných bloků cache (mění se pouze stav z exkluzivního na sdílený). Vstupní a skrytý vektor (kromě přidělené sekce) se jen čtou, takže se nemusí odsunovat do SM, pouze se přepisují.
- předpokládáme, že matice vah jsou rezidentní ve všech uzlech (např. v privátních ROM), přístup k vahám v 1 taktu.

Výsledky a závěry



Výsledky a závěry - pokrač.



Závěry

- simulované ANN: malé 32 - 24 - 12
  střední 100 - 48 - 12
  velké 200 - 60 - 12
- MP se SW řetězením předčí implementaci SM
- Nejlepší topologie pro implementaci DM-MP
  - úplné propojení do P = 6 procesorů
  - hyperkostka pro P = 8, 16, 32, …
  - jinak kruhová topologie, P = 7, 9,10, …,15,17,…
- Nepravidelná topologie AMP (A Minimum Path) s tabulkovým směrováním vykazuje horší výkonnost.

Příklad: výkonnost paralelní FFT

n-bodová Diskrétní Fourierova Transformace je def. jako součin komplexní matice $W_n[n, n] = \| w_n(j,k) \|$ a komplexního vektoru $x[n,1]$:

$$y = W_n x, \quad y_j = \sum_{k=0}^{n-1} w_n(j,k) x_k = \sum_{k=0}^{n-1} x_k \exp\left[-\frac{2\pi ijk}{n}\right]$$

kde $w_n(j,k)$ jsou tzv. *otáčecí činitelé* (twiddle factors), jinak též komplexní kořeny rovnice $x^n = 1$

$$w_n(j,k) = \exp(-ijk2\pi/n) = (\cos 2\pi/n - i \sin 2\pi/n)^{jk},$$

$i = \sqrt{-1}$, $j, k = 0,1,\ldots, n-1$.

Složitost přímého výpočtu je n^2 aritmetických operací (součin a akumulace dvou komplexních čísel).

n-bodová Diskrétní Fourierova Transformace

Příklady transformačních matic :

$W_1 = 1$

W_2

1	1
1	-1

jk (mod 4)

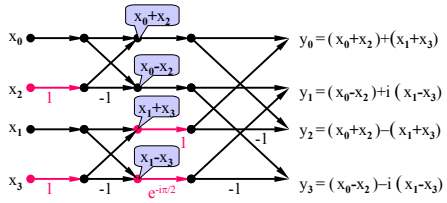
0	0	0	0
0	1	2	3
0	2	4	6≡2
0	3	6≡2	9≡1

W_4

1	1	1	1
1	$e^{(-i\pi/2)}$	-1	$-e^{(-i\pi/2)}$
1	-1	1	-1
1	$-e^{(-i\pi/2)}$	-1	$e^{(-i\pi/2)}$

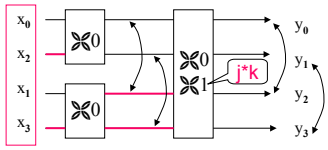
Algoritmus FFT využívá symetrii matice a opakující se hodnoty

Rychlá Fourierova transformace FFT (n = 4) = algoritmus výpočtu DFT



Stačí 6 operací ± a 4 operace * (obecně komplexní)
Přímý výpočet: 12 operací ± a 16 operací *

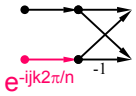
FFT symbolicky (n = 4)



Počet motýlků:

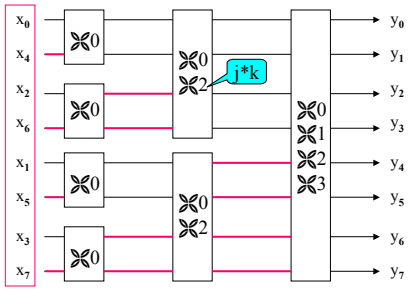
$\frac{n \log n}{2}$

reverzace adres vstupů



= operace „motýlek j*k“

Rychlá Fourierova transformace FFT (n = 8)



$\frac{n \log n}{2}$ - krát

reverzace adres vstupů

FFT (n = 32), uspořádání motýlků

0	0	0	0	0
0	8	4	2	1
0	0	8	4	2
0	8	12	6	3
0	0	0	8	4
0	8	4	10	5
0	0	8	12	6
0	8	12	14	7
0	0	0	0	8
0	8	4	2	9
0	0	8	4	10
0	8	12	6	11
0	0	0	8	12
0	8	4	10	13
0	0	8	12	14
0	8	12	14	15

Zisk při použití n-bodové FFT

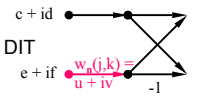
Složitost FFT:
$\log_2 n$ stupňů výpočtu,
n operací ±
n operací *
v každém stupni
(obecně komplexních)
→ O(n log n).

