COMPILER FROM C# LANGUAGE TO NVIDIA LANGUAGE

Jiří Zajíc

Master Degree Programme (2), FIT BUT

E-mail: xzajic10@stud.fit.vutbr.cz

Supervised by: Peter Jurnečka

E-mail: ijurnecka@fit.vutbr.cz

Abstract: This project is focused on GPU accelerated calculations on Nvidia grapics cards. CUDA technology is used and converted to implementation on a .NET platform. The problem is solved as compiler from C# programming language to Nvidia CUDA language with extension attributes of C# language that preserves the same action semantics of CUDA language. Application is implemented in C# programming language and uses NRefactory, the open-source library.

Keywords: CUDA, C#, .NET, compiler, NRefactory

1. ÚVOD

S vědomím, že se při programování potýkáme se stále větší mírou abstrakce, se vyvíjí programovací jazyky, které toto paradigma zohledňují. Současně je ale vyvíjen tlak na přesun náročných výpočtů z procesoru na grafickou kartu. Existuje několik technologií, jež umožňují programovat paralelní výpočty na grafických kartách. A jsou to zejména technologie CUDA (od společnosti NVidia) a ATI Stream od společnosti (AMD/ATI). Obě technologie využívají jazyky nízké úrovně (CUDA C a OpenCL) pro implementaci programů.

Práce je zaměřena na zpřístupnění paralelních výpočtů prostřednictvím technologie CUDA pro uživatele, kteří programují ve vysokoúrovňových jazycích pod platformou .NET [1], zejména pak v jazyce C# [2].

2. TECHNOLOGIE CUDA

Technologie CUDA [5] (Computed Unified Device Architecture) byla představena v roce 2006. Aktuální je verze 4 vydaná v květnu 2011, která podporuje výpočty na několika grafických kartách najednou. Každá karta je rozdělena na několik multiprocesorů (typicky složených z bloku 8 procesorů řízených jednou instrukční jednotkou) a globální paměť, která slouží pro kopírování dat z operační paměti na grafickou kartu a naopak.

Typický běh CUDA aplikace probíhá tak, že je nejprve inicializováno zařízení (grafická karta) a následně jsou na něj překopírovány operandy výpočtu. Nad těmito operandy se provede výpočet a výsledek je překopírován ze zařízení zpět do operační paměti. Hlavním stavebním kamenem je tzv. *kernel*, neboli funkce spouštěná na grafické kartě současně v několika vláknech a blocích.

Nativně je pro implementaci algoritmů podporován jazyk C v podobě jeho modifikace s názvem CUDA C. Hlavní odlišností tohoto jazyka od standardního jazyka C je volání *kernel* funkcí, které zahrnuje tzv. konfiguraci pro spuštění (*execution configuration*), jež udává počet bloků a vláken, ve kterých se má daná funkce současně spustit:

```
kernel_function<<10,10>>(argument1, argument2, … , result);
```

3. EXISTUJÍCÍ ŘEŠENÍ

Spojení technologie CUDA s platformou .NET existuje několik (viz např. [3]). Jedná se o řešení, která zahrnují syntaktický podklad pro převod C# kódu do jazyka CUDA C. Složení všech částí do spustitelného celku musí programátor provést ručně. Prezentovaná aplikace zahrnuje jednak převod syntaxe, ale i nástroje pro překlad, složení a spuštění celých projektů.

4. STRUKTURA APLIKACE

Aplikace je koncipována jako výukový systém pro začínající programátory, kteří mají zkušenost s jazykem C# a chtějí si vyzkoušet práci na paralelních výpočtech s využitím grafické karty jako prostředku pro urychlení výpočtu. Díky využití jazyka C#, tedy jazyka s vysokou mírou abstrakce, byl kladen zřetel na zapouzdření některých částí jazyka CUDA C, u kterých by abstrakce syntaktických konstrukcí neohrozila výpočetní sílu jazyka CUDA C.

Jedním z těchto prostředků je zapouzdření volání výpočtu na grafické kartě. Jelikož C# má automatickou správu paměti, ruční alokování paměti na zařízení není žádoucí. Veškerá režie spojená s alokací paměti, přenesením operandů, voláním funkce a přenesením výsledku zpět je zapouzdřena do volání jediné C# funkce. Ta se při převodu do CUDA C transformuje na zmíněné konstrukce.

Jazyk CUDA C používá rozšíření pro značení *kernel* funkcí. Tato rozšíření byla převedena do jazyka C# pomocí vlastních atributů. Byly též definovány konstanty a funkce v jazyce C#, které představují ekvivalentní vestavěné funkce jazyka CUDA C.

Dalším problémem je převod tříd a jmenných prostorů do prostředí standardního jazyka C. V rámci zjednodušení práce s třídami je převod proveden pomocí prefixů. Každá funkce a veřejná proměnná je upravena tak, aby začínala jménem třídy, ve které se nachází. Současně je předloženo jméno jmenného prostoru, ve kterém se nachází daná třída. Celý název funkcí je tedy složen ze jména jmenného prostoru, jména třídy a jména funkce. Kontrola přístupu k privátním proměnným je zajištěna sémantickou analýzou při překládání C# kódu.

