Buletinul Ştiinţific al Universităţii Politehnica Timişoara
Seria Automatică şi Calculatoare
SCIENTIFIC BULLETIN of The Politehnica University of Timişoara
Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE, Vol. 60(74), No. 1, March. 2015
ISSN 1224-600X

GPU Based Implementation of Inverse Heat Conduction Problem Solver

Sándor Szénási*, Imre Felde** and István Kovács***

** Óbuda University, Faculty of Informatics, Bécsi út 96/B, 1034 Budapest
Phone: +36 (1) 666-5551, E-Mail: szenasi.sandor@nik.uni-obuda.hu*

*** Óbuda University, Faculty of Informatics, Bécsi út 96/B, 1034 Budapest
Phone: +36 (1) 666-5528, E-Mail: felde.imre@nik.uni-obuda.hu*

**** Óbuda University, Faculty of Informatics, Bécsi út 96/B, 1034 Budapest
E-Mail: kovacs.istvan.perez@outlook.com*

Abstract ***–Inverse Heat Conduction Problem means that the surface Heat Transfer Coefficient (HTC)/Heat Flux (HF) must be determined from transient temperature measurements at given interior points. This is a typical ill-posed problem, because the solution's behaviour does not change continuously with the initial conditions; therefore, there are no already known direct solutions. There are several heuristic methods to solve the IHCP, and one of the most promising methods is Particle Swarm Optimization (PSO) developed by Eberhart and Kennedy in 1995. It has the ability to find the optimal solution in very large parameter spaces; however, it has some limitations. The main weaknesses are the high computational demand (and consequently a large runtime), and the unpredictable chance to find only a local but not the global optimum. This paper presents the implementation and the evaluation of a graphics accelerator-based parallel approach, which has significantly lower runtime (this GPU implementation is about three times faster than the original CPU-based sequential method). Furthermore, the authors examined the relationship between the size of the initial swarms and the final fitness values. The results indicate that it is worth it to generate a larger initial swarm and continue the processing using smaller further swarms. This technique combines the advantages of the large (better accuracy) and small (lower runtime) particle counts.***

Keywords: ***IHCP, HTC, GPU, PSO, Parallel Algorithm.***

I. INTRODUCTION

Reliable simulation of heat transfer processes requires functions and/or parameters describing the real heat transfer phenomena obtained during cooling. Typical quantities characterizes the heat extraction ability of the cooling medium is the Heat Transfer Coefficient (HTC) and/or the Heat Flux (HF). During quenching in liquids generating vapour and boiling stages (i.e. oils, water, polymer solutions) the heat transfer is altering significantly depending on the surface temperature. In addition to, heat transfer is not uniform at the whole surface domain of the workpiece. It follows that realistic simulation of quenching is assumed to apply the thermal boundary conditions (HTC or HF) as functions of surface temperature and local coordinates as well. We can describe the heat transfer process using Eq 1.

$$\frac{\partial}{\partial r}\left(k\frac{\partial T}{\partial r}\right)+\frac{k}{r}\frac{\partial T}{\partial r}+q_v=\rho C_p\frac{\partial T}{\partial t} \quad (1)$$

Where

- k - thermal conductivity
- C_p - specific heat capacity
- ρ – density
- t – time
- r – polar coordinate
- T – temperature

To solve the IHCP [1], there are several implicit and explicit formulations. In the first approach, the problem is formulated as a multivariable optimization problem [2], [3], while the latter method attempts to directly determine the unknown parameters using regularization techniques to solve the resulting system of equations [4]–[6]. In this paper, we use a stochastic approach that is based on particle swarm optimization.

