GPU accelerated RBF-FD solution of Poisson's equation

Mitja Jančič[1,2], Jure Slak[1,3], Gregor Kosec[1]
[1] "Jožef Stefan" Institute, Parallel and Distributed Systems Laboratory, Ljubljana, Slovenia
[2] "Jožef Stefan" International Postgraduate School, Ljubljana, Slovenia
[3] Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia
mitja.jancic@ijs.si, jure.slak@ijs.si, gregor.kosec@ijs.si

Abstract – The Radial Basis Function-generated finite differences became a popular variant of local meshless strong form methods due to its robustness regarding the position of nodes and its controllable order of accuracy. In this paper, we present a GPU accelerated numerical solution of Poisson's equation on scattered nodes in 2D for orders from 2 up to 6. We specifically study the effect of using different orders on GPU acceleration efficiency.

I. Introduction

In contrast to the traditional numerical methods for solving partial differential equations (PDE) that require connectivity between discretization nodes, meshless methods can operate on scattered nodes. Although seemingly minimal difference, this feature made meshless methods popular [1] in various fields of science and engineering ranging from computational fluid dynamics [2, 3, 4] to option pricing [5]. The key to becoming so popular is that discretization of arbitrary domain with scattered nodes is considered much easier problem in comparison with meshing. To some degree it can also be automated in dimensionless sense [6].

From a historical point of view, meshless methods were introduced in 1990s. Since then, many meshless methods have been developed, e.g. the Element Free Galerkin Method [7], the Diffuse Element Method [8], the Partition of Unity Method [9], the Local Petrov-Galerkin Methods [10], the h-p Cloud Methods [11], etc.

The radial basis function-generated finite differences (RBF-FD) was first mentioned in [12] as a local strong form meshless method for solving PDEs. The method approximates differential operators only using scattered nodes. However, often used RBFs, e.g. Gaussians, include a shape parameter that can be crucial to the overall method stability [13]. Stability issue has been recently addressed by using Polyharmonic splines (PHS) and additionally augmenting them with polynomials [14].

The generality of the meshless methods comes with the price, namely higher complexity. First, the shape functions of stencil weights have to be computed every time from scratch. Second, the support sizes are typically much bigger than in mesh-based methods, especially in high order RBF-FD methods. The natural way to accelerate most computationally demanding parts of the solution procedure is to employ parallel computing. There have been several reports on parallel meshless solution procedures in the past, including distributed computing as well as shared memory approaches [4, 15, 16, 17, 18]. In this paper we analyze the graphics processing unit (GPU) acceleration of RBF-FD explicit solution of Poisson problem with Dirichlet boundary condition. We are especially interested in the effect of using different polynomial orders.

The rest of the paper is organized as follows: in section II a short presentation of local meshless methods is given, in section III we present our workflow and how GPU was included in our computations, in section IV problem is explained, in section V results are presented, and in section VI final conclusions are given.

II. Local strong form meshless methods

In general, the idea of meshless methods is the use of local discretization points to construct an approximation of the considered field and later use it for manipulation with differential operators. Discretization points are referred to as computational nodes or just *nodes*. The nodes are placed within the domain and on its boundary, and their distribution can be scattered, as presented in Fig. 1.

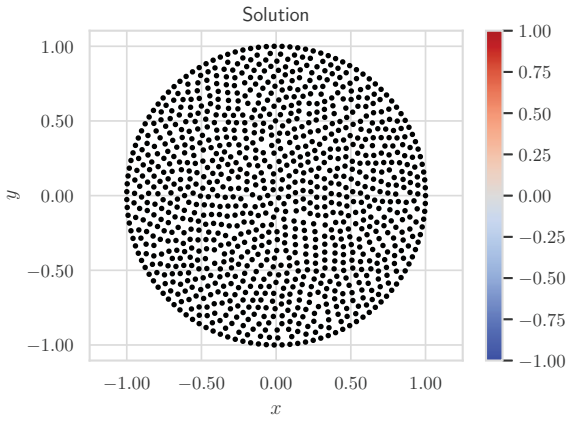

Figure 1. Solution of Poisson's problem with Dirichlet boundary conditions on scattered 2D nodes. Chosen highest polynomial degree $m = 2$, support size $n = 15$ and number of nodes $N = 1027$.

