Portál pre odborné publikovanie ISSN 1338-0087

Urýchlenie numerickej simulácie s využitím grafickej karty

Sebestyén Kamil · Informačné technológie, Študentské práce

07.07.2010

Práca sa zaoberá návrhom a implementáciou numerického algoritmu pre grafický procesor. Algoritmus rieši explicitnou metódou parabolickú parciálnu diferenciálnu rovnicu v dvojrozmernom priestore. Ukážeme, že aplikácia grafickej karty môže numerický výpočet zrýchliť až 9x. Tiež sa hľadá, pre aké hodnoty parametrov beží program najrýchlejšie.

1. Úvod

Vo fyzike aj v mnohých technických aplikáciách sme schopní získať relevantné výsledky len pomocou numerických simulácií. Napriek prudkému vývoju počítačov sú numerické simulácie často relatívne pomalé. Na výsledky sa často čaká aj niekoľko týždňov až mesiacov. Prevádzka výpočtových klastrov je finančne i energeticky veľmi náročná. Grafické karty ponúkajú možnosť podstatne urýchliť niektoré numerické úlohy. Paralelná výpočtová sila jedného grafického procesora (GPU) môže nahradiť celý klaster počítačov. Cieľom našej práce je otestovať možnosti GPU na jednom numerickom probléme.

1.1. História GPGPU

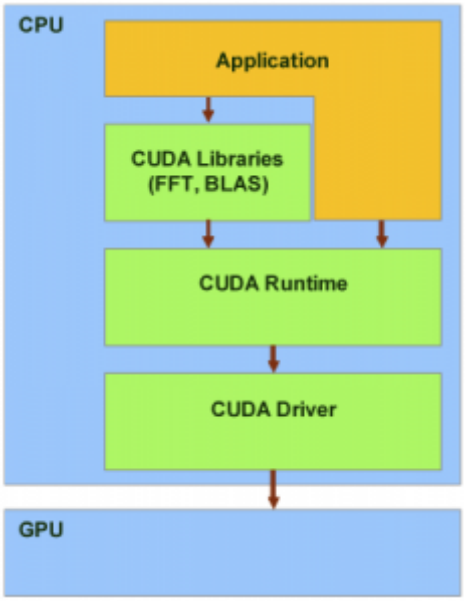
Koncom 90. rokov 20. storočia sa grafické karty stali aj výkonnými akcelerátormi 2D a 3D grafiky. V roku 1999 vyrobila spoločnosť Nvidia prvú grafickú kartu s GPU – GeForce 256. Druhá generácia GPU obsahovala prvú programovateľnú shader pipeline. Počet shaderov na GPU postupne narastal a tiež sa zlepšovali možnosti ich programovania. Vznikli špecializované programovacie jazyky pre vertex shadery – HLSL (Direct3D) a GLSL (OpenGL). Tieto jazyky umožnili využiť GPU aj pre negrafické výpočty – teda GPGPU (general-purpose computing on graphics processing units).

Dáta na spracovanie sa zakódovali do textúr a tie boli spracované shadermi. Neskôr sa objavili grafické karty s unifikovanými shadermi, ktoré mohli plniť rôzne úlohy ako pixel, vertex a geometrické shadery a tiež boli veľmi vhodné pre účely GPGPU. Unifikované shadery dali vzniknúť aj niekoľkým programovacím jazykom pre GPGPU ako C for CUDA a OpenCL.

1.2. CUDA

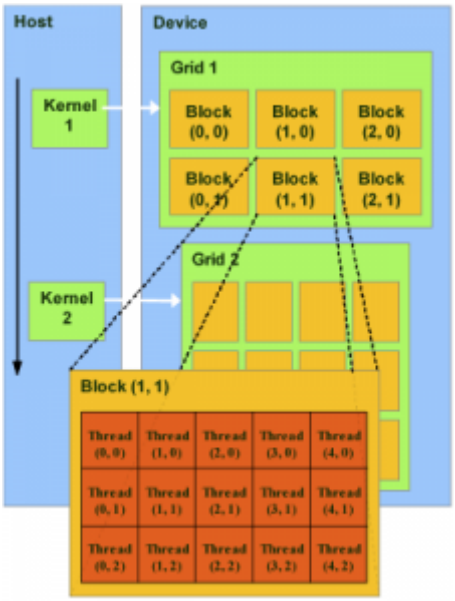
CUDA (Compute Unified Device Architecture) je architektúra GPU od spoločnosti

Nvidia používaná od grafických kariet GeForce 8 vhodná aj pre účely GPGPU. Užívateľ môže využiť CUDA na rôznych úrovniach, či už v podobe hotových aplikácií a knižníc, alebo napísať si vlastné funkcie v programovacom jazyku C for CUDA alebo CUDA Driver API (Obr. 1).