Buletinul Ştiinţific al Universităţii Politehnica Timişoara
Seria Automatică şi Calculatoare
SCIENTIFIC BULLETIN of The Politehnica University of Timişoara
Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE, Vol. 60(74), No. 1, March. 2015
ISSN 1224-600X

II. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) [7]–[9] is a computational method for optimization problems (it cannot guarantee the optimal solution). It was developed in 1995 by Eberhart and Kennedy [7]. It is a biologically-inspired heuristic optimization method [10], which is based on the behaviours of birds flocking or fish schooling. The main idea of the algorithm is the simulation of animal groups without any distinguished leader (such as birds and fishes).The main concept is to iteratively try to improve a candidate solution. From several aspects, it is similar to genetic algorithms [11]–[13], as the optimization is based on a population (swarm) of candidate solutions (particles). However, there are several differences between the two methods. In the case of PSO, the search algorithm moves these particles around in the problem specific search-space using a position and velocity value for each. The movement direction of each particle is influenced by its best local position and the best known positions of the entire swarm.
The main steps of the basic algorithm are visible in Algorithm 1.
The parameters of the function are as follows:

- N : size of swarm (number of particles)
- ε : minimum fitness change limit
- limit : maximum iteration limit

Algorithm 1 Basic particle swarm optimization

```
function PSO($N, \epsilon, limit$)
    $S[\ ] \leftarrow InitializeSwarm(N)$
    $bestFitness \leftarrow \infty$
    $cnt \leftarrow 1$
    repeat
        for $i \leftarrow 1, N$ do                          ▷ Move items
            $S_i.Velocity \leftarrow CalcVelocity(S_i)$
            $S_i.Position \leftarrow S_i.Position + S_i.Velocity$
        end for
        for $i \leftarrow 1 \rightarrow N$ do                   ▷ Calculate fitness
            $S_i.Fitness \leftarrow CalcFitness(S_i.position)$
        end for
        $cnt \leftarrow cnt + 1$
        $lastFitness \leftarrow bestFitness$
        $bestFitness \leftarrow \min_{i \in 1..N} x_i.Fitness$
    until $lastFitness - bestFitness < \epsilon \vee cnt > limit$
    return $bestFitness$
end function
```

Algorithm 1: Basic Particle Swarm Optimization

III. INCREASING PROCESSING SPEED

As it is visible in the algorithm, several parts of the PSO algorithm are well parallelizable [14]. Both loops process the items one by one, and it is clear that there is no interference between the iterations. Furthermore, we have to do the same calculations on all particles without any communication. These are all embarrassingly parallel calculations. Therefore, these parts are executable in a data parallel fashion, which is ideal for graphics accelerator (GPU) based implementation [15].

Algorithm 2 Data parallel particle swarm optimization

```
function PSO($N, \epsilon, limit$)
    $S[\ ] \leftarrow InitializeSwarm(N)$
    $bestFitness \leftarrow \infty$
    $cnt \leftarrow 1$
    repeat
        for all $x \in swarm$ do parallel
            $x.Velocity \leftarrow CalcVelocity(x)$
            $x.Position \leftarrow x.Position + x.Velocity$
        end for
        for all $x \in swarm$ do parallel
            $x.Fitness \leftarrow CalcFitness(x.position)$
        end for
        $cnt \leftarrow cnt + 1$
        $lastFitness \leftarrow bestFitness$
        $bestFitness \leftarrow \min_{i \in 1..N} x_i.Fitness$
    until $lastFitness - bestFitness < \epsilon \vee cnt > limit$
    return $bestFitness$
end function
```

Algorithm 2: Parallel Particle Swarm Optimization

The swarm creation is well parallelizable too, but this part only runs once; therefore, its execution speed is not relevant. As it is visible in Algorithm 2, the first parallel part is the particle movement section. We should start as many threads as the number of the particles. Each thread will calculate the new velocity and (based on this) the new location of each particle.
In the case of GPU development, the high ratio of arithmetic/memory operations is very important in regard to obtaining the maximum performance [16]. The above mentioned part does not fulfil this condition. The GPU multiprocessors have to load and save all swarm data from the device memory (for example, in the case of 1000 particles that use 10 dimensional double coordinates, it means loading/storing of 80K), meanwhile the number of arithmetic calculations is relatively low (we use a simple method to calculate the new velocity, and the calculation of the new position needs only one 10 addition operations). Nonetheless, it is worth implementing this part in the GPU, because this technique helps to avoid unnecessary memory copies from GPU device memory to host memory and vice versa.
The parallelization of the second loop is much more efficient. We should calculate the fitness values for each particle. The naïve method uses one thread for the calculation of the fitness of one particle. The memory