Local strong form meshless methods approximate derivatives in the form

$$(\mathcal{L}u)(x) \approx \sum_{p \in N(x)} w_i^{\mathcal{L}}(x) u(p), \qquad (1)$$

where $N(x)$ is a set of neighboring nodes of x and $\mathcal{L}$ is a differential operator. Different ways of computing the weights w exists. We will use the RBF-FD method [19].

We have implemented the RBF-FD based solution procedure using object oriented approach and C++'s strong template system. Node positioning, support selection, differential operator approximation and PDE discretization and other modules are all available in one package, the Medusa library [20]. Please refer to our open source Medusa library for more features and examples.

III. GPU IMPLEMENTATION

Compared to a CPU-based system, the memory bandwidth in GPU based systems is often an order of magnitude higher, however at the price of higher memory latency. A single access to on board GPU memory takes approximately 400-600 cycles compared to a floating point operation which takes approximately 1-2 cycles. Current GPUs support large numbers of processing elements and must be programmed in a high data-parallel way in order to observe the advantages of GPU programming and to help hide the memory latency. Large data is also needed to keep the processing units busy and good knowledge of GPU systems is required to reduce the memory communication and optimize the kernel functions [17].

In this work only part of the problem was dumped to the GPU. The illustration of the GPU implementation is in Fig. 2. Green boxes are specific to GPU programming while the yellow box presents the actual calculations done on the GPU. Shapes, node positions and support nodes were still tasks executed by the CPU, only the explicit time loop (yellow box in Fig. 2) was ported to the GPU. Time loop execution was also timed using high resolution timer, enabling us to estimate the speedups and thus evaluate the performance of CPU and GPU.

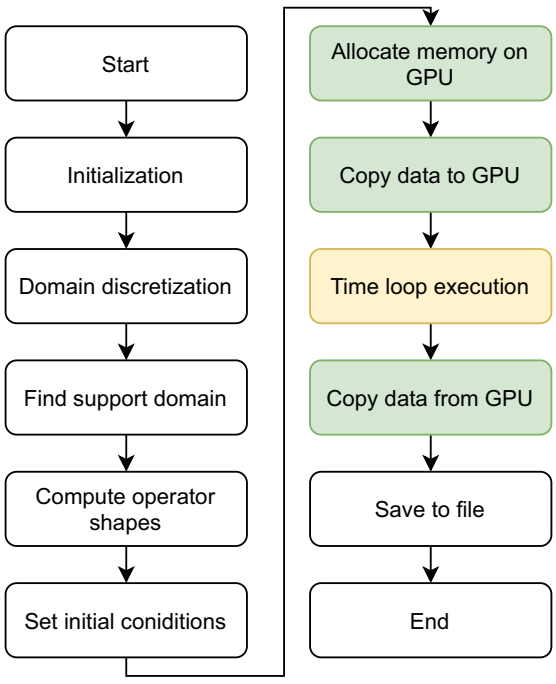

Figure 2. The implementation scheme.

A. Time loop implementation

Once the shapes, node positions and support nodes computation was carried out on the host (CPU), memory was allocated on the device (GPU) using `cudaMalloc` memory allocation function. As necessary data structures for numerically feasible problem sizes on the onboard GPU memory, we simply copied relevant data to the host using the `cudaMemcpy` function with `cudaMemcpyHostToDevice` argument and left the data there during time loop execution.

