Posudek diplomové práce

Matematicko-fyzikální fakulta Univerzity Karlovy

Autor práce	Bc. Petr Šťavík		
Název práce	Analysis and Optimizing GPU Kernels with Machine Learning		
Rok odevzdání	2020		
Studijní program	Informatika	**Studijní obor**	Softwarové systémy

Autor posudku	Milan Straka	**Role**	Oponent
Pracoviště	Ústav formální a aplikované lingvistiky		

Text posudku:

Diplomová práce se zabývá analýzou programů v jazyce PTX , což je mezijazyk používaný platformou CUDA pro programování GPU. Práce rozpracovává dvě konkrétní úlohy, a to

- rozpoznání, zda daný PTX kód poběží rychleji na CPU či GPU;
- předpověď vytížení GPU při vykonávání PTX kódu, konkrétně *achieved occupancy*, což je průměrný poměr aktivních warpů při výpočtu GPU.

Obě tyto úlohy jsou příkladem *text classification* problému, který je hojně studovaný (byť většinou v kontextu přirozeného místo programovacího jazyka) a který je řešitelný mnoha různými modely strojového učení – autor používá konkrétně rekurentní neuronové sítě.

Obsah práce

V první šestnáctistránkové kapitole popisuje autor základy programování GPU, metody optimalizace GPU kódu (auto-tuning) a úvod do strojového učení. Dalších devět stran popisuje současné nejlepší metody pro analýzu kódu pomocí strojového učení (DeepTune a NCC). Vlastní přínos je popsán v kapitole 3 (16 stran; popis dvou architektur ptx2vec a ptx_ab) a kapitole 4 (11 stran; výsledky navržených metod na dvou výše zmíněných úlohách).

Přínos práce

Práci bych popsal primárně jako vědeckou – popisuje dvě architektury pro analýzu PTX mezikódu, které pak aplikuje na dvou úlohách, z nichž je jedna existující a druhá nově navržená.

Oba navržené přístupy jsou založené na existujících architekturách – *ptx2vec* je velice podobná NCC, je pouze upravená pro PTX místo LLVM IR mezikódu. Předtrénovává embeddingy jednotlivých instrukcí na základě *contextual flow grafu*, což je zjednodušeně graf datových a řídících (*control*) závislostí daného programu. Druhá architektura *ptx_ab* nevyužívá předtrénované embeddingy a místo toho trénuje reprezentace instrukcí (obecnějších než v případě ptx2vec) zároveň se zbytkem modelu.

První z uvedených architektur je tedy reimplementace existujícího přístupu, a samotné modely strojového učení jsou bez větší změny založené na existujících zdrojových kódech. Přesto vyžádalo provedení experimentů jistě značnou práci – použitý mezikód je jiný, což znamenalo nejen převést do něj existující dataset, ale navrhnout správnou granularitu reprezentace jednotlivých instrukcí a v neposlední řadě získání velké kolekce PTX kódu (17 milionů řádek).

Na první existující úloze se řešitel porovnává s nejlepšími známými modely a dosahuje podobných výsledků jako DeepTune a NCC, avšak horších než NCC-imm. Druhá úloha je nově navržená – autor vytvořil veřejně dostupnou kolekci 244 PTX programů a jim příslušným vstupů a naměřil pro ně *achieved occupancy* metriku na existující grafické kartě. Nakonec použil popsané architektury a získal první výsledky v této nově navržené úloze.

