AKCELERACE EVOLUCE PRAVIDEL CELULÁRNÍCH AUTOMATŮ NA GPU

Luděk Žaloudek
Výpočetní technika a informatika, 2. ročník, prezenční studium
Školitel: Lukáš Sekanina

Fakulta informačních technologií, Vysoké učení technické v Brně
Božetěchova 2, 61266 Brno, Česká Republika

izaloude@fit.vutbr.cz

Abstract. Tento příspěvek se zabývá zejména způsoby akcelerace evolučního návrhu pravidel celulárních automatů pomocí GPU nVidia, který je předpokladem pro návrh pokročilých celulárních výpočetních systémů. Dále je naznačen další směr výzkumu s přihlédnutím k disertační práci.

Keywords. cellular automata, evolutionary design of rules, GPU, CUDA.

1 Úvod

V posledních několika letech se změnil trend ve způsobu zvyšování výkonu výpočetních systémů. Udržování růstu výkonu výpočetních systémů metodou zvyšování taktovací frekvence procesorů přestává být prakticky proveditelné z důvodů vysokého ztrátového výkonu. Dalším problémem je blížící se hranice, za kterou už nebude možné zmenšovat tranzistory za použití běžných technik [1]. Protože tyto problémy stále ještě nedokážeme vyřešit, mnoho výzkumů v oblasti zvyšování výpočetního výkonu se orientuje na paralelní výpočty. Mezi dva hlavní přístupy v masivních paralelních výpočtech můžeme považovat systémy s distribuovanou pamětí (DM), které jsou propojeny komunikační sítí (např. BOINC, SGE, MPI), a paralelizaci na úrovni výpočetních jader se sdílenou pamětí (SM), kde výpočty probíhají lokálně [4].

V současné době dochází díky pokroku v miniaturizaci k trendu, že namísto jednoho složitého procesoru již obsahují některé dnešní výpočetní systémy stále větší počty stále jednodušších jader. Příkladem mohou být procesory IBM Cell či shluky procesorů na grafických akcelerátorech nVIDIA (CUDA).

Zatímco zmíněné systémy s více jádry dnes obsahují obvykle desítky výpočetních jader, v budoucnu to budou stovky či tisíce [6]. Největšími problémy těchto tzv. masivně paralelních výpočetních systémů (DM i SM) jsou nutnost centrálního řízení a pomalá konfigurace a komunikace. Dalším nedostatkem může být nedostatečná odolnost proti poruchám. Z těchto důvodů se objevují nejrůznější výpočetní modely inspirované biologií, které by mohly znamenat řešení zmíněných problémů.

Jedním z často zmiňovaných možných modelů výpočtu pro budoucí platformy jsou celulární automaty (CA). CA je masivně paralelní výpočetní systém založený na lokální interakci jednoduchých buněk, které mohou nabývat různých stavů, periodicky se měnících na základě stavů svých sousedů. Stručný úvod do CA poskytuje 2. kapitola. Problematikou CA bych se chtěl zabývat ve své disertační práci na téma Sebeopravující se masivně paralelní výpočetní architektury.

Cílem disertační práce by mělo být prokázat, že **aplikačně specializované masivně paralelní výpočetní systémy založené na celulárních architekturách a pokročilých metodách sebereplikace**

(či samokonfigurace) mohou v některých aplikacích dosahovat lepšího poměru výkon/cena a větší odolnosti vůči chybám, než současné systémy založené na centrálním řízení.

Návrh pokročilých celulárních systémů je však obtížný vzhledem s emergentním vlastnostem CA a proto je vhodné využít pokročilých návrhových technik, mezi něž patří např. evoluční algoritmy (EA), které jsou stručně popsány v kapitole 3.

Mezi problémy EA se řadí i velká výpočetní náročnost, je tedy vhodné algoritmus nějakým způsobem urychlit, například paralelizací výpočtu. Jako levné a výkonné paralelní výpočetní platformy se ukazují komerčně dostupné grafické akcelerátory s multiprocesorovými GPU (zejména nVIDIA), jimž se věnuje kapitola 4. V 5. kapitole je pak nastíněna možnost, jak využít GPU při návrhu pravidel CA s využitím GPU, čímž se zabývá i můj současný výzkum.