+ přeskládání vstupní nebo výstupní posloupnosti: n log n kroků jako řazení.

n	n log n	n^2
64	384	4096
128	896	16384
256	2048	65536
512	4608	262144
1024	10240	1048576

Násobení matice vektorem podle definice DFT: $O(n^2)$

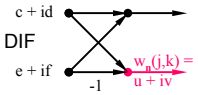
Motýlkové operace



C = c + ue - vf
D = d + uf + ve

E = 2c - C
F = 2d - D

4 x *
2 x *2
6 x ±



C = c + e
D = d + f

E = (c - e) u - (d - f) v
F = (c - e) v - (d - f) u

4 x *
6 x ±

n-bodová paralelní FFT, n=2ᵠ, P dělí n

$$y = W_n x = T\,(W_P \otimes I_{n/P})\,S\,(I_P \otimes W_{n/P})\,\Pi\,x$$

permutace s rozestupem n/P

P n/p-bodových FFT

n/P p-bodových FFT

přeskládání vektoru x (permutace) reverzací indexů

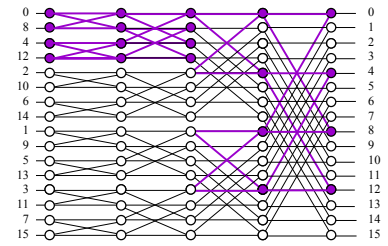
násobení diagonální maticí rotačních koeficientů
$T = \mathrm{diag}\,(I_{n/P}, \Omega, \Omega^2, \ldots, \Omega^{P-1})$,
$\Omega = \mathrm{diag}\,(1, \omega, \omega^2, \ldots, \omega^{n/P-1})$, $\omega = \exp(-2\pi i/n)$

Implementace n-bodové paralelní FFT

Sekvence n bodů je uložena ve sdílené paměti jako matice P řádků a n/P sloupců (jazyk C).

1. Procesory provedou paralelně přeskládání posloupnosti reverzací bitů.
2. Každý procesor provede n/P-bodovou sekvenční FFT na řádcích matice, nejlépe v rámci cache, bez komunikace s ostatními (prvních log (n/P) stupňů výpočtu).
3. Následuje globální transpozice matice (komunikace všichni všem)
4. Násobení maticí otáčecích činitelů
5. Každý procesor provede závěrečných log P stupňů výpočtu (P-bodové sekvenční FFT na sloupcích matice, každý procesor n/P^2 sloupců)
6. Globální transpozice matice

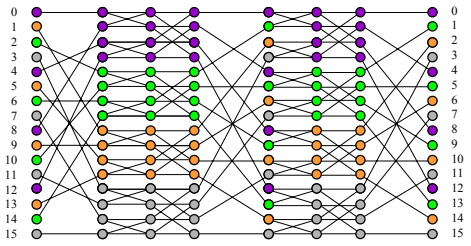
Paralelizace FFT (n = 16, P = 4)



reverzní adresace

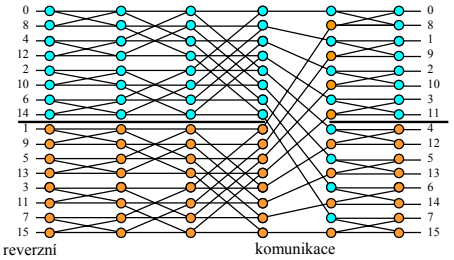
Body ● počítá první procesor

Paralelní FFT (n = 16, P = 4)



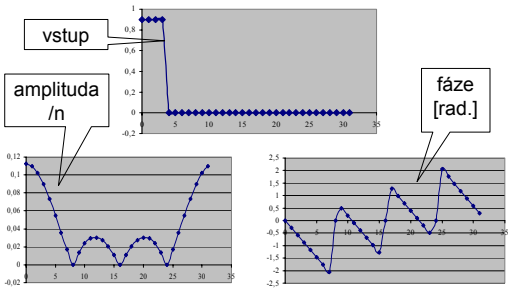
reverzní adresace | FFT | komunikace mezi procesory | FFT | permutace (každý čtvrtý)

