FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Samostatná práce

KIV/PPR

Patrik Harag

harag@students.zcu.cz

A18N0084P

20. listopadu 2019

Kapitola 1

Zadání

Standardní zadání Implementujte buď stávající, nebo si navrhněte vlastní, evoluční algoritmus dle specifikovaného rozhraní - viz zdrojové soubory, soubor solver.cpp, funkce do_solve_generic. Algoritmus implementujte alespoň ve dvou verzích (SMP, OpenCL nebo MPI), čímž získáte alespoň dva solvery. V souboru descriptor.h si vygenerujte nové GUID pro vaše solvery a zadejte jejich název dle uvedeného vzoru. Nic jiného v tomto souboru neměnte.

Kapitola 2

Analýza

Pro řešení byl zvolen Spiral Optimization Algorithm.

2.1 Spiral Optimization Algorithm

Spiral Optimization Algorithm (SPO) je evoluční algoritmus určený pro hledání minima funkce. Algoritmus byl průvodně popsán [2] pro použití v 2-rozměrném prostoru, ale v článku [1] byl upraven i pro použití v n-rozměrném prostoru.

Algoritmus

1. Nastavení parametrů:
   - k_{max} = počet iterací,
   - m = počet bodů prohledávání, $m > 1$,
   - θ = úhel rotace, $0 < \theta < 2\pi$,
   - r = velikost kroku, $0 < r < 1$.
2. Vygenerování m bodů prohledávání a určení bodu $x^\star$ jako bodu s nejmenší funkční hodnotou.
3. Rotace ostatních bodů kolem bodu $x^\star$ o úhel θ s velikostí kroku r.
4. Určení funkční hodnoty pro všechny body.
5. Určení bodu $x^\star$ jako bodu s nejmenší funkční hodnotou.
6. Kontrola zastavovací podmínky $k > k_{max}$, jinak zpět ke kroku 3.
7. Prohlášení $x^\star$ za řešení.

Kapitola 3

Implementace

Byly implementovány celkem tři solvery. Obecná problematika bude vysvětlena na prvním z nich. U ostatních budou vysvětleny už jen specifika související s použitím dané technologie.

3.1 Solver využívající sekvenční výpočet

Algoritmus samotný je poměrně jednoduchý a jeho implementace byla přímočará. Za parametry byly zvoleny $r = 0.95$ a $\theta = \pi/4$. Za zmínku však stojí problematika *generování startovacích bodů* a *zastavovací podmínka*.

Generování startovacích bodů Pro co nejvyšší pravděpodobnost nalezení řešení je vhodné prostor co nejrovnoměrněji pokrýt. Bylo provedeno mnoho experimentů a jako nejlepší řešení se ukázala kombinace určených bodů doplněná náhodnými body. Určenými body se myslí bod uprostřed prohledávaného prostoru a body v jeho rozích. Přičemž se body neumísťují úplně do rohu, ale pouze do jeho blízkosti, relativně podle velikosti prostoru. Tento parametr je nastaven na 0.1 z celkové šířky dané dimenze. Například pro 1D prostor (-1, 1) by to byly body 1.0 (uprostřed), -0.8 (jeden roh) a 0.8 (druhý roh).

Zastavovací podmínka Zavedení zastavovací podmínky je nejsnazší a nejjednodušší způsob jak urychlit výpočet. Na duhou stranu špatná podmínka může přivodit předčasné ukončení výpočtu. Po různých pokusech se jako nejlepší řešení ukázalo sečtení hodnot funkce pro všechny body a kontrola jestli se tato suma zmenšuje. Pokud se suma určitý počet iterací nezmenší (nastaveno na 256), tak se výpočet ukončí. Tak je možné ušetřit až 98 % času.

3.2 Solver využívající TBB

Jelikož je SPO iterativní algoritmus, je možné paralelizovat výpočty pouze v rámci jedné iterace.

Paralelizované části

- Počáteční výpočet funkční hodnoty (část kroku 2 algoritmu SPO). Uveden na Výpisu 3.1.
- Rotace bodu a výpočet funkční hodnoty (krok 3 a 4 algoritmu SPO).

