Local strong form meshless method on multiple Graphics Processing Units

G. Kosec[1,2], P. Zinterhof[3]

[1]Jožef Stefan Institute, Department of Communication Systems, Jamova 39 1000 Ljubljana, Slovenia
gkosec@ijs.si
[2]University of Nova Gorica, Laboratoty for Multiphase processes, Vipavska 13, 5000 Nova Gorica, Slovenia
[3]University of Salzburg, Jakob-Haringer-Str. 2, 5020 Salzburg, Austria
peter.zinterhof3@sbg.ac.at

Abstract This paper deals with the implementation of the local meshless numerical method (LMM) on general purpose graphics processing units (GPU) in solving partial differential equations (PDE). The local meshless solution procedure is formulated in a way suitable for parallel execution and has been implemented on multiple GPUs. The implementation is tested on a solution of diffusion equation in a 2D domain. Different setups of the meshless approach regarding the selection of basis functions are tested on an interval up to 2.5 million of computational points. It is shown that monomials are a good selection of the basis when working with a high number of nodes. The results are presented in terms of error analysis, convergence analysis and computational performance measurements.

Keywords: Local Meshless Method, LRBCFM, DAM, Dirichlet jump, GPU, CUDA, dynamic balancing,

1 Introduction

The developments in numerical modeling and computer science are tightly coupled scientific disciplines. Regardless the numerical method the solution algorithms are executed on computers and in most cases the accuracy of a computed solution is limited by the capacities of the available computer resources and the efficiency of computer implementation. One of the challenging tasks in the computational mechanics community is the effective implementation of novel numerical solution procedure in an effective way [Kirk and Hwu, (2010)]. The developments in the computer technology are extremely vivid. Almost all modern computer platforms are nowadays parallel; most computers use several computing cores sharing the same memory. For more complex computations computer clusters are used. Moreover, use of graphical processing units (GPUs) for accelerating the numerical simulations is becoming more and more attractive. Recently there have been several publications regarding the acceleration of computationally intensive programs with GPUs, e.g. cellular automata based lattice-Boltzman simulations [Rinaldi, Dari, Vénere and Clausse, (2012)], wave propagation [Molero and Iturrarán-Viveros, (2013)], environmental studies [Abouali, Timmermans, Castillo and Su, (2013)], turbulence modeling [Yokota, Narumi, Sakamaki, Kameoka, Obi and Yasuoka, (2009)], gas simulations [De Vuyst and Salvarani, (2013)], etc. GPU computing has already been adopted in the meshless context in connection with generalized meshless Finite Difference Method (FDM) methods [Bollig, Flyer and Erlebacher, (2012)] and particle based methods [Yokota, Narumi, Sakamaki, Kameoka, Obi and Yasuoka, (2009)].

From the implementation and execution point of view the simplest and most effective numerical method is explicit FDM since it is completely local and computationally simple. However, the FDM is limited only to consideration of simple domains and it has limited possibilities for upgrades. The promising alternative is a class of the local point interpolation meshless methods (LPIM) [Wang and Liu, (2002)]. The LPIM is based entirely on the approximation constructed over the local sub set of scattered computational points. In general, the meshless methods do not require any topological relations between points making treatment of complex geometries elegant, which, in the case FDM, might not even be possible. The LPIM can be also easily upgraded or altered to treat anomalies such as sharp discontinues or other obscure situations, which might occur in complex simulations. The LPIM offer countless options for upgrades, e.g. straightforward adaptation [Kosec and Šarler, (2011)], basis augmentation [Zhang, Sim, Tan and Sung, (2006)], conditioning of the approximation [Lee, Liu and Fan, (2003)], etc. The popular variant of LPIM where the Radial Basis Functions are used is also referred to as the Local Radial Basis Function Collocation Method (LRBFCM) [Šarler, (2005); Šarler, (2007)]. Another variant with the weighted least squares approximation and a monomial basis is referred to as Diffuse Approximate Method (DAM) [Wang, Sadat and Prax, (2012)]. To avoid confusion we use term Local Meshless Method (LMM) when referring to the local strong form meshless method. Meshless methods originate in the seventies when the Smoothed Particles Hydrodynamics (SPH) have been developed for astrophysical problems [Gingold and Monaghan, (1977)]. Many meshless methods have been developed, e.g. Element free Galerkin method, Meshless Petrov-Galerkin method, Point Interpolation Method, Point Assembly Method, Finite Point Method, Reproducing Kernel Particle Method, and Kansa [Kansa, (1990); Atluri and Shen, (2002a); Atluri and Shen, (2002b); Atluri, (2004); Gu, (2005); Fasshauer, (2006)] since then. The most general and "truly" meshless method is the weak form Meshless Local Petrov-Galerkin method (MLPG) [Atluri, (2004)]. The strong form meshless alternatives such as LRBFCM or DAM are simpler to handle and offer better possibilities for effective parallel execution since they do not require numerical integration [Trobec, Šterk and Robič, (2009)]. The intense development in a field of the meshless methods continues, which is reflected also in several relevant recent publications [Bourantas, Skouras, Loukopoulos and Nikiforidis, (2010); Stevens and Power, (2010); Guo, (2011); Bustamante, Power, Sua and Florez, (2012); Li, (2012); Wang, Sadat and Prax, (2012); Wen and Aliabadi, (2012); Arefmanesh, Najafi and Musavi, (2013); De Chowdhury and Sannasiraj, (2013); Duan and Rong, (2013); Techapiroma and Luadsonga, (2013)].

In this paper we explore the implementation of LMM on a GPU in solving the two dimensional transient partial differential equation (PDE). We deal with the solution of the time dependent second order partial differential equation. The numerical solution of identical problem has been recently published in [Trobec, Kosec, Šterk and Šarler, (2012)], where Finite Element Method (FEM), Meshless Local Petrov Galerkin Method, and DAM are compared in different regimes. The main message in [Trobec, Kosec, Šterk and Šarler, (2012)] is comparison of weak form and strong form methods. The results indicate that strong form methods perform comparable to the weak form methods in tested cases. It has been shown that DAM provides good results despite its simplicity. In this paper we extend the solution of a well-accepted benchmark problem to a much higher number of computational points, up to 2.5 million. We also analyze behavior of the LMM when working with different basis functions. Furthermore, the GPU acceleration performance is analyzed both; on a single and multiple GPUs, where different load-balancing approaches are tested in order to minimize communication overheads. We show that the method is very suitable for implementation on GPU based systems. The numerical methodology used in this paper has been recently tested on much more demanding technological related problems [Vertnik and Šarler, (2006); Kosec, Založnik, Šarler and Combeau, (2011); Vertnik and Šarler, (2011)] and also in connection with Cellular Automata [Lorbiecka and Šarler, (2010)].

2 Local meshless method

The general idea behind local meshless method is the use of local sub clusters of discretization points, termed as local support domains, to construct an approximation of a considered field, which can be further manipulated with differential operators. The discretization points are often also referred to as computational nodes or just nodes. In general, there is no need to create a mesh or any kind of other topological relations between nodes. The nodes can be arbitrarily distributed within the computational domain and its boundary (see Figure 1).