Zbytek příspěvku pak naznačuje plánovaný postup prací na disertaci a stručný přehled dalších souvisejících činností.

2 Celulární automaty

Jak už bylo zmíněno v úvodu, CA je masivně paralelní výpočetní systém, který se skládá z velkého množství jednoduchých buněk, které mohou nabývat předem definovaných stavů. Můžeme se setkat s jednorozměrnými, dvourozměrnými i třírozměrnými automaty. Nejčastěji jde o 2D ve tvaru čtvercové mřížky.

CA můžeme rozdělit na synchronní a asynchronní. To závisí na způsobu aktualizace stavů buněk. V synchronním CA dochází periodicky u každé buňky ke kontrole stavu okolních buněk. Těmto buňkám se říká okolí a podle jejich tvaru a typu automatu může být prakticky libovolně velké, obvykle jde o čtvercové buňky s 5- nebo 9-okolím (středová buňka se rovněž počítá).

Při každém synchronizačním taktu se zjistí stav okolí každé buňky a zjištěná kombinace se vyhledá v seznamu pravidel. Pravidla jsou zapsána ve tvaru NWCES => C (tedy sever, západ, střed, východ, jih => střed – nadále pracujeme s 2D automatem a 5-okolím). Pokud je pravidlo nalezeno, nový stav na pravé straně pravidla se zapíše do kopie stavů celého CA a po zjištění změn u všech ostatních buněk se stav z této kopie aktualizuje. Pokud pro zjištěný stav okolí žádné pravidlo neexistuje, ponechá se buňka v dosavadním stavu.

Použijeme-li pro všechny buňky stejnou sadu pravidel, říkáme takovému CA uniformní. Má-li však každá buňka svoji vlastní sadu, která se může lišit od ostatních buněk, získáme neuniformní CA. Neuniformní CA je důležitým krokem pro výpočetní univerzalitu, což je vlastnost, kterou bychom chtěli mít u každého výpočetního systému. Byly vytvořeny univerzální uniformní CA s mnoha stavy a složitým sousedstvím (přehled v [3]), nicméně např. u případu 5-okolí a dvou stavů bylo dokázáno, že univerzální CA sestavit nelze [8].

Neuniformita nám umožňuje to, abychom některé buňky použili jako vodiče a další buňky jako logická hradla. Stačí nám vytvořit hradla XOR kvůli křížení vodičů a hradla NAND kvůli kompletní množině logických funkcí, která nám umožní implementovat jakýkoli logický výpočet, čímž získáváme výpočetní univerzalitu. Podrobnější vysvětlení a důkaz v [9].

Pomineme-li problém počáteční konfigurace (tj. nastavení stavů či distribuce vhodných pravidel v neuniformním automatu), můžeme si všimnout toho, že CA funguje zcela autonomně a jen na základě lokálních interakcí a synchronizace. To je důležitá vlastnost právě kvůli urychlení výpočtu. Není třeba složitých algoritmů na zasílání zpráv mezi buňkami/procesory.

Další důležitou vlastností je emergence. To je vlastnost, která je založena právě na lokálních interakcích mezi buňkami CA. (Pozn.: Emergence však není výsadní vlastností CA.) Z jedné buňky a sady pravidel těžko určíme, jak bude vypadat výsledné chování CA s mnoha buňkami. Vhodným návrhem pravidel a také vhodným nastavením počátečního stavu automatu jsme schopni získat velice komplexní chování. Příkladem může být i experiment s jednoduchým automatem zvaný hra Life či princip sebereplikace, kdy v průběhu funkce CA dochází ke kopírování celých struktur vytvořených ze stavů buněk CA.

