Evoluční hardware v síťových aplikacích

David Grochol
2. ročník, prezenční studium
Školitel: Lukáš Sekanina

Vysoké učení technické v Brně
Božetěchova 2, 612 66, Brno, ČR
igrochol@fit.vutbr.cz

***Abstrakt*—Tento článek se zabývá využitím evolučních algoritmů v oblasti síťových aplikací. V rámci článku jsou představeny cíle disertační práce a dosavadní práce představující využití kartézského a lineárního genetického programování ve dvou případových studiích. První studie se zabývá klasifikací aplikačních protokolů v FPGA a druhá se věnuje návrhu speciální hašovací funkce pro síťové aplikace.**

***Klíčová slova*—síťová aplikace, lineární genetické programování, kartézské genetické programování, SDM**

I. Úvod

Počítačové sítě jsou v poslední letech využívány stále větším počtem zařízení a uživatelů. S tímto fenoménem roste množství dat, která musí být přenášená pomocí sítě. S rostoucím množstvím dat se musí přizpůsobovat technologie umožňující přenos těchto dat. Kromě přenosových technologií je potřeba přizpůsobovat rychlost i dalších aplikací starajících se o provoz sítě, sledování stavu síťových prvků, monitorování provozu a systémů zajišťujících bezpečnost.

U dnes používaných vysokorychlostních sítí s propustností až 100 Gb/s je stále častější hardwarová akcelerace síťových aplikací. Toto řešení s sebou nese i své problémy. Aby bylo řešení pomocí hardware robustní, byl by potřeba velmi specifický hardware, který se vyznačuje vysokou cenou. Proto je vhodnější použít obecnější ale konfigurovatelné hardwarové komponenty, které jsou řízeny pomocí software. Návrh hardwarových komponent je možný pomocí konvenčních metod, které vyžadují perfektní znalost problematiky, nebo využitím technik evolučního návrhu, které nevyžadují tak perfektní znalost problematiky, ale dovolují v některých případech dosáhnout lepších parametrů systémů.

Ve své práci se zabývám využitím evolučních technik v návrhu a optimalizaci síťových aplikací. Tyto síťové aplikace, určené pro sítě s rychlostí až 100 Gb/s, je potřeba navrhovat příp. optimalizovat zejména z pohledu zpoždění. Aby bylo možné využít evolučního algoritmů (EA), je potřeba vhodně upravit konstrukci EA s ohledem na použití v rámci síťových aplikací. V některých případech je potřeba také vylepšit efektivitu fungování EA, jak z pohledu evoluce, tak z pohledu rychlosti běhu samotného EA.

V kapitole II seznamuji čtenáře stručně s variantami genetického programování a pojmem síťová aplikace z pohledu této práce. V kapitole III popisují cíle práce a dosavadní výzkumnou činnost. Závěrečná kapitola IV shrnuje nejdůležitější body textu.

II. Evoluční algoritmy a síťové aplikace

Genetické programování (GP) [1], [2] umožňuje automaticky navrhovat programy. V práci využívám dvě varianty genetického programování a to konkrétně kartézské GP (CGP) [3], [4], které využívá pro reprezentaci problému acyklické orientované grafy. CGP používá relativně malé populace jedinců a využívá pouze mutaci pro vytváření nových jedinců. Jednou z aplikací CGP je optimalizace obvodů pro FPGA (Field Programmable Gate Array). Další variantou GP je lineární GP (LGP) [5]. V LGP jsou kandidátní programy reprezentovány posloupností instrukcí. Pomocí LGP jsou obvykle navrhovány relativně krátké programy, které je ale možné přesně doladit pro potřeby konkrétní aplikace.

A. Síťové aplikace

Síťové aplikace [6], [7] můžeme mimo jiné chápat jako komplexní systémy, které pracují v síti. Jejich úkolem je poskytovat služby uživatelům a zajišťovat bezpečnost sítí, monitorování provozu apod.

Většina síťových aplikací pracuje nad nejvyšší vrstvou TCP/IP modelu, tedy na aplikační vrstvě. Síťové aplikace (aplikační protokoly) můžeme rozdělit do dvou skupin. První skupinu tvoří uživatelské protokoly, které poskytují služby přímo uživateli (např. HTTP, SMTP, SSH, FTP) a druhou systémové protokoly, které zajišťují síťové funkce (např. SNMP, DNS).