Time loop computation on the GPU was executed by introducing a `solveOnGpu` kernel function as shown in code listing 1.

```
int tpb = 32;
int bpg = (N + tpb − 1) / tpb;

for (int step = 0; step < timeSteps; ++step) {
  solveOnGpu <<<bpg, tpb>>> (d_u1, d_u2, ...);
  cudaMemcpy(d_u1, d_u2, sizeof(double) ∗ N,
                  cudaMemcpyDeviceToDevice);
}
```

Listing 1: Sample code of time loop execution on GPU.

In Compute Unified Device Architecture (CUDA) the actual use of multi-processors and processing units is controlled by block size [21]. This reflects in the number of threads which act on each processor. In order to fully exploit the benefits of a GPU, proper definition of block size is of great importance. When chosen to small, processing units tend to stall, if chosen to large other effects (e.g. register spilling) might interfere with top performance [17].

Several tests of block size of multiples of 32 were made. Best time performance showed at $N_{tpb} = 32$ threads per block and $N_{bpg} = (N+N_{tpb}-1)/N_{tpb}$ blocks per grid with N nodes in the domain.

Once the time loop execution on the GPU is finished, `cudaDeviceSynchronize` function is used for synchronization and results are copied from the device to host using `cudaMemcpy` function with the `cudaMemcpyDeviceToHost` argument. The results are saved to a file and post-processing is done with python 3. To prevent memory leakage, `cudaFree(varibaleName)` was used before the shutdown.

IV. PROBLEM DESCRIPTION

With the aim to analyze the performance of CPU and GPU in terms of support size and highest augmented polynomial degree, proposed solution procedure and its implementation is studied on a Poisson problem with Dirichlet boundary conditions:

$$\frac{\partial}{\partial t}u(\boldsymbol{p}) - \nabla^2 u(\boldsymbol{p}) = f(\boldsymbol{p}) \qquad \text{in } \Omega, \qquad (2)$$

$$u(\boldsymbol{p}) = \prod_{i=1}^{d} \sin(\pi_i) \qquad \text{on } \partial\Omega, \qquad (3)$$

where $f(\boldsymbol{p}) = d\pi^2 \prod_{i=1}^{d} \sin(\pi p_i)$ and domain space Ω is a $d = 2$ dimensional unit disk with boundary $\partial\Omega$.

The problem is discretized in both time and spatial domain space

$$u_2(\boldsymbol{p}) = u_1(\boldsymbol{p}) + \mathrm{d}t(f(\boldsymbol{p}) + \nabla^2 u_1(\boldsymbol{p})), \qquad (4)$$

where $\mathrm{d}t$ is an infinitesimal time step and ∇^2 is approximated as described in section II.

The closed form solution of the above problem is $u(\boldsymbol{p}) = \prod_{i=1}^{d} \sin(\pi p_i)$ allowing us to validate the numerically obtained solution.

V. Results

Numerical results are computed using RBF-FD with PHS radial basis functions $\Phi(r) = r^3$ and monomial augmentation. Radial function was kept the same for all cases. The convergence order is directly controlled by the highest augmented polynomial degree $m \in \{2, 4, 6\}$, however the larger the polynomial degree, the larger the recommended support size $n = \binom{m+d}{m}$. Recommended support size in terms of domain space dimensionality and highest polynomial degree was first given by Bayona [14], however larger supports can also be used [22]. In our test $n \in \{12, 15, 20, 30, 45, 60\}$.

All computations were performed on a single core of a computer with `Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz` processor and 64 GB of DDR4 memory and graphical processing unit `NVIDIA GeForce RTX 2080 Ti, 11GB GDDR6` with `4352 CUDA cores`. Code was compiled using `g++ (GCC) 8.1.0` on Linux with `-O3 -DNDEBUG -std=c++11` flags while GPU code was compiled using `CUDA release V9.1.85`.

An example of numerical solution is shown in Fig. 1.

A. Execution times

The discretized Poisson problem (4) was solved for 10^5 time steps with time step $\mathrm{d}t = 10^{-6}$ seconds, resulting in total simulation time $t = 0.1$ seconds.

