FAKULTA STROJNÍHO INŽENÝRSTVÍ
ústav automatizace a informatiky

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

GPU AKCELEROVANÉ METAHEURISTIKY VE VYBRANÝCH ÚLOHÁCH GLOBÁLNÍ A KOMBINATORICKÉ OPTIMALIZACE

GPU ACCELERATED METAHEURISTICS FOR GLOBAL AND COMBINATORIAL OPTIMIZATION TASKS

TEZE K DISERTAČNÍ PRÁCI
SHORTENED VERSION OF DOCTORAL THESIS

AUTOR PRÁCE Ing. Ladislav Dobrovský
AUTHOR

ŠKOLITEL prof. Ing. Radomil Matoušek, Ph.D.
SUPERVISOR

BRNO 2024

ABSTRAKT

Získávání dostatečně dobrých řešení v rozumném čase je jednou z hlavních náplní inženýrské práce. Přesněji v kontextu inženýrské optimalizace je to hledání parametrů či vhodnější struktury s kompromisem protichůdných požadavků. Řešené problémy jsou stále větší a složitější výzvou. K jejich zvládnutí je třeba neustále inovovat řešící metody, či jejich implementace. Jednou skupinou těchto metod jsou právě metaheuristické algoritmy, kterými se tato práce podrobně zabývá. Jsou teoreticky představeny a prakticky implementovány dvě metody vhodné pro akceleraci na grafických procesorech pro obecné výpočty (GPGPU). První je rozšíření HC12 algoritmu o tzv. Tabu list. Druhá je nová varianta diferenciální evoluce s více ostrovními populacemi s GPU optimalizací genetických operátorů. Prostor je také věnován přehledu prostředků pro vysoce náročné výpočty (HPC). Metody jsou ověřeny na úlohách globální spojité a kombinatorické optimalizace (QAP a SAT). Pro problém SAT jsou verifikovány dvě varianty GPU akcelerace. Pro úplnost je provedeno porovnání implementací pro více jádrová CPU a pro GPU na HC12 a kombinatorické úloze QAP.

ABSTRACT

Achieving sufficiently good solutions in a reasonable time is one of the main focuses of engineering work. More precisely, in the context of engineering optimization, it involves searching for parameters' values or a better structure with. To reach a compromise is often needed for conflicting requirements.. New problems become increasingly challenging. To manage them, it is necessary to innovate methods and their implementation. One group of these methods are metaheuristic algorithms and this work describe their principles and implementation in detail. Two methods suitable for acceleration on general purpose graphics processors (GPGPUs) are theoretically presented and practically implemented. The first is the extension of the HC12 algorithm by the so-called Tabu list. The second is a new variant of differential evolution (DE) with multiple island populations suitable for GPU optimization of genetic operators. Space is also devoted to an overview of high-performance computing (HPC) hardware resources. The methods are verified on tasks of global continuous and combinatorial optimization (QAP and SAT). Two variants of GPU acceleration are verified for the SAT problem. For completeness, implementations for multi-core CPUs and for GPUs on HC12 and the combinatorial QAP task are compared.

KLÍČOVÁ SLOVA

Metaheuristiky, GPU akcelerace, CUDA, ISPC, HC12, diferenciální evoluce, QAP, SAT, HPC

KEYWORDS

Metaheuristics, GPU acceleration, CUDA, ISPC, HC12, differential evolution, QAP, SAT, HPC

KLÍČOVÁ SLOVA

Metaheuristiky, GPU akcelerace, CUDA, ISPC, HC12, diferenciální evoluce, QAP, SAT, HPC

KEYWORDS

Metaheuristics, GPU acceleration, CUDA, ISPC, HC12, differential evolution, QAP, SAT, HPC

DOBROVSKÝ, Ladislav. *GPU akcelerované metaheuristiky ve vybraných úlohách globální a kombinatorické optimalizace*. Brno, 2024. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/154877. Teze k disertační práci. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Školitel prof. Ing. Radomil Matoušek, Ph.D.

OBSAH

1 ÚVOD

Optimalizace je v širším kontextu vylepšování. Předpokládá tedy možnost kvalitativního hodnocení a následného seřazení řešení od horšího k lepšímu. Stanovení důležitosti dílčích požadavků je samo o sobě kompromisem. V užším kontextu matematické optimalizace jde o hledání globálních extrémů funkcí. Tato práce se věnuje problémům z pohledu tzv. inženýrské optimalizace, což často představuje hledání dostatečně dobrých řešení v rozumném čase a za rozumnou cenu (feasible solutions in feasible time at feasible cost).

Práce je rozdělena na teoretickou a praktickou část. Praktická část představuje základní výsledek a zahrnuje implementaci zvolených metaheuristik s orientací na datově paralelní výpočty, zvolených úloh globální spojité i kombinatorické optimalizace (taktéž s vhodnými datově paralelními implementacemi) a provedení experimentů s vyhodnocením, včetně experimentů porovnávající zvolené hardwarové platformy.

Teoretická část uvádí základní pojmy a doplňuje popis praktické částí. Jsou zde uvedeny principy jednotlivých metaheuristických algoritmů, formální definice a vlastnosti zvolených úloh globální spojité i kombinatorické optimalizace, přehled hardwarových platforem dostupných v současnosti pro vysoce náročné výpočty (HPC). Také je zde popis přístupů k programování více jádrových centrálních procesorových jednotek (multi-core CPU) a také grafických procesorových jednotek s podporou obecných výpočtů (GPGPU). Vše je podložené relevantními zdroji literatury.

Mezi hlavní autorovy cíle pro tuto práci patří praktické využití hardware vhodného pro vysoce náročné výpočty (HPC), primárně GPGPU, zkoumání variant více populační diferenciálních evoluce a efektivní řešení úlohy QAP.

Kapitola 2 se věnuje fungování algoritmů HC12, GAHC a diferenciální evoluce (DE). Jak jsou definovány problémy globální spojitá optimalizace, která hledá vhodné hodnoty pro argumenty funkcí reálných proměnných je popsáno v kapitole Error: Reference source not found. Také je v této kapitole definována kombinatorická optimalizace, která se pokouší najít nejvhodnější kombinaci, permutaci či variaci. Konkrétními úlohami k řešení je problém kvadratického přiřazení (QAP) a problém splnitelnosti Booleovské formule (SAT, satisfiability problem).

Přehledem a popisem dostupných HW platforem pro potřeby vysoce náročných výpočtů (HPC) se zabývá kapitola 3. Jsou zde zmíněny jak komplexní řešení (superpočítače, grid, cloud), tak jednotlivé součástky, ze kterých se skládají zařízení od mobilních zařízení po cloud (CPU, GPGPU, FPGA, ASIC). Konkrétní rozdíly v architektuře a preferovaných přístupech pro programování více jádrových CPU (kapitola 3.2, programovací jazyky C++ a ISPC) a GPGPU (kapitola 3.1, softwarové platformy CUDA a SYCL).

Popisem datově paralelních implementací jednotlivých metaheuristických algoritmů a optimalizačních úloh (16 spojitých funkcí, QAP a SAT) se zabývá kapitola 4. Metaheuristika HC12 je rozšířena o seznam zakázaných řešení, tzv. Tabu list, v implementaci vhodné pro GPGPU.

Diferenciální evoluce s více ostrovními populacemi je v unikátní implementací, která velice výhodně řeší problémy dostupných GPGPU implementací z pohledu aplikace

genetických operátorů mutace a křížení. Po efektivní transformaci dat zachovává uspořádání pro vhodnou paralelizaci účelové funkce na GPGPU.

Text je doplněn vhodnými pseudokódy a výňatky zdrojového kódu v programovacích jazycích C++, CUDA C, Python a Matlab (m-kód). Také jsou podrobněji popsány relevantní části veřejného programového rozhraní (*public API*) a příklady uživatelem definovaných účelových funkcí pro použití s implementací řešičů, které metaheuristiky prakticky zpřístupňují. Pro programování GPGPU je použita platforma NVIDIA CUDA. V kapitole 4.1.2 je také jako *benchmark* implementovaná metoda HC12 s úlohou QAP v pěti různých verzích. Čtyři pro více jádrová CPU s pomocí C++ a ISPC. Poslední, pátá, je upravená GPU verze pro co nejpřímější srovnání výkonu platforem.

Provedenými experimenty se zabývá text kapitoly 5. Nejprve je verifikována funkce implementace DE na příkladě analýzy vlivu volby strategie korekce nevhodných řešení SDIS a koeficientů *F*. Pak se u DE zkoumá vliv periody migrace u jednotlivých migračních strategií. Dalším experimentem je porovnání efektivity pěti různých implementací HC12 pro QAP na různých HW platformách CPU a GPGPU. Dva experimenty v podkapitole 5.3 ověřují úspěšnost implementace HC12 pro QAP. Posledním experimentem je zjišťování rozdílů mezi implementacemi (Python+NumPy, Matlab, CUDA C) testovací sady funkcí spojité optimalizace.

2 METAHEURISTIKY OPTIMALIZAČNÍ ÚLOHY

Optimalizační metaheuristiky jsou algoritmy, které prohledávají diskrétní (nebo diskretizovaný) stavový prostor parametrů účelové funkce. Takových strategií existuje značné množství, dělí se hlavně na lokální a globální prohledávání. Liší se tím, zda se pokouší nalézt globální extrém účelové funkce nebo z principu zůstávají v blízkém lokálním extrému.

Všechny optimalizační algoritmy pracují s konceptem účelové funkce (objective function), v algoritmech inspirovaných evoluční teorií (*bio-inspired* algoritmy) je označovaná jako *fitness* funkce, či zkráceně *fitness*. Poznamenejme, že v biologickém kontextu je tato funkce definovaná jako nezáporná, tedy určená k maximalizaci.

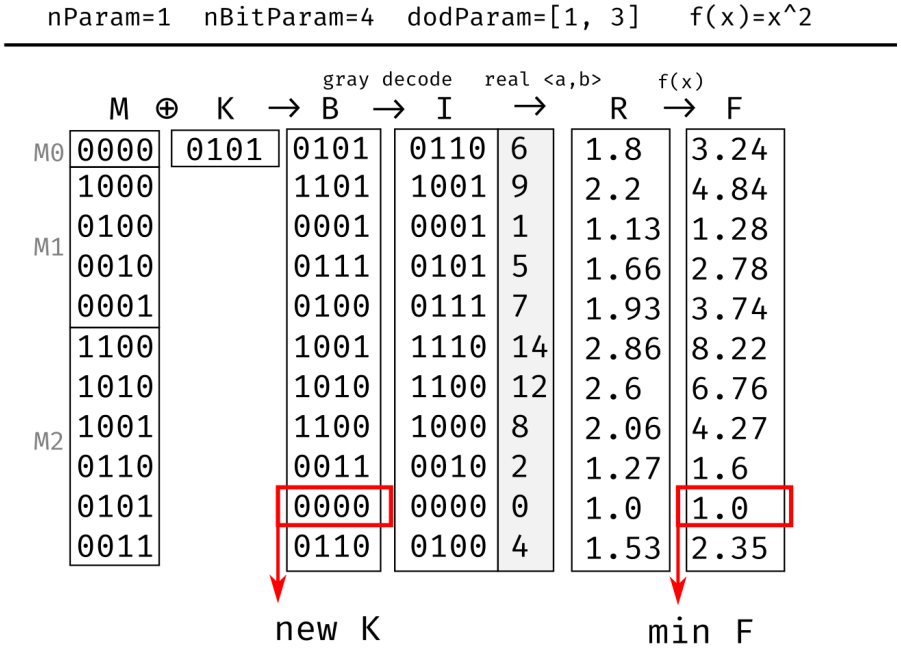
Od roku 1975, kdy byly představeny Genetické algoritmy (GA), byly vyvinuty stovky variant různých metod inspirovaných evolucí, biologicky či obecně přírodou. Taxonomie přírodních a populačních algoritmů uvedená v [6] jich zařazuje 323.

