VÝPOČETNÍ SYSTÉMY ZALOŽENÉ NA CELULÁRNÍCH AUTOMATECH

Luděk Žaloudek
Výpočetní technika a informatika (4-letý), 3.ročník, prezenční studium
Školitel: Lukáš Sekanina

Fakulta informačních technologií, Vysoké učení technické v Brně
Božetěchova 1, 612 66, Brno

izaloude@fit.vutbr.cz

Abstrakt. Článek čtenáře stručně seznamuje s problematikou celulárních automatů a výpočetní složitosti jejich evolučního návrhu. Jsou identifikovány dvě hlavní skupiny úloh pro celulární automaty včetně příkladů benchmarků. Dále je uveden způsob akcelerace vědeckých výpočtů pomocí grafických procesorů a jsou shrnuty výsledky, kterých bylo dosaženo při akceleraci evoluce pravidel celulárních automatů na těchto procesorech. V závěru je naznačen směr, kterým by se měla ubírat disertační práce zabývající se právě návrhem masivně paralelních výpočetních systémů založených na celulárních automatech s důrazem na využití principu sebereplikace.

Klíčová slova. Celulární automaty, evoluční návrh, GPU, CUDA, sebereplikace.

1 Úvod

V posledních několika letech pokračuje trend ve způsobu zvyšování výkonu výpočetních systémů paralelizací. Udržování růstu výkonu výpočetních systémů metodou zvyšování taktovací frekvence procesorů přestává být prakticky proveditelné z důvodů vysokého ztrátového výkonu či vzhledem k přibližující se hranici, 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 [5].

Inspirací pro budoucí masivně paralelní výpočetní systémy může být nastupující fenomén paralelních výpočtů na grafických procesorech (GPU), které díky rozvoji počítačové grafiky postupují odlišnou cestou než běžné procesory. Právě GPU v současné době udržují křivku růstu výkonu v trendu, kterým v duchu Moorova zákona postupoval výkon klasických procesorů ještě před několika lety. Na rozdíl od běžných CPU obsahujících dnes maximálně 6 jader, GPU snadno překonávají hranici 1 TFLOPu [15].

Zatímco zmíněné systémy s více jádry dnes obsahují obvykle až stovky výpočetních jader, v budoucnu to budou tisíce či desetitisíce [12]. 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) [4]. 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. Vzhledem k tomu, že příspěvek je psán z pozice studenta předposledního ročníku, je obsahem kapitol 2 až 4 hrubě naznačen přehled témat, která by měla být obsažena v disertační práci.

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 emergentnímu charakteru CA a proto je vhodné využít pokročilých návrhových technik, mezi něž patří např. evoluční algoritmy (EA), které jsou stručně popsány v na konci kapitoly 2. Mezi problémy EA se však řadí i velká výpočetní náročnost, a proto je 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 3, ve které jsou zmíněny i výsledky, kterých bylo dosaženo při snaze o urychlení evoluce pravidel CA na GPU.

Kapitola 4 naznačuje směr vývoje práce na disertaci a stručně diskutuje možnosti, které tato oblast skýtá, zejména pak výpočty za pomoci sebereplikujících se smyček.

2 Celulární Automaty

Celulární automat je zpravidla n-rozměrná mřížka identických buněk, z nichž každá funguje jako konečný stavový automat [4]. V synchronní verzi je stav buněk periodicky obnovován na základě lokální přechodové funkce. Pokud všechny buňky používají stejnou přechodovou funkci, mluvíme o CA jako o uniformním, v opačném případě pak o neuniformním. Následující stav každé buňky je funkcí jejího současného stavu a stavu jejích sousedních buněk, označovaných jako sousedství.

V 1D CA je sousedství definováno pomocí tzv. poloměru sousedství *r*, který určuje, kolik buněk z každé strany centrální buňky je do sousedství zahrnuto. V 2D CA může být sousedství definováno podobně, rovněž však existují sousedství s nepravidelným tvarem. Nejčastěji se však setkáváme se sousedstvím Moorovým, kde je v případě *r=1* zahrnuto všech 8 okolních buněk jak v rovných tak v šikmých směrech, respektive s von Neumannovým sousedstvím, které zahrnuje jen 4 buňky v rovných směrech.