Prostředky pro bezpečnost a monitorování provozu spolu úzce souvisí. Při monitorování sítě je kontrolován provoz na síti, a pokud je detekována anomálie, může být provoz filtrován. Filtrování síťového provozu je jedním z mechanismů posilující bezpečnost. S rostoucí rychlostí počítačových sítí (dnes až 100 Gb/s) jsou potřeba stále výkonnější aplikace/zařízení pro monitorování a bezpečnost sítí. Pro určité aplikace (zejména z pohledu bezpečnosti) je potřeba provádět analýzy v reálném čase.

1) Evoluční algoritmy v počítačových sítích: Evoluční algoritmy mohou být v počítačových sítích využity pro návrh a optimalizaci prakticky na všech úrovních, počínaje samotným návrhem topologie sítě [8]. EA byly úspěšně použity při směrování protokolů v síti na základě více QoS parametrů

[9]. Dalším příkladem je návrh speciální hašovací funkce. V rámci práce [10] byl navržen způsob, jak navrhovat speciální hašovací funkce pro hašování IP adres v hardware.

III. Cíle disertační práce a dosažené výsledky

EA v sítích již využity byly, ale zejména pro optimalizaci nebo návrh relativně jednoduchých komponent. Otázkou je, zda je evoluční přístup vhodný i pro optimalizaci a návrh složitých obvodových komponent moderních síťových zařízení, jako je například systém SDM (Software Defined Monitoring) [11].

Cíle disertační práce:

1) Seznámit se s vybranými síťovými aplikacemi.
2) Prostudovat stávající evolučními algoritmy.
3) Navrhnout a implementovat evoluční algoritmy pro vybrané síťové aplikace.
4) Vylepšit stávající síťové aplikace využitím evolučních algoritmů zaměřených na optimalizaci zpoždění.
5) Ověřit a experimentálně vyhodnotit vylepšení síťových aplikací na reálných síťových datech.

Ve své práci bych se rád zaměřil na systém softwarově řízeného monitorování (SDM). Jedná se o systém hardwarové akcelerace monitorovacích a bezpečnostních aplikací. Základní princip je postavený na softwarově řízeném předzpracování síťových dat v hardware. Systém se skládá z mnoha částí, které jsou navrženy konvenčním způsobem. Díky dělení zátěže mezi software a hardware může tento systém pracovat v sítích s propustností až 100 Gb/s. Cílem disertační práce bude ukázat, že optimalizací stávajících a návrhem nových aplikací pomocí evolučních algoritmů je možné zvyšovat efektivitu systému SDM. Dalším cílem bude vytvořit metodiku pro optimalizaci zpoždění struktur vytvářených pomocí GP, kde je možné problém řešit jako jednokriteriální, vícekriteriální. Následuje popis dvou případových studií.

A. Rychlá klasifikace aplikačních protokolů v FPGA optimalizovaná pomocí CGP

Klasifikace aplikačních protokolů je důležitá součást systému SDM. Na základě identifikovaného aplikačního protokolu mohou být nastavena pravidla pro další zpracování, případně odfiltrování dat. Důležitým požadavkem na klasifikátor je velmi nízká odezva, pro 100 Gb/s linky je nutné dosáhnout zpoždění menší než 7 ns. Proto je klasifikátor umístěn v hardwarové části systému SDM.

V rámci mé práce byl navržen klasifikátor aplikačních protokolů implementovaný v FPGA a optimalizovaný pomocí CGP, klasifikující protokoly HTTP, SMTP, SSH a později SIP na základě aplikačních dat. Pro určení přesnosti a úplnosti klasifikace byla použita reálná data zachycena na propojeních sítí CESNET a ACONET (CESACO) a CESNET a PIONIER (CESPIO), navíc byla použita i speciální sada pro protokol SIP a SSH (DATA SIP). Pro tuto práci jsou zajímavé pouze pakety protokolu TCP a UDP. Datová sada byla anotována pomocí L7 filtru [12].