The approximation function is constructed as a linear combination of basis functions

$$\theta(\mathbf{p}) = \sum_{n=1}^{N_b} \alpha_n \Psi_n(\mathbf{p}), \tag{1}$$

where N_b, α_n and Ψ_n, $\mathbf{p}(p_x, p_y)$ stand for the number of basis functions, the approximation coefficients, the basis functions and the position vector, respectively. The selection of basis functions is general; however, the most used basis functions are Hardy's multiquadrics (MQ)

$$\Psi_n^{MQ}(\mathbf{p}) = \sqrt{\frac{\|\mathbf{p}\|_n^2}{r_0 {\sigma_C}^2} + 1}, \tag{2}$$

Gaussians

$$\Psi_n^{G}(\mathbf{p}) = \exp\left(-\frac{\|\mathbf{p}\|_n^2}{r_0 {\sigma_C}^2}\right), \tag{3}$$

and monomials

$$\begin{aligned}\Psi_n^{M}(\mathbf{p}) = [1, \frac{p_x - p_x^0}{r_0}, \frac{p_y - p_y^0}{r_0}, \left(\frac{p_x - p_x^0}{r_0}\right)^2, \ldots \\ \ldots \left(\frac{p_y - p_y^0}{r_0}\right)^2, \left(\frac{p_y - p_y^0}{r_0}\right)\left(\frac{p_x - p_x^0}{r_0}\right), \ldots]\end{aligned}, \tag{4}$$

with

$$\|\mathbf{p}\|_n^2 = (\mathbf{p} - \mathbf{p}^n) \cdot (\mathbf{p} - \mathbf{p}^n), \tag{5}$$

and σ_C, $\mathbf{p}^n$, r_0 standing for free shape parameter, position vector of *n*-th support domain node and support normalization radius (Figure 1), respectively. The introduced free parameter σ_C can be used to shape the basis functions and their influence on the approximation function.

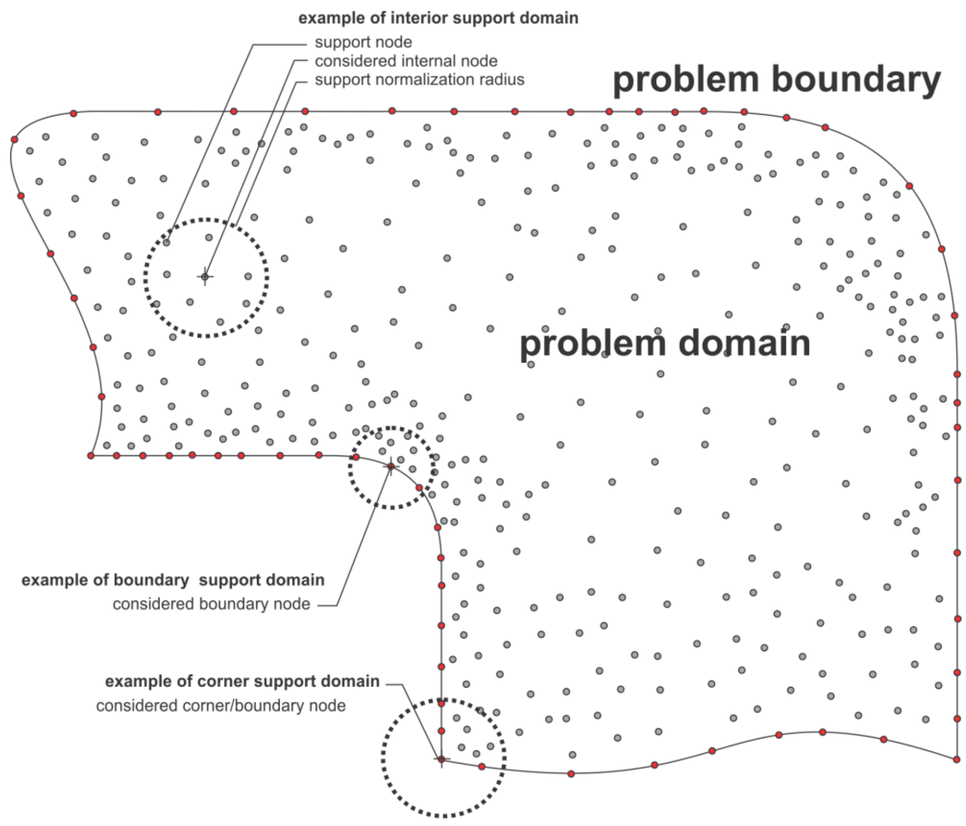
example of interior support domain
support node
considered internal node
support normalization radius
problem boundary
problem domain
example of boundary support domain
considered boundary node
example of corner support domain
considered corner/boundary node

Figure 1: The scheme of local meshless principle.

The stability of the approximation is characterized through the condition number, which can be controlled with the basis shape parameter $\left(\sigma_C\right)$. It has been shown, by numerical experiments, that using more sensitive interpolation system results in better accuracy [Lee, Liu and Fan, (2003)]. The phenomenon has been also theoretically confirmed by Schaback [Schaback, (1995)], who showed the uncertainty relation between attainable error and the condition number of the system. However, the condition number of the interpolation matrix should be kept below a critical value. The interpolation system becomes too ill conditioned for practical computations roughly at condition number over 10^{14}. With the approximation function created, the arbitrary spatial differential operation L can be applied on it

$$L\theta(\mathbf{p}) = \sum_{n=1}^{N_{Basis}} \alpha_n L\Psi_n(\mathbf{p}) \,. \tag{6}$$

The applicability of the differential operator on the approximation functions is limited only by the selection of the basis functions. If the operator can be applied on all basis functions then it can be applied also on the approximation function. In general, the approximation system has to be solved only at the beginning of the solution procedure or when the support domain topology changes, thus the computation can be optimized further. The computation of the coefficients and the evaluation of the differential operators can be combined in a single operation

$$L\theta(\mathbf{p}) = \sum_{n=1}^{N_S} \chi_m^L(\mathbf{p})\theta(\mathbf{p}_n)\,, \tag{7}$$

where the differential operator χ_m^L vector is introduced as

$$\chi_m^L(\mathbf{p}) = \sum_{n=1}^{N_B} \Psi_{nm}^{-1} L\left(\Psi_n(\mathbf{p})\right). \tag{8}$$

The introduced formalism holds in general and therefore the general notation for partial differential operator L is used. The formulation is convenient for implementation since most of the complex operations are performed in the pre-processing phase. During the simulation each numerical evaluation of an arbitrary operator L requires only N_s floating point operations. More details about the presented method can be found in [Kosec and Šarler, (2011)].

The temporal discretization is done through a two-level explicit time stepping

$$\frac{\partial \theta}{\partial t} = \frac{\theta - \theta_0}{\Delta t}, \tag{9}$$

where the zero-indexed quantities stand for the values at the previous time step and the time step is denoted with Δt.