Obr. 1. CUDA

Pri CUDA každý multiprocesor spúšťa tu istú inštrukciu, ale nad rozdielnymi dátami. GPU má k dispozícii tiež vlastnú pamäť. Kernel je funkcia určená pre GPU spúšťaná vláknami. Hlavnou výhodou GPU oproti CPU je to, že dokáže paralelne spúšťať veľké množstvo vlákien. Jednotlivé vlákna sú usporiadané do blokov (maximálne 512 vlákien/blok), ktoré sú usporiadané do mriežky (Obr. 2). Obe tieto štruktúry môžu byť 1-, 2- a 3-rozmerné [1]. Jednotlivé kernely sú spúšťané z hosťovského kódu.

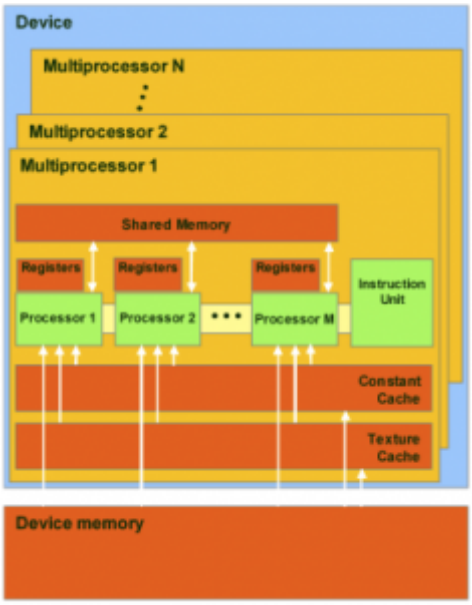


Obr. 2. CUDA programovací model

CUDA využíva nasledujúce typy pamäte (Obr. 3):

- Globálna pamäť – je najväčšia, môžu k nej pristupovať všetky vlákna, má veľkú šírku pásma, ale má veľmi vysoké latencie.

- Lokálna pamäť – môže k nej pristupovať len procesor, ku ktorému je priradená.
- Zdieľaná pamäť – je zdieľaná medzi vláknami v jednom bloku, umožňuje im komunikovať navzájom, má nízke latencie.
- Cache na konštanty, Cache na textúry



Obr. 3. CUDA pamäťový model

2. Postup riešenia

2.1. Definícia fyzikálneho problému

Budeme riešiť Schrödingerovu rovnicu pre kvantovú časticu v dvojrozmernom náhodnom prostredí:

$$i\frac{\partial\varphi}{\partial t} = -\frac{\partial^2\varphi}{\partial x^2} - \frac{\partial^2\varphi}{\partial y^2} + V(x,y)\varphi \tag{1}$$

kde φ(x, y, t) je vlnová funkcia častice. V rovnici (1) sme pre jednoduchosť všetky fyzikálne konštanty položili rovné 1. Schrödingerova rovnica je zaujímavá preto, lebo na rozdiel od difúznej rovnice vyžaduje prácu s komplexnými číslami a teda lepšie preverí možnosti GPU. Pre numerické riešenie aproximujeme derivácie diferenciami a nimi (1) prepíšeme do tvaru:

$$\begin{aligned}\varphi(x,y,t+\delta t) = \varphi(x,y,t) + \frac{i\delta}{\Delta^2}[\varphi(x+\Delta,y,t)+\\ +\varphi(x-\Delta,y,t) + \varphi(x,y+\Delta,t) + \varphi(x,y-\Delta,t)+\\ +\varphi(x,y,t)(E(x,y)-4)]\end{aligned} \tag{2}$$

Rovnica (2) predstavuje typickú explicitnú metódu na numerické riešenie parabolickej diferenciálnej rovnice [3]. Budeme hľadať hodnoty φ(x, y) v bodoch x = ixΔ, y = iyΔ, kde 1 ≤ ix, iy, ≤ N. N definuje veľkosť priestoru. Rovnica (1) umožňuje vypočítať časový priebeh vlnovej funkcie. Výpočet začneme s počiatočnou podmienkou φ(t = 0) = 1 pre bod v strede mriežky ix = iy = N/2. Presnosť výpočtu overíme v každom čase t

tak, že vypočítame normu:

$$\sum_{ix,iy} |\varphi(ix,iy,t)|^2 = 1 \tag{3}$$

pre všetky časy t. Stabilita riešenia samozrejme závisí od voľby parametrov δ a Δ. V našich výpočtoch používame $\delta/\Delta^2 = 0{,}0001$. Náhodný potenciál je modelovaný funkciou E(x, y) v rovnici (2). Pre každé x, y je náhodné číslo E(x, y) generované z intervalu (-W/2, W/2) a W = 3.