2.1 Algoritmy HC12 a GAHC

Oba algoritmy byly vyvinuté doc. Matouškem na VUT v Brně [7, 8, 9, 10]. První algoritmus, HC12, je variantou horolezeckého algoritmu nad binárními řetězci. Jeho relativní zvláštností je využití XORu binárních masek a Grayova kódování pro efektivnější generování bodů okolí. Téměř všechny operace lze velmi dobře provádět paralelně. Druhý algoritmus, GAHC, je hybridní modifikace genetického algoritmu, který integruje HC12 jako mutační operátor náhodně zvolených podřetězců v rámci mutace jedince.

2.1.1 Princip algoritmu HC12

Základní varianta pro problémy s binárními řetězci (například problém batohu), celočíselnými (integer) nebo racionálními (IEEE-754 floating point) argumenty účelové funkce se kóduje podle následujícího obrázku 1.



Obr. 1: Schéma iterace algoritmu HC12

- *nParam*: počet parametrů účelové funkce, *nBitParam*: počet bitů na parametr
- *dodParam*: intervaly *<a, b>*, na které se rovnoměrně pro každý parametr funkce rovnoměrně zobrazují hodnoty *I*.
- **M**: matice masek, složená z pod-matic M0, M1 a M2, které obsahují hodnoty binárních řetězců s Hammingovými vzdálenostmi 0, 1 a 2.
- **K**: binární řetězec k transformaci (počáteční nebo vybrán v předchozí iteraci).
- **B**: matice binárních řetězců v
- **I**: matice nezáporných (bez znaménka, unsigned) celých čísel vzniklá z částí binárních řetězců matice B převedených z Grayova kódu na přímý binární kód.

V každé iteraci se na řádku obsahujícím minimum vektoru F vybírá z matice **B** nová hodnota binárního řetězce *K* pro další iteraci. Pokud je nejmenší hodnota F v prvním řádku, algoritmus se ukončí.

2.1.2 GAHC

Genetický algoritmus (GA) je dnes již klasická metaheuristika [11] primárně vyvinutá pro účelové funkce nad binárními řetězci s následným překódováním. GA se inspiruje principy evoluční teorie a její fyzickou manifestací v nukleových kyselinách genotypu a jejich následném projevu jako fenotyp organismů. Základní kroky algoritmu jsou selekce (výběr jedinců do další generace/iterace), křížení (kombinace genotypu dvojice či více jedinců), mutace (náhodná úprava genotypu přeživších jedinců). Některé implementace sledují počet iterací, po které jedinec existuje a zavádí koncept stárnutí, jedinci po limitu "umírají" tj. jsou odstraňováni z populace [12]. Pokud varianta algoritmu nezaručuje nekonečné přežití nejlepšího jedince (některé varianty selekce a stárnutí), je třeba toto řešení vždy uschovat do paměti v kopii, aby algoritmus na konci neprohlásil horší řešení za finální výsledek.

GAHC je využití algoritmu HC12 (a dalších variant) jako (přídavného) mutačního operátoru, který náhodnou část binárního řetězce genomu optimalizuje. Při zastavení konvergence genetického algoritmu lze provést ještě dodatečně HC12 pro celý řetězec a ověřit tak pozici lokálního extrému v rámci dostupného okolí [13].

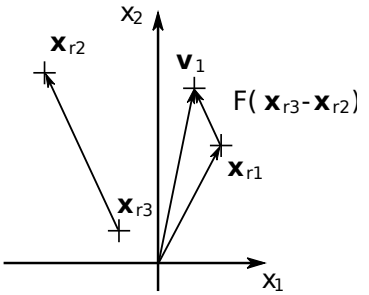
2.2 Diferenciální evoluce (DE)

Diferenciální evoluce (DE) je algoritmus podobný genetickému algoritmu. Představen byl v roce 1995 v technické zprávě. Upravený text byl vydán v časopise v roce 1997. [14, 15]. První dvě použití byly pro hledání 18 parametrů IIR-filtru a jako účastník soutěže ICEC'96 v hledání extrémů desítky testovacích funkcí s reálnými (racionálními) argumenty. [16, 17]

Na rozdíl od genetického algoritmu nepracuje diferenciální evoluce s binárními řetězci převedenými na racionální čísla, ale s racionálními čísly přímo. Dalším rozdílem je, že operátory mutace a křížení jsou definovány jinak. Využívají tři a více jedinců (vektorů) tak,

že k prvnímu je přičten v některých složkách rozdíl, diference; odtud název diferenciální evoluce. Vektor složený pouze z těchto diferencí je mutační. Vektory vzniklé křížením – výběrem složek buď původního nebo mutačního vektoru, zvané *trial* vektory, nahrazují původní vektor (selekce) pouze při zlepšení hodnoty fitness funkce.

Obrázek 2 ilustruje jak mutační vektor $\mathbf{v}_i$ vzniká. Z populace jsou náhodně zvoleny tři různé vektory $\mathbf{x}_{r1}$, $\mathbf{x}_{r2}$ a $\mathbf{x}_{r3}$ tak, že k vektoru $\mathbf{x}_{r1}$ je přičten rozdíl vektorů $\mathbf{x}_{r2}$ a $\mathbf{x}_{r3}$ násobený koeficientem *F*. *Trial* vektor $\mathbf{y}_i$ (často v literatuře značeny jako $\boldsymbol{u}_i$) vzniká z původního vektoru $\mathbf{x}_i$ a z mutačního vektoru $\mathbf{v}$ operátorem křížení. Zda ve složce vektoru dojde ke křížení závisí na pravděpodobnosti křížení dané koeficientem *CR*. Některé varianty DE vybírají také náhodně jednu složku, která se nahradí vždy, nezávisle na výsledku porovnání s *CR*. To zaručí, aby se vždy *trial* vektor odlišoval od vektoru původního alespoň v jedné složce.



Obr. 2: Vznik mutačního vektoru, DE/rand/1 (2D)

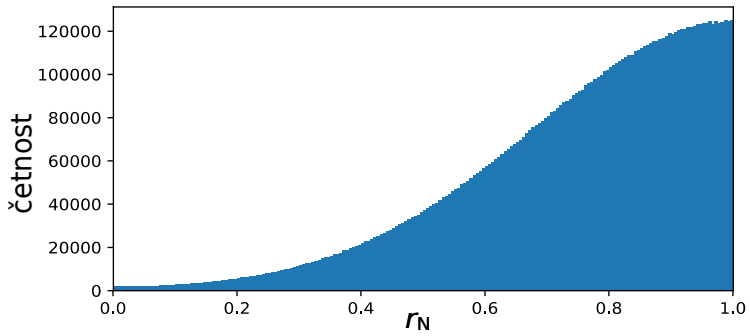
Tato základní mutační strategie se značí DE/rand/1. Mutační strategie se liší počtem použitých diferencí a volbou použitých vektorů (původní $\mathbf{x}_i$, náhodný $\mathbf{x}_{rk}$, nejlepší $\mathbf{x}_{best}$). Jsou klasifikovány v [18]: DE/rand/1, DE/rand/2, DE/best/1, DE/best/2, DE/current-to-rand/1, DE/current-to-best/1.

Existuje několik opravných zobrazení klasifikovaných dle [19] jako *Strategy of dealing with infeasible solutions – SDIS*: *saturation (sat), mirror (mir), toroidal (tor), halfway -to-violated-bounds (HVB), uniform* (*uni*); *complete one-sided truncated normal* (*COTN*).

Metoda *COTN* není v literatuře nijak dobře popsaná: pseudonáhodné číslo r_N má funkci hustoty pravděpodobnosti přibližně (1) na intervalu <0; 1>. Prakticky není třeba zjišťovat hodnotu koeficientu *c* a při generování r_N je použito pseudonáhodné číslo r_{N0} z normální rozdělení $N(0; 0{,}32^2)$ a korekce do intervalu <0; 1> se provede dle (2). Hodnota σ byla volena tak, aby $f(r_N)$ odpovídala přibližně grafu (a) na „Figure 1“ z [19]. Histogram generovaných pseudonáhodných čísel je na obrázku 3.

$$f(r_N) = 2\,\mathrm{N}(\mu;\sigma^2)+c \approx 2\,\mathrm{N}(0;\,0{,}32^2)+c$$
$$\int_0^1 f(r_N)\,dx = 1 \tag{1}$$

$$r_N = \begin{cases} r_{N0} & ;r_{N0}\in\langle 0,1\rangle \\ \left|\mathrm{fmod}(r_{N0},1)\right| & ;r_{N0}<0 \\ \left|\mathrm{fmod}(1-r_{N0},1)\right| & ;r_{N0}>1 \end{cases} \tag{2}$$



Obr. 3: Histogram pseudonáhodných čísel pro SDIS COTN

2.2.1 Diferenciální evoluce s více ostrovními populacemi

Dosavadní implementace algoritmu diferenciální evoluce (DE) na CUDA [21, 22] neřeší některé problémy protichůdných požadavků efektivního výpočtu genetických operátorů na GPU. Také se zabývají faktem, že DE při přílišném zvýšení počtu jedinců v populaci má výrazně negativní vliv na rychlost konvergence [23]. Kompromisem je zavedení více populací, které se obrazně řečeno vyvíjejí samostatně na tzv. ostrovech (island). Každá ostrovní populace může mít jiné parametry a vyvíjí se samostatně. V obecném případě se mohou populace lišit v počtech jedinců, koeficientech *F* a *CR*, mutaci i způsobu opravy nevyhovujících řešení (SDIS). Vždy po určitém počtu generací dojde k migraci, tj. přesunu (kopírování) části jedinců mezi jednotlivými ostrovy. V některé literatuře, například [24], je nazývána migrace jako propagace.

Základní migrační strategie jsou dle [24]: *One to one*, *One to N*, *N to one*, *N to N*. Další dvě migrační strategie mohou být:

- *PermuteN*, kdy je každé populaci přiřazena jiná cílová populace, které předá nejlepšího jedince tak, že žádné dvě populace nemají stejný cíl.
- *RandTarget*, kdy si každá populace náhodně vybere jinou cílovou populaci, které předá svého nejlepšího jedince. Pokud je populace cílová pro více jedinců, dohodne se, který bude přijat.

2.3 Spojitá optimalizace

Spojitá optimalizace se zabývá hledáním globálních extrémů funkcí s reálnými (racionálními) argumenty. Mnoho technických problémů lze vyjádřit jako takovou funkci. Výsledný problém je pak formulován jako hledání globálního extrému funkce; minimalizace (3) nebo maximalizace. Definičním obor D může být různě omezen pro jednotlivé parametry. Tato práce se zabývá metodami, které o funkci nepředpokládají diferencovatelnost ani spojitost na celém intervalu či po částech.

$$\arg\min_{x \in D} f(x) = \{x \in D : \; f(x) = \min_{y \in D} f(y)\} \tag{3}$$

Aby bylo možné porovnávat robustnost jednotlivých metod, byly různými autory definovány vhodné testovací funkce. Při jejich návrhu byla snaha o dosažení různých

vlastností (ne všech najednou): obecný počet dimenzí, známá poloha optima, vysoká hustota lokálních extrémů, nediferencovatelnost, nespojitost, konstantní oblasti a oblasti se zanedbatelnou změnou, lineární neseparabilita a posunutí optima ze středu soustavy.

Pro testování byly vybrány funkce F1 až F12 z The Genetic and Evolutionary Algorithm Toolbox for Matlab (GEATbx) [26] a funkce F4 až F10 z "The 100-Digit Challange" [27], označované dále jako CEC04 až CEC10. A to v základní verzi pro více dimenzí bez transformací a s transformacemi (škálování, rotace a posun) ve 2 a 10 dimenzích, zde značené jako CEC04t až CEC10t.