One of the most convenient features of the presented meshless spatial discretization is its generality. One can change the number of basis functions, the shape of basis, the size of influence domain, position of computational points, etc, in order to treat numerical anomalies or problematic parts of the domain, without any kind of special treatment, either of the solution procedure or the program implementation. All information about the numerical scheme and topology of computational nodes is stored in partial differential vectors. Such an approach is also very suitable for GPU implementation since it is similar to filtering the image with spatially dependent filter.

3 GPU implementation

GPU accelerator hardware has been adopted widely in many areas of scientific, technical, and even financial computing. Current GPUs sport large numbers of Processing Elements (PE), usually in the range of 512 (e.g. NVIDIA GTX 580) up to some 2700 (e.g. NVIDIA Tesla K20) which have to be programmed in a highly data-parallel way. Processing elements are grouped into multi-processors in which teams of 32 PEs act as a Single Instruction/Multiple Data (SIMD) processor. Memory bandwidth on GPUs tends to be at least one order of magnitude larger than on traditional CPU-based systems, but memory latency is a big issue as a single access to on-board GPU-memory takes some 400 to 600 cycles whereas one floating point operation takes 1-2 cycles. To effectively harness the enormous computational capabilities of GPUs one is inclined to employ massively (data-) parallel algorithms which helps to hide memory latencies and thus fully utilize both the processing elements as well as available memory bandwidth.

We have implemented the presented LMM numerical solver for single-, dual-, and quad-GPU setups, all based on the Compute Unified Device Architecture (CUDA) for C. As necessary data structures for numerically feasible problem sizes fit into on-board GPU memory, we can copy relevant data to the GPU once and leave them there during a whole run.

In each time step the GPU kernel is invoked by the main (CPU-based) application. The GPU kernel multiplies the local partial differential vector and the values of the field in the local support domain (see equation (8)) in order to update the considered field. According to the explicit time stepping (9) the transfer of the “current” results into the “previous” time step is performed by alternating the pointers to “previous” and “current” time step data between consecutive kernel invocations.

Workloads are parallelized by assigning a single CUDA thread to each point of the array N_D. Current GPU hardware based on NVIDIA designs is structured as a collection of multi-processors (MP) that incorporate a multitude of processing elements (PE), which again form an entity called Warp. On our given hardware Warp-size is 32, that is, 32 PEs act as a single instruction/multiple data (SIMD) processor. PEs have access to both global and local memories, thus allowing distributed and shared memory programming paradigms at the same time. At the CUDA software layer the actual use of MPs and PEs is controlled by the so-called block-size. This block-size parameter reflects the number of threads which act in SIMD-mode on each multi-processor. Interestingly, its value may be much larger than the actual number of physical PEs. In conjunction with very low latency task switching and asynchronous memory transfers this setup allows the system to hide memory stalls in the physical PEs. In order to fully exploit the capabilities of a GPU, proper definitions of block-sizes is a key. If chosen too small, PEs will in general tend to stall, if chosen to large, other effects such as register-spilling might hinder top performance. In several tests with block-sizes being multiples of the Warp-size of 32 PEs a block-size of 128 CUDA threads showed best performance, thus the number of necessary CUDA blocks is set by $B = N/128$. On the NVIDIA pre-Kepler architectures the implementation is limited to $B = 65536$, thus resulting in up to 65536x128 array points.

3.1 Multiple GPU implementation

We have chosen a coarse grained parallelization scheme in which we simply divide the workspace N_D into a lower half $\left[0, N_D/2-1\right]$and an upper half $\left[N_D/2, N_D-1\right]$. Each GPU employs the same data structures as in the single GPU implementation but its workload is restricted to one of the two partitions. Each GPU also gets the memory addresses of the data its sibling has stored in its local memory. By means of the recently introduced CUDA Unified Virtual Addressing (UVA) mode, memory access to non-local memory is viable without any direct CPU intervention. This feature is important as it enables the kernel to access memory that is stored on another GPU in a very fine grained fashion. Practical experiments with CPU governed block memory transfers between two GPUs resulted in no speedup at all; therefore we have concluded that use of UVA is absolutely mandatory in order to gain speedups from multi-GPU setups. In the quad GPU implementation we employ a very similar scheme in which each of the four GPUs is responsible for the computation of a quarter instead of a half of the total workload. The scheme of the implementation is presented in Figure 2.

3.2 Optimization

In general, the LMM introduces scattered memory accesses, which are distributed unevenly over the workspace size N_D. These scattered accesses undermine naive load-balancing on multiple GPUs in which one would simply attribute evenly sized parts of workspace N_D to the participating GPUs. We approach this problem by choosing a coarse-grained partitioning of the total workspace with the intent to reflect this imbalance of memory access operations between the co-acting GPUs. In other words, we aim to hide the inherently varying numbers of memory access operations for evenly distributed partitions on multiple GPUs by introducing unevenly-sized workloads. Due to the complexity of the problem, which depends on multiple parameters such as GPU memory bandwidth/latency, cache hierarchies and also speed and type of the underlying host-PCI bus, a solution in closed form is not possible. Therefore we turn to an auto-tuning approach in which at least parts of the search space are being explored automatically.

We introduce an auto-tuning phase in which we check out all possible tilings t_n of two partitions P_0, P_1 that together cover the entire workspace $N_D = P_1 + P_0$, with $P_0 = [0, t_{n-1}]$, $P_1 = [t_n, N_D - 1]$ with $t_n \in [0, N_D - 1]$. The computational complexity grows linearly with workspace size N_D in the dual GPU setup. Each tiling t_n is benchmarked and the tiling corresponding to the minimal run time will be taken as the performance-optimal setup for all further computations on that given hardware setup. Clearly, the possibly time consuming auto-tuning phase needs to be run only once for the hardware setup and its results can be reused from then on. We observed typical run times for auto-tuning in the range of minutes ($N_D = 10^4$) to several hours ($N_D = 10^6$).

In the quad GPU setup we cannot explore the search space exhaustively anymore, as the computational complexity grows with $O(N_D^3)$. Therefore we turn to random sampling in a Monte Carlo fashion. Random partitions of the form $P_0[0, s_1]$, $P_0[s_1 + 1, s_2]$, $P_3[s_2 + 1, s_3]$ and $P_3[s_3 + 1, N_D - 1]$ with $s_1 < s_s < s_3 < N_D$ are generated, repeatedly tested and benchmarked. The tiling that corresponds to the minimal execution time is being preserved for further production runs of the solver. Auto-tuning obviously takes away the burden of proper workload-parameterization from the user and it should work well on heterogeneous setups of different (CUDA-enabled) GPU models, although this has not been in the scope of our practical experiments.