Na obě části byla použita funkce `tbb::parallel_for` s `tbb::blocked_range`. Jiná vhodná místa k paralelizaci nebyla nalezena.

Listing 3.1: Ukázka použití `tbb::parallel_for`

```
1 void calculate_fitness_smp(solver::TSolver_Setup &setup, double **search_points,
      double *fitness_results) {
2     tbb::parallel_for(tbb::blocked_range<int>(0, (int)setup.population_size), [&](
          tbb::blocked_range<int> r) {
3         for (int i = r.begin(); i < r.end(); ++i) {
4             fitness_results[i] = count_fitness(setup, search_points[i]);
5         }
6     });
7 }
```

3.3 Solver využívající OpenCL

Pro algoritmus SPO není implementace pro grafickou kartu příliš vhodná a solver tak byl vytvořen spíše z výukových důvodů.

Přesnost čísel s plovoucí desetinnou čárkou Při implementaci jsem řešil problém, že žádné ze zařízení nepodporovalo `double`. Standardně se podpora pro `double` povoluje příkazem: `#pragma OPENCL EXTENSION cl_khr_fp64: enable`, ovšem pouze pokud je rozšíření `cl_khr_fp64` podporované. V mém případě tomu tak nebylo, viz Výpis 3.2, a tak jsem musel na grafické kartě počítat s hodnotami typu `float`.

Listing 3.2: Informace o dostupných zařízeních

```
1 Devices: 2
2 --------
3 CL_DEVICE_NAME (26): Intel(R) HD Graphics 4600
4 CL_DEVICE_VERSION (12): OpenCL 1.2
5 CL_DEVICE_EXTENSIONS (0):
6 --------
7 CL_DEVICE_NAME (41): Intel(R) Core(TM) i3-4000M CPU @ 2.40GHz
8 CL_DEVICE_VERSION (25): OpenCL 1.2 (Build 10094)
9 CL_DEVICE_EXTENSIONS (0):
```

Paralelizované části Paralelizováno bylo maticové násobení použité při výpočtu rotace (část kroku 3 algoritmu SPO). Původně bylo paralelizováno také odčítání vektorů, ale později byla tato část z výkonnostních důvodů odebrána. Jako zajímavost však byl tento kód ponechán v příloze.

Kapitola 4

Výsledky

Testy probíhaly na notebooku s procesorem Intel Core i3-4000M (2.4 GHz, 2 jádra) a 8 GB RAM, který je vybaven integrovaným grafickým čipem.

4.1 Přesnost

Tabulky 4.2, 4.3 a 4.4 ukazují rozdíl nalezeného optima od optima dané funkce pro každý solver, pro dimenze 3, 8, a 40. Výsledek je vždy stejný pro každý běh.

4.2 Rychlost

Tabulky 4.5, 4.6 a 4.7 ukazují průměrnou dobu výpočtu pro dimenze 3, 8, a 40. Výpočet byl proveden 3x a byla určena i směrodatná odchylka.

4.3 Urychlení

Tabulka 4.1 porovnává časy běhu sériového a paralelního solveru. Čas běhu je součtem průměrných časů běhu pro všechny problémy přes všechny měřené velikosti populace.

Tabulka 4.1: Porovnání časů běhu

Dimenze	Doba výpočtu Serial v s	Doba výpočtu TBB v s	Urychlení
3	0.20	0.34	1.7
8	2.64	4.18	1.58
20	8.75	8.18	0.93
40	16.48	10.05	0.61

