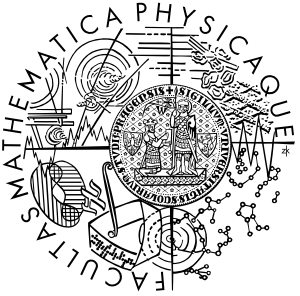
Even Faster Channelizer of Litoměřice



David Klusáček
MFF UK
6.10.2016
`brmlab.cz/user/david`

1 Channelizer – princip

Mame signal x, chceme C kanalu (oddelenych filtrem s impulsni odezvou h), vysledky D krat decimovane.

Vyber kanalu

$$z_n := x_n \exp(-2\pi i n c/C) \tag{1}$$

jeho odfiltrovani od ostatnich

$$\hat{y}_n := \sum_{k=0}^{L-1} h_k z_{n-k} \tag{2}$$

a decimace

$$y_n := \hat{y}_{nD} \tag{3}$$

Dosadime:

$$\begin{aligned} y_n &= \sum_{k=0}^{L-1} h_k x_{(nD-k)} \exp(-2\pi i (nD-k) c/C) \\ &= \sum_{k=0}^{C-1} \left(\sum_{l=0}^{L/C-1} h_{(k+lC)} x_{(nD-lC-k)} \exp(-2\pi i l C c/C) \right) \exp(-2\pi i (nD-k) c/C) \\ &= \sum_{k=0}^{C-1} w_{n,k} \exp(-2\pi i (nD-k) c/C) = \sum_{k=0}^{C-1} w_{n,(nD-k)\%C} \exp(-2\pi i k c/C) \end{aligned} \tag{4}$$

coz je zjevna FFT.

V predchozim vzorci je

$$w_{n,k} := \sum_{l=0}^{L/C-1} h_{k+lC} x_{nD-lC-k} \tag{5}$$

a to je cast ktera bere vetsinu casu, proto se ji budeme venovat. V implementaci je dobre cist x postupne z leva do prava, proto zavedem substituce $k = C-1-\hat{k}$ a $l = L/C-1-\hat{l}$.

Ty prevedou $k+lC$ na $C-1-\hat{k}+(L/C-1-\hat{l})C = C-1+L-C-\hat{k}-\hat{l}C = L-1-\hat{k}-\hat{l}C$, odkud dostanem

$$w_{n,C-1-\hat{k}} = \sum_{\hat{l}=0}^{L/C-1} h_{L-1-\hat{k}-\hat{l}C} \cdot x_{nD-L+1+\hat{k}+\hat{l}C} \tag{6}$$

Nyni obratime (i v pameti) h tak ze definujeme $\hat{h}_k = h_{L-1-k}$

$$w_{n,C-1-k} = \underbrace{\sum_{l=0}^{L/C-1} \hat{h}_{lC+k}\, x_{nD-L+1+lC+k}}_{u_k} \tag{7}$$

Jelikoz vysledna sada C kanalu v case n se spocita pomoci FFT

$$y_n = \sum_{k=0}^{C-1} v(n)_k \exp(-2\pi ikc/C) \tag{8}$$

kde $v(n)_k := w_{n,(nD-k)\%C}$, chceme najit zpusob jak vyplnit pole $v(n)$ hodnotami u_k ze vzorce '`ordered-polyphase`'. Jelikoz plati $w_{n,k} = u_{C-1-k} = u_{(-k-1)\%C}$, dostanme

$$v(n)_k = w_{n,(nD-k)\%C} = u_{(-(nD-k)\%C-1)\%C} = u_{(k-nD-1)\%C} \tag{9}$$

a tedy

$$v(n)_{(k+nD+1)\%C} = u_k \tag{10}$$

Coz je vzorec pro vypocet adresy na kterou ulozit hodnotu u_k.

Fastest Channelizer in Litomeřice

- Podle Jednova mereni 2 az 4 krat rychlejsi nez `gnuradio`.
- Podle meho mereni (bez zapocteni casu I/O operaci) dava v konfiguraci C=72 D=50 l=1151 na procesoru 3.7 GHz "AMD Phenom II X4 980" nasledujici vysledky (prelozeno pomoci gcc-4.9.2):

27.3 MS/s 39.8 MS/s 47.5 MS/s 48.2 MS/s

Serazeno podle poctu pouzitych worker-threadu. MS/s jsou komplexni megasamply za sekundu vystupniho signalu.

Kde by sel zlepsit

- Pouzivat `__builtin_assume_aligned()` bez `__restrict__` kdyz predavas vic poli do funkce skoro nema smysl. Ale ani s nimi nedokazalo gcccko slusne vektorizovat.
- Predavat koeficienty filtru v textovem souboru vede k jejich zaokrouhleni a odezva dlouhych filtru je na jejich male zmeny dost citliva. Lepsi je nacitat to binarne.
- Prilis mnoho slozite synchronizace (condition variables) mezi thready dusi vykon.
- Da se vymyslet lepsi rozvrzeni dat v pameti pro lepsi vyuziti L1-cachi

Even Faster Channelizer of Litomeřice

• Obsahuje nekolik implementaci, jak se postupne vyvijel, vcetne puvodni nemultithreadove (oproti ktere se pak automaticky testuji pokrocilejsi implementace).