Navržený klasifikátor identifikuje aplikační protokol na základě několika bajtů aplikačních dat prvního datového paketu toku. Tyto bajty budou v dalším textu nazývány "vzor". Například pro protokol HTTP a metodu GET je nejdelší vzor prvního paketu: "GET /". Nejprve byl navržen co možná nejpřesnější klasifikátor (CL-acc) obsahující co možná nejkomplexnější vzory protokolů. Další klasifikátor CL-cmp je kompromisem, kde je délka vzorů zkrácena na 4 znaky (např. "GET ") a poslední verze klasifikátoru je snahou o vytvoření co možná nejmenší implementace (CL-lat), kde jsou pouze znaky, které se vyskytují alespoň dvakrát ve vzorech na stejné pozici a každý vzor musí obsahovat alespoň 3 znaky (např. "*ET /"). Všechny vzory jsou prezentovány v [13].

V FPGA je klasifikátor připojen k sběrnici o šířce 512 bitů (součást SDM), přes kterou jsou posílána všechna data ze sítě. Každý rámec začíná hlavičkami nejnižších úrovní jako Ethernet, IPv4 nebo IPv6 a TCP nebo UDP. Důsledkem je, že aplikační data mohou začínat na různých pozicích, konkrétně s ofsetem 2 bajty z pozice 0 nebo 2 + 4k bajtů, kde $k = 1, ..., 16$.

První úroveň klasifikátoru tvoří kodéry s osmi vstupy kódující znaky vzorů aplikačních protokolů (celkem je 64 kodérů). Jelikož aplikační data mohou začínat s ofsetem 4, jsou navrženy 4 typy kodérů (c1, c2, c3, c4). Kodéry na výstupu produkují kód 2 z N. Rozdělení znaků mezi kodéry je uvedeno v [13].

Druhou úrovní klasifikátoru představují komparátory. Komparátor porovnává výstupy kodérů, a pokud nalezne požadovaný vzor, je na jeho výstupu nastaven odpovídající bit identifikující daný aplikační protokol (0001 - HTTP, 0010 - SMTP, 0100 - SSH, 1000 - SIP, 0000 - neznámý). Velikost vstupu komparátoru odpovídá délce nejdelšího vzoru. Třetí úroveň tvoří hradlo OR, které spojuje výstupy komparátorů do jediného čtyřbitového výstupu klasifikátoru.

Implementace probíhala v následujících krocích: Klasifikátor byl implementován konvenčním způsobem a následně byly optimalizovány vybrané komponenty (kodéry). Poté byl klasifikátor implementován s využitím optimalizovaných kodérů a na závěr byla provedena verifikace kvality klasifikace.

Všechny tři klasifikátory (CL-acc, CL-cmp, CL-lat) byly popsány behaviorálně pomocí VHDL a syntetizovány do FPGA systému SDM. Optimalizační kritérium syntézy bylo nastaveno na zpoždění obvodu.

Pro optimalizaci všech kodérů každého klasifikátoru byla použita standardní verze CGP pracující s dvou-vstupovými hradly. Použití hradel s šesti vstupy se nezdá být pro návrh vhodné z důvodu velkého prohledávaného prostoru. Detailní nastavení CGP je popsáno v [13].

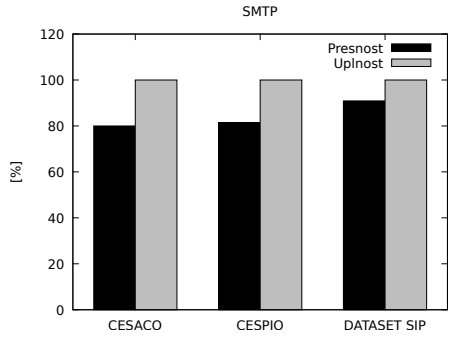
Tabulka I
Výsledky syntézy pro Xilinx Virtex-7 XC7VH580T FPGA.

Klasifikátor	LUTs	Flip Flop	Zpoždění [ns]
CL-acc	2352	0	6.410
CL-acc+CGP	1909	0	6.113
CL-cmp	1549	0	6.093
CL-cmp+CGP	1073	0	5.604
CL-lat	1625	0	5.943
CL-lat+CGP	1217	0	5.139
Yamagaki/Clark	10431	2326	77.504 (16 x 4.844)
AMTH	10547	2190	71.536 (16 x 4.671)

Získané výsledky návrhu jsou pro všechny kodéry všech klasifikátorů publikovány v [13]. Výsledky potvrdily předpoklad, že optimalizované kodéry klasifikátoru CL-lat jsou menší než kodéry CL-cmp a kodéry CL-cmp jsou menší než kodéry CL-acc. Tyto kodéry byly syntetizovány do FPGA. V tabulce I vidíme počet LUT, Flip-Flop a zpoždění pro všechny syntetizované obvody (optimalizované obvody pomocí CGP jsou označeny jako "+CGP"). Pro srovnání jsou přidány dvě implementace klasifikátoru vytvořené konvenčně pomocí regulárních výrazů. Tyto klasifikátory obsahují 16 paralelních jednotek, aby bylo dosaženo propustnosti 100 Gb/s.