Tabulka 4.2: Přesnost solverů na dané úloze při dimenzi 3

Solver	Chyba fitness funkce							
	Sphere	Rosenbrock	AbsSum	DeJong4	Rastrigin	Schwefel	Griewank	Masters
Serial_7	0.00e+00	1.82e-01	2.22e-16	3.45e-32	2.51e-08	8.14e-10	2.22e-02	0.00e+00
Serial_15	0.00e+00	5.34e-01	0.00e+00	1.05e-10	9.95e-01	8.14e-10	9.86e-03	0.00e+00
Serial_25	0.00e+00	7.51e-02	3.33e-16	1.52e-64	9.95e-01	8.14e-10	5.18e-02	0.00e+00
Serial_40	0.00e+00	3.46e-01	2.20e-05	1.42e-36	1.99e+00	8.14e-10	9.12e-02	0.00e+00
Serial_60	0.00e+00	8.44e-01	0.00e+00	7.29e-63	9.95e-01	8.14e-10	2.22e-02	0.00e+00
Serial_100	0.00e+00	2.14e-01	1.11e-16	0.00e+00	0.00e+00	8.14e-10	2.96e-02	0.00e+00
SMP_7	0.00e+00	1.82e-01	2.22e-16	3.45e-32	2.51e-08	8.14e-10	2.22e-02	0.00e+00
SMP_15	0.00e+00	5.34e-01	0.00e+00	1.05e-10	9.95e-01	8.14e-10	9.86e-03	0.00e+00
SMP_25	0.00e+00	7.51e-02	3.33e-16	1.52e-64	9.95e-01	8.14e-10	5.18e-02	0.00e+00
SMP_40	0.00e+00	3.46e-01	2.20e-05	1.42e-36	1.99e+00	8.14e-10	9.12e-02	0.00e+00
SMP_60	0.00e+00	8.44e-01	0.00e+00	7.29e-63	9.95e-01	8.14e-10	2.22e-02	0.00e+00
SMP_100	0.00e+00	2.14e-01	1.11e-16	0.00e+00	0.00e+00	8.14e-10	2.96e-02	0.00e+00
OpenCL_7	0.00e+00	1.82e-01	1.19e-07	4.04e-28	1.89e-09	1.08e-08	2.22e-02	0.00e+00
OpenCL_15	0.00e+00	5.34e-01	1.19e-06	1.06e-10	9.95e-01	1.08e-08	9.86e-03	0.00e+00
OpenCL_25	0.00e+00	7.51e-02	3.58e-07	1.01e-26	9.95e-01	1.46e-08	5.18e-02	0.00e+00
OpenCL_40	0.00e+00	3.46e-01	2.96e-05	7.88e-25	1.99e+00	1.31e-09	9.12e-02	0.00e+00
OpenCL_60	0.00e+00	8.44e-01	2.38e-07	4.04e-28	9.95e-01	8.09e-10	2.22e-02	0.00e+00
OpenCL_100	0.00e+00	2.14e-01	4.17e-07	8.08e-28	1.05e-09	3.81e-09	2.96e-02	0.00e+00

Tabulka 4.3: Přesnost solverů na dané úloze při dimenzi 8

Solver	Chyba fitness funkce							
	Sphere	Rosenbrock	AbsSum	DeJong4	Rastrigin	Schwefel	Griewank	Masters
Serial_7	0.00e+00	1.88e-02	6.66e-16	1.25e-61	0.00e+00	2.17e-09	2.02e-01	0.00e+00
Serial_15	0.00e+00	7.99e-03	1.11e-16	1.51e-61	0.00e+00	2.17e-09	2.02e-01	0.00e+00
Serial_25	0.00e+00	9.86e-01	4.44e-16	1.47e-62	0.00e+00	2.17e-09	1.60e-01	0.00e+00
Serial_40	0.00e+00	9.47e-01	3.33e-16	5.77e-63	0.00e+00	2.17e-09	1.16e-01	0.00e+00
Serial_60	0.00e+00	8.58e-04	2.22e-16	1.22e-62	0.00e+00	2.17e-09	5.05e-01	0.00e+00
Serial_100	0.00e+00	1.17e-02	3.33e-16	9.72e-63	0.00e+00	2.17e-09	1.97e-01	0.00e+00
SMP_7	0.00e+00	1.88e-02	6.66e-16	1.25e-61	0.00e+00	2.17e-09	2.02e-01	0.00e+00
SMP_15	0.00e+00	7.99e-03	1.11e-16	1.51e-61	0.00e+00	2.17e-09	2.02e-01	0.00e+00
SMP_25	0.00e+00	9.86e-01	4.44e-16	1.47e-62	0.00e+00	2.17e-09	1.60e-01	0.00e+00
SMP_40	0.00e+00	9.47e-01	3.33e-16	5.77e-63	0.00e+00	2.17e-09	1.16e-01	0.00e+00
SMP_60	0.00e+00	8.58e-04	2.22e-16	1.22e-62	0.00e+00	2.17e-09	5.05e-01	0.00e+00
SMP_100	0.00e+00	1.17e-02	3.33e-16	9.72e-63	0.00e+00	2.17e-09	1.97e-01	0.00e+00
OpenCL_7	0.00e+00	1.86e-02	2.98e-07	1.92e-26	1.13e-10	1.22e-09	2.02e-01	0.00e+00
OpenCL_15	0.00e+00	7.74e-03	2.98e-07	3.76e-27	3.27e-10	1.92e-09	2.02e-01	0.00e+00
OpenCL_25	0.00e+00	9.42e-01	2.38e-07	8.84e-28	1.47e-10	1.72e-09	1.60e-01	0.00e+00
OpenCL_40	0.00e+00	9.02e-01	2.38e-07	1.16e-26	5.64e-11	1.92e-09	1.16e-01	0.00e+00
OpenCL_60	0.00e+00	1.75e-04	8.94e-07	5.74e-27	1.35e-10	1.46e-11	5.05e-01	0.00e+00
OpenCL_100	0.00e+00	1.02e-02	5.96e-08	1.21e-27	1.13e-11	1.40e-09	1.97e-01	0.00e+00