Některé postupy aplikované na CA můžeme použít i na složitější zařízení, protože buňky CA mohou být brány jako abstrakce, např. jednoduchých procesorů. Zároveň existují i některé složitější modely založené na CA, vhodným příkladem je systém Cell Matrix, který byl již i fyzicky implementován [2]. Rovněž není nutné představovat si implementaci CA pouze ve známých křemíkových technologiích. Široké možnosti se díky jednoduchosti modelu CA nabízejí i na poli nanotechnologií (např. membránové výpočty, DNA mřížky apod.).

3 Evoluční algoritmy

Evoluční algoritmy jsou stochastické metody prohledávání stavového prostoru, které se požívají jak pro návrh, tak pro optimalizaci rozličných problémů nejen v oblasti HW a SW. EA jsou inspirovány teorií biologické evoluce, přičemž vyvíjení jedinci jsou reprezentacemi možných řešení daného problému.

Pro úspěšnou implementaci EA je nutné zvolit vhodnou reprezentaci řešení, která jsou pak zakódována jako genotypy, dále je potřeba definovat genetické operátory (křížení a mutace) a rovněž způsob vyhodnocení kvality jedince, tzv. fitness funkci. Podle toho, jakým způsobem se tyto hlavní činnosti provádí, můžeme uplatnit tzv. evoluční tlak a působit tak na konvergenci populace směrem k optimálnějším výsledkům. Zároveň tak můžeme rozlišit hlavní typy EA. Tento příspěvek se nadále omezí na genetické algoritmy (GA).

3.1 Genetické algoritmy

Genetický algoritmus je typickým představitelem EA. Řešený problém je obvykle zakódován do tzv. chromozomu v podobě řetězce celých nebo binárních čísel. Populace kandidátních řešení se podle složitosti problému může skládat z desítek až stovek jedinců, přičemž při evolučním návrhu je počáteční populace obvykle náhodně inicializována. Dalším krokem je vyhodnocení populace fitness funkcí, aby bylo možné podle kvality potenciálních řešení vybrat ta nejvhodnější pro aplikaci genetických operátorů křížení a mutace. Aplikací genetických operátorů je ze stávající generace vytvořena nová, které může dle míry tzv. elitismu obsahovat jednoho či více nejlepších jedinců ze všech dosavadních generací. GA je ukončen, pokud je dosaženo ukončovací kritérium, přičemž jím obvykle bývá určitá hodnota fitness nebo počet generací.

GA jsou nejuniverzálnější EA a používají se pro řešení řady problémů, zejména těch, které se dají dobře diskretizovat. Mezi jejich hlavní problémy patří škálovatelnost a uváznutí v lokálním minimu.

4 Paralelizace výpočtů na GPU

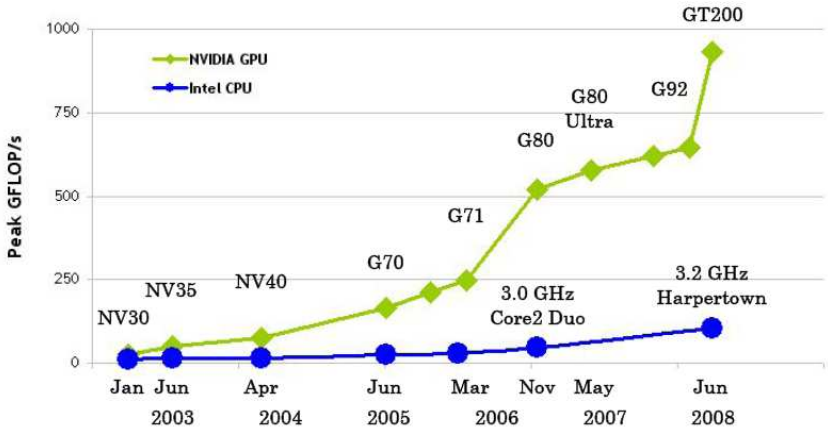
Než přejdeme k možnostem evoluce pravidel CA na GPU, nejprve se stručně seznámíme s touto platformou a jejím rozhraním.