Buletinul Ştiinţific al Universităţii Politehnica Timişoara
Seria Automatică şi Calculatoare
SCIENTIFIC BULLETIN of The Politehnica University of Timişoara
Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE, Vol. 60(74), No. 1, March. 2015
ISSN 1224-600X

requirement for each thread is a little lower (they only need the position of the swarm, and there is no modification and store operations), while the arithmetical complexity is significantly higher.
For the fitness calculation, we are using the explicit finite difference method (FDM) presented by Smith [17]. In the case of one-dimensional heat transfer simulation, our goal is to solve the following equations:

- For the middle line (Eq. 2):

$$T(n+1,0)=T(n,0)+(dt*\alpha(T))*$$
$$(\frac{1}{dx^2})*2*(T(n,1)-T(n,0)) \qquad (2)$$

- For the surface (Eq. 3):

$$T(n+1,N)=T(n,N)+(dt*\alpha(T))*$$
$$((\frac{1}{dx^2})*2*(T(n,N)-1)-T(n,N)-(\frac{dx}{k})*$$
$$(HTC.(t)*(T(n,N)-T_{cm}))+\frac{1}{N*dx}*\frac{-1}{k(T)}*$$
$$(HTC.(t)*(T(n,N)-T_{cm}) \qquad (3)$$

- And for the inner points (Eq. 4):

$$T(n+1,i)=T(n,i)+(dt*\alpha(T))*$$
$$(\frac{1}{dx^2}*(T(n,i-1)-T(n,i+1)*2*T(n,i))+$$
$$\frac{1}{i*dx}*\frac{1}{2*dx}*(T(n,i+1)-T(n,i-1))) \qquad (4)$$

Where

- alpha - thermal diffusivity (Eq. 5)

$$\alpha(T)=\frac{k(T)}{cp(T)*\rho} \qquad (5)$$

- k - thermal conductivity
- cp - specific heat capacity
- ρ - density
- HTC - heat transfer coefficient
- N - number of points

In this case, the ratio of arithmetic/memory operations is high, which leads to good GPU performance. It is difficult to theoretically estimate the speed-up of a GPU algorithm; therefore, we implemented this naive method and ran several tests to measure the execution time.

IV. INCREASING ACCURACY

Another weakness of the PSO method is that it often finds a local optimum (but not the global one), because the parameter space of the given problem is too large. In our tests, all particles are located in an 11-dimensional parameter space (in the case of 2D calculations, this dimension is significantly higher). In a typical case, a PSO swarm consists of 1000-10000 pieces of particles. It is not possible to cover the entire space with these, however we should seek to the best possible coverage. The final result of the optimization steps depends strongly on the initial random starting positions [18]–[20]. If none of the particles start near the global optimum; this can lead to weak final results.
Increasing the number of particles can solve this problem, but it raises some new issues:

- It requires higher computing capacity and it leads to higher runtime
- This way neither ensures a global optimum

Based on these, we developed a hybrid approach [21], [22]. Our idea is to extend the PSO method by one additional step. The modified algorithm generates more particles in the first (initial) generation than the common swarm size. It evaluates all these temporary particles, and it keeps only the best ones. Fig. 1 shows the main steps of this modified method.