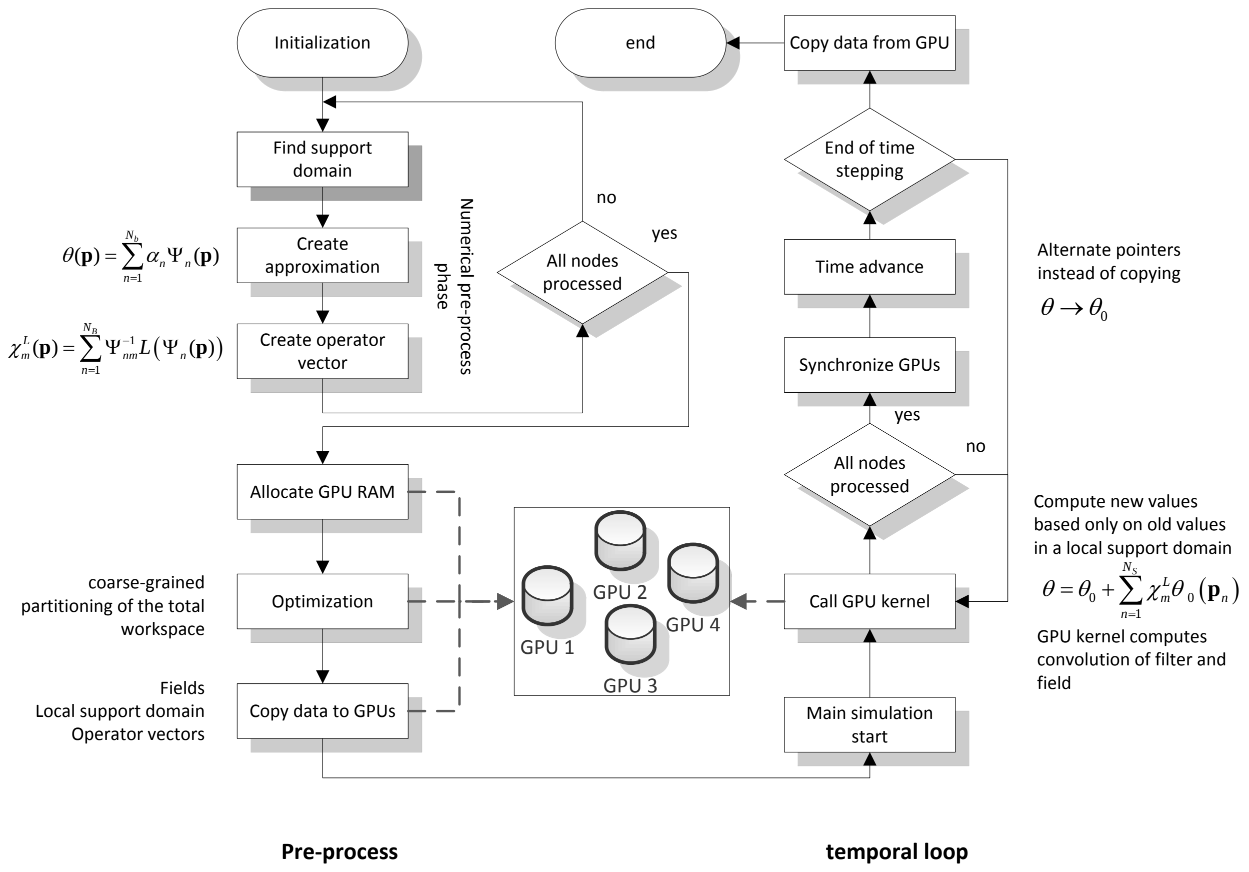
Initialization
end
Copy data from GPU
Find support domain
End of time stepping
no
yes
Numerical pre-process phase
$\theta(\mathbf{p}) = \sum_{n=1}^{N_b} \alpha_n \Psi_n(\mathbf{p})$
Create approximation
All nodes processed
Time advance
Alternate pointers instead of copying
$\theta \to \theta_0$
$\chi_m^L(\mathbf{p}) = \sum_{n=1}^{N_B} \Psi_{nm}^{-1} L(\Psi_n(\mathbf{p}))$
Create operator vector
Synchronize GPUs
yes
no
All nodes processed
Allocate GPU RAM
Compute new values based only on old values in a local support domain
$\theta = \theta_0 + \sum_{n=1}^{N_S} \chi_m^L \theta_0(\mathbf{p}_n)$
coarse-grained partitioning of the total workspace
Optimization
GPU 1
GPU 2
GPU 3
GPU 4
Call GPU kernel
GPU kernel computes convolution of filter and field
Fields
Local support domain
Operator vectors
Copy data to GPUs
Main simulation start
Pre-process
temporal loop

Figure 2: The scheme of implementation.

4 Test problem

The two dimensional second order PDE, describing the heat transfer phenomenon, is considered as a test problem

$$\frac{\partial T}{\partial t} = D\nabla^2 T \,. \tag{10}$$

We deal with the numerical implementation performance and thus the heat diffusion coefficient D is set to 1 and a square domain of dimensionless size 1×1 with Dirichlet boundary conditions and uniform initial conditions is considered. The initial dimensionless temperature is set to 1 and at all the boundaries to 0, i.e.

$$T(\mathbf{p}_\Omega, t=0) = 1 \,. \tag{11}$$

$$T(\mathbf{p}_\Gamma, t) = 0 \,, \tag{12}$$

with $\mathbf{p}_\Omega$ and $\mathbf{p}_\Gamma$ standing for interior and boundary nodes, respectively. Equation (10) with given boundary conditions (11) and initial state (12) can be evaluated in a closed form by a double series summation

$$T_a(\mathbf{p},t) = \sum_n \sum_m \frac{1}{\pi^2}\frac{16}{nm}\sin(\pi n p_x)\sin(\pi m p_y) e^{-\pi^2\left(n^2+m^2\right)t} \quad , \tag{13}$$

where m and n are odd integers, and T_a stands for a closed form solution of the temperature field. With the known closed form solution the numerical accuracy and stability is assessed through the maximal error defined as

$$E = \max\left(\left|T_a(\mathbf{p}_\Omega,t) - T(\mathbf{p}_\Omega,t)\right|\right), \tag{14}$$

where all domain nodes are taken into account. The symmetry of the problem can be exploited to assess the behavior of the method when working with Neumann boundary conditions by considering just one quarter of the domain. It is commonly accepted that the “quarter domain case” is more demanding due to the Neumann boundaries, however it has been showed in [Trobec, Kosec, Šterk and Šarler, (2012)] that this does not hold for LMM approach. The numerical error for the proposed test has a maximum at the beginning of the simulation where the steep gradients occur near boundaries. The intense time dynamics due to the high second derivatives of the temperature, together with the error from construction of the approximation function and its derivatives result in high errors. However, the error diminishes rapidly with time and therefore a suitable simulation time, where the results are to be analyzed, has to be chosen in order to get as much information about the behavior of the method as possible. With this in mind the dimensionless time $t = 0.005$ is chosen for further analysis. The same time for analysis is chosen in [Trobec, Kosec, Šterk and Šarler, (2012)]. Scheme of the problem and its closed form solution are presented in Figure 3.