Například funkce „CEC08 Expanded Schaffer's F6 Function":

$$
\begin{gathered}
f_{CEC\,08}(\boldsymbol{x}) = g(x_1, x_D) + \sum_{i=1}^{D-1} g(x_i, x_{i+1}) \\
g(x, y) = \frac{1}{2} + \frac{\sin^2\sqrt{x^2 + y^2} - \frac{1}{2}}{1 + \frac{x^2 + y^2}{1000}} \\
x_i \in \langle -100\,;\,100 \rangle
\end{gathered}
\tag{4}
$$

Metoda HC12 vyžaduje diskretizaci hodnot argumentů na omezeném intervalu. Interval lze pak dále zmenšovat a dosáhnout tak postupně jemnějšího pokrytí. Naopak metoda diferenciální evoluce pracuje přímo s reálnými čísly (racionálními, s plovoucí desetinnou čárkou) a může fungovat na neomezeném intervalu. DE trpí opačným problémem, kdy je třeba zajistit, aby hodnoty argumentů nově navrhovaných řešení ležely v definičním oboru.

2.4 Kombinatorická optimalizace

Kombinatorická optimalizace se zabývá řešením problémů, u kterých mohou být kombinace, variace či permutace vstupem optimalizované funkce, tj. existuje vhodné zobrazení *n*-prvkové relace do množiny s definovanou operací porovnávání (většinou reálná/racionální/celá čísla). Při vhodném kódování se některé metaheuristiky, navržené pro nespojité či nediferencovatelné funkce, ukazují jako vhodné nástroje pro získávání dostatečně dobrých řešení.

2.4.1 Problém kvadratického přiřazení (QAP)

Quadratic Assignment Problem (QAP) patří mezi klasické NP-těžké (NP-*hard*) problémy a je příbuzný problému obchodního cestujícího (TSP). Jeho použití v základní variantě jako *facility placement* metody při již známých možných umístěních (*places*) jednotlivých *facilities* nachází využití v logistice a technice [28]. Mezi umístěními je definována matice vzdáleností D a mezi *facilities* je definována matice toků F. Samotným problémem je nalezení permutace π, která by přiřadila *facilities* na místa tak, aby byl minimální součet všech součinů vzdáleností a toku (5). [29]

Následně je tedy QAP problém:

$$\min_{\pi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} D_{\pi(i)\pi(j)} F_{ij} \tag{5}$$

kde S_n je množina všech permutací množiny {1, 2, 3, …, n}. Pokud jsou matice D a F symetrické podle hlavní diagonály, jedná se o symetrický QAP (6) a lze jej vyjádřit pomocí sdružené matice T, kde jsou nad hlavní diagonálou prvky matice F a pod hlavní diagonálou prvky matice D.

$$\begin{gathered} \min_{\pi \in S_n} \sum_{i=1}^{n} \sum_{j=i+1}^{n} 2T_{\pi(\max(i,j))\pi(\min(i,j))} T_{ij} \\ T_{ij} = \begin{cases} F_{ij} & pro\ i > j \\ D_{ij} & pro\ i < j \end{cases} \end{gathered} \tag{6}$$

Pro testování stochastických i exaktních řešičů QAP existuje knihovna problémů QAPLIB [30]. U některých instancí stále není ověřeno exaktně nejlepší řešení a je zde uvedena vzdálenost (GAP) od odhadu spodní meze.

Mezi základní metody odhadu spodní meze patří Gilmore-Lawler bound. Vychází z myšlenky, že pokud se prvky matice D seřadí vzestupně a prvky matice F sestupně, musí být součet násobků mezi nimi nejmenším možným pro tato čísla. Nebývá však možné pro původní matice sestavit permutaci, která by k této sumě dospěla.

2.4.2 Problém splnitelnosti Booleovské formule (SAT)

Tento klasický NP-úplný (NP-complete) problém je významný v teoretické i aplikované informatice. Při řešení obecného SAT problému (6) se hledá odpověď na jednoduchou otázku: „Pro jaké hodnoty logických literálů x_i je konjunkce klauzulí obsahujících tyto literály či jejich negace pravdivá?“.

$$\begin{aligned} l_{po} &\in \{x_1, \neg x_1, x_2, \neg x_2, \ldots,\} \\ q_i &= l_{i1} \vee l_{i2} \vee \ldots \vee l_{io} \\ \boldsymbol{x} &= (x_1, x_2, \ldots, x_n) \\ s(\boldsymbol{x}) &= q_1 \wedge q_2 \wedge \ldots \wedge q_m \\ \mathrm{s}(\boldsymbol{x}) &= 1 \end{aligned} \tag{7}$$

Kde „1“ reprezentuje pravdivou hodnotu, q_i jsou klauzule, $\boldsymbol{x}$ je vektor literálů, l_{po} jsou literály x_o a jejich negace vyskytující se v klauzuli q_p. Funkce $s(\boldsymbol{x})$ je výsledek přiřazení konkrétních pravdivých či nepravdivých hodnot literálům a vyhodnocení konjunkce klauzulí. Ověřuje se zda je výsledek pravdivá hodnota.

K porovnání výkonu či vhodnosti metod pro řešení SAT problému je k dispozici knihovna instancí různých variant SATLIB [31].

3 GPU A HPC VÝPOČTY

V souvislosti s tím, jak se dále nedaří příliš vylepšovat běh programu v jednom vlákně procesu (*single thread performance*), vzniká tlak na vývoj software s použitím více vláken, procesů či přímo jako distribuované systémy spolupracujících programů s využitím dalších akcelerátorů v hybridních architekturách.

Superpočítače jsou specializované a integrované distribuované systémy, které obsahují unifikované výpočetní uzly, propojené pomocí speciální sítě s vysokou rychlostí, případně jsou tyto uzly vybavené dalšími akcelerátory (GPU, FPGA, ASIC). Celý superpočítač je provozován jako celek se specializovaným operačním systémem, který se rovněž stará o rozdělení úlohy na jednotlivé výpočetní uzly více méně transparentně.

Clustery hlediska HPC připomínají superpočítače, ale na každém výpočetním uzlu běží operační systém zvlášť s instancí výpočetní úlohy. Jejich předností je redundance, odolnost vůči částečným výpadkům a rozšiřitelnost. Z hlediska programování jde již o distribuovaný systém a je třeba řešit komunikaci mezi uzly samostatně nebo pomocí knihoven.

Cloudy jsou často velké clustery (a soustavy clusterů), které používají virtualizované servery a kontejnery. Jsou flexibilnější z hlediska alokace dodatečných HW prostředků pro další instance úloh či jejich uvolnění a řízení těchto přidělených kapacit.

Grid je obecně vysoce heterogenní distribuovaný systém, kde se mění dostupnost výpočetních uzlů s přerušováním výpočtu a to klade vysoké nároky na granularitu úlohy a vhodné sestavování mezivýsledků a případných změn v přiřazení uzlů při nedodržení *deadline* pro mezivýsledek.

S růstem počtu diskrétních jader procesorů vzrůstají nároky na programovací jazyky, operační systémy a různá API v oblasti horizontálního škálování, optimalizace lokálnosti přístupů do paměti a vyhnutí se konfliktům paralelních procesů (uváznutí, vyhladovění).

Field Programmable Gate Array, programovatelná hradlová pole, jsou zvláštním druhem obvodu, který se s výhodou využívá pro prototypování integrovaných obvodů a tvorbu specializovaných akcelerátorů. Programování probíhá dvojúrovňově jako konfigurace zapojení buněk, které pak realizují prvky obvodu: logická hradla, statická paměť a datové cesty. Samotné takto vzniklé zapojení může být programovatelné (tzv. *softcore*) nebo poskytovat pouze urychlení konkrétního výpočtu.

Pro některé aplikace se vyplatí na zakázku vyrobit aplikačně specifický integrovaný obvod. Většinou s prototypem realizovaným FPGA, ovšem s dodatečnými simulacemi a testováním při výrobě. Oproti FPGA dosahuje obecně ASIC provozu na vyšších frekvencích a nižší spotřeby energie. Velkou nevýhodou je náročnost, délka a cena celého procesu, tzv. *tapeout*.

Společnost Cerebras uvedla na trh produkt Wafer-Scale Engine. Při výrobě integrovaných obvodů jako křemíkových chipů používá kruhová křemíková destička (*wafer* neboli „oplatka“). Na tu se postupně pomocí litografických, mechanických a chemických procesů vytvoří funkční zapojení jednotlivých kopií obvodu a následně dojde k rozřezání na jednotlivé kusy. Protože proces není dokonalý, některé kusy je nutno vyřadit jako vadné. Někdy je možné část vadných kusů prodat jako méně kvalitní produkt, pokud to architektura obvodu umožňuje. Například 8 jádrový procesor, který má 6 jader v pořádku lze prodat s tím, že se dvě vadná deaktivují. Pokud však tato výrobní chyba byla v obvodech sdílené vyrovnávací

paměti, paměťového řadiče, vnitřní sběrnice nebo řadiče externích sběrnic, nelze vadný kus použít vůbec. Procesor společnosti Cerebras je navržen tak, že může zabírat výrazně větší plochu (více než 50x) křemíkové destičky než jiné velké chipy. Je navržený modulárně tak, aby se vyrovnal s výrobními vadami výrazně lépe než konkurenční architektury, včetně chyb na sběrnic

Neustálé zvyšování nároků počítačových her na flexibilitu grafické pipeline vedlo k náhradě pomocí programovatelných jednotek zvaných *shadery*. S příchodem SW platformy CUDA C společnosti NVIDIA bylo možné tyto jednotky programovat mimo grafický kontext. Skupina Khronos reagovala otevřenými standardy OpenCL a SYCL, též založených na programovacích jazycích C a C++. Další standardy pro GPU výpočty jsou implementovány v rámci grafických API OpenGL, Vulkan a DirectX jako speciální *compute shadery*. Nově se pro webové aplikace zpřístupňují GPGPU výpočty přes dvojici rozhraní: WebGL 2.0 Compute (opuštěno, původně podporováno firmou Intel) a WebGPU, které je novější a nejspíše bude standardizováno W3C [34] (vývoj byl započat společnostmi Mozilla, Google a Apple).

3.1 Programování GPU

Od běžných CPU se GPU fundamentálně liší v přístupu vícevláknového programování. Pro CPU je vlákno (*thread*) procesu poměrně vysoká abstrakce a je spravováno operačním systémem. Také v poměru k množství operační paměti je využíváno relativně málo vláken, které mohou pracovat na větším bloku paměti.

GPU vlákna jsou velice “štíhlá” (lightweight). Organizují se do hierarchických skupin. Napříč skupinami nelze provádět synchronizaci, vlákna jiné skupiny nemusí ani zároveň běžet ani mít vytvořený kontext. Veškerý kontext vlákna v globální RAM musí být předem explicitně definován, jsou však také dostupné hierarchie lokálních pamětí vlákna i skupiny (CUDA nazývá shared memory) pro určitou míru synchronizace. O plánování spuštění a udržování kontextu skupin vláken se stará přímo HW plánovač v kooperaci s ovladačem OS. V rámci skupin nejnižší hierarchie (CUDA je nazývá *block*) jsou vlákna dále dělena na podskupiny, které už zaručeně běží zároveň (CUDA je nazývá *warp*) a přiřazují se na SIMD jednotky. Tento model NVIDIA nazývá *Single Instruction Multiple Thread* (SIMT). Velikost podskupin se může lišit podle dostupného HW, z hlediska programovacího modelu a kompilace není přímo podstatná. Znalost této velikosti však může rozhodovat o vhodné optimalizaci rozdělení výpočtu.

Spuštění celé úlohy sestává v definování hierarchie rozdělení skupin (v CUDA jako *grid* a *block*) a spuštění programu, který všechna vlákna sdílejí, zvaného *kernel*. Tento kernel je psaný z pohledu jednoho vlákna, u klasického případu paralelizace nahrazuje vnější cyklus for. Nejsou zde žádné triky vektorizace kompilátoru známé z programování CPU, veškerý paralelismus je explicitní. Organizace skupin může být vícerozměrná. Každé vlákno pak přímo v HW kontextu má k dispozici svoji pozici v hierarchii a velikost buněk. Není pak třeba u vláken řešit uložení kontextu indexů v globální paměti na rozdíl od CPU řešení.