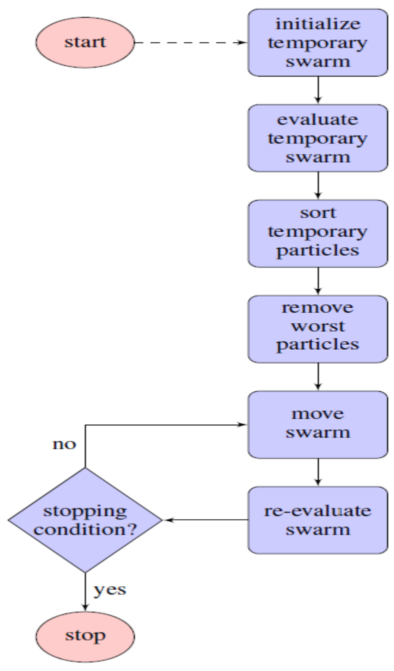

Figure 1: main steps of modified PSO algorithm

Buletinul Ştiinţific al Universităţii Politehnica Timişoara
Seria Automatică şi Calculatoare
SCIENTIFIC BULLETIN of The Politehnica University of Timişoara
Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE, Vol. 60(74), No. 1, March. 2015
ISSN 1224-600X

TEST RESULTS

A. Test configurations

Hardware configurations

We used the following configurations for the tests:

- CPU configuration
  - Processor: Intel Core™ i7-2600
  - Architecture: Sandy Bridge
  - Number of cores: 4
  - Memory: 16GB DDR2
- GPU configuration
  - Graphics accelerator: NVIDIA Tesla K40c
  - Architecture: Kepler (GK110B)
  - Number of shaders: 2880
  - SMX Count: 15
  - Memory: 12GB GDDR5
  - Host: the same as the CPU config

B. Runtime test

In the case of the CPU, we use the C# built-in "parallel for" statement to parallelize the calculations (using all available cores). As expected, the running time is linearly dependent on the number of cores. We need a minimum number of particles to utilize the processing power of all CPU cores, but this number is relatively small. Using swarms with 10 or more particles is enough to reach this maximum performance. Above this limit, the processing time is linearly increasing with the number of items. Thread scheduling is the responsibility of the C# compiler and the .NET environment, but in practice, we can assume that the number of threads is equal to the number of cores; therefore, every thread handles more than one particle.

In the case of the GPU, according to the architecture of the device, the calculation of the estimated runtime is much more complex. First of all, the used graphics accelerator (Tesla K40 Active) has 2880 processing elements; therefore, it is obvious that we need a lot of threads to utilize all of them. The thread to particle assignment is manual, and our main concept was that every thread handles one particle; therefore, we need at least 2880 particles to use all the cores. In fact, a lot more than this is needed as the GPU tries to hide latency using fast context switching mechanism and thus, we need three-four times more particles to fully utilise the processing power of the GPU.

As it is expected, the evaluation of swarms containing few particles is faster in the CPU implementation (one GPU core is significantly slower than one CPU core). As it is visible in Table 1, increasing the swarm size leads to a smaller difference between the measured running times. In addition, there is a limit (about 350 items) where the lead position is swapped. For larger swarms, the GPU becomes faster (Fig. 2).

As we discussed before, we have to start a lot of threads to utilize all processing elements of the GPU. It is visible in Fig. 3 that the speed-up (the ratio of the CPU and the GPU running time) increases rapidly for a smaller number of elements, and becomes a constant at about 8000-10000 or more elements. In these cases, the GPU is three times faster than the CPU, and this condition persists in the case of larger swarms.