• Tri implementace jsou Cckove, ostatni (SSEx86, SSE, AVX a NEON) jsou psany v assembleru (AVX a NEON jeste nejsou). Assemblerove implementace jsou za behu prelozeny s pevnymi parametry C, D a L pomoci assembleru `nasm` a prilinkovany k bezicimu programu.

• Channelizer je distribuovan jako stand-alone program `efcl` i jako knihovna `libefcl`. Knihovna je thread-safe, je mozne mit v jednu chvili vice ruznych bezicich channelizeru.

• Obsahuje VSCL Very Slow Channelizer of Litomeřice ktera je v podstate opsana definice. Oproti ni se testuje vyse zminena nemultithreadova implementace.

• Behem vypoctu si automaticky meri vykon, coz iterne pouziva pro load balancing (ten ale jeste potrebuje vylepsit) a pro vypis jak dlouho co trvalo, coz bylo vyhodne behem vyvoje.

Na stejnem pocitaci ve stejne konfiguraci:

C1 implementace:

34.2 MS/s 67.1 MS/s 98.3 MS/s 102.5 MS/s

C2 implementace:

35.8 MS/s 71.2 MS/s 103.9 MS/s 106.3 MS/s

C3 implementace:

43.8 MS/s 87.2 MS/s 125.7 MS/s 136.0 MS/s

SSE implementace (assembler):

122.3 MS/s 222.7 MS/s 335.8 MS/s 342.9 MS/s

```
CHANNELIZER SETUP:
  SSE code compiled (5715+455 bytes), overhang = 2*(12*2+12*1)-72 = 0 csamples
  efcl 0.1 (SSE implementation) compiled by gcc-4.9.2 using fftw-3.3.3-fma-sse2
  filter length l=1151, 0-padded to L=1152
  block length BL=29710 using 4-buffering, input buffer of 118849 samples
  number of channels C=72, downsampling D=50, maximum threads=4
  FFT:  add=  195  mul=   84  mac=   90,   6.375 FLOP/csample
  Polyphase filter: add= 2160  mul= 2304,  62 FLOP/csample
SPEED:
  Throughput in complex valued output samples:  342.89 MS/s
        Complex samples (csamples) generated: 1152000000
PERFORMANCE: (clocks are approximate as f_TSC is constant 3700000881 Hz)
     ---  per-thread CPU clocks  ---    MIN      AVG     MAX    AVG per SIMDop
  polyphase filter clocks per csample:  14.58   17.56   19.02   | 2.265 mac
  FFT CPU clocks per 1 output csample:  11.24   11.79   11.90   | 3.698 +|*
  memcpy clocks per 1 output csample:    3.46    3.62    3.64   | 3.620 ld|st
  management clocks per csample:         1.40    1.80    1.89   +-------------
  stall clocks per 1 output csample:    44.69    7.81    1.43
  per thread FFT performance:       2.00087 GFLOPS  0.540775 FLOP/clock
  per thread filter performance:    13.0668 GFLOPS   3.53157 FLOP/clock
  Total performance (all threads):  23.4453 GFLOPS   6.33657 FLOP/clock
  Work balance times 4:  112.8% 113.2% 117.3% 56.6%
```

7 Na co bylo treba dat pozor

• Pro dobry multithreadovy vykon je potreba aby se temer nenarazilo na zamceny mutex. Peer-to-peer mutexy, pyramidova struktura...

• Proto se meri vykon a predikuji se casy dokonceni bufferu a podle nich se prideluje velikost bufferu. K tomu ale potrebuju predikovatelnou dobu behu.

• SSE je potreba prepnout do "flush deromal results to zero" a "denormals are zero" modu. Jinak operace muze trvat podstatne dele kdyz se narazi na denormalizovane cislo.

• Hack pro constant TSC, aby slo pocitat clocky.

• Cim blize jsme k maximalnimu vykonu CPU tim je vetsi variabilita vykonu – cache, TLB reload, rozpad sychronismu. Mozna by pomohlo barveni stranek v alokatoru pameti v jadru Linuxu...

• Co jedne architekture prospiva, to druhe skodi. Napriklad AMD K7 pomaha `MOVNTQ`, ostatnim skodi. Naopak mu neprospiva `PREFETCHT0`.

Naproti tomu Intelu pomuze `CLFLUSH` ktery velmi skodi AMD Phenom II.

Proto je kod plny `#ifdefu` pomoci kterych se na zaklade `/proc/cpuinfo` zvoli muj odhad vlastnosti, ktere pobezi rychle

• LIMIT: I/O subsystem linuxu

Cteni ze souboru v ramdisku 4 GB/s = 500 MS/s

Zapis do souboru v ramdisku 2.1 GB/s = 262 MS/s

Predavani dat rourou 1.2 GB/s = 150 MS/s