Pokud je třeba ve výpočtu synchronizovat obecně vlákna napříč skupinami, je třeba výpočet rozdělit na spuštění více podúloh jako jednotlivých kernelů. Nevýhodou je zvýšená režie běhového prostředí (run-time) a ovladače (driver). Je tedy vhodné počet spuštění kernelů minimalizovat. Aby byly některé problémy řešitelné byla také přidána podpora atomických operací v globální paměti, avšak jejich využívání by mělo být omezeno na nezbytně nutné. Atomické operace nemohou sloužit ke vzájemné synchronizaci skupin (bloků), protože není zaručeno, že budou tyto skupiny spuštěny opravdu paralelně.

Při programování mikroprocesorů jsou velmi důležité vzory přístupu do paměti z hlediska *cache* paměti. Pro GPU to platí také stále více, ovšem je zde specifická potřeba využívat tzv. sdružené (*coalesced*) přístupy, kdy vlákna ve stejné skupině (nebo alespoň podskupině) přistupují k paměti přímo za sebou. To umožňuje paměťovému řadiči tyto přístupy sdružit do jednoho přenosu. Jelikož se ke každému přenosu váže latence, je nutno jejich počet minimalizovat pro dosažení vysoké propustnosti. Možnosti paměťových řadičů se neustále zlepšují, kdy zvládají takto sdružovat i přístupy permutované či adresově nezarovnané (*misaligned*).

Z těchto důvodů je preferované uložení dat jako struktura polí (Structure of Arrays, SOA) na rozdíl od skalárního kódu pro CPU, kde se preferuje rozdělení do polí struktur (Array of Structures, AOS), které jsou pro jedno vlákno rozprostřeny v co nejméně buňkách vyrovnávací paměti (*cache lines*). Předchozí tvrzení je zavádějící a pro některé výpočty se na CPU také preferuje uložení jako AOS. Pro 2 a více rozměrná pole se zarovnávají (alignment, padding, pitched) adresy začátků řádků v paměti na určité násobky (podle architektury paměťového řadiče, například na násobky 16). Pak je sdružený přístup zaručen. Jak se možnosti HW zvyšují a objevují se časté vzory, je sdružování přístupů i při určité permutaci v rámci pořadí sousedních vláken vůči pozicím v bloku paměti či zvýšené benevolence offsetu bloku paměti (misaligned). V první generaci CUDA však každý nesdružitelný přístup vedl na 16 oddělených přístupů, což často anulovalo výhody použití GPU pro urychlení daného problému.

Pro SIMD/SIMT je problémem divergence v rámci algoritmu, prakticky v rámci podskupiny vláken, která fyzicky běží zároveň na SIMD jednotkách. Menší problém je, pokud jen některá vlákna mají dělat práci navíc - neexistuje *else* větev, pak ji dělají všechna ovšem některé výsledky se nepoužijí. Pokud větev *else* existuje, může nastat příznivá situace, že pro všechna vlákna je podmínka nepravdivá. Jinak je třeba provést kód této větev se všemi vlákny v podskupině a uložit pouze relevantní výsledky (nazýváno jako *masking*). Právě zde může docházet ke značným ztrátám výkonu.

Často lze divergenci podmínek skrýt přímo do výpočtu (*branchless arithmetic*) a vyhnout se tak instrukcím skoků, když už není možné se výpočtu pro všechna vlákna v podskupině vyhnout. Booleovské hodnoty pravdy a nepravdy se zobrazují na datové typy `int` a `float` jako 1 a 0. Pro předchozí příklad: Tato technika je používaná také pro skalární kód pro CPU pokud četné a nebo špatně předvídatelné větvení snižuje výkon *branch predictoru* a zabraňuje *pipeliningu* instrukcí [38]. Kompilátory (např. GCC a clang) v některých případech zvládnou generovat instrukce podmíněného přesunu, aby předešly použití instrukcí skoku a *branch predictor* je tak úplně vynechán [39].

Součástí vývojového balíku CUDA SDK je knihovna cuRAND, která obsahuje několik paralelních implementací více typů generátorů pseudonáhodných čísel. Každé vlákno programu musí mít vlastní kontext generátoru, alokuje pro něj paměť. Při inicializaci je zadáváno jak semínko (*seed*), tak číslo vlákna. To zjednodušuje inicializaci, protože není třeba generovat semínko pro každé vlákno zvlášť. Jsou dostupná dvě API, více kontroly nabízí tzv. *device API*, kdy se z kernelů řídí inicializace a generování čísel dle potřeby výpočtu.

3.2 Architektura a programování více-jádrových CPU

Současné více jádrové CPU se vyznačují poměrně velkými jádry, které jsou vnitřně paralelní, tzv. superskalární. Instrukce mohou být prováděny v jiném pořadí než jsou v kódu (*out of order execution*) a paralelně na více ALU (aritmeticko-logická jednotka).

K paralelizaci na jednotlivá jádra lze využít procesy a vlákna poskytovaná operačním systémem. V jazyce C++ je od verze C++11 poskytována ve standardní knihovně třída `std::thread`, která abstrahuje použití vláken přenositelně napříč platformami. Spouštění a ukončování vláken vyžaduje určitou režii operačního systému. Pro HPC výpočty bývá vhodné použít tzv. *thread pool*, což je skupina předem spuštěných vláken (ne více než je dostupných výpočetních jader), které si postupně rozebírají úlohy k výpočtu. Jednou z implementací, která takto využívá vlákna ze standardní knihovny `std::thread` je [40] a poskytuje rozhraní ve třídách `BS::thread_pool` a `BS::thread_pool_light`.

3.2.1 Datový paralelismus v C++ a standardizace

Ve standardu C++17 bylo přijaté rozšíření standardní knihovny pro možnost paralelního výpočtu standardních algoritmů, tj. těch definovaných v hlavičkovém souboru `<algorithm>`. V technické specifikaci ISO/IEC TS 19570:2018 [43] jsou deklarovány postupy pro další rozšiřování standardu jazyka C++ o možnosti zápisu paralelních algoritmů i jinak než pomocí standardních algoritmů a vláken poskytovaných operačním systémem. Jedním ze vzniklých návrhů je dokument P1928R8 [44]. Stejně jako pro GPU je možné SYCL využít pro programování více jádrových CPU. U některých CPU (AMD je nazývá APU) lze takto také využít integrované iGPU.

3.2.2 Intel ISPC

Je jednou z možností jak implementovat datově paralelní algoritmy na CPU [45, 46]. Ten realizuje paradigma Single Program Multiple Data (SPMD) pomocí Single Instruction Multiple Data (SIMD). V současné době jsou nejpoužívanější SIMD rozšíření na ISA x86-64 skupiny instrukcí AVX2 a AVX512, na procesorech s ISA ARM je to rozšíření NEON. Kompilátor ISPC zvládá kompilovat pro obě platformy, protože generuje LLVM IR a nechává tak na LLVM finální rozhodnutí o použitých instrukcích.

4 IMPLEMENTACE METOD

Byl modifikován algoritmus HC12 s novou implementací Tabu listu na GPU. Dále byla vyvinuta nová varianta Diferenciální evoluce s více ostrovními populacemi vhodná pro paralelní implementaci na GPU.

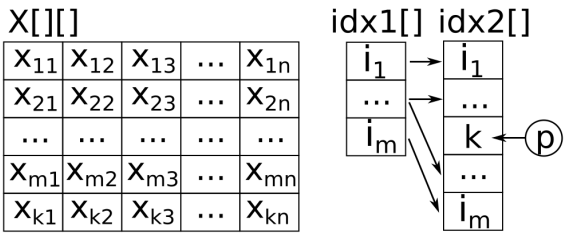
Známý algoritmus HC12 vyvinutý prof. Matouškem byl postupně implementován i s využitím GPU. Jeho třetí verze na GPU ve variantách původního Matlab/C++ řešiče s obecnou fitness funkcí a `hc12qs2_v2` pro řešení QAP problému pomocí překódování *qapswap2* byla rozšířena o seznam zakázaných (již navštívených) řešení, tzv. tabu list. [2]

Binární vyhledávání v seřazeném poli je klasický algoritmus. Zde jsou seřazenými prvky celé vektory a tak při vyhledávání postupně po sloupcích, tj. prvcích vektorů, je třeba ošetřit novou situaci. V případě když se rovnají hodnoty ve sloupci hledané hodnotě, zjišťuje se rozsah řádků se stejnou hodnotou a ty se pak stávají novými mezními řádky pro hledání v dalším sloupci. Pokud však hodnota shodná není, algoritmus se dalším prohledáním sloupce nezdržuje. Podrobný popis je uveden v algoritmu 1.

Pro efektivní přidávání položek tabu listu je seřazení struktury nepřímé. Zvlášť existuje neseřazené pole řádků a pole indexů seřazených řádků. Proces přidání položky ilustruje obrázek 4.

Prototypování implementace probíhalo ve 4 krocích: jednodušší program s NumPy, implementace binárního vyhledávání pro více sloupců v Pythonu nad NumPy poli, ale pouze skalární přístup po jednom prvku, program v C++ pro hledání a přidávání položek tabu listu na CPU s omezením na čisté C ve funkcích pro další krok, výsledná implementace tabu listu s kernely v CUDA C.

Tabu list se rozšiřuje přidáním řádku na konec a pak úpravou pole indexů. Po neúspěšném vyhledání řádku algoritmus již zná pozici pro vložení indexu. Udržují se dvě pole indexů, vždy z jednoho se kopíruje a do druhého se vkládá upravená verze a následně se vymění jejich ukazatele. Může tak probíhat paralelně kopírování indexů s vynecháním místa pro vkládaný prvek.



Obr. 4: Stav paměti po přidání řádku do tabu listu

Algoritmus 1: nalezení řádku v nepřímo seřazeném poli

```
vstup:
X[1 … m, 1 … n] současný obsah tabu listu, m řádků, n sloupců
V[1 … n] hledaný řádek (paralelně pro každé vlákno)
idx1[1 … m] indexy seřazených řádků

inicializace:
   a ⟵ 1 ;   b ⟵ m  ;   f ⟵ -1   ; p ⟵ 1
   funkce Xi(r, s) → X[idx1[r]][s]
```