4.1. ARCHITEKTURA APLIKACE

Předpokládaný postup při práci s aplikací je následující – uživatel (programátor) napíše zdrojový kód v jazyce C# za pomoci výrazových prostředků, které mají stejnou sémantiku jako v jazyce CUDA C. Nad tímto kódem proběhne syntaktická a sémantická kontrola pro jazyk C#. Pokud tato analýza proběhne v pořádku, je kód rozdělen na úroveň jednotlivých instrukcí. Pro jednotlivé instrukce je vygenerován jejich ekvivalent v CUDA C v kontextu zdrojového kódu. Výsledný kód je přeložen překladačem. Pokud překlad proběhne v pořádku, je program spuštěn s prezentací výsledků. Pokud v kterékoli části převodu dojde k chybě, je tato nahlášena, prezentována a běh převodu se zastaví.

Aplikace je rozdělena do čtyř vzájemně spolupracujících projektů: knihovna rozšíření (CUDALib), překladač zdrojového kódu v jazyce C# (Parser), generátor cílového kódu v jazyce CUDA C (Generator) a interpret výsledků a zároveň vstupní bod aplikace (Compiler).

4.2. KNIHOVNA CUDALIB

Knihovna poskytuje ekvivalentní rozšíření k vestavěným funkcím CUDA C. Obsahuje definice atributů pro označování funkcí, ekvivalenty vestavěných funkcí a proměnných CUDA C a zejména konstrukci pro volání *kernel* funkcí.

Jelikož je *kernel* funkce v CUDA C definována vždy bez návratové hodnoty a součástí jejího volání je konfigurace pro spuštění, byla vytvořena generická funkce *Call*, která vrací výsledek datového typu podle volané funkce. Pomocí argumentů jsou funkci předány počty bloků a vláken a odkaz na *kernel* funkcí, která se má spustit, pomocí anonymní funkce.

4.3. PŘEKLADAČ C# KÓDU

Nejjednodušší projekt celé aplikace obsahuje syntaktickou a sémantickou analýzu C# zdrojového kódu. Z velké části je využita open-source knihovna NRefactory [4], která slouží pro syntaktickou a sémantickou analýzu. Jejím výsledkem je chybové hlášení v případě neúspěchu překladu. V případě úspěchu je vytvořen abstraktní syntaktický strom, který je předán projektu generátoru. Projekt tvoří třída, která zabaluje volání knihovny NRefactory a zpracovává její výstup.

4.4. GENERÁTOR CÍLOVÉHO KÓDU V CUDA C

Rozhraní s okolím tohoto projektu tvoří třída, která přejímá abstraktní syntaktický strom, nachází v něm definice jmenných prostorů a volá odpovídající generátorové třídy. Kromě těchto tříd se v projektu nacházejí manažeři různých částí generování a tabulka symbolů.

Generátorové třídy tvoří jádro celé aplikace. Tvoří hierarchickou strukturu a všechny dědí od základní abstraktní třídy, která zahrnuje reprezentaci cílové instrukce pomocí řetězce, abstraktní metodu pro zpracování instrukce a potlačení funkce *ToString*(), jež vrací textovou reprezentaci instrukce v CUDA C. Hierarchicky jsou pak seskupeny třídy do skupin podle toho, zda se jedná o výraz, deklaraci nebo kontrolní blok jako podmíněný příkaz, příkaz cyklu apod. Pro každou skupinu je definována taktéž abstraktní třída skupiny, která dědí od zmíněné základní třídy.

Manažeři jsou třídy navržené podle vzoru jedináček. Obstarávají funkce pro správu CUDA identifikátorů, datových typů nebo volání *kernel* funkcí. Součástí je i manažer pro tabulku symbolů.

Tabulka symbolů tvoří samostatný modul tohoto projektu, s definicí samostatných tabulek pro třídy, jmenné prostory a celou aplikaci. Ukládá se každý nalezený identifikátor. Tyto informace slouží při generování pro kontrolu typů a platnost rozsahu jednotlivých proměnných. Výsledkem celého projektu je vygenerovaný zdrojový kód v jazyce CUDA C.

4.5. INTERPRET VÝSLEDKŮ

Vstupní bod celé aplikace tvoří projekt s grafickým uživatelským rozhraním ve WPF (Windows Presentation Foundation). Tento projekt spojuje celou aplikaci dohromady. Vstupní zdrojové kódy v jazyce C# předkládá překladači s knihovnou NRefactory, výsledný abstraktní syntaktický strom předává generátoru a vygenerovaný zdrojový kód překládá a spouští. Prezentuje též případné chyby vzniklé při převodu.

5. ZÁVĚR

Aplikace nabízí možnost uživatelům seznámit se s architekturou CUDA pomocí jazyka vysoké úrovně. Uživatelé mohou pracovat s paralelními algoritmy s omezenou množinou výrazových možností jazyka C#. Výsledky a problémy jsou jim zprostředkovány pomocí přehledného uživatelského rozhraní.

REFERENCE

[1] *.NET Framework Conceptual Overview* [online]. 2012. [cit. 1.3.2012] Dostupné z WWW: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx

[2] WATSON, Ben.: *C# 4.0*. Brno: Zoner Press, 2010. ISBN 978-80-7413-094-6

[3] NICK, Kopp.: *CUDAfy.NET* [online]. 14.11.2012. [cit 25.3.2012] Dostupné z WWW: http://cudafy.codeplex.com/

[4] GRUNWALD, Daniel.: *NRefactory* [online]. 19.11.2011. [cit. 29.2.2012] Dostupné z WWW: http://wiki.sharpdevelop.net/NRefactory.ashx

[5] *NVIDIA CUDA Library Documentation* [online]. 2012. [cit. 1.3.2012] Dostupné z WWW: http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/online/index.html