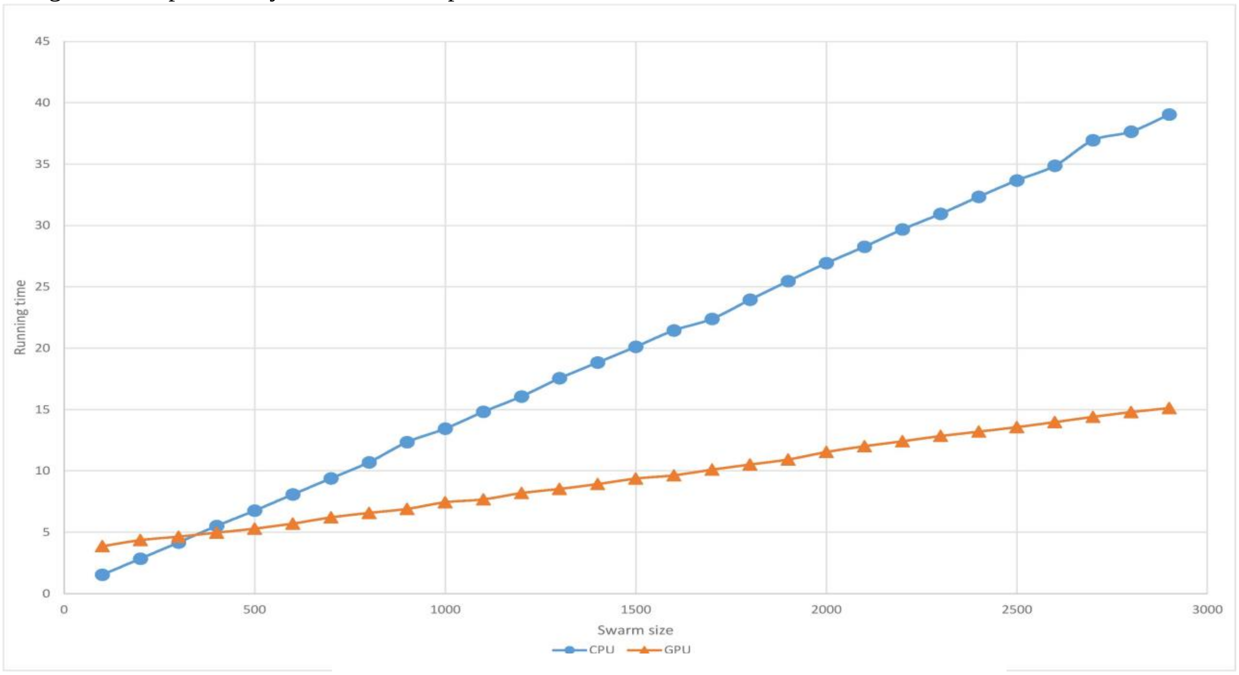

Figure 2: Running time based on swarm size

TABLE 1. Detailed run-time results

Test	Swarm	CPU runtime	GPU runtime	Speed-up
1	100	1.53	3.86	0.40
2	200	2.85	4.36	0.65
3	300	4.17	4.64	0.90
4	400	5.52	4.96	1.11
5	500	6.76	5.29	1.28
6	600	8.09	5.70	1.42
7	700	9.39	6.21	1.51
8	800	10.70	6.58	1.63
9	900	12.35	6.90	1.79
10	1000	13.44	7.44	1.81
11	1100	14.82	7.68	1.93
12	1200	16.06	8.19	1.96
13	1300	17.55	8.53	2.06
14	1400	18.83	8.92	2.11
15	1500	20.11	9.37	2.15
16	1600	21.45	9.64	2.23
17	1700	22.39	10.10	2.22
18	1800	23.95	10.52	2.28
19	1900	25.46	10.92	2.33
20	2000	26.93	11.54	2.33
21	2100	28.26	12.01	2.35
22	2200	29.69	12.41	2.39
23	2300	30.95	12.84	2.41
24	2400	32.34	13.20	2.45
25	2500	33.68	13.56	2.48
26	2600	34.87	13.97	2.49
27	2700	36.95	14.40	2.57
28	2800	37.65	14.79	2.55
29	2900	39.04	15.12	2.58
30	3000	40.44	15.63	2.59
31	3000	40.24	15.64	2.57
32	4000	53.97	19.81	2.73
33	5000	67.73	23.86	2.84
34	6000	80.31	28.01	2.87
35	7000	93.55	32.01	2.92
36	8000	106.91	36.29	2.95
37	9000	120.45	40.24	2.99
38	10000	134.37	44.61	3.01
39	11000	147.24	48.72	3.02
40	12000	160.25	53.18	3.01
41	13000	174.81	57.26	3.05
42	14000	187.32	61.84	3.03
43	15000	201.47	65.74	3.06
44	16000	215.69	73.27	2.94
45	17000	227.72	77.70	2.93
46	18000	242.07	81.79	2.96
47	19000	253.72	85.90	2.95
48	20000	268.96	89.97	2.99