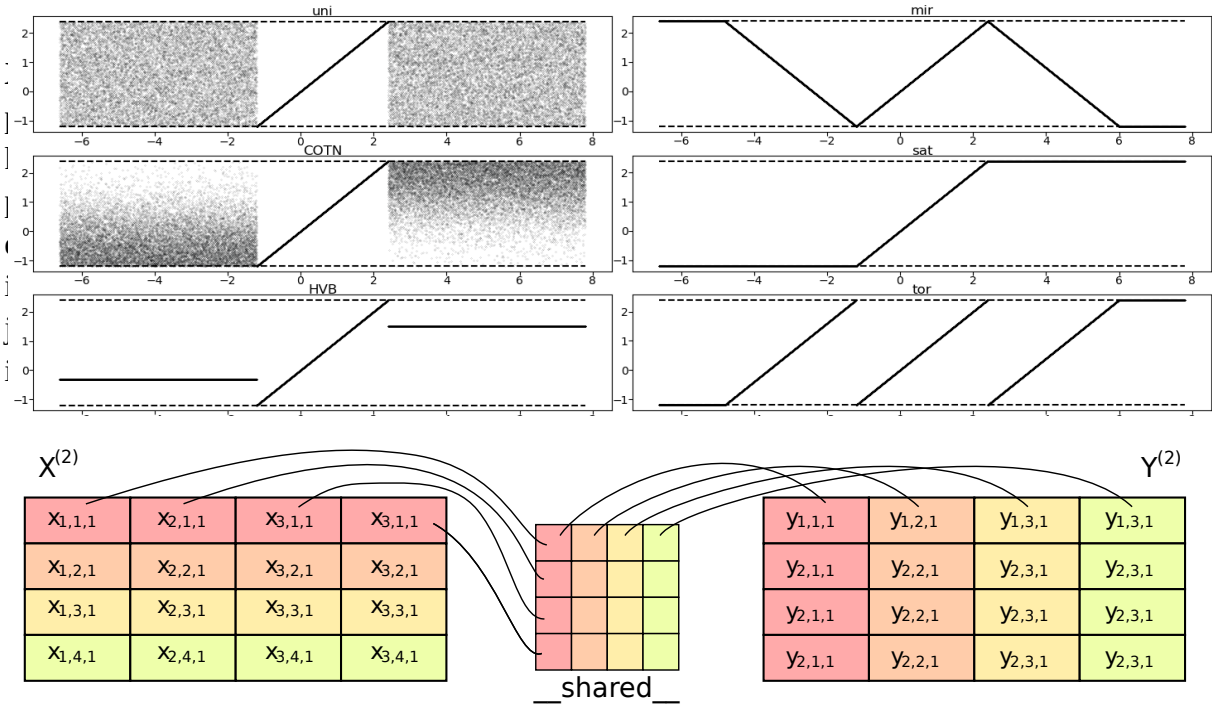
```
pro j ⟵ 1 … n   (cyklus A)
|   pokud Xa > Xb potom
|   └   b ⟵ a
|   pokud V[ j ] < Xi(a, j) potom
|   |   p ⟵ a
|   └   přerušení cyklu A
|   pokud V[ j ] > Xi(b, j) potom
|   |   p ⟵ a
|   └   přerušení cyklu A
|   dokud b - a ≥ 2    (cyklus B)
|   |   s ⟵ a + ⌊ (b - a) / 2) ⌋
|   |   Xsj ⟵ Xi(s, j)
|   |   pokud Xs = V[ j ] potom
|   |   |   na ⟵ a
|   |   |   pro i ⟵ m … a   (cyklus C1) sestupně
|   |   |   |   pokud Xsj = Xi(i, j) potom
|   |   |   |   └ na ⟵ i
|   |   |   |   jinak
|   |   |   └   └   přerušení cyklu C1
|   |   |   a ⟵ na
|   |   |   nb ⟵ b
|   |   |   pro i ⟵ b … m   (cyklus C2)
|   |   |   |   pokud Xs = Xi(i, j) potom
|   |   |   |   └   nb ⟵ i
|   |   |   |   jinak
|   |   |   └   └   přerušení cyklu C2
|   |   |   b ⟵ nb
|   |   └   přerušení cyklu B    byly nalezeny meze pro další sloupec
|   |   jinak pokud Xs > V [ j ] potom
|   |   └   b ⟵ s
|   |   jinak
|   |   └   a ⟵ s
|   |
|   |   pokud       Xi(a, j) = V[ j ]
|   |   a zároveň Xi(b, j) ≠ V[ j ] potom
|   |   └   b ⟵ a
|   |   jinak pokud Xi(a, j) ≠ V[ j ]
|   |   a zároveň Xi(b, j) = V[ j ] potom
|   |   └   a ⟵ b
|   |
|   |   pokud b - a = 1 a zároveň Xi(a, j) < V[j] < Xi(b, j)
└   └   └   p ⟵ a + 1
výstup: (f, p)
```

4.1 Diferenciální evoluce s více ostrovními populacemi

V implementaci je konzistentně používáno značení: číslo populace *p*, počet populací `n_populations`, číslo jedince *m*, počet jedinců v každé populaci `n_members`, číslo dimenze problému *n*, počet dimenzí problému `n_dims`.

Strategie korekce chybných řešení (*Strategy of dealing with infeasible solutions – SDIS*) byly implementovány a verifikovány. Korekci pro interval <–1,2; 2,4> znázorňuje obrázek 5. Uložení v paměti, které pro sdružený přístup pro genetické operátory do matice **X** s prvky

$x_{p,m,n}$, ale selhává pro efektivní transformaci uložení *trial* vektorů v matici **Y** s prvky $y_{p,m,n}$. Matice **Y** musí být uložená tak, aby sdružený přístup byl možný při výpočtu účelové funkce.



Obr. 6: Kopírování X do Y s využitím sdílené paměti bloku vláken

Hlavním programovým rozhraním (API) je šablonová třída `DEMultipop<FP, FPf>` a pomocné typy pro nastavení režimů genetických operátorů, migrace a korekce neplatných řešení (SDIS). Kód odpovídá normě ISO C++17. Šablonový typ `FP` je použitý pro uložení matic X a Y, typ `FPf` je použitý pro uložení výsledku fitness funkce, která je předávána pomocí ukazatele na abstraktní bázovou třídu `FitnessFunctorInterface<FP, FPf>`.

4.1.1 Testovací funkce s reálnými argumenty

Všechny funkce F1 až F12 a CEC04 až CEC10 byly implementovány ve 4 verzích: Matlab (CPU), Python+NumPy (CPU), CUDA C pro HC12 a pro DE (GPU).

4.1.2 Problém kvadratického přiřazení (QAP)

Pro porovnání bylo vyvinuto několik implementací řešiče HC12 pro QAP jako *benchmark*. Pro GPU je s CUDA C mírně upravený řešič, aby bylo srovnání co nejpřímější. Instance problému byla zvolena tho150 z QAPLIB. Důvody jsou: matice jsou symetrické, rozměr není mocnina dvou, matice vzdáleností a toků obsahují poměrně málo nul, problém je obtížný a není ověřena optimálnost řešení. Pro CPU jsou implementovány 4 varianty:

Normal je referenční implementací. Jednovláknový kód, případná vektorizace je ponechána na schopnostech kompilátoru *clang LLVM*.

NormalBS má stejný kód výpočtu jako u varianty *Normal*. Ten je vložený do *lambda* funkce s rozdělením do více úloh. Tyto úlohy jsou předávány k výpočtu skupině předem spuštěných vláken (*thread pool*), využívá `BS::thread_pool_light` [40].

ISPCqap varianta vytváří permutace skalárně, výpočet QAP je paralelizován pro tuto jednu permutaci. Pro paralelizaci na jednotlivá jádra CPU je také použit *thread pool* `BS::thread_pool_light`. Sdružená matice T (`DF`) je transponovaná T^T (`DFt`) a tak je třeba pouze jediná *gather* operace, když je SIMD vektorizována vnitřní smyčka.

ISPCtasks využívá pro paralelizaci na jednotlivá jádra CPU *thread pool* `tasksys.cpp`. Lze tak použít v kódu klíčová slova `task` a `launch`. Pro SIMD je paralelizována v každém *tasku* vnější smyčka.

4.1.3 Problém splnitelnosti Booleovské formule (SAT)

Implementace výpočtu obecného SAT problému – klauzule obsahují různý počet literálů a jejich negací. Klauzule jsou kódovány klasicky jako vektory celých čísel *i*, kde $|i|$ je index literálu $x_{|i|}$ a záporné hodnoty značí negaci literálu $\neg x_{|i|}$. **MCSX** je paralelní pro jednotlivé klauzule a počítá pouze jeden vektor hodnot literálů **x**. **DXAC** je paralelní pro vektory hodnot literálů **x**, ale jednotlivé klauzule se pro každý vektor počítají sekvenčně.

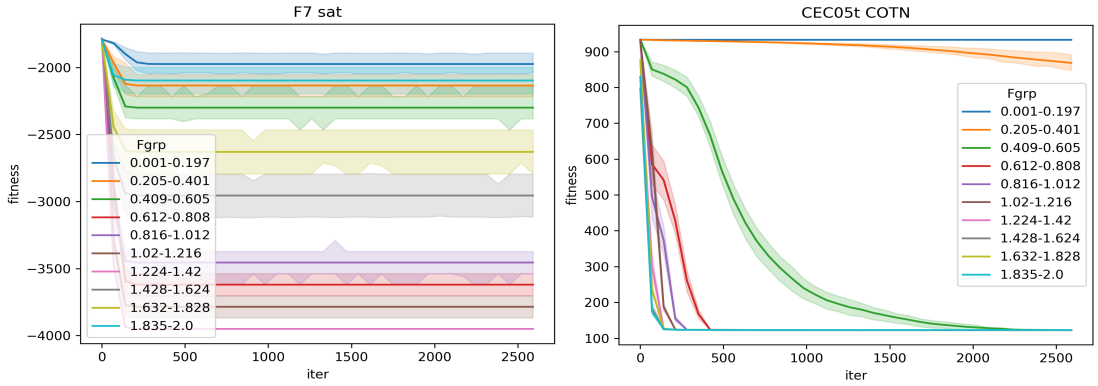
5 EXPERIMENTY

5.1 Diferenciální evoluce s více ostrovními populacemi

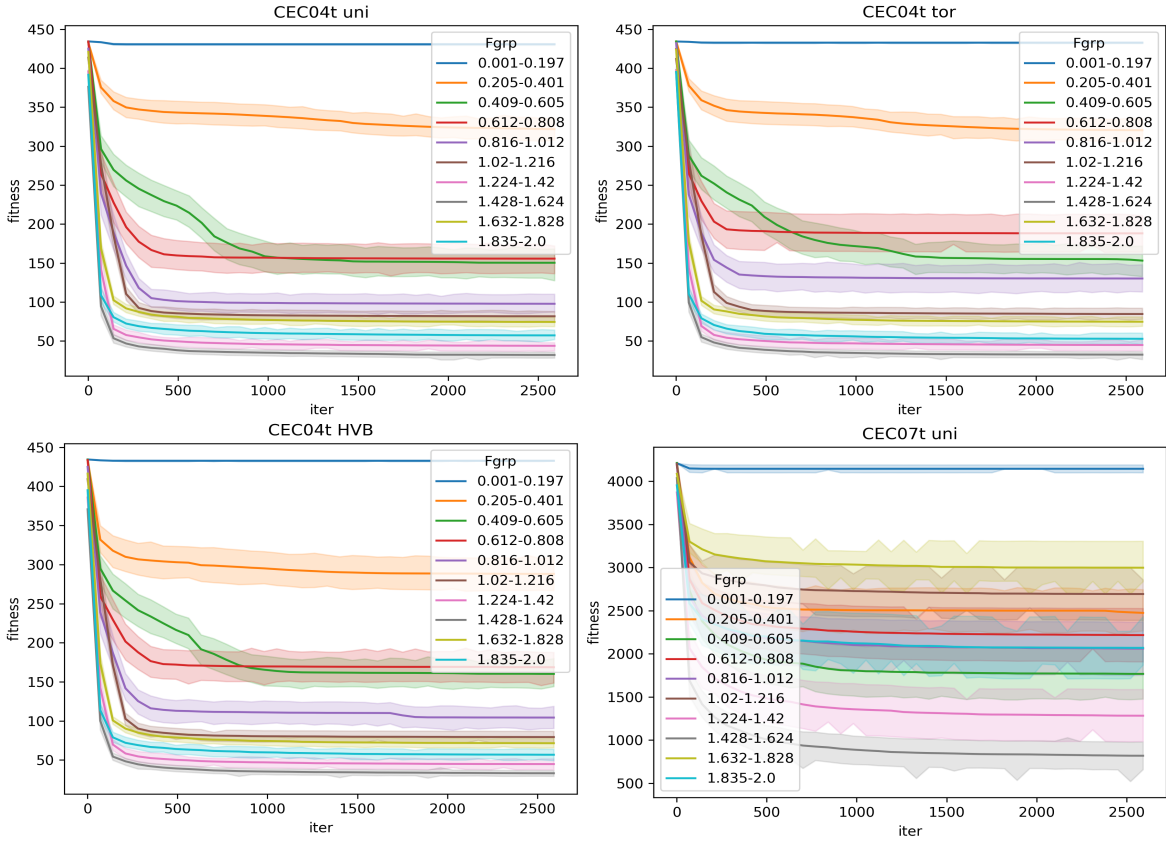
První porovnání se věnuje zkoumání vlivu *SDIS* na konvergenci při určitém koeficientu F_p. Migrace je zakázána (*None*), každá populace *p* je nezávislá. (Výraz pro F_p odpovídá použití funkce `linspace`). Testovací účelové funkce jsou použity CEC04t až CEC10t a F7. Počet populací N_p.

