ACCELERATION OF SIGNAL PROCESSING USING GPU

Milan Bartl

Master Degree Programme (2), FEEC BUT

E-mail: xbartl02@stud.feec.vutbr.cz

Supervised by: Michal Trzos

E-mail: xtrzos00@stud.feec.vutbr.cz

Abstract: This paper focuses on using parallel General Purpose GPU computing for accelerating digital signal filtering and its benefits in comparison with common CPU processing. First part describes a basic algorithm for simulation of Finite Impulse Response (FIR) filter and its implementation into GPU. Second part shows benchmark results, measuring CPU and GPU computation time of this algorithm.

Keywords: GPGPU, parallel computing, CUDA, FIR, convolution

1. ÚVOD

Tato práce se zabývá možností využití paralelních výpočtů na GPU při simulaci signálových filtrů.

Při implementaci číslicových filtrů (např. v programu MATLAB) je zapotřebí velkého množství výpočtů. Tyto výpočty mohou dosahovat v závislosti na typu filtru různé složitosti. Faktor nejmarkantněji ovlivňující dobu trvání simulace není složitost filtru, nýbrž množství prvků vstupního signálu. Působení filtru lze zpravidla vypočítat pro každý prvek signálu zvlášť, nezávisle na ostatních. To činí z tohoto druhu zpracování vhodného kandidáta pro paralelní zpracování na grafickém procesoru.

V první části této práce je popsán algoritmus, jenž umožňuje využít grafického akcelerátoru ke zrychlení výpočtů během filtrace. Tento algoritmus je naprogramován v jazyce C s implementovaným rozhraním CUDA. Druhá část se zabývá měřením a porovnáním hodnot doby zpracování algoritmu na CPU a na GPU. Více informací o měření a srovnávání výpočetního času (benchmarku) CPU a GPU lze nalézt v článku [3].

2. VÝSLEDKY

2.1. ROVNICE FILTRU

Pro demonstraci byl vybrán jednoduchý filtr s konečnou délkou impulzní odezvy (FIR). Patří mezi diskrétní lineární filtry používané za účelem potlačení či zvýraznění určitých spektrálních složek digitálního signálu, případně změny jejich fázového posunutí. [1]

V praxi je výstup filtru vypočítán konvolucí prvků vstupního signálu s prvky impulzní odezvy. Konvoluční výpočet reprezentuje velké množství dílčích operací násobení a sčítání. Počet těchto výpočtů stoupá s rostoucím počtem prvků vstupního signálu. Matematické vyjádření použití konvoluce pro výpočet výstupu FIR filtru reprezentuje rovnice

$$y[n] = \sum_{j=0}^{M-1} h[j]x[n-j], \tag{1}$$

kde x[n] je N-vzorkový vstupní signál mající vzorky 0 až $N-1$ a h[n] je impulzní odezva filtru o M vzorcích (0 až $M-1$). Výstupní signál y[n] bude mít celkem $N+M-1$ vzorků.

2.2. ALGORITMUS

Jádrem paralelního kódu pro GPU je tzv. kernel. Vstupními hodnotami kernelu jsou ukazatele na impulzní odezvu, vstupní signál, výstupní signál a délky vstupního signálu a impulzní odezvy.

Celý proces konvoluce je rozdělen do jednotlivých vláken, přičemž každému vláknu náleží výpočet jednoho prvku výstupního signálu. Počet vláken je tedy roven $N + M - 1$.

Do celočíselné proměnné *n* je uloženo ID reprezentující pořadí vlákna v celkovém sledu vláken, spuštěných daným kernelem. Hodnota tohoto ID se pohybuje v intervalu od 0 do $N + M - 2$. Tuto informaci nesou proměnná *threadIdx*, udávající pozici vlákna v bloku, a proměnná *blockIdx*, specifikující pozici bloku v mřížce (*grid*), násobená velikostí (počtem prvků) v bloku. [2]

Pro ukládání mezivýpočtů je deklarována proměnná *buffer* a je inicializována hodnotou 0. Následuje smyčka, v níž jsou vzájemně násobeny patřičné prvky vstupního vektoru a impulzní odezvy podle rovnice (1). Výsledek násobení je přičten k hodnotě v proměnné *buffer*. Nakonec je obsah této proměnné uložen na náležité místo ve výstupním vektoru.