C. Accuracy test

The CPU implementation of the PSO algorithm was an already tested application; therefore, we used this as a reference. To compare the results, we deactivated the random number generator functions (which were always started with the same starting seed number), and ran run several tests using this configuration. The final results of all the CPU and GPU executions were exactly the same.
To increase the accuracy of both implementations, we implement the improvement introduced in Section IV. We ran a lot of tests to evaluate this new algorithm using different number of particles. The tested swarm size was given by the following parameters:

- M – size of the (temporary) initial swarm.
- N – further swarm size (as used in the original PSO algorithm).

Table 2. shows the results of these tests.

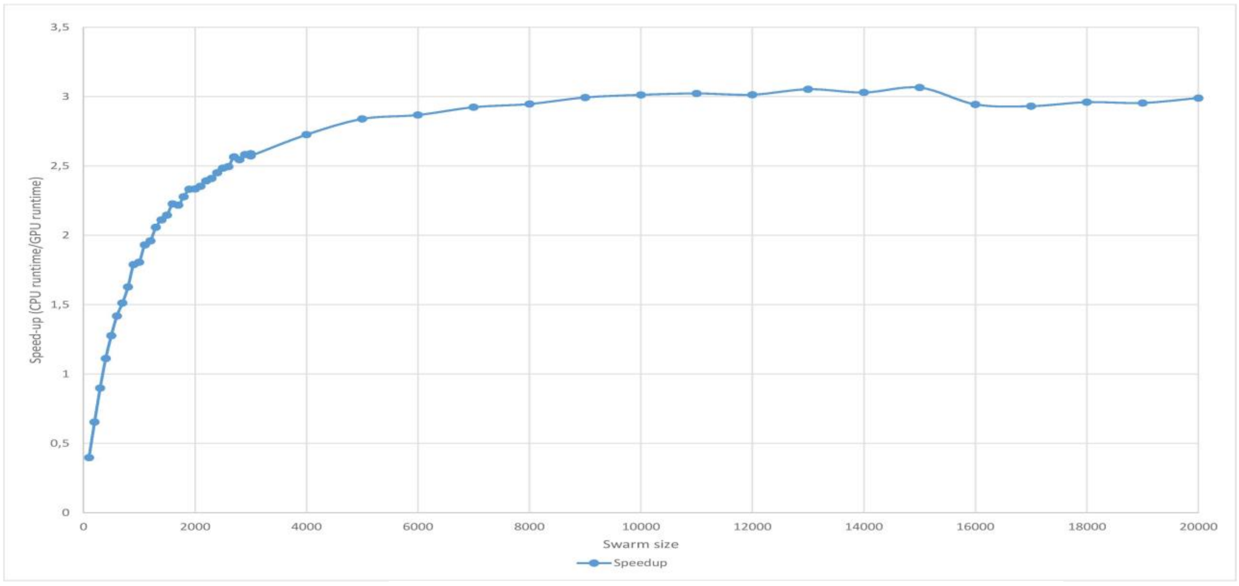

Figure 3: Speed-up based on swarm size

Buletinul Ştiinţific al Universităţii Politehnica Timişoara
Seria Automatică şi Calculatoare
SCIENTIFIC BULLETIN of The Politehnica University of Timişoara
Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE, Vol. 60(74), No. 1, March. 2015
ISSN 1224-600X

TABLE 2. Accuracy test results.