Teoreticky je celulární prostor nekonečný, prakticky však musí být omezen. K tomuto účelu se definují tzv. okrajové podmínky. Nejčastěji se využívají buď cyklické resp. konstantní. V takovém případě se na okrajích berou sousední buňky z opačného okraje CA, resp. se jejich stav považuje za konstantní buď podle předem definované hodnoty nebo podle aktuálního stavu poslední okrajové buňky.

Stav všech buněk na počátku běhu CA se nazývá počáteční konfigurace.

Přechodovou funkci CA, jejíž plný formální popis je možné najít např. v [16], můžeme např. pro případ 2D CA s Moorovým okolím zjednodušeně znázornit jako množinu pravidel tvaru C, N, NE, E, SE, S, SW, W, NW → C', kde C je stav středové buňky, C' je její nový stav a ostatní symboly reprezentují stavy sousedství podle anglického popisu světových stran. Pokud pro danou kombinaci stavů množina pravidlo neobsahuje, znamená to, že ve středové buňce nedojde ke změně.

2.1 Evoluční algoritmy a návrh CA

Při snaze o využití CA se snažíme: (a) navrhnout taková pravidla, která nám umožní provádět požadovaný výpočet nebo (b) chceme předvídat chování automatu s předem známými pravidly

a počáteční konfigurací. Tento příspěvek se zabývá pouze možností (a), která v sobě skrývá zásadní nedostatek: Vzhledem k složitosti CA se ruční návrh pravidel CA stává značně problematickým a je třeba přejít k pokročilým návrhovým metodám, jako např. k evolučnímu návrhu [17].

Evoluční algoritmy [2] jsou stochastické prohledávací metody inspirované biologií. EA se podobají přírodní evoluci v tom, že hledání řešení daného problému probíhá na populaci jedinců, na kterou jsou v každé generaci běhu algoritmu aplikovány biologické operátory jako křížení a mutace, pomocí kterých dochází k vytvoření nové populace. Jedinci z předchozí populace jsou k aplikaci operátorů vybíráni na základě jejich kvality, kterou označujeme jako fitness a která znamená, jak dobře daný jedinec splňuje cíl evoluce.

Nejběžnější forma EA označovaná jako genetický algoritmus (GA) vypadá následovně:

1. Vytvoř náhodnou počáteční populaci kandidátních jedinců.
2. Ohodnoť populaci – každému jednici přiřaď fitness hodnotu.
3. Na základě fitness hodnoty vyber nejlepší jedince.
4. Na vybrané jedince aplikuj genetické operátory (křížení, mutace) a vytvoř novou populaci.
5. Pokud jsou splněna ukončovací kritéria (fitness, počet generací), ukonči se, jinak pokračuj bodem 2.

V našem případě je jedincem celulární automat, resp. jeho přechodová funkce a fitness se vyhodnocuje tak, že na základě dané počáteční konfigurace proběhne určitý počet kroků CA a na poté je vyhodnocen konečný stav CA vzhledem k zadané úloze.

V minulosti bylo již mnohokrát ukázáno, že EA mohou dosáhnout inovativních výsledků v mnoha odvětvích. Bohužel, největším problémem EA je škálovatelnost a to jak v případě reprezentace problému, tak v případě výpočtu fitness, jejíž výpočet je zpravidla nejsložitější částí EA [19]. Jak je ukázáno v kapitole 3, tento problém se dá částečně řešit paralelizací EA.

2.2 Benchmarky pro CA

Úlohy pro výpočet v CA mohou být dvou základních typů: (i) Úlohy, kde vstupem problému mohou být jakékoliv počáteční konfigurace omezené jen množstvím stavů a velikostí automatu, přičemž hledáme vhodnou kombinaci stavů v konečné konfiguraci; (ii) Úlohy, kde známe jedinou nebo omezený počet počátečních konfigurací a hledáme určité chování CA v celém průběhu jeho výpočtu.

Příkladem úloh typu (i) mohou být např. problém majority či problém synchronizace. Majorita je problém, kdy se na základě počáteční kombinace snažíme najít převládající stav, který je CA indikován tak, že se celý celulární prostor po určitém počtu kroků zaplní převažujícím stavem. U větších automatů je prakticky nemožné otestovat všechny možné vstupní konfigurace a proto se problém často řeší náhodným vygenerováním omezeného počtu počátečních konfigurací, kterým říkáme trénovací vektory. Fitness je pak přímo úměrná úspěšnosti na množině trénovacích vektorů.