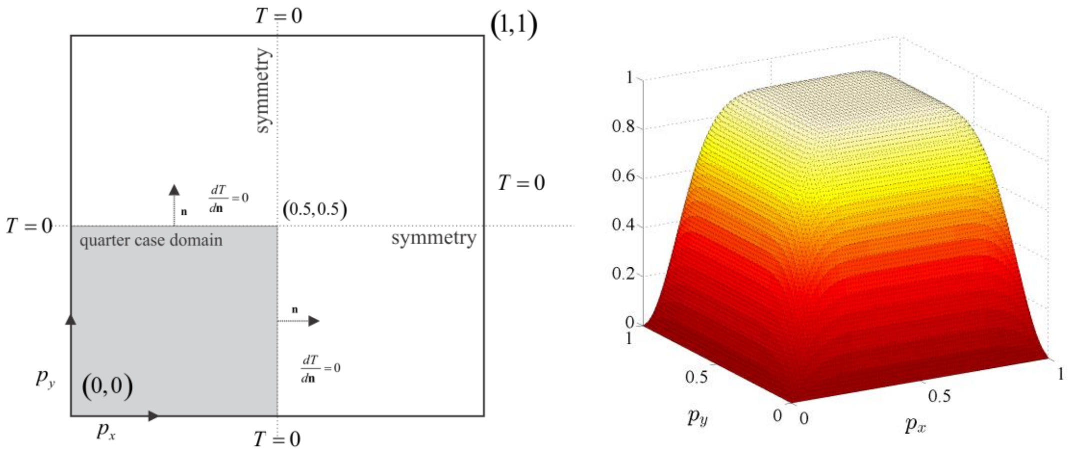

Figure 3: Scheme of the test problem (left) and closed form solution at t=0.005 (right).

5 Results

5.1 The convergence behavior

Since the presented diffusion problem has been already analyzed in the past, we focus only on topics related to the present research. One of the main messages of this paper is the analysis of the solution procedure on the multiple GPU, which are designed to process huge amounts of floating point operations. In order to fully exploit the power of GPUs the number of computational nodes has to be as high as possible, but before we can move to the analysis of the GPU efficiency we have to analyses numerical solution in such computational regimes. For the sake of readability of the paper we will limit some LMM parameters. In this paper we use the simplest 5 nodded support domain with a collocation approach. When the RBFs, i.e. the MQs or Gaussians, are used for a basis the methods is known as LRBFCM. There are several papers proposing the MQs are suitable basis selection, starting with Kansa in [Kansa, (1990)] and latter in [Zhang, Song, Lu and Liu, (2000); Young, Jane, Lin, Chiu and Chen, (2004); Vertnik and Šarler, (2011)]. In this paper we extend the tests with the computations based on the monomial basis. In Figure 4 convergence rate for calculations with different basis functions are presented. In the left plot, the MQs, Gaussians and monomials are compared; in the right plot influence of MQ shaping is presented. The shaping of the basis is directly connected with the conditioning of the approximation system, the higher is the shape parameter (σ_C) the higher is the condition number of the local approximation system. On the first plot it can be seen that all three setups perform well as long as the number of nodes is "small" (up to $N_D = 10^5$), which is also the case in [Kansa, (1990); Zhang, Song, Lu and Liu, (2000); Young, Jane, Lin, Chiu and Chen, (2004); Vertnik and Šarler, (2011)]. For higher number of nodes the RBF based computations start losing accuracy. From the right plot it is evident that the effect is much more pronounced when low σ_C is used or in another words, with good conditioned systems the results are poorer. Increasing the σ_C and consequently the condition number improves the convergence behavior. On the other hand, too high σ_C results in ill-conditioned systems, and consequently unstable divergent numerical solution. The monomial based computations are stable on a whole interval. Based on the results we decide to use simple monomials without any free parameters as such configuration shows good convergence rate. The monomials are, from the computational point of view, much simpler to compute and do not require any kind of special treatment, such as system conditioning, to achieve good results.

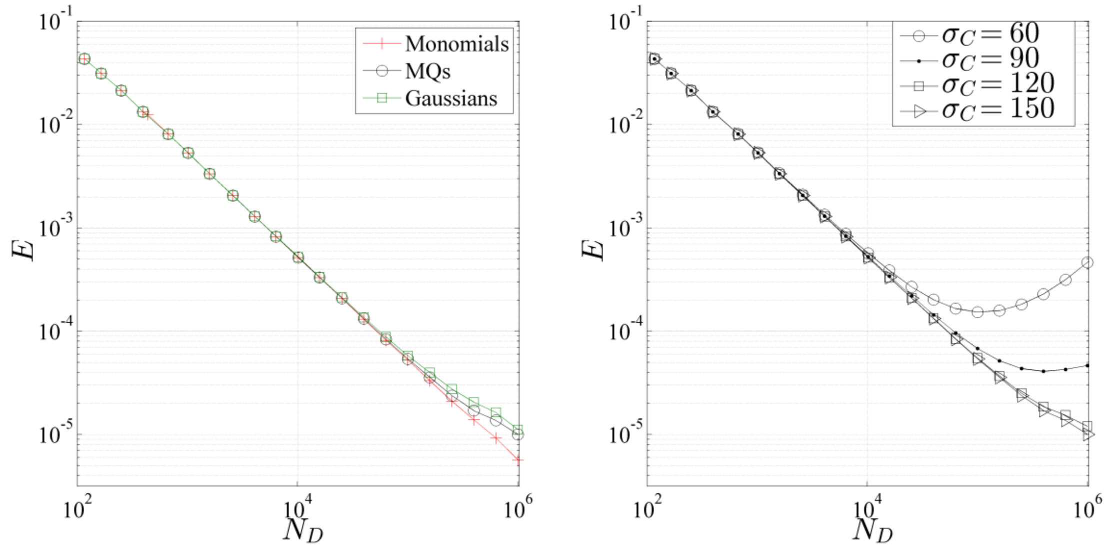

Figure 4: Convergence rate for different types of basis functions with $\sigma_C = 150$ (left) and convergence rate with MQ basis for different shape parameters.

Next analysis is focused on the time step effect on the convergence rate. Previous analysis has been done with time step $dt = 10^{-8}$. To create a better insight on the topic we also test the influence of the data type used in the simulations, as it is generally accepted that GPUs perform much better with a single precision data type (32 bit) in comparison to double precision (64 bit) data type. However, before we can speculate about impact of the computational precision on the performance, the impact on the accuracy should be discussed. In Figure 5 convergence plots regarding the data type and temporal discretizations are presented. The conditional stability of the explicit stepping can be recognized on both plots; sudden extreme increase in error indicates the unstable solution. It is evident that for computations with more than $N_D = 10^5$ nodes and time step $\Delta t = 10^{-6}$ diverges, similar for $N_D = 10^{6.2}$ even $\Delta t = 10^{-7}$ does not suffice anymore. The behavior is in accordance to the expectations since the LMM can be understood as a meshless generalization of the FDM. In the present context, where 5 nodded regular support domains are used, the stability problems should be similar to FDM. Second message from Figure 5 is focused on the impact of data type on the accuracy. It is well known that accuracy of computer simulation is governed by interplay of rounding and truncation error [Heath, (2002)]. Refining the spatial discretization causes the reduction of the truncation error, but on the other hand it increases the rounding error. It is also important to note that with refining the temporal discretization the number of iterations is increased and consequently the rounding error is increased. All this is confirmed in Figure 5. The single precision computations cover the simulations with time step down to $\Delta t = 10^{-6}$ and spatial discretizations up to $N_D = 10^5$. The finer discretizations require finer time step, but with finer time step, the rounding error becomes more important than the truncation error. Double precision data type, conversely, covers much higher interval and the rounding limit is not reached anymore in our present analysis. Slight decrease in the convergence rate in a high N_D is not a consequence of a rounding error. The rate drops due to the numerical limitations of the solution of the meshless approximation system, but this is already out of the scope of the present paper.