2.2. Požiadavky na riešenie

Funkcia by mala vedieť počítať s komplexnými číslami s dvojitou presnosťou. Na tento účel bol použitý dátový typ cuDoubleComplex, ktorý je zadefinovaný v knižnici cuBLAS (BLAS pre CUDA). Dvojitú presnosť podporuje CUDA od verzie 2.3, ale len pre grafické karty s GPU GT200 a GF400. Funkcia by mala byť volateľná nielen z programu napísaného v C, ale i vo Fortrane. Toto je riešené pomocou tzv. „wrapperu“, ktorý je napísaný v C, obaľuje kernel a dá sa použiť ako externá funkcia. Dátový typ cuDoubleComplex je kompatibilný s fortranovským dátovým typom CMPLX*16.

2.3. Implementácia

Na implementáciu samotného kernelu bol použitý programovací jazyk C for CUDA vo verzii 2.3. Na kompiláciu kernelu bol použitý kompilátor nvcc od spoločnosti Nvidia a na kód v C a Fortrane gcc/gfortran. C kód načíta potrebné parametre a skopíruje vstupné matice na grafickú kartu. Potom v cykle podľa počtu časových krokov spúšta samotný kernel.

Ten je spustený vo vláknach pre každý prvok matice. Jeho úlohou je podľa rovnice (2) vypočítať hodnotu prvku v čase t + δ na základe susedných prvkov. Aby sa zabránilo zbytočnému kopírovaniu, po každom kroku sa len prehodia smerníky matíc φ(t) a φ(t + δ). Po skončení cyklu sa skopíruje výsledok naspäť do pamäte RAM PC.

3. Výsledky

Výsledný program bol otestovaný na nasledujúcej zostave:

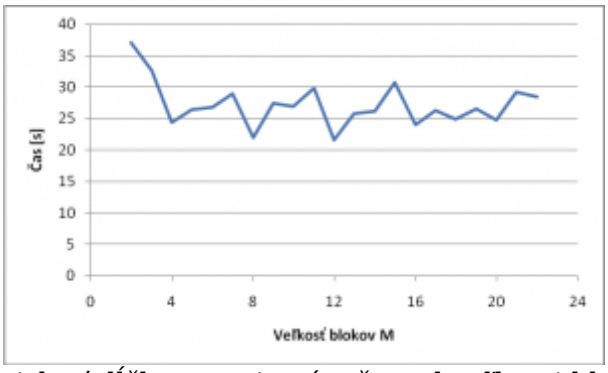
- Intel 2 Duo E8400, 3 GHz, 6 MB L2 Cache
- 4 GB DDR2 RAM
- GeForce GTX 285 (Tab. 1)
- Ubuntu 9.10

Tab. 1. Špecifikácia GPU.

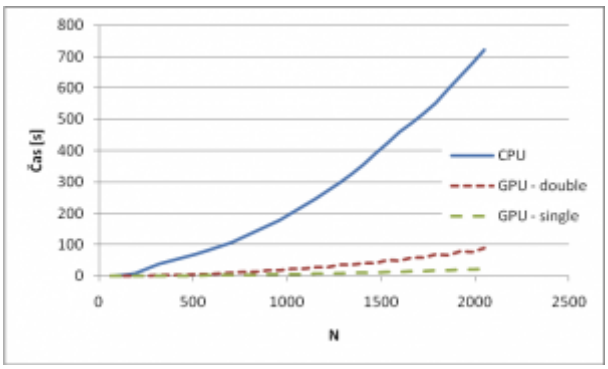
GPU	
CUDA jadrá	240
Frekvencia jadra	648 MHz
Frekvencia shaderov	1476 Mhz
Pamäť	

Frekvencia pamäte	1242 MHz
Konfigurácia	1 GB GDDR3
Šírka pamäťovej zbernice	512 bitov
Šírka pamäť. pásma	159,0 GB/s

Verzia použitých ovládačov grafickej karty je 190.18.
Vo všetkých nasledujúcich výsledkov bol použitý počet iterácií 10 000. Algoritmus bol testovaný v jazykoch Fortan, C a C for CUDA a bola porovnaná presnosť výsledkov, ktorá bola samozrejme totožná. Najprv sme experimentálne zistili optimálne usporiadanie vlákien M = 8 testovaním rôznych veľkostí blokov (Obr. 4). Pre M = 12 bol výpočet síce rýchlejší, ale numericky nestabilný.