Tabulka 4.4: Přesnost solverů na dané úloze při dimenzi 40

Solver	Chyba fitness funkce							
	Sphere	Rosenbrock	AbsSum	DeJong4	Rastrigin	Schwefel	Griewank	Masters
Serial_7	0.00e+00	2.07e+02	3.60e+00	1.42e-06	3.88e+00	1.14e+02	3.99e+02	0.00e+00
Serial_15	0.00e+00	1.54e+02	3.35e+00	1.42e-06	2.02e+00	5.45e+01	3.91e+02	0.00e+00
Serial_25	0.00e+00	2.28e+02	3.31e+00	7.22e-07	1.57e+00	6.84e+01	2.72e+02	0.00e+00
Serial_40	0.00e+00	6.05e+01	2.71e+00	4.77e-07	5.14e-01	1.97e+01	2.56e+02	0.00e+00
Serial_60	0.00e+00	6.70e+01	2.20e+00	2.27e-07	1.87e-01	8.03e+00	3.14e+02	0.00e+00
Serial_100	0.00e+00	1.03e+02	3.21e+00	4.11e-07	2.74e-01	1.01e+01	9.51e+01	0.00e+00
SMP_7	0.00e+00	2.07e+02	3.60e+00	1.42e-06	3.88e+00	1.14e+02	3.99e+02	0.00e+00
SMP_15	0.00e+00	1.54e+02	3.35e+00	1.42e-06	2.02e+00	5.45e+01	3.91e+02	0.00e+00
SMP_25	0.00e+00	2.28e+02	3.31e+00	7.22e-07	1.57e+00	6.84e+01	2.72e+02	0.00e+00
SMP_40	0.00e+00	6.05e+01	2.71e+00	4.77e-07	5.14e-01	1.97e+01	2.56e+02	0.00e+00
SMP_60	0.00e+00	6.70e+01	2.20e+00	2.27e-07	1.87e-01	8.03e+00	3.14e+02	0.00e+00
SMP_100	0.00e+00	1.03e+02	3.21e+00	4.11e-07	2.74e-01	1.01e+01	9.51e+01	0.00e+00
OpenCL_7	0.00e+00	2.06e+02	3.60e+00	1.02e-06	3.67e+00	1.09e+02	3.99e+02	0.00e+00
OpenCL_15	0.00e+00	1.54e+02	3.35e+00	1.22e-06	1.99e+00	5.08e+01	3.91e+02	0.00e+00
OpenCL_25	0.00e+00	2.28e+02	3.31e+00	6.30e-07	1.18e+00	4.73e+01	2.72e+02	0.00e+00
OpenCL_40	0.00e+00	6.04e+01	2.71e+00	3.89e-07	4.40e-01	1.79e+01	2.56e+02	0.00e+00
OpenCL_60	0.00e+00	6.70e+01	2.20e+00	2.16e-07	1.65e-01	8.11e+00	3.14e+02	0.00e+00
OpenCL_100	0.00e+00	1.03e+02	3.21e+00	3.92e-07	2.47e-01	1.01e+01	9.51e+01	0.00e+00