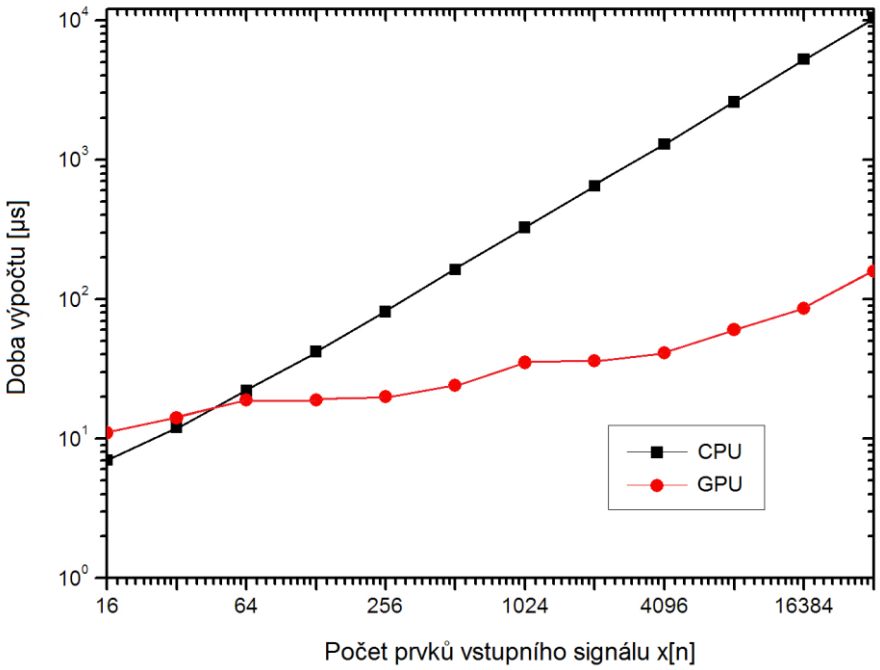
Uvnitř smyčky je kromě samotného násobení implementován také mechanizmus pro kontrolu indexování. Ošetřuje případy, kdy výraz $n - j$ nabývá hodnot mimo rozsah vektoru *x*.

```
__global__ static void conv(float * h, float * x, float * y,
      int N, int M)
{
      int n = blockIdx.x * blockDim.x + threadIdx.x;
      float buffer = 0;

      for(int j = 0; j < N; j++)
      {
            if(n - j < 0) break;
            if(n - j > M - 1) j = n - M + 1;

            buffer += h[j] * x[n-j];
      }
      y[n] = buffer;
}
```

2.3. BENCHMARK

Srovnávací test byl proveden na PC s CPU AMD Athlon 64 X2 Dual Core 5200+ s taktovací frekvencí 2.61 GHz a grafickou kartou NVIDIA GeForce GTX 460. Graf na obrázku 1 zachycuje naměřené časy potřebné ke zpracování daného algoritmu na CPU a GPU v závislosti na délce vstupního signálu. Délka impulzní odezvy má konstantní hodnotu 64 prvků.

Graf ukazuje, že při malých délkách signálu zabírá sériový výpočet na CPU kratší dobu než paralelní na GPU. Je tomu tak především díky vyšší taktovací frekvenci CPU oproti multiprocesorům, tvořícím základní operační jednotky na GPU, technologii hyperthreadingu a superskalární architektuře CPU. S rostoucím počtem prvků však získává GPU značnou převahu, která nakonec přesahuje 10 ms.



Obrázek 1: Graf závislosti doby výpočtu konvoluce na délce vstupního signálu

Zpracování dat na grafickém akcelerátoru vyžaduje od řídícího programu jistou režii. Tato režie je časově velice drahá. Řídícímu programu konvolučního jádra zabralo zpracování režijních operací od 1 do 2 milisekund. Detailní průzkum tohoto jevu je mimo oblast zaměření této práce (více v [3]). Režií se rozumí například přenos dat mezi RAM a pamětí GPU. Minimalizace tohoto negativního efektu lze dosáhnout provedením maximálního množství výpočtů přímo na GPU. V kontextu zpracování signálů to implikuje např. modelování systému filtrů, kde jsou data přenášena z výstupu jednoho filtru na vstup druhého, tedy v rámci paměti na GPU bez nutnosti intervence RAM.

3. ZÁVĚR

Při zpracovávání signálů počítačem vykonává CPU velké množství na sobě nezávislých výpočtů. Moderní technologie paralelního zpracování dat pomocí grafických karet umožňují tento sériový proces zrychlit o několik řádů.

Jako modelový příklad byl zvolen FIR filtr, jehož výstup představuje konvoluci vstupního signálu s impulzní odezvou filtru. V jazyce CUDA C byl zpracován algoritmus pro paralelní výpočet konvoluce na GPU. Výpočetní rychlost tohoto algoritmu byla porovnávána s týmž algoritmem upraveným pro sériový výpočet na CPU.

Se vzrůstající délkou vstupního signálu markantně rostla doba výpočtu algoritmu na CPU. U GPU došlo k výraznějšímu prodloužení výpočetního času až při vysokém počtu prvků vstupního signálu vlivem vyčerpání zdrojů na GPU. Rozdíl mezi časy CPU a GPU dosahoval až dvou řádů ve prospěch GPU.

REFERENCE

[1] SMÉKAL, Z. *Číslicové zpracování signálů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky komunikačních technologií, 2010. s. 17-26.

[2] NVIDIA: *NVIDIA CUDA C Programming Guide, Version 3.2*. Santa Clara, CA, USA, 2010

[3] PÁLENÍK, T. Utilizing Parallelism in Simulation of a Communication System Using LDPC Codes. In *Telecommunications and Signal Processing TSP-2009 : 32nd International Conference*. Dunakiliti, Hungary : Asszisztencia Szervezö Kft., 2009. ISBN 978-963-06-7716-5.