Paralelní FFT (n = 16, P = 2)



reverzní adresace | komunikace mezi procesory

Příklad: FFT pravoúhlého impulsu, n=32



DFT ve zpracování obrazů

2-D DFT: $$X_{lm} = \sum_{j=0}^{N-1}\sum_{k=0}^{M-1} x_{jk} \exp\left[-2\pi i\left(\frac{jl}{N}+\frac{km}{M}\right)\right]$$

kde j a k jsou indexy řádků a sloupců, $0 \le j \le N-1$, $0 \le k \le M-1$.
Pro čtvercový obraz N = M a

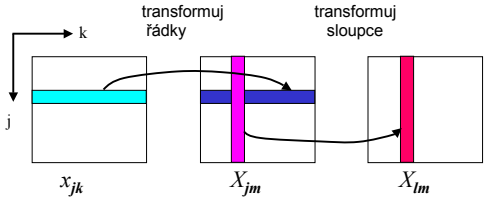
$$X_{lm} = \sum_{j=0}^{N-1}\left[\sum_{k=0}^{N-1} x_{jk} \exp\left[-2\pi i\left(\frac{km}{M}\right)\right]\right]\exp\left[-2\pi i\left(\frac{jl}{N}\right)\right]$$

Vnitřní součet je 1-D DFT na N bodech řádku, výsledkem je transformovaný řádek. Vnější součet je 1-D DFT na N bodech sloupce. Můžeme psát:

$$X_{lm} = \sum_{j=0}^{N-1} X_{jm} \exp\left[-2\pi i\left(\frac{jl}{N}\right)\right]$$

Tudíž 2-D DFT se dá rozdělit do dvou sekvenčních fází, jedna zpracovává řádky, druhá vzniklé sloupce.

DFT ve zpracování obrazů



Aplikace DFT:
Kmitočtová filtrace, DFT obrazu se bod po bodu násobí DFT filtru.
Inverzní DFT získáme vyfiltrovaný obraz.

Vyhlazování obrazů

např. nahrazením hodnoty pixelu průměrem v 9-okolí

x0	x1	x2
x3	x4	x5
x6	x7	x8

$$x'_4 = \frac{x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8}{9}$$

Počet kroků lze redukovat na 4 vektorové operace:
1. Přičtení sloupce pixelů zleva — $x_4 + x_3$
2. Přičtení sloupce pixelů zprava — $x_4 + x_3 + x_5$
3. Přičtení řádku pixelů shora — $x_4 + x_3 + x_5 + x_0 + x_1 + x_2$
4. Přičtení řádku pixelů zdola — $x_4 + x_3 + x_5 + x_0 + x_1 + x_2 + x_6 + x_7 + x_8$

Obecně lze použít u každého x_k použít váhu w_k (masky s vahami)

Vzájemná diskrétní korelace f ⊗ w

w0	w1	w2
w3	w4	w5
w6	w7	w8

⊗

x0	x1	x2
x3	x4	x5
x6	x7	x8

k =

	x'4	

k = 1/9

1	1	1
1	1	1
1	1	1

zprůměrování

1/16

1	1	1
1	8	1
1	1	1

redukce šumu

1/9

-1	-1	-1
-1	8	-1
-1	-1	-1

zostření

Snadná paralelizace jako u metody konečných diferencí.

Masky pro detekci hran - Prewittův operátor

Používá se aproximace gradientu ∇f (= vektor, jehož složky jsou derivace skalárního pole f podle souřadnic). Prewitt:

$$\frac{\partial f}{\partial x} \approx (x_2 - x_0) + (x_5 - x_3) + (x_8 - x_6)$$

$$\frac{\partial f}{\partial y} \approx (x_6 - x_0) + (x_7 - x_1) + (x_8 - x_2)$$

x0	x1	x2
x3	x4	x5
x6	x7	x8

$$\nabla f \approx |x_2 - x_0 + x_5 - x_3 + x_8 - x_6| + |x_6 - x_0 + x_7 - x_1 + x_8 - x_2|$$

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Masky pro detekci hran - Sobelův operátor

Sobel :

$$\frac{\partial f}{\partial x} \approx (x_2 + 2x_5 + x_8) - (x_0 + 2x_3 + x_6)$$

$$\frac{\partial f}{\partial y} \approx (x_6 + 2x_7 + x_8) - (x_0 + 2x_1 + x_2)$$

x0	x1	x2
x3	x4	x5
x6	x7	x8

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1