Tabulka 4.5: Průměrná doba výpočtu při dimenzi 3 (první 4 problémy)

Solver	Doba výpočtu v sekundách			
	Sphere	Rosenbrock	AbsSum	DeJong4
Serial_7	3.07e-04 ± 9.03e-06	3.33e-04 ± 3.48e-06	3.12e-04 ± 6.06e-06	1.39e-03 ± 6.78e-06
Serial_15	6.67e-04 ± 4.18e-06	7.54e-04 ± 5.53e-06	6.88e-04 ± 4.45e-06	3.85e-03 ± 5.35e-06
Serial_25	1.21e-03 ± 8.55e-05	1.26e-03 ± 8.37e-06	1.15e-03 ± 1.58e-05	5.46e-03 ± 1.42e-05
Serial_40	2.01e-03 ± 3.58e-04	2.04e-03 ± 1.35e-05	1.88e-03 ± 1.19e-05	9.03e-03 ± 1.59e-04
Serial_60	3.24e-03 ± 5.42e-04	3.11e-03 ± 1.69e-05	2.81e-03 ± 1.46e-05	1.38e-02 ± 2.78e-05
Serial_100	4.56e-03 ± 3.37e-05	5.23e-03 ± 2.61e-04	4.53e-03 ± 2.06e-05	2.26e-02 ± 2.40e-05
SMP_7	3.25e-03 ± 5.12e-04	2.93e-03 ± 2.50e-05	2.92e-03 ± 2.43e-05	3.47e-03 ± 3.30e-05
SMP_15	4.38e-03 ± 5.68e-05	4.45e-03 ± 1.48e-05	4.38e-03 ± 3.69e-05	6.46e-03 ± 3.60e-05
SMP_25	6.32e-03 ± 7.66e-04	6.14e-03 ± 1.83e-04	5.83e-03 ± 1.41e-05	7.42e-03 ± 4.45e-05
SMP_40	8.32e-03 ± 1.18e-03	7.76e-03 ± 4.27e-05	7.64e-03 ± 8.43e-05	1.01e-02 ± 4.55e-05
SMP_60	9.78e-03 ± 1.00e-03	9.39e-03 ± 1.89e-05	9.08e-03 ± 2.57e-05	1.28e-02 ± 1.19e-04
SMP_100	1.28e-02 ± 2.76e-04	1.28e-02 ± 8.09e-05	1.22e-02 ± 6.46e-05	1.85e-02 ± 2.25e-04
OpenCL_7	1.79e+00 ± 1.69e-01	1.69e+00 ± 2.07e-02	1.70e+00 ± 2.77e-02	1.61e+00 ± 1.37e-02
OpenCL_15	3.76e+00 ± 7.53e-01	3.15e+00 ± 3.06e-01	3.88e+00 ± 3.42e-02	2.95e+00 ± 2.43e-02
OpenCL_25	5.51e+00 ± 1.32e+00	5.15e+00 ± 3.91e-02	4.64e+00 ± 7.76e-02	5.61e+00 ± 1.33e+00
OpenCL_40	7.44e+00 ± 3.79e-02	7.83e+00 ± 4.58e-02	7.78e+00 ± 1.43e-01	7.38e+00 ± 3.28e-02
OpenCL_60	1.14e+01 ± 4.66e-01	1.14e+01 ± 2.64e-02	1.10e+01 ± 1.31e+00	1.12e+01 ± 1.36e+00
OpenCL_100	1.83e+01 ± 9.47e-01	1.83e+01 ± 1.63e-01	1.90e+01 ± 6.83e-02	1.80e+01 ± 4.47e-02

Tabulka 4.6: Průměrná doba výpočtu při dimenzi 8 (první 4 problémy)