$$\begin{gathered} SDIS \in \{sat, mir, tor, HVB, uni, COTN\} \\ F_p \subset (0{,}001; 2) \rightarrow F_p = 0.001 + \frac{p(2-0.001)}{N_p - 1} \\ CR = 0{,}7 \;\; ; \;\; N_p = 256 \end{gathered} \tag{8}$$

Průběhy hodnot účelové funkce (*fitness*) byly seskupeny pro 10 intervalů hodnot F_p. Pro skupinu zobrazují grafy průměrnou hodnotu a interval spolehlivosti 50%. Pro všechny funkce CEC*XX*t a všechny varianty *SDIS* byl nejúspěšnější (nebo stejný) interval koeficientů F_p = <1,428; 1,828>. Obrázek ukazuje, že naopak u funkce F7 byl nejúspěšnější interval koeficientů F_p = <1,224; 1,42>, ale na různá nastavení *SDIS* také nebyla pozorována žádná podstatná reakce. Také zobrazuje, že funkce CEC05t je k volbě *F* koeficientů tolerantnější. Obrázek 8 ilustruje, že u funkce CEC04t k největším rozdílům vedly *SDIS* = {*uni*, *tor*, *HVB*} pro některé intervaly F_p. Naopak pro funkci CEC07t neměla *SDIS* žádný zřejmý vliv a grafy vypadají stejně jako u *SDIS*=*uni*. Autor považuje za možné, že sada testovacích funkcí nevykazuje velkou citlivost na volbu *SDIS*, protože se ani v jedné dimenzi globální extrém nenachází blízko hranice intervalu.

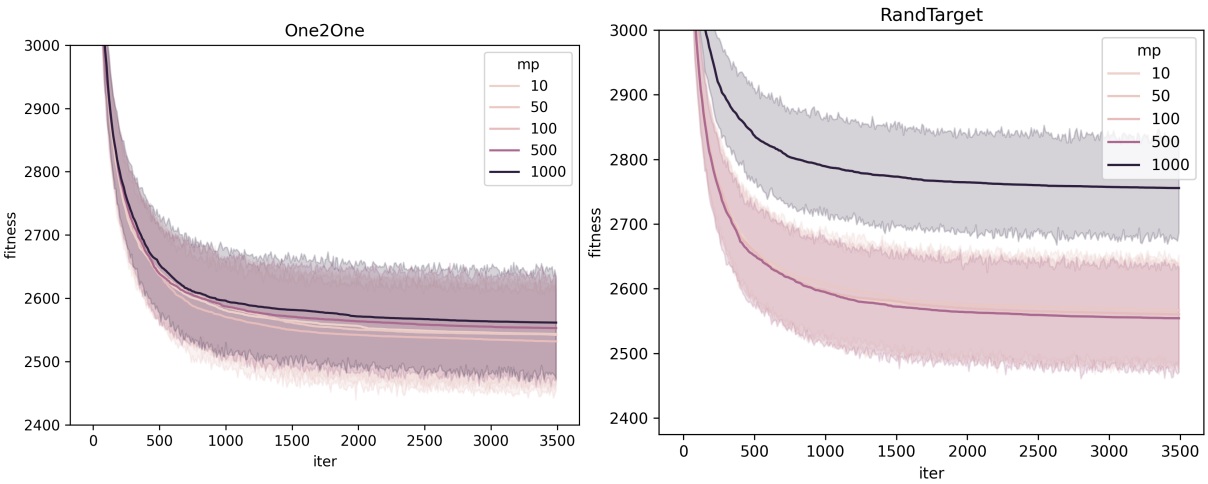


Obr. 7: Průběhy fitness u funkcí F7 a CEC05t pro různé intervaly F koeficientů



Obr. 8: Výběr průběhů fitness CEC04t a CEC07t pro různé skupiny koeficientů F a SDIS

Následující druhý experiment zkoumal vliv periody migrace na hodnotu fitness pro různé migrační strategie. Parametry DE jsou téměř stejné jako u předchozího experimentu, SDIS je volena COTN a fitness funkce CEC07t. Poměrně překvapivá je bifurkace u strategie *RandTarget*.

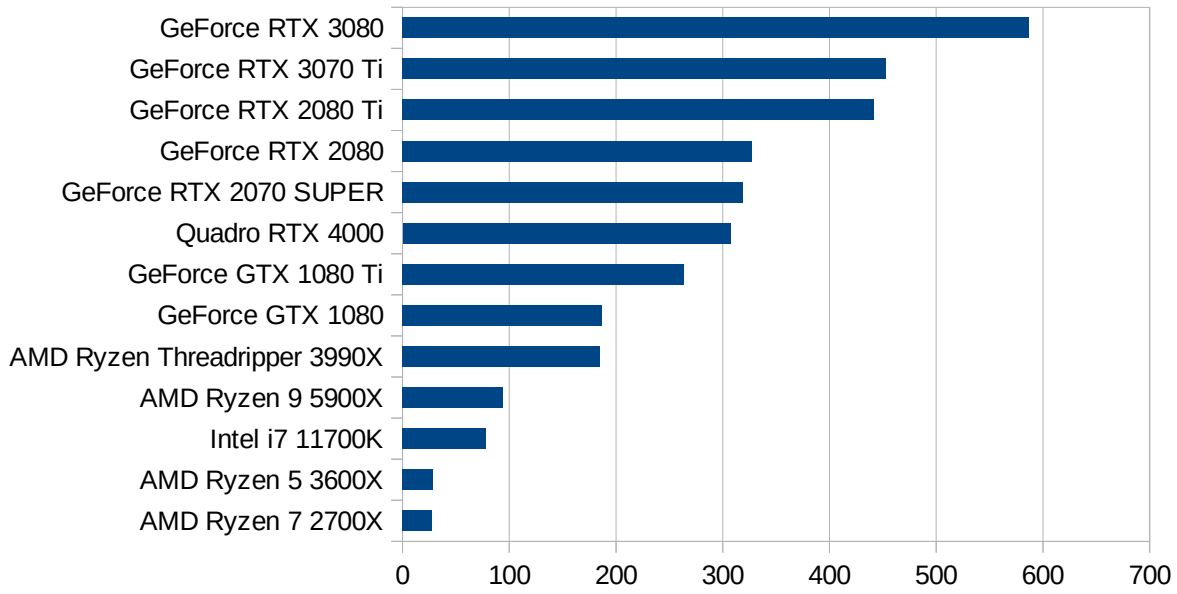


Obr. 9: Vliv periody migrace na fitness u strategií One to one a RandTarget(CEC07t)

5.2 Porovnání implementací HC12 pro QAP na CPU a GPU

Varianta *ISPCqap* byla vyvíjena dodatečně, protože hlavní stroj použitý pro vývoj byl s procesorem AMD Ryzen 7 2700X, na kterém varianta *ISPCtasks* byla pomalejší než základní varianty *Normal(BS)* v rozporu s hypotézou autora. Další testy však prokázaly, že při použití procesorů s novější architekturou je naopak *ISPCtasks* výhodnější. Benchmark použil QAP úlohu *tho150* a HC12 pro 15 výměn (swapů), tedy 30 parametrů a 28 921 řádků.

Přestože se podařilo pomocí ISPC dosáhnout výhodného urychlení algoritmu na CPU, v porovnání s CUDA implementací vycházejí GPU výhodněji (obrázek 10). Použitá implementace pro benchmark v zájmu objektivnosti porovnání používá shodné datové typy.

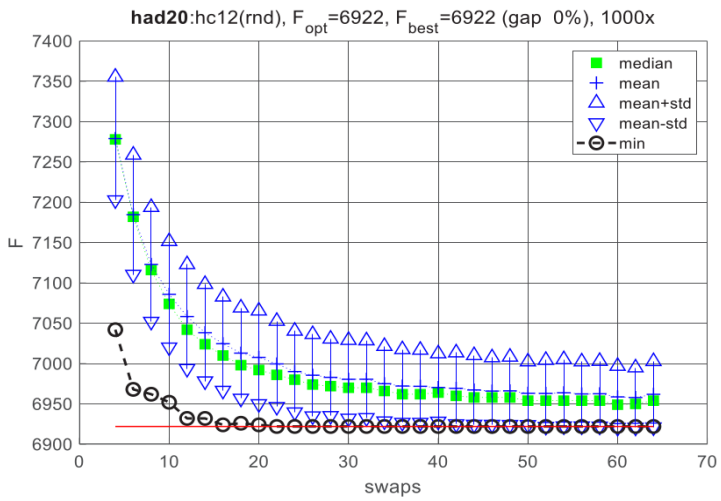


Obr. 10: Porovnání CPU a GPU v benchmarku HC12qapswap2

Ani 64 jádrový procesor AMD Ryzen Threadripper 3990X se základní frekvencí 2,9 GHz (4,3 GHz *boost*) a tepelným výkonovým limitem TDP 280W nestačil na starší grafickou kartu MSI GeForce GTX 1080 ARMOR 8G OC (chip NVIDIA GP104, generace Pascal) s frekvencí 1,657 GHz (1,797GHz *boost*) a tepelným výkonovým limitem TDP 180W. Chip GP104 má 7,2 miliard tranzistorů, byl vyráběn 16nm procesem u TSMC a má pouze 3968 KB *cache* se dvěma úrovněmi. Naopak zmíněný Threadripper je soustava 9 *chipů* (či *chippletů*), 8 z nich vyráběných u TSMC výrazně novějším 7nm procesem a I/O *chip* vyráběný 14nm procesem GlobalFoundries. Celkově procesor obsahuje 30,4 miliard tranzistorů. Velká část z nich je však využita na obří 292MB *cache* se třemi úrovněmi, řídící logiku a spekulativní vyhodnocování instrukcí.

5.3 HC12qs2 na QAPLIB

Na osmi menších úlohách z QAPLIBu, které mají známé optimální řešení, bylo testováno vhodné množství výměn (swapů) a poměr úspěšnost k množství restartů algoritmu. Obrázek 11 ilustruje jak zvyšující se počet výměn vede k lepšímu průměrnému řešení a zvyšuje se pravděpodobnost dosáhnutí optima. Byla použita grafická výpočetní karta NVIDIA RTX 2080 (8 GB). Výsledky byly prezentovány na konferenci GECCO [2].



Obr. 11: HC12 QAPLIB had20: Závislost fitness na počtu výměn

Další aplikací řešiče bylo porovnání jeho řešení inicializovaných náhodně a řešení, kterých dosáhl pomocí inicializace permutacemi získanými projekcí odhadů spodní hranice. Tabulka 1 ilustruje kolik procent chybělo do nejlepšího známého řešení u vybraných úloh. Použita byla výpočetní grafická karta NVIDIA RTX 2080 (8GB). Výsledky byly publikovány v International Journal of Industrial Engineering Computations [1].

Tabulka 1: Procentuální vzdálenost od nejlepšího známého řešení pro vybrané QAP.

QAP	`chr12a`	`chr18b`	`nug30`	`sko100a`	`tho150`	`tai100a`	`wil100`
GAP %	0	0	1,54	2,64	3,8	3,84	1,50

5.4 Rozdíly přesnosti výpočtu funkcí spojité optimalizace

Pro porovnání bylo v 10 dimenzích vygenerováno 10 000 bodů a byla spočítána funkční hodnota pomocí Variable Precision Arithmetic (`vpa` funkce) z Matlab Symbolic Math Toolbox s přesností na 50 desetinných míst (v desítkové soustavě). Porovnávané implementace využívaly datový typ IEEE-754 double precision, který je přesný maximálně na 16 desetinných míst (v desítkové soustavě).

Výsledky byly srovnatelné, průměrná odchylka byla vždy menší než jeden řád.