Do druhé skupiny úloh (ii) patří např. simulace čítače. CA zadáme počáteční konfiguraci odpovídající počáteční hodnotě čítače zakódované v číselné soustavě o základu odpovídajícímu počtu stavů CA a necháme automat běžet tak dlouho, jak má být dlouhá jeho sekvence čítání. Za každý krok, kdy konfigurace CA odpovídá číslu v požadované sekvenci čítání, zvýšíme fitness daného jedince.

3 Akcelerace běhu CA na GPU

Jak vyplývá z popisu algoritmu v podkapitole 2.1, EA jsou časově velice náročné. Populace mohou mít stovky jedinců, v případě (i) typu úloh z podkapitoly 2.2 je třeba otestovat velké množství trénovacích vektorů, které často dosahuje řádově tisíců. I samotné CA mohou být značně velké, čímž se zvyšuje výpočetní náročnost. Z těchto důvodů je výhodné výpočet CA paralelizovat, aby se snížila celková doba návrhu. Ukazuje se, že využití grafických akcelerátorů k paralelizaci výpočtů je v současnosti levnou alternativou k využívání drahých, či v případě jejich dostupnosti často jinými úlohami přetížených výpočetních serverů (např. Sun Grid Engine).

3.1 Užitečné výpočty na GPU

Moderní grafické akcelerátory, přesněji čipy uvnitř, nabízejí sadu velice výkonných výpočetních jednotek, které se dají využít k obecným výpočtům. Z dostupné literatury [14] je patrné, že GPU svým teoretickým maximálním výkonem vysoce převyšují běžné univerzální procesory, včetně nejnovějších 6-jádrových. U nejnovějších čipů Fermi od NVIDIA přesahuje výpočetní výkon 1 TFLOP [15].
Výkon grafických čipů již v minulosti přilákal pozornost jak při akceleraci evolučních výpočtů [7][8], tak při akceleraci celulárních automatů [6]. Vzhledem k tomu, že GPU jsou primárně navrženy pro výpočty s grafikou, bylo až do nedávné doby pro jejich využití k obecnějším výpočtům využíváno nejrůznějších „triků“ s rozhraními sloužícími k programování grafiky, jako jsou např. Open GL či DirectX, což je patrné i z výše zmíněných publikací.

Teprve před několika lety začali výrobci podporovat přímé programování GPU pomocí rozšíření běžných programovacích jazyků. Mezi nejznámější patří v současné době CUDA od NVIDIA a ATI Stream od AMD. Mezi nevýhodu zmíněných rozhraní patří jejich závislost na HW konkrétního výrobce. Proto se objevila i obecná rozhraní pro programování GPU či dokonce jiných HW platforem. Nejznámější je OpenCL. Urychlení zmiňované v tomto příspěvku bylo dosaženo s rozhraním CUDA.

3.2 Akcelerace evolučního návrhu pravidel CA

Vzhledem k neexistenci vhodného paralelního simulátoru CA bylo rozhodnuto o implementaci vlastního, který bude využívat rozhraní CUDA. Protože bylo předem počítáno s evolučním návrhem pravidel CA, byly nakonec navrženy 3 možnosti, jak implementovat akcelerovanou evoluci pravidel CA [22]: (a) Paralelizace na úrovni jednotlivých buněk CA, (b) paralelizace na úrovni trénovacích vektorů a (c) paralelizace na úrovni celých jedinců EA.

Pro experimenty se zrychlením výpočtu evoluce s CA byl nejprve vybrán benchmark majority, spadající do kategorie (i) z podkapitoly 2.2, později byl přidán benchmark 4-bitového čítače spadající do kategorie (ii) [23]. Vše bylo důkladně testováno na rúzných grafických akcelerátorech NIVIDIA od méně výkonného GeForce 8600M GS určeného pro laptopy, přes středně drahý GeForce 9600 GT až po nejdražší a vysoce výkonný GeForce GTX 285. Zmíněné akcelerátory mají různé parametry [14], nejdůležitější z nich jsou však počty výpočetních jader. Zatímco 8600M GS má jen 4 multiprocesory (MP) po 8 jádrech (tedy celkem 32 jader), GTX 285 má k dispozici celkem 30 MP s celkem 240 jádry.