Solver	Doba výpočtu v sekundách			
	Sphere	Rosenbrock	AbsSum	DeJong4
Serial_7	3.19e-04 ± 9.78e-05	5.39e-03 ± 7.95e-05	8.58e-04 ± 7.82e-06	3.76e-03 ± 1.58e-05
Serial_15	6.57e-04 ± 1.26e-04	3.28e-02 ± 5.14e-04	2.15e-03 ± 5.57e-06	9.68e-03 ± 8.12e-06
Serial_25	9.37e-04 ± 1.19e-04	3.26e-01 ± 4.81e-03	3.09e-03 ± 1.01e-05	1.43e-02 ± 2.44e-05
Serial_40	1.73e-03 ± 4.72e-04	5.16e-01 ± 1.17e-03	5.90e-03 ± 1.74e-05	2.31e-02 ± 1.69e-05
Serial_60	2.23e-03 ± 6.88e-04	1.29e-01 ± 3.62e-04	8.65e-03 ± 3.26e-05	4.13e-02 ± 3.20e-04
Serial_100	5.23e-03 ± 5.55e-04	1.15e+00 ± 4.44e-02	1.77e-02 ± 1.34e-05	5.56e-02 ± 1.38e-04
SMP_7	1.53e-03 ± 3.85e-04	2.27e-02 ± 7.86e-05	3.80e-03 ± 9.14e-06	5.26e-03 ± 5.01e-05
SMP_15	1.78e-03 ± 7.04e-05	9.38e-02 ± 4.30e-04	6.68e-03 ± 3.20e-05	9.57e-03 ± 3.05e-05
SMP_25	2.52e-03 ± 8.66e-05	7.78e-01 ± 1.66e-03	7.92e-03 ± 2.26e-05	1.18e-02 ± 2.01e-05
SMP_40	3.34e-03 ± 1.02e-04	1.03e+00 ± 6.16e-03	1.25e-02 ± 7.05e-05	1.66e-02 ± 2.67e-04
SMP_60	5.66e-03 ± 1.79e-03	2.12e-01 ± 3.82e-04	1.51e-02 ± 1.05e-04	2.71e-02 ± 4.83e-05
SMP_100	7.08e-03 ± 2.29e-03	1.61e+00 ± 2.90e-03	2.64e-02 ± 7.26e-05	3.22e-02 ± 1.02e-04
OpenCL_7	1.03e+00 ± 1.60e-01	1.89e+01 ± 1.23e-01	1.92e+00 ± 9.16e-03	2.07e+00 ± 2.39e-02
OpenCL_15	1.53e+00 ± 9.42e-03	5.38e+01 ± 1.52e+00	3.92e+00 ± 3.34e-02	4.59e+00 ± 1.37e+00
OpenCL_25	2.33e+00 ± 3.58e-02	8.12e+02 ± 1.12e+00	5.75e+00 ± 7.51e-02	5.63e+00 ± 2.78e-02
OpenCL_40	3.54e+00 ± 1.88e-02	1.27e+03 ± 6.37e+00	8.34e+00 ± 5.60e-02	7.60e+00 ± 3.89e-02
OpenCL_60	5.10e+00 ± 3.59e-02	1.78e+03 ± 2.47e+00	1.17e+01 ± 6.22e-02	1.14e+01 ± 9.17e-02
OpenCL_100	8.22e+00 ± 5.26e-02	1.20e+03 ± 7.17e+00	2.84e+01 ± 8.94e-02	2.24e+01 ± 1.18e+00

Tabulka 4.7: Průměrná doba výpočtu při dimenzi 40 (první 4 problémy)