Zjednodušením klasifikátoru CL-acc na klasifikátory CL-cmp a CL-lat a s využitím optimalizace pomocí CGP bylo uspořeno 48,2 % prostoru (LUT) a zpoždění zkráceno o 19,8 % oproti implementaci CL-acc.

Kvalita klasifikace byla ověřena offline s využitím softwarového modelu klasifikátoru na všech třech datových sadách. Jako metriky kvality byly zvoleny přesnost a úplnost.



Obrázek 1. Přesnost a úplnost v procentech pro protokol SMTP na všech třech datových sadách s použitím klasifikátoru CL-lat.

Přesnost určuje počet toků správně klasifikovaných. Na obrázku 1 jsou výsledky pro protokol SMTP pro nejméně přesný klasifikátor CL-lat. Nejméně přesná je klasifikace pro protokol SMTP, který je identifikován pomocí relativně krátkého a obecného vzoru. Pro systém SDM je ovšem důležitější metrika úplnosti, která určuje počet identifikovaných toků. Pro systém SDM je z pohledu bezpečnosti důležité zachytit všechna hledaná data (protokoly). Pokud jsou nějaká data zachycena navíc, mohou být později odstraněna. Úplnost 100 % znamená, že žádná data nebyla ztracena.

Návrh, implementace a optimalizace základní funkční jednotky klasifikátoru CL-acc (9 kodérů a komparátor) pro protokoly HTTP, SMTP a SSH a ohodnocení kvality tohoto klasifikátoru bylo publikováno na mezinárodní konferenci *European Conference on the Applications of Evolutionary Computation* [14]. Výše popsaná rozšířená verze byla publikována v impaktovaném časopise *Applied Soft Computing* (IF = 2,857) [13].

B. Evoluce hašovací funkce pro hašování síťových toků

Hašovací funkce se v síťových aplikacích používají k mnoha účelům, například, jak bylo popsáno výše, k hašování IP adres. Další možností je hašování síťových toků, tedy pětice zdrojová a cílová adresa (2 x 32b), zdrojový a cílový port (2 x 16b) a transportní protokol (8b), tedy 104 bitů. Pomocí této pětice je možno jednoznačně v síťových aplikacích identifikovat každý tok. V systému SDM se hašování síťových toků využívá zejména v softwarové části, pro ukládání informací o jednotlivých tocích, případně informací o jejich zpracování pro různé uživatelsky specifické aplikace. V systému SDM je použita hašovací tabulka s lineárním seznamem pro řešení kolizí [15]. Tento typ tabulky klade speciální požadavky na hašovací funkci. Pro tabulku je nejvhodnější hašovací funkce, která produkuje co nejméně kolizí. Ovšem při hašování síťových toků vzniká poměrně velké množství kolizí, proto je vhodnější hašovací funkce, která neprodukuje dlouhé seznamy, ale produkuje větší množství krátkých seznamů. Pokud budou seznamy u jednotlivých záznamů krátké, bude jejich prohledávání méně časově náročné.

Evolučně navržená hašovací funkce má být určena primárně pro software. Z toho důvodu se jeví jako vhodné použít LGP pro tento návrh. První experimenty s návrhem ukázaly, že navržené hašovací funkce se stejnou množinou funkcí, jaké obsahují konvenční hašovací funkce (prvočísla, logické operace, sčítaní a násobení) a fitness funkcí zaměřenou pouze na minimalizaci kolizí, nepřinesly výrazné vylepšení, jak z pohledu kvality hašování (počet kolizí), tak rychlosti výpočtu.