Za poslední rok se v oblasti paralelních výpočtů dostaly do popředí zájmu aplikace využívající výpočetního výkonu moderních GPU. Zatímco výkon tradičních CPU začíná zaostávat za očekáváními danými Moorovým zákonem, nárůst výkonu několika posledních generací GPU tato očekávání udržuje při životě. Ilustrací současného stavu může být graf na obrázku 2 [7], který porovnává výkon současných CPU Intel s GPU nVIDIA. Tento článek se bude na GPU nVidia nadále omezovat, nicméně podobné výsledky slibuje i konkurenční značka ATI/AMD.

4.1 nVIDIA a CUDA

nVidia CUDA je rozhraní sloužící k paralalnímu programování aplikací pro GPU nVIDA. Před tím, než bylo zavedeno, paralelní aplikace pro výkonná GPU se tvořily pomocí nejrůznějších „triků“

s grafickými knihovnami OpenGL a DirectX. CUDA rozšiřuje programovací jazyk C (plánují se další jako např. C++ [7]) a umožňuje přímý přístup k multiprocesorům a paměti na GPU.



Obrázek 2: Srovnání výkonnosti mezi CPU Intel a GPU nVIDIA [7]

GPU obsahuje podle typu 2 až 30 (GeForce 8100m až GeForce GTX 280) tokových multiprocesorů, z nichž každý disponuje 8 jednoduššími skalárními výpočetními jádry, sdílenou pamětí na čipu, vícevláknovou instrukční jednotkou a dalšími dvěmi speciálními výpočetními jednotkami. Výpočetní jádra jsou optimalizována pro operace s plovoucí čárkou a jsou schopna obstarávat vícevláknové aplikace s nulovou režií plánování.

Výpočet libovolného algoritmu na GPU s použitím CUDA je třeba rozdělit na sériovou (host) a paralelní (device) část. Paralelní část se pak podle nastavení v programu provede v udaném počtu vláken, přičemž vlákna je ještě možné hierarchicky přidělit k tzv. výpočetním blokům, které mají k dispozici vlastní sdílenou paměť a každý je prováděn na samostatném multiprocesoru. Sdílená paměť, která je k dispozici, je velice rychlá (uvádí se až 100x rychlejší než globální paměť [7]), avšak její využití s sebou nese problémy: Výpočet zpomaluje nutnost nahrání dat z globální paměti do sdílené a rovněž není možné synchronizovat vlákna mezi bloky.

5 Evoluce pravidel CA na GPU

Jak již bylo uvedeno v úvodu, EA jsou výpočetně náročné. Dalším problémem je skutečnost, že výkonné výpočetní systémy (jako např. servery zapojené do Sun Grid Engine) schopné paralelního provádění EA jsou obvykle velice drahé a v univerzitním prostředí často vytížené. S příchodem výkonných vícejádrových GPU se tento problém prakticky omezuje na investici v ceně jednoho výkonnějšího osobního počítače. To je i jednou z motivací výzkumu směřovaného tímto směrem.

5.1 Možné přístupy

Chceme-li paralelizovat EA, je třeba si uvědomit, která jeho část je výpočetně nejnáročnější. V případě evolučního návrhu pravidel CA se bezpochyby jedná o vyhodnocení vyvinutých pravidel, tedy o simulaci celulárního automatu. Máme-li vyhodnotit výkonnost simulace CA, musíme počítat s tím, že v případě evolučního návrhu budeme pracovat se automatem obsahujícím až desítky či stovky tisíc buněk, přičemž je třeba vyzkoušet množství počátečních konfigurací, např. při řešení tradičních benchmarkových problémů majority a synchronizace. V takovém případě se nám nabízí několik hlavních možností, paralelizace algoritmu, samozřejmě s přihlédnutím k cílové architektuře.

První možností je paralelizace na úrovni buněk CA, která s sebou však nese několik problémů. Při každém kroku CA je třeba synchronizovat vlákna, aby nedošlo k porušení posloupnosti kroků CA.