Solver	Doba výpočtu v sekundách			
	Sphere	Rosenbrock	AbsSum	DeJong4
Serial_7	9.29e-02 ± 1.90e-02	2.74e-01 ± 7.07e-04	8.30e-02 ± 1.65e-04	2.01e-01 ± 3.45e-04
Serial_15	8.75e-02 ± 1.62e-05	9.12e-01 ± 3.53e-03	8.77e-02 ± 2.85e-04	5.14e-01 ± 2.69e-03
Serial_25	9.32e-02 ± 1.58e-04	9.37e-02 ± 6.16e-04	9.35e-02 ± 5.96e-04	7.33e-01 ± 4.93e-04
Serial_40	1.02e-01 ± 1.19e-04	1.81e-01 ± 1.22e-03	1.03e-01 ± 6.46e-04	6.08e+00 ± 2.38e-02
Serial_60	1.15e-01 ± 9.36e-05	1.16e-01 ± 1.77e-04	1.20e-01 ± 6.10e-03	7.20e-01 ± 7.65e-02
Serial_100	1.36e-01 ± 3.09e-04	1.38e-01 ± 2.84e-04	1.37e-01 ± 3.48e-04	6.14e-01 ± 4.95e-03
SMP_7	8.28e-02 ± 5.84e-04	2.56e-01 ± 1.07e-03	8.28e-02 ± 3.11e-04	1.66e-01 ± 5.95e-04
SMP_15	8.49e-02 ± 9.90e-06	6.49e-01 ± 2.16e-03	8.52e-02 ± 3.02e-04	3.17e-01 ± 9.21e-04
SMP_25	8.78e-02 ± 2.69e-04	8.81e-02 ± 4.80e-04	8.80e-02 ± 4.45e-04	3.94e-01 ± 2.20e-03
SMP_40	9.31e-02 ± 3.90e-04	1.36e-01 ± 1.93e-04	9.30e-02 ± 2.82e-04	2.87e+00 ± 1.34e-03
SMP_60	9.92e-02 ± 1.57e-04	1.00e-01 ± 2.06e-04	9.95e-02 ± 1.30e-05	3.39e-01 ± 4.77e-04
SMP_100	1.09e-01 ± 3.04e-04	1.10e-01 ± 3.14e-04	1.09e-01 ± 3.39e-04	3.10e-01 ± 7.46e-04
OpenCL_7	1.14e+00 ± 1.92e-01	2.21e+01 ± 1.33e+00	1.04e+00 ± 1.22e-02	3.29e+01 ± 1.37e+00
OpenCL_15	1.70e+00 ± 2.52e-02	3.16e+01 ± 4.84e-02	1.69e+00 ± 1.35e-02	3.84e+01 ± 2.38e+00
OpenCL_25	2.53e+00 ± 2.76e-02	2.57e+00 ± 4.45e-02	2.57e+00 ± 3.44e-02	3.78e+01 ± 1.34e+00
OpenCL_40	3.70e+00 ± 5.25e-02	2.29e+01 ± 1.77e-01	3.80e+00 ± 7.72e-02	9.63e+01 ± 4.52e-01
OpenCL_60	5.27e+00 ± 5.97e-02	5.42e+00 ± 6.83e-02	5.47e+00 ± 6.97e-02	3.39e+01 ± 1.44e+00
OpenCL_100	9.23e+00 ± 1.25e+00	8.65e+00 ± 6.83e-02	8.76e+00 ± 4.76e-02	6.18e+01 ± 1.41e+00

Kapitola 5

Závěr

Implementovaný algoritmus zafungoval na některé problémy velmi dobře a na jiné hůře. U některých problémů bylo nalezeno optimum.

Sekvenční solver a SMP solver dávají vždy stejné výsledky. Výsledky OpenCL solveru se někdy liší kvůli převodu na `float`. Výsledek závisí především na vygenerovaných startovacích bodech, jelikož algoritmus nemá prostředky, jak by zabránil uváznutí v lokálním minimu.

Pro algoritmus SPO není implementace pro grafickou kartu příliš vhodná a solver pro OpenCL tak byl vytvořen spíše z výukových důvodů. K horšímu výkonu přispěl také fakt, že se čísla musela převádět z typu `double` na typ `float`.

Pro menší počet dimenzí je efektivnější sériový solver. S rostoucí dimenzí se začíná paralelizace TBB solveru vyplácet. Práci by vylepšilo měření na stroji s více jádry, aby bylo více patrné urychlení TBB solveru vůči sériovému solveru.

Literatura

[1] K. Tamura and K. Yasuda. Spiral optimization -a new multipoint search method. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1759–1764, Oct 2011.

[2] Kenichi Tamura and Keiichiro Yasuda. Primary study of spiral dynamics inspired optimization. *IEEJ Transactions on Electrical and Electronic Engineering*, 6(S1):S98–S100, 2011.