Pro zjednodušení návrhu hašovací funkce byla dimenze vstupního vektoru snížena z 5 na 3 slova velikosti 32 bitů. Zdrojová a cílová adresa je zachována v původní podobě, ale ze zdrojového portu (zp), cílového portu (cp) a transportního protokolu (tp) je vytvořeno jedno 32 bitové slovo následujícím způsobem:

$$((zp << 16) \vee cp) \oplus tp.$$

Reálná data obsahují zejména dva transportní protokoly (TCP a UDP), takže snížení dimenze nezpůsobí výraznou ztrátu informace. Fitness funkce je založena na počtu kolizí, ale zohledňuje i délky seznamu. Nechť K_i je počet klíčů mapovaných do slotu i v hašovací tabulce h. Pak je fitness funkce $f(h)$ počítána jako:

$$f(h) = \sum_{i=1}^{s} g_i, \text{ kde} \quad (1)$$

$$g_i = \begin{cases} 0 & \text{if } K_i \leq 1 \\ \sum_{j=2}^{K_i} j^2 & \text{if } K_i \geq 2 \end{cases} \quad (2)$$

a s je počet slotů. Tato fitness funkce penalizuje kandidátní funkce, které produkují dlouhé seznamy v hašovací tabulce s lineárním seznamem. Menší hodnota fitness znamená lepší řešení.

Délka kandidátního programu byla omezena na maximálně 12 instrukcí, bylo použito osm 32 bitových registrů. Sada instrukcí obsahuje rotaci vpravo, XOR a sčítání. Populace měla velikost 200 jedinců a maximálně bylo provedeno 1000 generací. Na obrázku 2 je hašovací funkce, která reprezentuje nejlepší nalezené řešení. Dále byly pro srovnání vybrány další evolučně navržené hašovací funkce. LGPhash2 je vybrána jako velice jednoduchá funkce, LGPhashMult je navržena s instrukční sadou která navíc obsahuje násobení a LGPhash20inst

```
unsigned int LGPHash1 (unsigned int * input ){
  r[0] = input[0]
  r[1] = input[1]
  r[2] = input[2]

  r[1] = r[1] + r[2]
  r[2] = r[1] + r[2]
  r[4] = r[0] + r[2]
  r[0] = r[1] + r[4]
  r[3] = 0x5BE0CD19
  r[2] = rotr(r[3], r[4])
  r[0] = r[0] + r[2]
  r[0] = 0xA54FF53A + r[0]
  return r0 ⊕ (r0 >> 16)
}
```

Obrázek 2. Evolučně navržená hašovací funkce LGPHash1.

je omezena na maximálně 20 instrukcí. Další nastavení LGP a navržené hašovací funkce jsou publikovány v [16].

V rámci experimentů byly všechny hašovací funkce implementovány v jazyce C a všechna měření probíhala na procesoru Intel XEON E5-2630. Byly vytvořeny tři datové sady obsahující 20000 (DataSet1, použitá jako trénovací sada), 50000 (DataSet2) a 100000 (DataSet3) trénovacích vektorů. Pro srovnání byly vybrány obecné konvenční hašovací funkce (DJBHash, DEKHash, FVNHash, One At Time, lookup3, Murmur2, Murmur3, CityHash), speciální hašovací funkce pro hašování síťových toků (XORhash) a obecné evolučně navržené hašovací funkce (GPhash a EFhash). V tabulce II jsou výsledky měření doby výpočtu hašovací funkce pro každý vektor trénovací sady (průměrná hodnota 20 nezávislých měření). Navržené hašovací funkce LGPhash1 a LGPhash2 jsou nejrychlejší. Dále byl proveden test na počet kolizí, kde se ukázalo, že počet kolizí je srovnatelný s ostatními hašovacími funkcemi. Podrobné výsledky jsou publikovány v [16].

Použitím evolučních technik pro návrh specializované hašovací funkce jsme dosáhli u funkce LGPhash1 zkrácení výpočetního času o 10.4%, 4.2% a 3.0% pro DataSet 1, 2 a 3 oproti nejlepší konvenčně navržené specializované XORhash. Navíc LGPhash1 dosahuje zlepšení výpočetního času 48.5%, 31.4% a 26.9% oproti hašovací funkce murmur3, která je typicky používána v SDM. Pro návrh hašovacích funkcí byla využita vlastní paralelní varianta LGP prezentovaná v [17].