6 ZÁVĚR

Téměř celá práce se zabývá převážně využitím GPGPU společnosti NVIDIA pomocí softwarové platformy CUDA a jazyka C++. Kapitola 3 se věnuje přehledu hardwarových systémů a prostředků používaných v oblasti vysoce náročných výpočtů (HPC). Kromě více jádrových CPU a GPGPU jsou uváděny osvědčená řešení jako superpočítače, clustery, grid, cloud, FPGA a ASIC. Také je zmíněna novinka Cerebras Wafer-Scale Engine.

Jedním z hlavních cílů práce bylo vytvořit a demonstrovat plně funkční, komplexní a datově paralelní implementace vybraných a pro daný účel vhodných metaheuristických algoritmů a úloh spojité a kombinatorické optimalizace. Metaheuristiky prokazatelně umožňují nacházet dostatečně dobrá řešení v aplikacích inženýrské optimalizace. Je třeba v praktické aplikaci přistoupit k tzv. inženýrské optimalizaci a najít dostatečně vhodná řešení v rozumném čase (feasible solution in feasible time) a za přijatelnou cenu (feasible solution at acceptable cost).

Prvním implementovaným metaheuristickým algoritmem je HC12 rozšířený o novou implementaci seznamu zakázaných řešení (Tabu list), která je vhodná pro GPU. Obecný řešič spouštěný z Matlabu umožňuje uživateli dodat vlastní účelovou funkci pro GPU (CUDA), a také jí před spuštěním samotného algoritmu připravit obecně jakákoliv data. Je také implementován a popsán specializovaný řešič *hc12qs2_v2*, který efektivně kombinuje HC12 algoritmus přímo s řešením QAP problému pomocí překódování *qapswap2*. Také je část implementace a jejího popisu věnována praktickému propojování software a tvorbě knihoven a *wrapperů* pro různé programovací jazyky a SW balíky.

Druhou implementovanou metaheuristikou je zcela nová varianta diferenciální evoluce (DE) s více ostrovními populacemi vhodná právě pro GPU jak z pohledu genetických operátorů (mutace, křížení, selekce) tak pro výpočet účelové funkce. Navržená implementace DE vhodně využívá sdílenou paměť dostupnou na Streaming Multiprocesorech v GPU NVIDIA. Při vývoji byl kladen důraz na podporu více variant algoritmu a různých nastavení (feature complete). Návrh je realizován tak, aby bylo poměrně snadné dále rozšiřovat funkcionalitu. Uživatelem definovaná fitness funkce se předává jako objekt s rozhraním definovaným pomocí šablonové abstraktní bázové třídy a může si tak udržovat jakýkoliv kontext svých dat. Ve veřejném API i interně implementace využívá C++ šablony (*templates*). Kapitola 5.1 prezentuje dva experimenty. Prvním je zkoumání vliv SDIS a koeficientů F na fitness na nezávislých populacích pro různé funkce. Druhý experiment zkoumá vliv periody migrace na fitness pro čtyři migrační strategie u transformované funkce CEC07t.

Bylo implementováno šestnáct známých testovacích účelových funkcí spojité globální optimalizace z literatury ve třech různých programovacích prostředích: Matlab, Python+NumPy, CUDA C. Verze v CUDA C jsou dvě, první pro spolupráci s HC12 řešičem a druhá pro řešič Diferenciální evoluce s více ostrovními populacemi. K sedmi funkcím byly také přidány další modifikace v podobě transformací, aby se lépe porovnávala robustnost metod, tak jak byly definovány v literatuře. Bylo verifikováno jak moc se výsledky jednotlivých implementací liší vůči přesnější metodě výpočtu.

Klasický kombinatorický optimalizační problém kvadratického přiřazení (QAP) byl s HC12 rigorózně testován a výsledky experimentů v kapitole 5.3 byly publikovány na konferenci a v odborném časopise.

QAP v kombinaci s HC12 byl také využit pro srovnání GPU (CUDA) a CPU (ISPC) implementací. Přestože se podařilo zefektivnit využití SIMD jednotek pomocí explicitního popisu paralelnosti výpočtu v jazyce ISPC na CPU a urychlit výpočty oproti optimalizujícímu C++ kompilátoru LLVM/clang, ukázalo se v experimentech popsaných v kapitole 5.2, že pro HC12 a QAP je architektura GPU významně výhodnější.

Problém splnitelnosti Booleovské formule (SAT) byl analyzován z pohledu akcelerace výpočtu paralelního pro více klauzulí při jediném ohodnocení a pak paralelního pro vícero ohodnocení. Verifikovaná implementace minimalizuje velikost použitých datových typů a maximálně využívá bitových operací pro efektivní určení platnosti formule a počtu platných klauzulí.

V oblasti spojité optimalizace by se autor chtěl dále věnovat využití své varianty diferenciální evoluce při návrhu parametrů modelů pro simulaci a řízení komplexních dynamických systémů popsaných soustavou diferenciálních rovnic a implementací s tím souvisejících paralelních implementací řešičů. Autor by se rád v budoucnu také soustředil na úpravu metaheuristiky GAHC zobecněním schématu HC12 mutace.

Dalším cílem autora pokračovat ve výzkumu v rámci tohoto velmi dynamického oboru aplikace metod umělé inteligence a vysoce náročných výpočtů (HPC).

7 PUBLIKACE AUTORA

[1] MATOUSEK, Radomil, Ladislav DOBROVSKY a Jakub KUDELA. How to start a heuristic? Utilizing lower bounds for solving the quadratic assignment problem. International Journal of Industrial Engineering Computations [online]. 2022, 13(2), 151-164. ISSN 1923-2934. Dostupné z: doi:10.5267/j.ijiec.2021.12.003

[2] MATOUSEK, Radomil, Ladislav DOBROVSKY a Jakub KUDELA. The Quadratic Assignment Problem: Metaheuristic Optimization Using HC12 Algorithm. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: ACM, 2019, 2019-07-13, s. 153-154. ISBN 978-1-4503-6748-6. Dostupné z: doi:10.1145/3319619.3322088

[3] MATOUŠEK, Radomil, Tomáš HŮLKA, Ladislav DOBROVSKÝ a Jakub KŮDELA. Sum Epsilon-Tube Error Fitness Function Design for GP Symbolic Regression: Preliminary Study. In: *International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*. Athens, Greece: IEEE, 2019, 2019, s. 78-83. ISBN 978-1-7281-3572-4. Dostupné z: doi:10.1109/ICCAIRO47923.2019.00021

[4] HŮLKA, Tomáš, Radomil MATOUŠEK, Ladislav DOBROVSKÝ, Monika DOSOUDILOVÁ a Lars NOLLE. Optimization of Snake-like Robot Locomotion Using GA: Serpenoid Design. *MENDEL*. 2020, 26(1), 1-6. ISSN 2571-3701. Dostupné z: doi:10.13164/mendel.2020.1.001

[5] MIŠKAŘÍK, Kamil, Radomil MATOUŠEK, Ladislav DOBROVSKÝ a George HANNA. About controller design by means of grammatical evolution and bees algorithm. In: *MENDEL 2015: 21 st International Conference on Soft Computing*. Brno, 2015, s. 65-70. ISSN 1803-3814.

8 LITERATURA

[6] MOLINA, Daniel, Javier POYATOS, Javier Del SER, Salvador GARCÍA, Amir HUSSAIN a Francisco HERRERA, 2020. Comprehensive Taxonomies of Nature- and Bio-inspired Optimization: Inspiration Versus Algorithmic Behavior, Critical Analysis Recommendations. Cognitive Computation [online]. 12(5), 897-939 [cit. 2024-02-07]. ISSN 1866-9956. Dostupné z: doi:10.1007/s12559-020-09730-8

[7] MATOUSEK, Radomil. HC12: Highly Scalable Optimisation Algorithm. GONZÁLEZ, Juan R., David Alejandro PELTA, Carlos CRUZ, Germán TERRAZAS a Natalio KRASNOGOR, ed. *Nature Inspired Cooperative Strategies for Optimization* (NICSO 2010) [online]. 1. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, s. 177-183 [cit. 2022-01-21]. Studies in Computational Intelligence. ISBN 978-3-642-12537-9. Dostupné z: doi:10.1007/978-3-642-12538-6_15

[8] MATOUSEK, Radomil a Eva ZAMPACHOVA. Promising GAHC and HC12 algorithms in global optimization tasks. *Optimization Methods and Software* [online]. 2011, 26(3), 405-419 [cit. 2021-06-10]. ISSN 1055-6788. Dostupné z: doi:10.1080/10556788.2011.556826

[9] MATOUSEK, Radomil a Petr MINAR. Stabilization of Chaotic Logistic Equation Using HC12 and Grammatical Evolution. *Nostradamus 2013: Prediction, Modeling and Analysis of Complex Systems*. 1. Heidelberg: Springer International Publishing, 2013, 2013, s. 137-146. *Advances in Intelligent Systems and Computing*. ISBN 978-3-319-00541-6. ISSN 2194-5357. Dostupné z: doi:10.1007/978-3-319-00542-3_14

[10] MATOUSEK, Radomil. HC12 Implemented Using the CUDA Platform. In: *APPLIED COMPUTER SCIENCE: International Conference on Applied Computer Science (ACS)*. Malta: WSEAS, 2010, s. 649-652. ISBN 978-960-474-225-7. ISSN 1792-4863. Dostupné z: http://www.wseas.us/e-library/conferences/2010/Malta/ACS/ACS-106.pdf

[11] GOLDBERG, David Edward. *Genetic algorithms in search, optimization, and machine learning*. Boston: Addison-Wesley. ISBN 978-0201157673.

[12] GHOSH, Ashish, Shigeyoshi TSUTSUI a Hideo TANAKA. Individual aging in genetic algorithms. In: *1996 Australian New Zealand Conference on Intelligent Information Systems. Proceedings. ANZIIS 96* [online]. Adelaide, SA, Australia: IEEE, s. 276-279 [cit. 2021-10-15]. ISBN 0-7803-3667-4. Dostupné z: doi:10.1109/ANZIIS.1996.573957

[13] MATOUSEK, Radomil. GAHC: Hybrid Genetic Algorithm. AO, Sio-Iong, Burghard RIEGER a Su-Shing CHEN, ed. *Advances in Computational Algorithms and Data Analysis* [online]. Dordrecht: Springer Netherlands, 2009, s. 549-562 [cit. 2021-06-10]. Lecture Notes in Electrical Engineering. ISBN 978-1-4020-8918-3. Dostupné z: doi:10.1007/978-1-4020-8919-0_38

[14] STORN, Rainer a Kenneth PRICE. Differential Evolution - A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces: Technical Report TR-95-012 [online]. Berkeley, CA: *International Computer Science Institute*, 1995 [cit. 2021-08-10]. Dostupné z: https://cse.engineering.nyu.edu/~mleung/CS909/s04/Storn95-012.pdf

[15] STORN, Rainer a Kenneth PRICE. Differential Evolution: A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization* [online]. 1997, 11(4), 341-359 [cit. 2021-08-10]. ISSN 0925-5001. Dostupné z: doi:10.1023/A:1008202821328

[16] STORN, R. Differential evolution design of an IIR-filter. In: *Proceedings of IEEE International Conference on Evolutionary Computation* [online]. Nagoya, Japan: IEEE, 1996, s. 268-273 [cit. 2021-08-10]. ISBN 0-7803-2902-3. Dostupné z: doi:10.1109/ICEC.1996.542373

[17] STORN, Rainer a Kenneth PRICE. Minimizing the real functions of the ICEC'96 contest by differential evolution. In: P*roceedings of IEEE International Conference on Evolutionary Computation* [online]. Nagoya, Japan: IEEE, 1996, s. 842-844 [cit. 2021-08-10]. ISBN 0-7803-2902-3. Dostupné z: doi:10.1109/ICEC.1996.542711