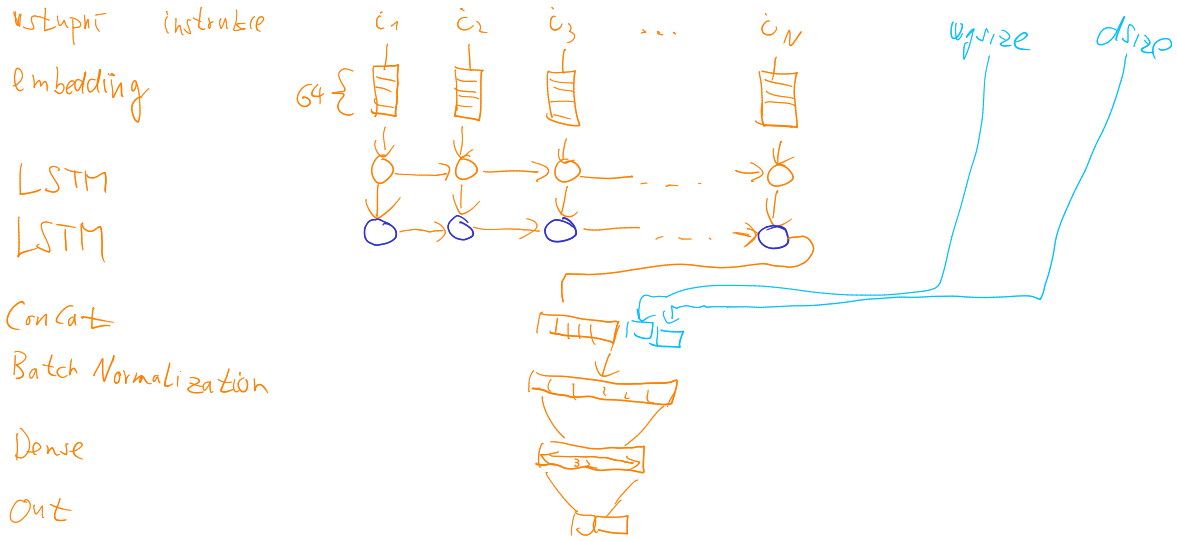
Pokládám přínos práce za dostatečný, prokazující schopnost samostatné vědecké práce a splňující požadavky na diplomovou práci.

Nedostatky práce

- Zásadní metodologický problém je způsob získání výsledků u obou úloh. Autor využívá 10-fold cross-validation, což je zcela v pořádku, ale provede ji pětkrát a prezentuje **výsledky nejlepšího běhu**. Tento postup je v kontextu strojového učení **zcela chybný** – je pravda, že je použitý i v článku popisující NCC (žádné z tří reviews na něj neupozorňuje), ale to ho nijak nelegitimizuje. Správné je prezentovat buď výsledky náhodného běhu, nebo ještě lépe **průměru** jednotlivých běhů (včetně standardních odchylek, tj. hodnoty v tabulkách 4.1 a 4.3 by byly ve formátu *průměr±odchylka*).
  Sám autor na problémy této evaluace upozorňuje na straně 51, kde uvádí
  > „this value [of ptx2vec-imm] is more of an outlier and the average achieved speedup is actually lower than that of ptx_ab-imm“
- Autor prezentuje dvě varianty obou architektur, základní ptx2vec a ptx_ab a pak rozšířenou ptx2vec-imm a ptx_ab-imm. Ovšem rozšířená varianta -imm zjevně nedosahuje výsledků existující NCC-imm, což autor na straně 51 opět sám deklaruje
  > „… we can see that none of them was able to outperform NCC-imm model. Instead, our ptx2vec-imm and ptx_ab-imm models are more comparable to DeepTune or NCC without immediate values“

  Autor dále spekuluje, že horší výsledky jsou způsobené rozdíly mezi LLVM IR a PTX. Tento závěr je dle mého mínění zcela nepodložený. Rozdíly ve výsledcích jsou totiž způsobené špatným pochopením NCC-imm architektury – zatímco varianty ptx2vec-imm a ptx_ab-imm z práce odpovídají použití *auxiliary values* (v terminologii NCC), které se používají v předchozích pracích vždy, tj. jak v DeepTune tak v NCC, varianta NCC-imm přidává do reprezentace konkrétní *konstanty* ze zdrojového kódu, což autor ve svých -imm variantách nedělá, a přímé porovnání pak ztrácí smysl.
- Metrika navržená autorem pro měření *achieved occupancy* je úspěšnost predikce po oříznutí na celé desítky procent (a varianta, kdy dovolujeme, aby se výsledky oříznuté na celé desítky procent lišily nanejvýš o daný počet desítek). Tato metrika mi přijde **velmi nevhodná**, protože nezohledňuje skutečnou chybu jednotlivých predikcí (tj. řešení, které pro cílovou hodnotu 5% předpovídá 5% by mělo být lepší než řešení předpovídající 3% nebo 0%). Přitom je zcela standardní řešení použít metriku MSE.
  Chápu, že navržená metrika je založená na zvolené architektuře, kdy autor použil klasifikaci do deseti tříd namísto regrese. To ale neospravedlňuje použijí nevhodné evaluace. Navíc samotné řešení regrese klasifikací není vyřešeno ideálně – místo použití nejpravděpodobnější třídy je možné použít **váženou kombinaci** jednotlivých tříd, čímž i klasifikační model může předpovídat libovolnou reálnou hodnotu (v této situaci by se asi použilo jedenáct tříd odpovídající hodnotám 0%, 10%, …, 100%, a hodnotu 5% je pak možné reprezentovat jako distribuci (0.5, 0.5, 0, 0, ...)).