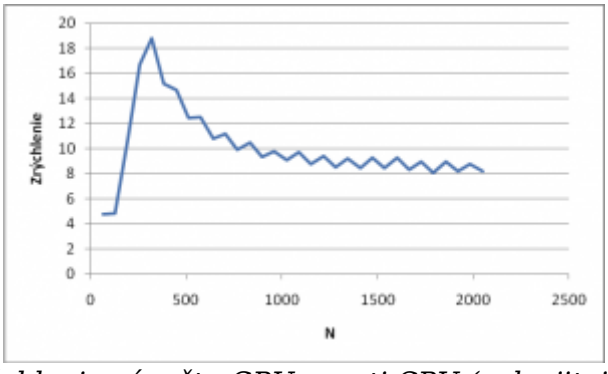


Obr. 4. Závislosť dĺžky trvania výpočtu od veľkosti blokov MxM.



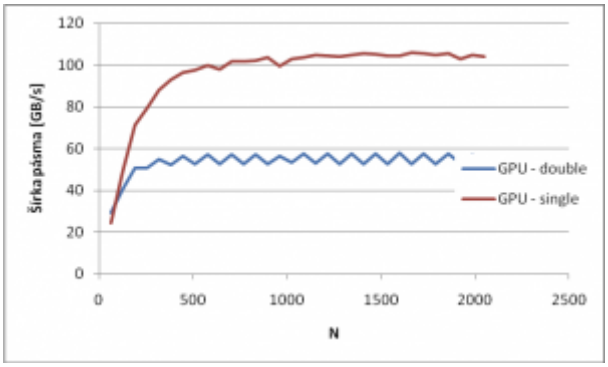
Obr. 5. Porovnanie výkonu CPU a GPU.

Podstatný rozdiel bol v dobe trvania výpočtu. Obr. 5 porovnáva výkon CPU a GPU v jednoduchej aj dvojitej presnosti komplexných čísel (Obr. 6). V obidvoch prípadoch je GPU oveľa rýchlejšia ako CPU. Pri dostatočne veľkých maticiach sa urýchlenie ustáli na približne 9x.

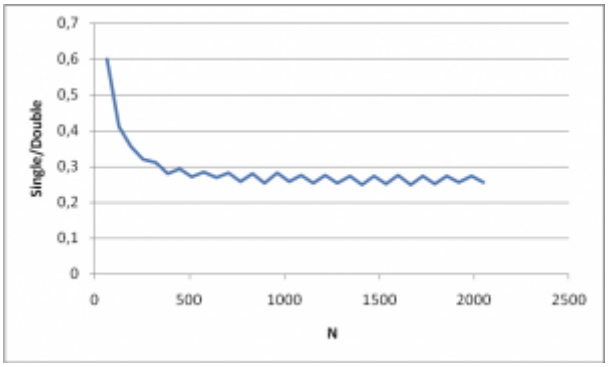


Obr. 6. Urýchlenie výpočtu GPU oproti CPU (v dvojitej presnosti).

Program dosiahne maximálnu efektívnu šírku pásma pri maticiach 512×512 a väčších 55 GB/s pri dvojitej presnosti a až 105 GB/s pri jednoduchej presnosti (Obr. 7).



Obr. 7. Závislosť efektívnej šírky pásma od veľkosti matice



Obr. 8. Spomalenie pri dvojitej presnosti oproti jednoduchej presnosti.

Pri porovnaní výkonu pri jednotlivých presnostiach je vidno, že program využívajúci dvojitú presnosť je približne 4x pomalší. Tento jav je spôsobený hlavne tým, že GPU postavené na báze GT200 majú shadery s 8 SP (single precision) jednotkami, ale len jednou DP (double precision) jednotkou. Nasledujúca generácia GPU GF400 od spoločnosti Nvidia obsahuje rovnaký počet SPU aj DPU, a preto by mali byť tieto rozdiely vo výkone pre rôzne presnosti oveľa nižšie.

4. Zhodnotenie

Vytvorili sme program, ktorý rieši explicitnou metódou parabolickú diferenciálnu rovnicu a na všetky hlavné výpočty využíva GPU namiesto CPU. Ukázali sme, že táto úloha je veľmi dobre paralelizovateľná a pri jej implementácii pre CUDA sme dosiahli urýchlenie 9x pre veľké matice. Program využíva efektívne dostupnú pamäť na grafickej karte.

Moderné GPU sú kvôli obrovskej paralelnej výpočtovej sile vhodné na náročné numerické výpočty ako je napríklad numerické riešenie diferenciálnych rovníc, problémy lineárnej algebry a pod., ktoré tvoria základ väčšiny numerických algoritmov používaných v technických aplikáciách.

5. Odkazy na literatúru

1. CUDA C PROGRAMMING GUIDE 2.3, 2009
   http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide_2.3.pdf

2. CUDA C PROGRAMMING BEST PRACTICES GUIDE, 2009 http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_BestPracticesGuide_2.3.pdf
3. Press, William a kol., Numerical Recipes in C, Cambridge University Press, 1992

Spoluautorom článku je Peter Markoš, Fakulta elektrotechniky a informatiky STU Bratislava