[18] TAN, Zhiping a Kangshun LI, 2021. Differential evolution with mixed mutation strategy based on deep reinforcement learning. Applied Soft Computing [online]. 111 [cit. 2024-01-14]. ISSN 15684946. Dostupné z: doi:10.1016/j.asoc.2021.107678

[19] KONONOVA, Anna V., Diederick VERMETTEN, Fabio CARAFFINI, Madalina-A. MITRAN a Daniela ZAHARIE, 2023. The Importance of Being Constrained: Dealing with Infeasible Solutions in Differential Evolution and Beyond. *Evolutionary Computation* [online]. 2023-11-29, 1-46 [cit. 2024-01-15]. ISSN 1530-9304. Dostupné z: doi:10.1162/evco_a_00333

[20] KUSHIDA, Jun-ichi, Kazuhisa OBA, Akira HARA a Tetsuyuki TAKAHAMA. Solving quadratic assignment problems by differential evolution. In: *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems* [online]. IEEE, 2012, 2012, s. 639-644 [cit. 2021-08-10]. ISBN 978-1-4673-2743-5. Dostupné z: doi:10.1109/SCIS-ISIS.2012.6505170

[21] DE P. VERONESE, Lucas a Renato A. KROHLING, 2010. Differential evolution algorithm on the GPU with C-CUDA. In: *IEEE Congress on Evolutionary Computation* [online]. IEEE, s. 1-7 [cit. 2023-11-01]. ISBN 978-1-4244-6909-3. Dostupné z: doi:10.1109/CEC.2010.558621

[22] QIN, A. K., Federico RAIMONDO, Florence FORBES a Yew Soon ONG, 2012. An improved CUDA-based implementation of differential evolution on GPU. In: *Proceedings of the 14th annual conference on Genetic and evolutionary computation* [online]. New York, NY, USA: ACM, 2012-07-07, s. 991-998 [cit. 2023-11-01]. ISBN 9781450311779. Dostupné z: doi:10.1145/2330163.2330301

[23] PIOTROWSKI, Adam P., 2017. Review of Differential Evolution population size. *Swarm and Evolutionary Computation* [online]. 32, 1-24 [cit. 2023-11-01]. ISSN 22106502. Dostupné z: doi:10.1016/j.swevo.2016.05.003

[24] CHARILOGIS, Vasileios a Ioannis G. TSOULOS, 2023. A Parallel Implementation of the Differential Evolution Method. Analytics [online]. 2(1), 17-30 [cit. 2022-11-01]. Dostupné z: doi:10.3390/analytics2010002

[25] SKAKOVSKI, Aleksander a Piotr JĘDRZEJOWICZ, 2019. An island-based differential evolution algorithm with the multi-size populations. *Expert Systems with Applications* [online]. (vol. 126), 308-320 [cit. 2023-10-10]. ISSN 09574174. Dostupné z: doi:10.1016/j.eswa.2019.02.027

[26] POHLHEIM, Hartmut. GEATbx: The Genetic and Evolutionary Algorithm Toolbox for Matlab [online]. 2007-03 [cit. 2021-11-08]. Dostupné z: http://www.geatbx.com/

[27] PRICE, KV, NH AWAD, MZ ALI a PN SUGANTHAN. Problem definitions and evaluation criteria for the 100-digit challenge special session and competition on single objective numerical optimization. Technical Report. Nanyang Technological University, 2018.

[28] ABDEL-BASSET, Mohamed, Gunasekaran MANOGARAN, Heba RASHAD a Abdel Nasser H. ZAIED. A comprehensive review of quadratic assignment problem: variants, hybrids and applications. Journal of Ambient Intelligence and Humanized Computing [online]. 20 June 2018, 2018 [cit. 2021-08-10]. ISSN 1868-5137. Dostupné z: doi:10.1007/s12652-018-0917-x

[29] ÇELA, Eranda. The Quadratic Assignment Problem: Theory and Algorithms. 1. Springer Science & Business Media, 2013. ISBN 978-1-4419-4786-4. eISBN 978-1-4757-2787. Dostupné z: doi:10.1007/978-1-4757-2787-6

[30] BURKARD, Rainer E, Stefan E KARISCH a Franz RENDL. QAPLIB–a quadratic assignment problem library. *Journal of Global optimization*. Springer, 1997, 10(4), 391-403. ISSN 1573-2916. Dostupné z: doi:10.1023/A:1008293323270, (QAPLIB stránka přesunuta na: https://coral.ise.lehigh.edu/data-sets/qaplib/)

[31] HOOS, Holger H a Thomas STÜTZLE. SATLIB - The Satisfiability Library [online]. Canada: Computer Science Department of the University of British Columbia in Vancouver, 2000 [cit. 2021-11-02]. Dostupné z: http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

[32] SHAH, Agam. Proposed RISC-V vector instructions crank up computing power on small devices: When you need to do audio, voice or image processing at the network edge or on a battery budget. *The Register* [online]. 8 Oct 2021 [cit. 2021-11-10]. Dostupné z: https://www.theregister.com/2021/10/08/riscv_vector_instructions/

[33] CARBONNEAUX, Quentin. X86 Instruction Set Reference: Repeat String Operation Prefix. *c9x.me* [online]. [cit. 2021-11-11]. Dostupné z: https://c9x.me/x86/html/file_module_x86_id_279.html

[34] NINOMIYA, Kai, Brandon JONES a Jim BLANDY, 2024. *WebGPU: W3C Working Draft* [online]. [cit. 2024-02-06]. Dostupné z: https://www.w3.org/TR/webgpu/

[35] KILGARIFF, Emmett, Henry MORETON, Nick STAM a Brandon BELL. NVIDIA Turing Architecture In-Depth. DEVELOPER BLOG [online]. NVIDIA, 2018, Sep 14, 2018 [cit. 2021-10-15]. Dostupné z: https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth/

[36] WONG, Adrian. AMD CDNA Architecture: Tech Highlights!. TechARP [online]. [cit. 2021-10-08].
Dostupné z: https://www.techarp.com/computer/amd-cdna-architecture/

[37] CUDA Toolkit Documentation: CUDA C++ Programming Guide [online]. [cit. 2021-10-05]. Dostupné z: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[38] UZUN, Alp Bintuğ. Branchless programming. GitHub: alpbintug [online]. Jul 24, 2020 [cit. 2021-10-15]. Dostupné z: https://github.com/alpbintug/Branchless-programming

[39] DOBROVSKÝ, Ladislav. Clamp with if statements generate branchless x86-64 assembly. *CompilerExplorer* [online]. [cit. 2021-10-15].
Dostupné z: https://godbolt.org/z/MGf3Gsv93

[40] SHOSHANY, Barak, 2021. A C++17 Thread Pool for High-Performance Scientific Computing. *ArXiv e-prints* [online]. Dostupné z: doi:10.48550/arXiv.2105.00613

[41] CHAKRABARTI, Gautam, Vinod GROVER, Bastiaan AARTS, et al., 2012. CUDA: Compiling and optimizing for a GPU platform. *Procedia Computer Science* [online]. 9, 1910-1919 [cit. 2024-02-03]. ISSN 18770509.
Dostupné z: doi:10.1016/j.procs.2012.04.209

[42] KRETZ, Matthias. s*td::experimental::simd: portable, zero-overhead C++ types for explicitly data-parallel programming* [online]. [cit. 2024-02-05].
Dostupné z: https://github.com/VcDevel/std-simd

[43] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2018. *ISO/IEC TS 19570:2018: Programming Languages - Technical Specification for C++ Extensions for Parallelism*. Geneva.

[44] KRETZ, Matthias, 2023. *std::simd: merge data-parallel types from the Parallelism TS 2* [online]. [cit. 2024-02-01].
Dostupné z: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p1928r8.pdf

[45] INTEL. *Intel® Implicit SPMD Program Compiler: An open-source compiler for high-performance SIMD programming on the CPU and GPU* [online]. [cit. 2024-02-05]. Dostupné z: https://ispc.github.io

[46] DU BOIS, Marissa, Pete BRUBAKER a Dominic MILANO. *Single Instruction Multiple Data Made Easy with Intel® Implicit SPMD Program Compiler* [online]. [cit. 2024-02-05]. Dostupné z:
https://www.intel.com/content/www/us/en/developer/articles/technical/simd-made-easy-with-intel-ispc.html

[47] RIGO, Armin a Maciej FIJALKOWSKI. *CFFI documentation* [online]. [cit. 2024-02-01]. Dostupné z: https://cffi.readthedocs.io/

[48] WENZEL, Jakob. *Pybind11 Documentation* [online]. Jul 17, 2023 [cit. 2024-02-01]. Dostupné z: https://pybind11.readthedocs.io/_/downloads/en/stable/pdf/

9 CURRICULUM VITAE

Osobní údaje

Jméno a přijmení: Ladislav Dobrovský
Datum narození: 27. 12. 1987
Státní příslušnost: česká
Kontakt: l.dobrovsky@seznam.cz

Vzdělání

PhD. VUT v Brně, Fakulta strojního inženýrství 2018 – 2024
Disertační práce: GPU akcelerované metaheuristiky ve vybraných úlohách globální a kombinatorické optimalizace

Ing. VUT v Brně, Fakulta strojního inženýrství 2016 – 2017
Program strojní inženýrství, obor aplikovaná informatika a řízení
Diplomová práce: Automatické čtení SPZ s využitím technik hlubokého učení

Bc. VUT v Brně, Fakulta informačních technologií 2007 – 2010
Bakalářská práce: Synchronizace a řízení pohonů s využitím sběrnice CAN

- Střední odborná škola Strážnice, obor Technické lyceum, 2003 – 2007

Pedagogická praxe

1IN (Informatika), cvičení, 2018 až 2023
VSC (Neuronové sítě a *název se měnil*), cvičení, 2019 až 2024
VPD (Programování v Pythonu – Data Science), cvičení, 2021 až 2024
VDS/VDC (Databázové systémy), cvičení, 2019 až 2024
FSI (Simulace dynamických systémů), cvičení, 2019 až 2023

Vedení diplomových prací

Hlasové ovládání průmyslových a medicínských zařízení v rušných prostředích, Vymětalíková Lucie, Bc., 2023
Webová aplikace pro plánování rozmístění sběrných míst odpadu, Vidlička Adam, Bc., 2023
Webová aplikace pro vyhodnocování invest. záměrů v odpadovém hospodářství, Červenka Roman, Bc. 2023
Optimalizace rychlosti výpočtu knihovny PetNetSim, Dražka Vojtěch, Bc., 2022
Webová aplikace pro plánování svozu frakcí odpadu, Kubowský Jiří, Bc., 2022
Konfigurátor rozvodny velmi vysokého napětí, Pokorný Patrik, Bc., 2022

Vedení bakalářských prací:
Přestavba průmyslového počítače a vývoj webové aplikace, Rusnák Filip, 2023
Přestavba průmyslového počítače pro aplikaci kiosku, Bužga Petr, 2023
Simulace kolapsu dopravní sítě s Petriho sítěmi, Ondřej Dofek, 2023
Neeukleidovské geometrie a počítačové hry, Jůda Štěpán, 2022
Simulace a řízení silničního provozu pomocí Petriho sítí, Zbranek Matyáš, 2022
Využití WebAssembly v průmyslu 4.0, Chvátal Petr, 2022
Propojení CNC laserové vyřezávačky s ERP informačním systémem, Černocký Libor, 2021
Aplikace hledání cesty v počítačové hře, Tihlařík Miroslav, 2020
Aplikace smíšené reality v průmyslu 4.0 a robotice, Jedovnický Michal, 2019

Ocenění

Cena rektora (individuální cena za spolupráci v týmu OpenTube), VUT v Brně, 2020
Stříbrná medaile (týmová cena, také OpenTube), VUT v Brně, 2020

Pracovní zkušenosti

VUT v Brně, Fakulta strojního inženýrství, Programátor, od 03/2011 dosud.
REMAK a. s., Programátor, od 05/2017 do 10/2018.
Promoteri.eu, Programátor a administrátor, od 05/2015 do 12/2019.