Test	*M (swarm)*	*N (initial)*	*Result fitness*
1	10	10	893
2	20	10	957
3	30	10	853
4	40	10	860
5	50	10	907
6	60	10	873
7	70	10	793
8	80	10	849
9	90	10	803
10	100	10	705
11	110	10	875
12	120	10	777
13	130	10	843
14	140	10	853
15	20	20	689
16	40	20	793
17	60	20	712
18	80	20	730
19	100	20	707
20	120	20	652
21	140	20	753
22	160	20	775
23	180	20	660
24	200	20	684
25	220	20	813
26	240	20	788
27	260	20	669
28	280	20	656
29	40	40	571
30	80	40	655
31	120	40	665
32	160	40	636
33	200	40	620
34	240	40	685
35	280	40	596
36	320	40	628
37	360	40	599
38	400	40	599
39	440	40	596
40	480	40	619
41	520	40	589
42	560	40	629
43	80	80	599
44	160	80	571
45	240	80	587
46	320	80	573
47	400	80	560
48	480	80	551
49	560	80	636
50	640	80	558
51	720	80	558
52	800	80	588
53	880	80	598
54	960	80	600
55	1040	80	562
56	1120	80	577
57	160	160	546
58	320	160	540
59	480	160	539
60	640	160	562
61	800	160	540
62	960	160	543
63	1120	160	540
64	1280	160	547
65	1440	160	542
66	1600	160	541
67	1760	160	548
68	1920	160	535
69	2080	160	538
70	2240	160	532

The results of these tests show that (in most cases) a larger number of initial temporary particles lead to better results.

V. CONCLUSIONS

We have implemented the particle swarm optimization based IHCP method for both the CPU and the GPU. The original algorithm has several well parallelizable parts; therefore, our expectation was that the GPU implementation would need significantly less time to solve the given equations. As it is clearly visible from the results, our expectations were justified; the GPU implementation is about three times faster.

This speed-up strongly depends on the size of the swarm. In the case of small swarms, it is worth using the original CPU base implementation. Though, in the case of larger swarms (350 particles or more), the GPU code becomes faster (this speed-up increase continues until the number of particles is equal to 10000). However, there are no strict rules for determining the number of elements, but in practice, it is common to use more than 350 particles.

Deeper analysis of the GPU code and execution statistics shows that the largest hindering limit is the high number of device memory access operations. Our future plan is to optimize the GPU code to use the shared memory of the multiprocessors to avoid device memory access where possible.

Our assumption that a larger initial swarm can cover a larger part of the search space and lead to better fitness values has been confirmed by the test results. Larger initial swarms lead to better fitness values. The runtime of the evaluation of the initial swarm is higher than in the case of the original algorithm, but this disadvantage does not significantly affect overall runtime.

Based on these experiences, it is worth it to start a larger initial swarm at the start phase of the PSO algorithm, and continue the process with a smaller group of the best selected particles.

ACKNOWLEDGMENTS

We acknowledge the financial support of this work by the Hungarian State and the European Union under the TÁMOP-4.2.2/A-11/1-KONV-2012-0029 and the Chinese-Hungarian S&T bilateral project TÉT_12_CN-1-2012-0027 projects. The authors would like to thank NVIDIA Corporation for providing graphics hardware for the GPU benchmarks through the CUDA Teaching Center program. The authors also wish to thank the members of the CUDA Teaching Center at Óbuda University for their constructive comments and suggestions. The content of the paper however does not necessarily express the views of these companies and persons, the authors take full responsibility for any errors or omissions.

REFERENCES

[1] I. Felde and W. Shi, “Hybrid approach for solution of inverse heat conduction problems,” in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, 2014, pp. 3896–3899.

[2] O. M. Alifanov, *Inverse Heat Transfer Problems*. Springer, 1994.

[3] M. N. Özisik and H. R. B. Orlande, *Inverse Heat Transfer: Fundamentals and Applications*. Taylor & Francis, 2000.