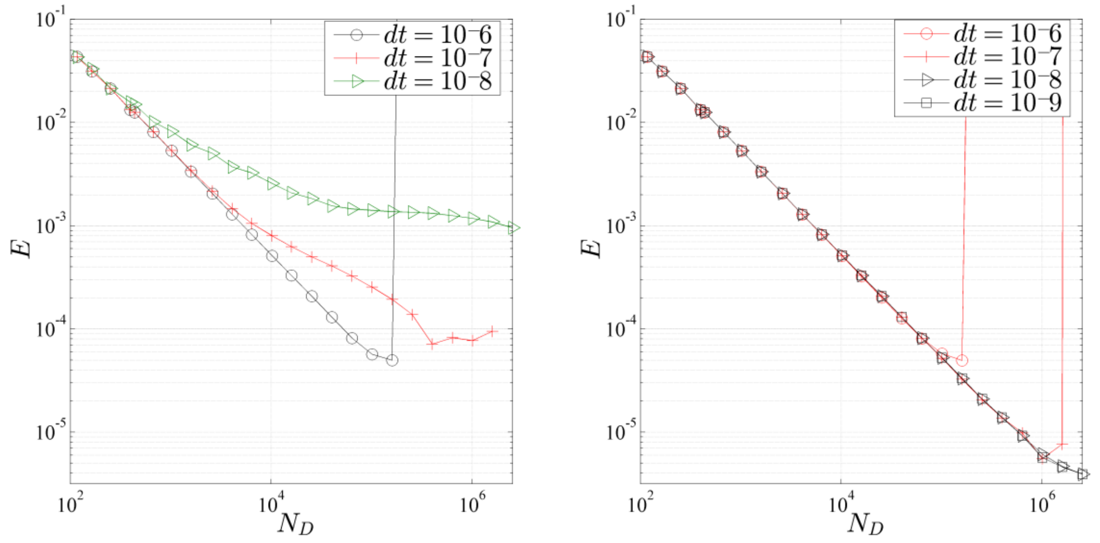

Figure 5: Convergence rate for different time steps and data types. Left plot presents the results computed with 32 bit and right plot with 64 bit data type.

5.2 Execution performance

In previous section the accuracy and stability of the numerical integration have been covered, whereas the present section focuses on the computational efficiency. The results of the execution performance analysis are presented in terms of computational times and speedups regarding the CPU execution. We use the following computer setup for testing: SuperMicro server with dual AMD Opteron(TM) Processor 6274, 2.2 GHz, 128 GB RAM, NVIDIA Tesla S2050 GPU (4 GPUs), RocksCluster 6.0 Linux, and NVIDIA CUDA 5.0. Although the multicore CPU is used for the computations, only single core computations are used for a reference, since the memory architecture of the test machine introduces several anomalies (accumulating caches, bandwidth limitation, etc.) in the CPU execution timings which would severely distort the speedup representation if all cores would be involved. The resulting performance from CPU-only code (Figure 6, left diagram) is directly proportional to the number of nodes. The differences between single and double precision arithmetic is negligible. In the single-GPU case we observe two different regimes of execution times. In the first regime all the data needed for the computations fits into 2nd level GPU cache which can be accessed with maximum speed by the processing elements operating in SIMD lockstep fashion of groups of 32 PEs. Hence, it makes no significant difference whether a data sets occupies only one third or all of the cache. For larger data sets which do not fit entirely into the caches, an increase of execution times can be seen that is solely governed by the number of nodes, which is an indication for the problem to be memory- rather than compute-bound. The timings are done for 5000 time steps, where we measured the whole process presented in Figure 2, except the numerical pre-process, which has been computed in advance and stored as a set of filters.

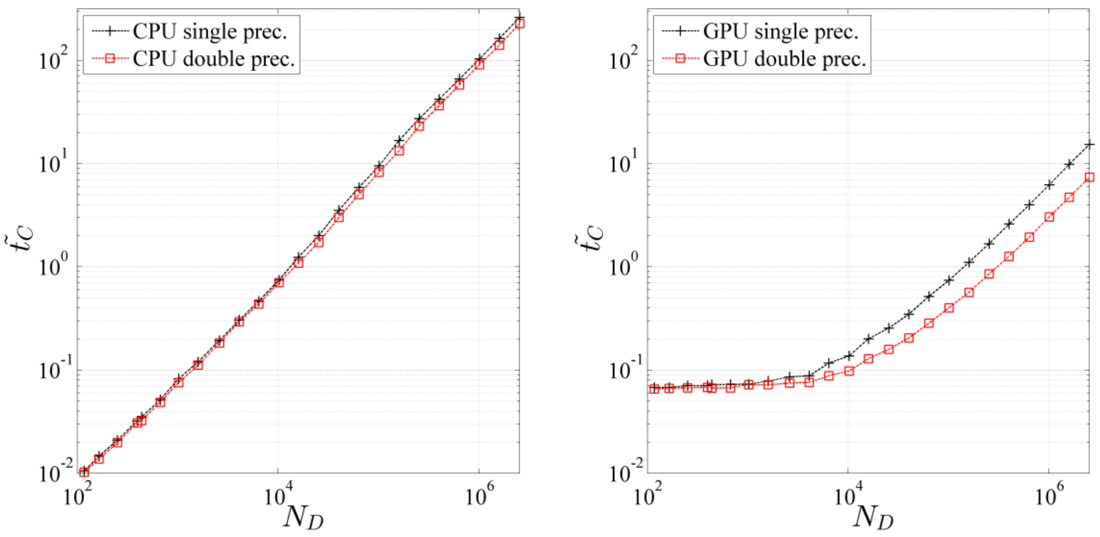

Figure 6: CPU timings and GPU timings

Unified Virtual Addressing (UVA) allows seamless integration of multiple GPUs, but use of the underlying PCI-bus introduces additional latency and bandwidth limitations for inter-GPU communications. In our case this performance-adverse effect can be overcome by choosing increased problem sizes, as can easily be seen in Figure 7, where data sets beyond $N_D = 10^{5.5}$ points clearly benefit from the use of two and four GPUs.