IV. Závěr

Cílem disertační práce je aplikování evolučních algoritmů na vybrané síťové aplikace, nebo jejich netriviální části a dosáhnout tím jejich vylepšení a to z pohledu funkčnosti, časové náročnosti nebo příkonu. První výsledky návrhu a optimalizace aplikací v rámci systému SDM již byly publikovány. Otevřenou otázkou je, jak pracovat se zpožděním v GP.

Plán dalších prací zahrnuje využití multikriteriální optimalizaci v LGP i CGP, návrh programů pomocí LGP s ohledem na možnosti vektorizace instrukcí a zohlednění velikosti obvodu (počet LUT) v FPGA na základě predikce během evoluce. Dále tyto přístupy budou porovnány s dalšími optimalizačními metodami např. s ABC.

Tabulka II
Průměrný čas pro vložení datové sady do hašovací tabulky.

Hašovací funkce	Čas $[ms]$ DataSet1	DataSet2	DataSet3
DJBHash	1.783	5.036	13.254
DEKHash	1.592	4.591	12.199
FVNHash	1.678	4.647	12.373
One At Time	2.365	6.269	15.763
lookup3	1.275	3.736	9.931
Murmur2	1.314	3.820	10.153
Murmur3	1.590	4.434	11.568
CityHash	3.089	7.883	19.237
XORHash	0.913	3.174	8.708
GPHash	1.936	6.229	15.813
EFHash	2.323	16.282	56.921
LGPhash1	*0.818*	*3.039*	*8.446*
LGPhash2	***0.756***	***2.852***	***8.057***
LGPhashMult	0.912	3.349	9.096
LGPhash20inst	0.916	3.242	8.954

Poděkování

Tato práce byla podporována projektem Vysokého učení technického v Brně FIT-S-14-2297.

Literatura

[1] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[2] W. Banzhaf and P. Nordin et al., *Genetic programming: an introduction*. Morgan Kaufmann San Francisco, 1998, vol. 1.

[3] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming*. Springer, 2000, pp. 121–132.

[4] J. F. Miller, *Cartesian genetic programming*. Springer, 2011.

[5] M. F. Brameier and W. Banzhaf, *Linear genetic programming*. Springer Science & Business Media, 2007.

[6] A. S. Tanenbaum, "Computer networks, 4-th edition," 2003.

[7] F. Halsall, *Computer Networking and the Internet, 5/e*. Pearson Education India, 2006.

[8] R. M. Morais and C. Pavan et al., "Genetic algorithm for the topological design of survivable optical transport networks," *Journal of Optical Communications and Networking*, vol. 3, no. 1, pp. 17–26, 2011.

[9] L. Barolli and A. Koyama et al., "A genetic algorithm based routing method using two qos parameters," in *Proceedings. 13th International Workshop on Database and Expert Systems Applications.* IEEE, 2002, pp. 7–11.

[10] R. Dobai and J. Kořenek, "Evolution of non-cryptographic hash function pairs for fpga-based network applications," in *2015 IEEE Symposium Series on Computational Intelligence*. IEEE, 2015, pp. 1214–1219.

[11] L. Kekely, V. Pus, and J. Korenek, "Software defined monitoring of application protocols," in *INFOCOM, 2014 Proceedings IEEE*, 2014, pp. 1725–1733.

[12] E. Sommer and M. Strait, "Application Layer Packet Classifier for Linux," Jan. 2009. [Online]. Available: http://l7-filter.sourceforge.net/

[13] D. Grochol and L. Sekanina et al., "Evolutionary circuit design for fast fpga-based classification of network application protocols," *Applied Soft Computing*, vol. 38, pp. 933–941, 2016.

[14] D. Grochol and L. Sekanina et al., "A fast fpga-based classification of application protocols optimized using cartesian gp," in *18th European Conference, EvoApplications*. Springer, 2015, pp. 67–78.

[15] W. D. Maurer and T. G. Lewis, "Hash table methods," *ACM Computing Surveys (CSUR)*, vol. 7, no. 1, pp. 5–19, 1975.

[16] D. Grochol and L. Sekanina, "Evolutionary design of fast high-quality hash functions for network applications," 2016, GECCO, Denver.

[17] D. Grochol and L. Sekanina, "Comparison of parallel linear genetic programming implementations," in *Mendel 2016: Recent Advances in Soft Computing*. Springer, 2016.