- Popis architektur, tj. jak text v sekci 4.1.2 tak obrázky 4.1 a 4.2, není jasný, a naznačuje nepochopení ze strany autora. Speciálně obrázky naznačují, že dvě vrstvy LSTM jsou obdobné (ani v textu se obě LSTM od sebe nijak neodlišují), a že vstup modelů má pevnou velikost 64 hodnot nezávisle na délce programu. To ale není pravda – velikost vstupu je závislá na délce programu, první LSTM vrstva vrací kontextualizovanou reprezentaci jednotlivých instrukcí, a druhá LSTM vrstva je použita jiným způsobem a vrací již pevně velkou preprezentaci celé posloupnosti (což je vidět v implementaci modelů). Obrázek 4.1 by měl vypadat spíš jako



  Zároveň je nestandardní řešení použít aktivační funkci *sigmoid* v případě dvou výstupů modelu, protože výstup modelu tak není distribuce – buď se používá jeden výstup plus *sigmoid*, nebo dva výstupy plus *softmax* (řešení 2 výstupy plus *sigmoid* se objevuje již v existující implementaci NCC a je odtud předpokládám přejato).

Návrhy ke zlepšení

- Přestože je na straně 52 uvedeno
  „It is worth mentioning that we also tried experimenting with a few different settings with the goal of improving the performance of our models. For example, because of a slight overfitting, we tried a few different regularization techniques like dropout or reduction of the complexity of the models. However, these attempts did not bring any conclusive improvements“
  autor nakonec prezentuje výsledky architektur v podstatě přejatých z NCC, což je trochu škoda.
  Úlohy klasifikace textu jsou řešitelné velkým množstvím způsobů, takže by bylo možné vyzkoušet nějaké běžné architektury (autor explicitně neříká, že to nezkusil, ale ani nepíše, že to zkusil). Zároveň má úspěšnost klasifikace textu většinou velký rozptyl, takže použít *ensembling* by velmi pravděpodobně výsledky měřitelně zlepšilo (stejně se trénování opakuje pětkrát, tak je škoda toho nevyužít).
- Bylo by zajímavé porovnat předtrénování ptx2vec embeddingů s čistě sekvenčním předtrénováním (tj. v situaci, kdy se jako kontext berou předcházející a následné instrukce ve zdrojovém kódu) – takové předtrénování je v porovnání s implementovanou metodou velmi jednoduché a více odpovídá tomu, jak jsou instrukce zpracovávány (tj. LSTM také zpracovává instrukce v pořadí, jak jsou uvedené ve zdrojovém kódu).

- Přirozené by bylo také implementovat varianty architektur odpovídající NCC-imm, což by nejspíš znatelně zlepšilo výsledky (osobně bych očekával dosažení úrovně NCC-imm).
- Vytvořená úloha předpovídání *achieved occupancy* má relativně malý dataset 244 kernelů – chápu, že je komplikované vytvořit dataset větší, ale možná by bylo možné vytvořit více cílových dat. Ideální by bylo například změřit nějaké hodnoty pro každou instrukci/řádku kódu (nvprof vypadá, že dokáže provést i nějakou *source-level analysis,* ale možná ne pro PTX kód) – tím by se množství hodnot předpovídaných modelem řádově zvětšilo, a model by mohl dokázat předpovídat *achieved occupancy* znatelně lépe na základě hodnot naměřených pro jednotlivé instrukce.