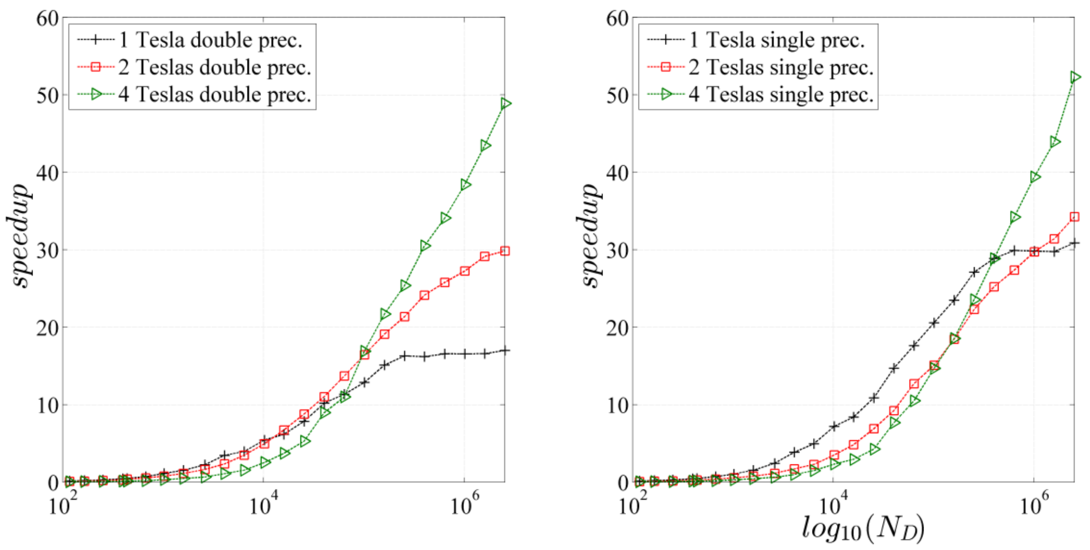

Figure 7: GPU execution speedup regarding the CPU execution.

Figure 8 displays the achieved additional speedups which are governed by the optimization techniques we discussed in section 3.2. Auto-tuning in a setup of four GPUs (upper curve) typically yields better speedup than in a setup of only two GPUs. This is a result of the higher degree of freedom (DOF) that the auto-tuning algorithm is able to exploit, as we see a DOF=1 in two GPUs acting together versus a DOF=3 in a setup of four GPUs.

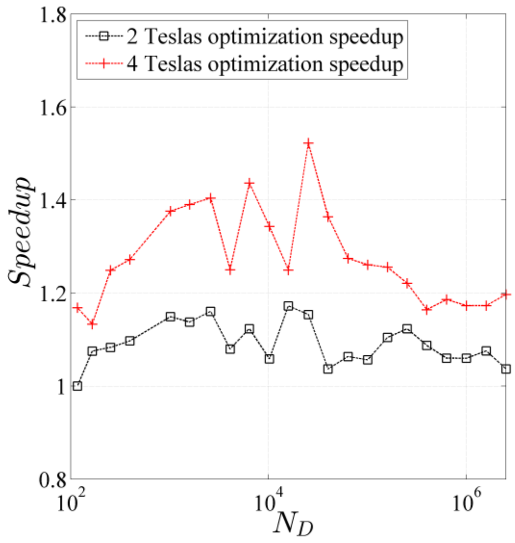

Figure 8: Optimized GPU execution speedup regarding the straightforward GPU execution.

Our method for auto-tuning proper work distributions for four GPU setups and optimal work distributions for two GPUs, shows rather high variations in the gained speedups (Figure 8). This is not being caused by the type of the tuning algorithm but it is a consequence of the high variation of the distribution of support domain nodes within the support domain. Auto tuning of the load results in additional speedups against the standard GPU algorithm in the range of 10-40%.

6 Conclusions

In the present paper we have introduced a convenient formulation of the LMM and have analyzed its behavior on the GPU accelerated computer system. The implementation has been tested on a well-known diffusion problem and since the problem has a closed form solution an accurate convergence analysis has been done. We have shown that the LMM based on the RBFs experiences problems if a high number of nodes is involved; however, changing the basis to monomials or using almost ill-conditioned approximation improves the performance of the method. The performance could be also improved by adding monomials to the RBF basis, i.e. augmentation of the basis, but this introduces unnecessary complications and computational overheads. The convergence has been also tested with respect to the time step where instabilities due to the explicit stepping are clearly identified. As expected the five nodded LMM behaves similar to the explicit FDM. Using wider support domain with more nodes would ease the stability criterion but would also introduce severe computational overheads. In addition, we have presented convergence with respect to the data type used for computations, where the interplay between rounding and truncation error, which play an important role, especially in cases of high number of nodes, has been clearly shown.

We have shown that LMM is amenable for computation on modern GPU hardware with speedups in the range of 50 compared to a server class Opteron CPU single core execution. Furthermore, we have shown that additional high speedups can be gained from multi-GPU setups. This result seems quite counter-intuitive since the algorithm is memory bound due to its high degree of small, scattered memory access operations, which generally pose a difficulty for GPU memory subsystems in terms of the high inherent access latency of some 400-600 cycles. Thanks to peer-to-peer computing and unified virtual addressing modes in CUDA 4 implementation gets a lot easier and performance becomes very competitive compared to the CPU. Scattered and unevenly distributed data access imposes imbalances of the load which is an obstacle towards high speedups. By automatically tuning proper workload distribution of the parallel implementation for two and four GPUs, we have observed additional speedups. The high variances in the additional speedups achieved by auto-tuning are an indication of the high complexity of the optimization problem. The absolute speedup gained from auto-tuning has been found to be in the range of 10-40%, which obviously can be helpful when it comes to the repeated solution of large problems. Auto-tuning of workload sizes on multiple GPUs hence proves to be a valuable tool, especially when we recall that the described auto-tuning phase can be implemented easily and has to be executed only once for a given set of nodes.

Acknowledgment

We acknowledge the financial support from the state budget by the Slovenian Research Agency under the grants P2-0095. We are also grateful to professor Božidar Šarler for expert assistance and helpful discussions.

7 References

Abouali, M.; Timmermans, J.; Castillo, J. E.; Su, B. Z. (2013): A high performance GPU implementation of Surface Energy Balance System (SEBS) based on CUDA-C. *Environmental Modelling & Software*, vol. 41, pp. 134–138.

Arefmanesh, A.; Najafi, M.; Musavi, S. H. (2013): Buoyancy-driven fluid flow and heat transfer in a square cavity with a wavy baffle—Meshless numerical analysis. *Engineering Analysis with Boundary Elements*, vol. 37, pp. 366–382

Atluri, S. N. (2004): *The Meshless Method (MLPG) for Domain and BIE Discretization*, Tech Science Press, Forsyth.

Atluri, S. N.; Shen, S. (2002a): The meshless local Petrov-Galerkin (MLPG) method: a simple & less-costly alternative to the finite element and boundary element methods. *CMES: Computer Modeling in Engineering & Sciences*, vol. 3, pp. 11-52.

Atluri, S. N.; Shen, S. (2002b): *The Meshless Method*, Tech Science Press, Encino.

Bollig, E. F.; Flyer, N.; Erlebacher, G. (2012): Solution to PDEs using radial basis function finite-differences (RBF-FD) on multiple GPUs. *Journal of Computational Physics* vol. 231, pp. 7133-7151.