CUDA však zatím neumožňuje synchronizaci vláken mezi bloky, takže paralelizace na úrovni buněk CA je značně omezena na jeden multiprocesor s osmi jádry. Dalším problémem je nejednoznačné pořadí provádění vláken v rámci bloku v jedné iteraci kroku CA (jeden blok může obsahovat více než 8 vláken), takže není možné nastavit přístupy do paměti tak, aby se vzájemně nepřekrývaly, když dochází k zjišťování stavu okolí sousedících buněk. Není tak možné provést výpočet s prokládaným přístupem do paměti, při kterém nedochází k současnému čtení stavu jedné buňky z více vláken.

Druhou možností je paralelizace algoritmu na úrovni různých počátečních konfigurací. Máme-li např. 1D CA s 64 buňkami, existuje 2^{64} možných počátečních kombinací. V praxi pak ověříme pouze několik tisíc náhodných konfigurací. Pro paralelizaci je pak možné úlohu rozdělit buď na několik tisíc vláken nebo na několik tisíc bloků a zkombinovat přístup s první možností. Pokud však chceme vyvíjet CA, kde testujeme pouze malé omezené množství pevně daných počátečních konfigurací, ztrácí tato metoda svůj smysl.

Třetí možností je paralelizace na úrovni jedinců v populaci EA. Tato možnost je středně paměťově náročná a její využitelnost závisí na použitém EA. Pokud by byl vyvíjen CA např. pomocí evoluční strategie, zrychlení by dosáhlo maximálně počtu jedinců v populaci, která je u ES malá. Tato metoda se tak jeví jako vhodnější pro GA.

Rovněž je možné paralelizovat další dílčí úkony EA jako inicializaci, výběr jedinců pro rekombinaci, aplikaci genetických operátorů či případnou kontrolu syntaktické správnosti nově vytvořených jedinců.

5.2 Dosažené výsledky

S akcelerací evolučního návrhu pravidel CA byly provedeny experimenty na modelovém problému majority na 1D CA se šířkou 64 buněk a 7-okolím. K měření hodnot byly použity dva různé PC: Notebook s CPU Intel Core 2 Duo 1,83 GHz a GeForce 8600M GS s 4 multiprocesory a pracovní stanice s Intel Core 2 Duo 3,33 GHz a GeForce FX 280 s 30 multiprocesory. Zrychlení bylo měřeno v porovnání se sériovým výpočtem na obou strojích při překladu s MS Visual C++ 9.0. Výsledky jsou průměrem několika běhů a měřena byla doba výpočtu celého programu, ne jen paralelní části.

Samotná simulace s paralelizací na úrovni buněk CA (bez EA) dosáhla zrychlení řádů stovek na notebooku i na pracovní stanici (621x a 489x). Větší zrychlení GPU na notebooku je dáno výkonnějším CPU na pracovní stanici. Při převedení paralelizace na úrovni buněk CA se však projevily přenosy mezi VRAM a RAM a řešení žádného zrychlení nedosáhlo.

GA s populací 100 jedinců a 10 generacemi s paralelizací na úrovni počátečních konfigurací CA, kterých bylo 240, dosáhl zrychlení 7,12x na notebooku a 12,16x na pracovní stanici. Při zvýšení počtu trénovacích vektorů na 24000 však došlo na pracovní stanici k zrychlení až 314,97x, což naznačuje dobrou škálovatelnost.

Stejný GA nedosáhl tak dobrých výsledků s paralelizací na úrovni jedinců v populaci. Tento způsob se ukázal jako nevhodný pro cílovou architekturu, protože je nevýhodné, aby paralelní jádro počítalo úkoly déle, než je několik sekund. Nejvyšší zrychlení v tomto případě bylo 192,88x.

5.3 Shrnutí

Výsledky ukázaly, že paralelizace evoluce pravidel CA na GPU má smysl. Při rozsáhlejších problémech se ukazuje, že s CUDA je nejvýhodnější velké množství jednoduchých paralelních výpočtů vysoce převyšujících počet výpočetních jader, na rozdíl od menšího množství delších výpočtů. To se projevilo ve výsledcích experimentu s paralelizací výpočtu trénovacích vektorů.