[4] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, “Convergence Properties of the Nelder--Mead Simplex Method in Low Dimensions,” *SIAM J. Optim.*, vol. 9, no. 1, pp. 112–147, 1998.

[5] R. Das, “A simplex search method for a conductive–convective fin with variable conductivity,” *Int. J. Heat Mass Transf.*, vol. 54, pp. 5001–5009, 2011.

[6] O. Nelles, *Nonlinear system identification*. Springer-Verlag, 2001.

[7] J. Kennedy and R. Eberhart, “Particle swarm optimization,” *Neural Networks, 1995. Proceedings., IEEE Int. Conf.*, vol. 4, pp. 1942–1948, 1995.

[8] R. C. David, R. E. Precup, E. M. Petriu, M. B. Rădac, and S. Preitl, “Gravitational search algorithm-based design of fuzzy control systems with a reduced parametric sensitivity,” *Inf. Sci. (Ny).*, vol. 247, pp. 154–173, 2013.

[9] N. A. El-Hefnawy, “Solving Bi-level Problems Using Modified Particle Swarm Optimization Algorithm,” *International Journal of Artificial Intelligence^{TM}*, vol. 12, no. 2. pp. 88–101, 27-Feb-2014.

[10] E. Toth-Laufer, M. Takacs, and I. J. Rudas, “Real-time fuzzy logic-based sport activity risk calculation model optimization,” in *IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI)*, 2013, pp. 291–295.

[11] S. Szénási and Z. Vámossy, “Evolutionary Algorithm for Optimizing Parameters of GPGPU-based Image Segmentation,” *Acta Polytech. Hungarica*, vol. 10, no. 5, pp. 7–28, 2013.

[12] F. Valdez, P. Melin, and O. Castillo, “An improved evolutionary method with fuzzy logic for combining Particle Swarm Optimization and Genetic Algorithms,” in *Applied Soft Computing Journal*, 2011, vol. 11, no. 2, pp. 2625–2632.

[13] A.-C. Zăvoianu, G. Bramerdorfer, E. Lughofer, S. Silber, W. Amrhein, and E. Peter Klement, “Hybridization of multi-objective evolutionary algorithms and artificial neural networks for optimizing the performance of electrical drives,” *Eng. Appl. Artif. Intell.*, vol. 26, no. 8, pp. 1781–1794, 2013.

[14] S. Szénási, I. Felde, and I. Kovács, “Solving One-dimensional IHCP with Particle Swarm Optimization using Graphics Accelerators,” in *10th Jubilee IEEE International Symposium on Applied Computational Intelligence and Informatics*, 2015, pp. 365–369.

[15] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[16] NVIDIA, “CUDA C Programming Guide.” 2014.

[17] G. D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Clarendon Press, 1985.

[18] E. Campana and G. Fasano, “Dynamic system analysis and initial particles position in particle swarm optimization,” *IEEE Swarm Intell.*, no. 1, 2006.

[19] D. Palupi Rini, S. Mariyam Shamsuddin, and S. Sophiyati Yuhaniz, “Particle Swarm Optimization: Technique, System and Challenges,” *Int. J. Comput. Appl.*, vol. 14, no. 1, pp. 19–27, 2011.

[20] Y. del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J. C. Hernandez, and R. G. Harley, “Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems,” *Evol. Comput. IEEE Trans.*, vol. 12, no. 2, pp. 171–195, 2008.

[21] A. Bogárdi-Mészöly, A. Rövid, H. Ishikawa, S. Yokoyama, and Z. Vámossy, “Tag and Topic Recommendation Systems,” *Acta Polytech. Hungarica*, vol. 10, no. 6, pp. 171–191, 2013.

[22] S. Sergyán and L. Csink, “Automatic Parametrization of Region Finding Algorithms in Gray Images,” in *4th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2007, pp. 199–202.