Bourantas, G. C.; Skouras, E. D.; Loukopoulos, V. C.; Nikiforidis, G. C. (2010): Meshfree point collocation schemes for 2d steady state incompressible navier-stokes equations in velocity-vorticity formulation for high values of reynolds number. *CMES - Computer Modeling in Engineering and Sciences*, vol. 59, pp. 31-63.

Bustamante, C. A.; Power, H.; Sua, Y. H.; Florez, W. F. (2012): A global meshless collocation particular solution method (integrated Radial Basis Function) for two-dimensional Stokes flow problems. *Applied Mathematical Modelling*, vol. pp.

De Chowdhury, S.; Sannasiraj, S. A. (2013): SPH Simulation of shallow water wave propagation. *Ocean Engineering*, vol. 60, pp. 41–52

De Vuyst, F.; Salvarani, F. (2013): GPU-accelerated numerical simulations of the Knudsen gas on time-dependentdomains. *Computer Physics Communications*, vol. 184, pp. 532-536.

Duan, Y.; Rong, F. (2013): A numerical scheme for nonlinear Schrödinger equation by MQ quasi-interpolation. *Engineering Analysis with Boundary Elements*, vol. 37, pp. 89–94.

Fasshauer, G. (2006): Radial basis functions and related multivariate meshfree approximation methods: Theory and applications - Preface. *Computers & Mathematics with Applications*, vol. 51, pp. 1223-1366.

Gingold, R. A. ; Monaghan, J. J. (1977): Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Mon. Not. Roy. Astron. So*, vol. 181, pp. 375-389.

Gu, G.R. Liu; Y.T. (2005): *An Introduction to Meshfree Methods and Their Programming*, Springer, Dordrecht.

Guo, Y. M. (2011): An overrange collocation method. *CMES - Computer Modeling in Engineering and Sciences*, vol. 73, pp. 1-22.

Heath, M. T. (2002): *Scientific Computing*, McGraw-Hill, New York.

Kansa, E. J. (1990): Multiquadrics - a scattered data approximation scheme with application to computational fluid dynamics, part I. *Computers & Mathematics with Applications*, vol. 19, pp. 127-145.

Kirk, D. B.; Hwu, W. W. (2010): *Programming Massively Parallel Processors*, Morgan Kaufmann Publishers, Burlington.

Kosec, G.; Šarler, B. (2011): H-adaptive local radial basis function collocation meshless method. *CMC: Computers, Materials, & Continua*, vol. 26, pp. 227-253.

Kosec, G.; Založnik, M.; Šarler, B.; Combeau, H. (2011): A Meshless Approach Towards Solution of Macrosegregation Phenomena. *CMC: Computers, Materials, & Continua*, vol. 580, pp. 1-27.

Lee, C. K. ; Liu, X.; Fan, S.C. (2003): Local muliquadric approximation for solving boundary value problems. *Computational Mechanics*, vol. 30, pp. 395-409.

Li, Xiaolin (2012): Application of the meshless Galerkin boundary node method to potential problems with mixed boundary conditions. *Engineering Analysis with Boundary Elements*, vol. 36, pp. 1799–1810.

Lorbiecka, A. Z.; Šarler, B. (2010): Simulation of dendritic growth with different orientation by using the point automata method. *CMC: Computers, Materials, & Continua*, vol. 18, pp. 69-103.

Molero, M.; Iturrarán-Viveros, U. (2013): Accelerating numerical modeling of wave propagation through 2-D anisotropic materials using OpenCL. *Ultrasonics*, vol. 53, pp. 815-822.

Rinaldi, P.R. ; Dari, E.A.; Vénere, M.J.; Clausse, A. (2012): A Lattice-Boltzmann solver for 3D fluid simulation on GPU. *Simulation Modelling Practice and Theory*, vol. 25, pp. 163-171.

Schaback, R. (1995): Error estimates and condition numbers for radial basis function interpolation. *Advances in Computational Mathematics*, vol. 3, pp. 251-264.

Stevens, D.; Power, H. (2010): A scalable meshless formulation based on RBF hermitian interpolation for 3D nonlinear heat conduction problems. *CMES - Computer Modeling in Engineering and Sciences*, vol. 55, pp. 111-145.

Šarler, B. (2005): A radial basis function collocation approach in computational fluid dynamics. *CMES: Computer Modeling in Engineering & Sciences*, vol. 7, pp. 185-193.

Šarler, B. (2007): From global to local radial basis function collocation method for transport phenomena, *Advances in Meshfree Techniques*, Springer, Berlin, pp. 257-282.

Techapiroma, T.; Luadsonga, A. (2013): Improved weight function in mlpg method for the two-dimension diffusion equation. *Far East Journal of Mathematical Sciences*, vol. 72, pp. 175-189.

Trobec, R.; Kosec, G.; Šterk, M.; Šarler, B. (2012): Comparison of local weak and strong form meshless methods for 2-D diffusion equation. *Engineering Analysis with Boundary Elements*, vol. 36, pp. 310-321.

Trobec, R.; Šterk, M.; Robič, B. (2009): Computational complexity and parallelization of the meshless local Petrov-Galerkin method. *Computers & Structures*, vol. 87, pp. 81-90.

Vertnik, R.; Šarler, B. (2006): Meshless local radial basis function collocation method for convective-diffusive solid-liquid phase change problems. *International Journal of Numerical Methods for Heat & Fluid Flow*, vol. 16, pp. 617-640.

Vertnik, R.; Šarler, B. (2011): Local collocation approach for solving turbulent combined forced and natural convection problems. *Advances in Applied Mathematics and Mechanics*, vol. 3, pp. 259-279.

Wang, C. A.; Sadat, H.; Prax, C. (2012): A new meshless approach for three dimensional fluid flow and related heat transfer problems. *Computers and Fluids*, vol. 69, pp. 136-146.

Wang, J. G.; Liu, G. R. (2002): A point interpolation meshless method based on radial basis functions. *International Journal for Numerical Methods in Engineering*, vol. 54, pp. 1623-1648.

Wen, Pi Hua; Aliabadi, M. H. (2012): Meshless Local Integral Equation Method with Analytical Formulation and its Application to Computational Fracture Mechanics. *Key Engineering Materials*, vol. 488, pp. 791–794.

Yokota, R.; Narumi, T.; Sakamaki, R.; Kameoka, S.; Obi, S.; Yasuoka, K. (2009): Fast multipole methods on a cluster of GPUs for the meshless simulation of turbulence. *Computer Physics Communications*, vol. 180, pp. 2066–2078.

Young, D. L.; Jane, S. C.; Lin, C. Y.; Chiu, C. L.; Chen, K. C. (2004): Solutions of 2D and 3D Stokes laws using multiquadrics method. *Engineering Analysis with Boundary Elements*, vol. 28, pp. 1233-1243.

Zhang, X.; Song, K. Z.; Lu, M. W.; Liu, X. (2000): Meshless methods based on collocation with radial basis functions. *Computational Mechanics*, vol. 26, pp. 333–343.

Zhang, Y.; Sim, T.; Tan, C. L.; Sung, E. (2006): Anatomy-based face reconstruction for animation using multi-layer deformation. *Journal of Visual Languages and Computing*, vol. 17, pp. 126-160.