Přílohy

Odčítání vektorů v prostředí OpenCL

```
1
2  const char *opencl_source = "                                \
3          __kernel void vec_subst(                             \
4                  __global int *A,                             \
5                  __global int *B,                             \
6                  __global int *C) {                           \
7              int idx = get_global_id(0);                      \
8              C[idx] = A[idx] - B[idx];                        \
9          }                                                    ";
10
11 void test() {
12     // ---------- data
13
14     const int elements = 12;
15
16     int *A = new int[elements];  // in
17     int *B = new int[elements];  // in
18     int *C = new int[elements];  // out
19     for (int i = 0; i < elements; i++) {
20         A[i] = i + 1;
21         B[i] = i;
22     }
23
24     // ---------- inicializace zarizeni
25
26     cl_int status;
27
28     // nalezeni a inicializace platform
29     cl_uint num_platforms = 0;
30     status = clGetPlatformIDs(0, NULL, &num_platforms);
31     cl_platform_id *platforms = new cl_platform_id[num_platforms];
32     status = clGetPlatformIDs(num_platforms, platforms, NULL);
33
34     // nalezeni a inicializace zarizeni
35     cl_uint num_devices = 0;
36     status = clGetDeviceIDs(platforms[0], CL_DEVICE_TYPE_ALL, 0, NULL, &num_devices
           );
37     cl_device_id *devices = new cl_device_id[num_devices];
38     status = clGetDeviceIDs(platforms[0], CL_DEVICE_TYPE_ALL, num_devices, devices,
            NULL);
39
40     // vytvoreni kontextu a command queue
41     cl_context context = clCreateContext(NULL, num_devices, devices, NULL, NULL, &
           status);
42     cl_command_queue cmd_queue = clCreateCommandQueue(context, devices[0], 0, &
           status);
43
44     // ---------- inicializace kernelu
45
46     // priprava programu
47     cl_program program = clCreateProgramWithSource(context, 1, (const char**)&
           opencl_source, NULL, &status);
```

```
48      status = clBuildProgram(program, num_devices, devices, NULL, NULL, NULL);
49
50      // vytvoreni bufferu
51      cl_mem buffer_A = clCreateBuffer(context, CL_MEM_READ_ONLY, sizeof(int) *
            elements, NULL, &status);
52      cl_mem buffer_B = clCreateBuffer(context, CL_MEM_READ_ONLY, sizeof(int) *
            elements, NULL, &status);
53      cl_mem buffer_C = clCreateBuffer(context, CL_MEM_WRITE_ONLY, sizeof(int) *
            elements, NULL, &status);
54
55      // vytvoreni kernelu a asociace bufferu
56      cl_kernel kernel = clCreateKernel(program, "vec_subst", &status);
57      status = clSetKernelArg(kernel, 0, sizeof(cl_mem), &buffer_A);
58      status |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &buffer_B);
59      status |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &buffer_C);
60
61      // ---------- vykonani
62
63      status = clEnqueueWriteBuffer(cmd_queue, buffer_A, CL_FALSE, 0, sizeof(int) *
            elements, A, 0, NULL, NULL);
64      status = clEnqueueWriteBuffer(cmd_queue, buffer_B, CL_FALSE, 0, sizeof(int) *
            elements, B, 0, NULL, NULL);
65
66      size_t global_work_size[1] = { elements };
67      status = clEnqueueNDRangeKernel(cmd_queue, kernel, 1, NULL, global_work_size,
            NULL, 0, NULL, NULL);
68
69      clEnqueueReadBuffer(cmd_queue, buffer_C, CL_TRUE, 0, sizeof(int) * elements, C,
             0, NULL, NULL);
70
71      // ---------- kontrola vysledku
72
73      bool result = true;
74      for (int i = 0; i < elements; i++) {
75          if (C[i] != 1) {
76              result = false;
77              break;
78          }
79      }
80      printf(result ? "Output is correct\n" : "Output is incorrect\n");
81
82      // ---------- uvolneni zdroju
83
84      clReleaseKernel(kernel);
85      clReleaseProgram(program);
86      clReleaseCommandQueue(cmd_queue);
87      clReleaseMemObject(buffer_A);
88      clReleaseMemObject(buffer_B);
89      clReleaseMemObject(buffer_C);
90      clReleaseContext(context);
91
92      free(A);
93      free(B);
94      free(C);
95      free(platforms);
96      free(devices);
97  }
```