Simulace CA na úrovni buněk se neukázala jako lepší, kvůli vysoké paměťové režie při přesunech velkého množství dat do globální paměti GPU, mnohonásobně delšími přenosy s globální do sdílené paměti a nemožností prokládat paměťové přístupy či využít více multiprocesorů.

V budoucnu hodlám vylepšit výsledky vyladěním přístupů k sdílené paměti a využít je jako prostředek pro akceleraci výzkumu nových modelů CA.

6 Další činnost na disertaci

Po dohodě se školitelem bylo rozhodnuto rozdělit práci na disertaci na dílčí části, přičemž akcelerace simulace a evolučního návrhu je jednou takovou částí, jejíž výsledky budou poté využity pro urychlení výzkumu vylepšení modelu CA tak, aby bylo možné prokázat hypotézu tezí disertační práce (viz úvod tohoto příspěvku). Mezi hlavní uvažovanou možnost přístupu k tomuto problému patří zavedení omezené globální informace do CA, aby tak mohly být lépe kontrolovány fáze výpočtu. Objevila se už práce, která v omezeném měřítku prokazuje využitelnost této myšlenky, přičemž bylo dosaženo urychlení sebereplikace smyčky v CA [5]. Předmětem zájmu disertace by bylo dosáhnout podobných výsledků na smyčce s reálnou funkcí (např. filtrování obrazu).

Poděkování

Práce na tomto příspěvku a souvisejícím výzkumu byly provedeny s podporou grantu **Návrh a obvodová realizace zařízení pro automatické generování patentovatelných invencí**, GAČR, GA102/07/0850, 2007-2009; a výzkumného záměru **Výzkum informačních technologií z hlediska bezpečnosti**, MSM0021630528, 2007-2013.

Literatura

[1] Beckett, P., Jennings, A.: Towards Nanocomputer Architecture, Proceedings of the Seventh Asia-Pacific Conference on Computer Systems Architecture (Melbourne, Victoria, Australia). Conferences in Research and Practice in Information Technology Series, vol. 19. Australian Computer Society, Darlinghurst, Australia, 2002, p. 141-150

[2] Cell Matrix Corporation home page, http://www.cellmatrix.com, červen 2008

[3] Culik II, K., Hurd, L. P., and Yu, S.: Computation theoretic aspects of cellular automata, Physica D, vol. 45, p. 357-378, 1990

[4] Dvořák, V.: Architektura a programování paralelních systémů, Vutium, Brno, 2004

[5] Komenda, T.: Sebereplikace v celulárních systémech, diplomová práce, FIT VUT v Brně, Brno, 2009.

[6] Margolus, N.: Crystalline Computation. In The Feynman Lecture Series on Computation, Volume 2. A. Hey (ed), Addison-Wesley, 1998

[7] nVIDA CUDA Programming Guide, Version 2.2, 4.2.2009, http://www.nvidia.com/object/cuda_develop.html

[8] Sipper, M.: Evolution of Parallel Cellular Machines - The Cellular Programming Approach, Lecture notes in Computer Science, vol. 1194, Springer, 1997

[9] Tempesti, G.: A New Self-Reproducing Cellular Automaton Capable of Construction and Computation, Advances in Artificial Life, Lecture notes in Computer Science, vol. 929, Springer, 1995

Publikace Autora

[10] Žaloudek, L.: Coevolution of Sorting Networks and Training Vectors, Proceedings of the 13th Conference Student EEICT 2007, vol. 1, p. 184-186, FEKT a FIT VUT v Brně, Brno, 2007

[11] Žaloudek Luděk, Sekanina Lukáš: Transistor-level Evolution of Digital Circuits Using a Special Circuit Simulator, Lecture Notes in Computer Science, roč. 2008, č. 5216, DE, s. 320-331, ISSN 0302-9743

[12] Žaloudek Luděk: Sebereplikace ve výpočetních systémech, Počítačové architektury a diagnostika 2008, Liberec, CZ, TUL, 2008, s. 131-136, ISBN 978-80-7372-378-1