Konkrétní připomínky k textu

Diplomová práce je psaná v anglickém jazyce. Přestože text obsahuje větší množství gramatických chyb (chybějící členy, špatné tvary sloves, čárky), je bez problémů pochopitelný a je čtivý. I přes chyby hodnotím použití angličtiny kladně (lepší s chybami anglicky, než česky bez chyb).

- Na straně 5 autor tvrdí, že ANN jsou známé od čtyřicátých let. To není přesné, například samotný algoritmus backpropagation, který je základem pro trénování neuronových sítí, byl popsán v roce 1986. Obecně je rozvoj hlubokých sítí nejen otázka výpočetních prostředků, ale také vhodných metod pro trénování hlubokých sítí.
- Konvergence algoritmů SGD a GD na straně 16 nejsou přesné. Speciálně GD garantuje konvergenci k lokálnímu optimu pouze pro vhodnou posloupnost learning rate (což není zmíněno). Naopak SGD, kde autor tvrdí
  > „despite the small inaccuracies, it has been empirically shown that in practice, SGD works very well“

  garanguje (s vhodnou posloupností learning rate) konvergenci i teoreticky (téměř jistě – *almost surely*).
- Na straně 16 autor zmiňuje, že velikost batche se pohybuje běžně v jednotkách až stovkách – nicméně dnes se vcelku běžně vyskytují batche o stovkách tisíc či dokonce jednotkách miliónů prvků.
- Tvrzení ze strany 17
  > „Capacity of a model depends on the programmer. For simple models, this approach works reasonably well. However, it may not be feasible for more complex models. Therefore, there are approaches that try to avoid the inevitable errors caused by human subjective judgement. One such approach are neural networks.“

  je poněkud zavádějící – neuronové sítě stále vyžadují vhodné určení kapacity. Zároveň není nevhodné určení kapacity „inevitable error caused by human subjective judgement“, protože se typicky hodnotí objektivně měřením generalizační chyby modelu.
- Na straně 19 se používá $f^{(1)}$ k označení dvou různých funkcí (generujících jednou $x^{(1)}_0$ a podruhé $x^{(1)}_1$).
- Na straně 27 autor uvádí, že
  > „[one-hot encoding] has several limitations that make it inappropriate for more complex tasks“

  s čímž nesouhlasím – one-hot encoding je zcela obecná reprezentace kategorických hodnot a ostatní používané jsou na ní založené (embeddingy jsou jen one-hot

reprezentace krát matice).

- Na straně 28 autor zmiňuje *categorical crossentropy* loss bez dalšího vysvětlení – bylo by dobré buď pojem vysvětlit, odkázat nebo vůbec neuvádět (například vzhledem k tomu, jak podrobně byla vysvětlena MSE).
- Při popisu konstrukce XFG by asi bylo vhodné zmínit, že PTX je skoro v SSA formě (dokud jsem to nenašel ve zdrojích, nebylo mi jasné, jak může dávat smysl mít instrukce jako hrany XFG grafu; LLVM IR v této formě je, proto asi autoři NCC navrhli XFG tak, jak ho navrhli).
- Nejsem si jistý přínosem obrázku 3.4 – z mého pohledu v podstatě nic neříká, a tvrzení

  „we can at least state that, to some degree, the training process has the desired effect of mapping semantically related instructions to similar embedding vectors“

  mi přijde poněkud odvážné.

Příloha práce

Příloha práce obsahuje implementaci všech algoritmů a zároveň data k trénování obou úloh – první z nich jsem vyzkoušel natrénovat a bez problémů se to podařilo. Překvapivé bylo, že trénování na GPU bylo pomalejší než na CPU – jeden z možných důvodů může být to, že se všechny kernely doplňují na délku nejdelšího programu v datasetu (což je pro první úlohu 1025, zatímco průměrný program má jen 25 instrukcí).

Práci doporučuji **k obhajobě.**

Práci nenavrhuji **na zvláštní ocenění.**

Pokud práci navrhujete na zvláštní ocenění (cena děkana apod.), prosím uveďte zde stručné zdůvodnění (vzniklé publikace, významnost tématu, inovativnost práce apod.).

Datum 25. 6. 2020 **Podpis**