3.3 Dosažené výsledky

Během experimentování byly změřeny časy pro evoluci jak pro sériovou verzi výpočtu na jednom jádru CPU, tak pro paralelní verzi s různým počtem využitých jader na GPU. Oproti výsledkům v [21] byly experimenty značně rozšířeny a vylepšeny a pro variantu (c) a benchmark majority **bylo dosaženo nejlepšího zrychlení blížícího se hodnotě 420x** oproti sériové verzi [23]. Rovněž se ukázal očekávaný výsledek, a sice že výpočty z kategorie (ii) z podkapitoly 2.2 jsou nevhodné pro paralelizaci na GPU, která se lépe hodí pro paralelní výpočty s možností zvětšování paralelní zátěže, řídící se Gustafsonovým zákonem, spíše než pro úlohy s konstantní velikosti, které se řídí Amhdalovým zákonem [9]. Podrobnější výsledky lze najít v souvisejících publikacích, přičemž první je příspěvek na konferenci [22] a druhá je článek do časopisu rozšiřující předchozí příspěvek [23].

4 Úlohy pro masivně paralelní celulární výpočetní systémy

Pokud se podařilo alespoň částečně vyřešit problém akcelerace výpočtů CA, je nutné přejít k tématu, které by mělo být stěžejním obsahem disertační práce. Tímto tématem by mělo být využití CA jako modelu pro HW architekturu, která by měla na základě svých vlastností v určitých aplikačně specializovaných úlohách dosahovat lepších výkonů či odolnosti proti poruchám, než je tomu u konvenčních výpočetních systémů založených na centrálním řízení. Kromě nejdůležitějších vlastností

CA, jako jsou masivní paralelismus, lokálnost interakcí a jednoduchost výpočetních elementů, by mělo být využito známých principů vycházejících z emergentního charakteru CA.

Práce na disertaci je v současné době ve stadiu, kdy probíhá hledání vhodných úloh a vylepšování metod návrhu (zejména evolučních) CA, které by měly vést k vytvoření sady dostatečně složitých ukázkových problémů, které se dají efektivně řešit v CA. Inspirací mohou být již existující příklady [3] implementované na fyzické platformě založené na CA. Nejpravděpodobnější oblastí úloh se jeví zpracování obrazu (např. filtry).

4.1 Výpočty v sebereplikujících se smyčkách

Mezi jeden z nejznámějších emergentních jevů v CA patří sebereplikace. Jde o princip známý už od poloviny minulého století od matematika von Neumanna [4], který však během desetiletí zaznamenal řadu vylepšení a modifikací. Von Neumann zavedl tzv. Univerzální konstruktor, obrovsky složitý stroj založený na CA schopný konstrukce sebe sama a provádění přidruženého výpočtu. To se ukázalo jako prakticky nerealizovatelné a tak asi největší revoluci přinesl až Langton [11], který upustil od výpočetní univerzality a soustředil se jen na sebereplikaci. Vytvořil tzv. sebereplikující se smyčku (SR loop – self-replicating loop), která se dokáže v celulárním prostoru donekonečna kopírovat s použitím pouhých 8 stavů a stovek buněk oproti 29 stavům a desetitisícům buněk von Neumannovým.

Sebereplikující smyčky se dočkaly mnoha modifikací, ale asi nejdůležitější jsou ty, které dokáží provádět užitečné výpočty. Mezi ně patří zejména Tempestiho smyčka [18]. Schopnost provádět užitečné výpočty umožňuje využít sebereplikující se smyčky jako prostředek k rychlé konfiguraci celulárního systému sloužícímu pro nějakou specializovanou úlohu (obsaženou ve smyčce).

Nedávné práce [10][13] ukázaly, že sebereplikující se smyčky ještě nebyly zcela prozkoumány a skýtají řadu možností pro vylepšení ať už z pohledu rychlosti sebereplikace, tak z pohledu jejich samotné funkčnosti. Možnosti lepšího řízení funkce CA doplňuje např. i využití omezené globální informace, které bylo úspěšně použito v [10].

5 Závěr

Vylepšení metod sebereplikace s přihlédnutím k odolnosti proti poruchám a jejich využití právě k řešení některých specializovaných úloh je oblastí, kterou bych se chtěl v rámci tématu disertace zabývat. CA využívající metod sebereplikace k rychlé konfiguraci a odolnosti proti poruchám by měly být navrženy jak ručně se znalostí již zaběhnutých postupů, tak za pomoci akcelerované evoluce na GPU. Na vybraných úlohách (např. z oblasti zpracování obrazu) by pak měla být demonstrována jejich efektivita ve srovnání s jinými návrhovými a výpočetními metodami.

Poděkování

Tato práce byla částečně podpořena z grantu **Natural Computing na nekonvenčních platformách** GP103/10/1517, grantu FITu FIT-10-S-1 a z výzkumného záměru **Výzkum informačních technologií z hlediska bezpečnosti**, MSM0021630528.

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] Bentley, P. (ed.): Evolutionary Design by Computers, Morgan Kaufmann, San Francisco, 1999, Chapter 1

[3] Cell Matrix publications, Cell Matrix Corporation,
URL: < http://www.cellmatrix.com/entryway/products/pub/publications.html >, [cit. 25.6.2010]

[4] Codd, E., Cellular Automata, Academic Press, 1968

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

[6] Gobron, S., Devillard F., Heit B., Retina simulation using cellular automaton and GPU programming, Machine Vision and Applications Journal 66, Springer, 2007, pp. 331–342

[7] Harding, S.: Evolution of Image Filters on Graphics Processor Units Using Cartesian Genetic Programming, 2008 IEEE World Congress on Computational Intelligence, Hong Kong: IEEE CIS, 2008, pp. 1921–1928

[8] Harding, S., Banzhaf, W.: Fast genetic programming on GPUs, Proceedings of the 10th European Conference on Genetic Programming, LNCS 4445, Springer, 2007, pp. 90–101

[9] Henessy, J., Patterson, D.: Computer Architecture A Quantitative Approach, The Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufmann Publishers, 2003

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

[11] Langton, C.G., Self-Reproduction in Cellular Automata, Physica D: Nonlinear Phenomena 10(1-2), Elsevier, 1984, pp. 135-144

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

[13] Novák, R.: Evoluce emergentního chování v celulárních systémech, diplomová práce, Brno, FIT VUT v Brně, 2010

[14] NVIDIA CUDA Programming Guide, Version 3.0,
URL: <http://developer.nvidia.com/object/cuda_3_0_downloads.html.>, [cit. 14.6.2010]

[15] Next Generation CUDA Architecture, Code Named Fermi,
URL: <http://www.nvidia.com/object/fermi_architecture.html>, [cit. 31.1.2010]

[16] Sekanina, L.: Evolvable Components: From Theory to Hardware Implementations, Natural Computing Series, Springer-Verlag, Berlin Heidelberg, DE, 2004

[17] Sipper, M.: Evolution of Parallel Cellular Machines: The Cellular Programming Approach, Springer Verlag, Heidelberg, 1997

[18] Tempesti, G.: A New Self-Reproducing Cellular Automaton Capable of Construction and Computation, Advances in Artificial Life, Proc. of 3rd European Conference on Artificial Life, Granada, Spain: Lecture Notes in Artificial Intelligence, 929, Springer Verlag, Berlin, 1995, pp. 555–563

[19] Yao, X., Higuchi, T., Promises and Challenges of Evolvable Hardware, IEEE Transactions on Systems, Man, and Cybernetics 29(1), IEEE, 1999, pp. 87–97

Vybrané publikace autora

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

[21] Žaloudek Luděk: Akcelerace evoluce pravidel celulárních automatů na GPU, Počítačové Architektury a diagnostika 2009, Zlín, CZ, UTB, 2009, s. 173-178, ISBN 978-80-7318-847-4

[22] Žaloudek, L., Sekanina, L., Šimek, V.: GPU Accelerators for Evolvable Cellular Automata, Computation World: Future Computing, Service Computation, Adaptive, Content, Cognitive, Patterns, Athens, GR, IEEE, 2009, pp. 533-537

[23] Žaloudek, L., Sekanina, L., Šimek, V.: Accelerating Cellular Automata Evolution on Graphics Processing Units, in International Journal On Advances in Software, online, IARIA, 2010 – v tisku