The polynomial degree takes an important role in the shape computation stage. After shapes are stored the differential operator only depends on the support. As illustrated on implementation scheme from Fig. 2, the shapes and supports are in our case calculated on the CPU. Different time loop execution times for various polynomial degrees m are therefore not expected neither are they observed.

In Fig. 3 we observe how the performance from CPU-only code is directly proportional to number of nodes, while multiple regimes can be seen in the single-GPU implementation.

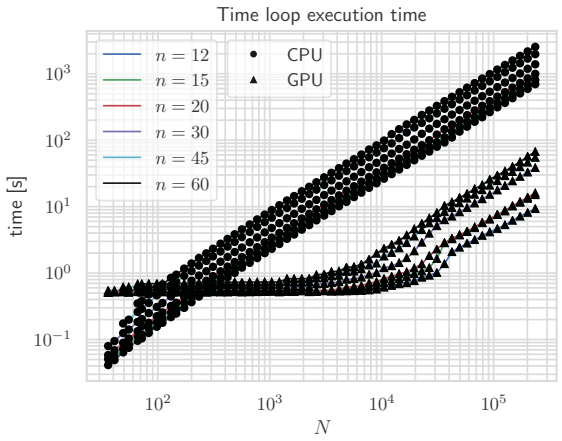

Figure 3. Time loop execution times on single CPU and GPU. Chosen highest polynomial degree $m = 2$.

In the first regime it makes no significant difference what portion of the available L2 cache is occupied by the data set. There is just not enough data dumped to the GPU to fully exploit the advantages of parallelization as most of the time is spent on communication and data distribution rather than on computation itself. For larger data sets ($N > 10^3$), the computation part prevails and faster computation times are observed. Increasing data set size even further again leads to computation times proportional to the number of nodes - similar to the CPU. As expected generally higher execution times are observed for larger support sizes.

Speedup is defined as ratio between the execution of the time loop on a single CPU t_{CPU} and single GPU t_{GPU}. Speedups are shown on Fig. 4. Similar regimes as in Fig. 3 can be seen starting with low or zero benefit from GPU implementation. Increasing the data exploits the parallelization advantages which peak at certain data set size dumped to the GPU.

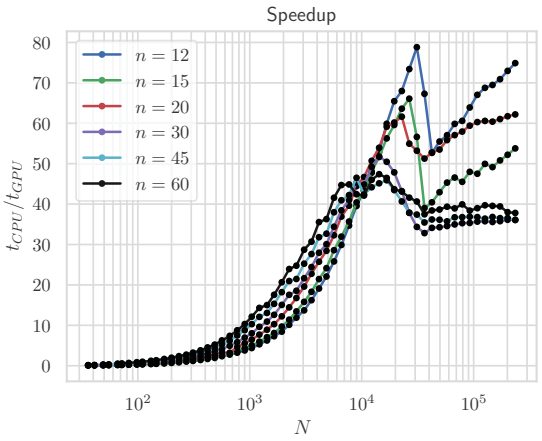

Figure 4. Observed speedups.

To explain the notable drop of speedup in Fig. 4, we have to understand hardware structure in more detail. The L2 cache size of the `NVIDIA GeForce RTX 2080 Ti` graphics processing unit is 5.5 MB [23]. Therefore increasing the data set size exploits the advantages of GPU implementation but only until the L2 cache of the GPU is

almost, but not yet, full. Once the L2 cache is filled, more memory communication with higher latency is needed.

A rough estimation of how much data is copied to the GPU can be made. In our implementation we used doubles and integers, $N(4+n)$ double-precision and N_i+Nn integer numbers, where N_i is number of nodes in interior. Knowing that size of double and integer is 8 and 4 bytes respectively, we can estimate when the L2 cache is full and approximately estimate the speedup peak position. The calculations are gathered in table 1. Note that the estimated $N_{5.5\,\mathrm{MB}}$ is calculated by enforcing $N_i = N$. The larger the number of nodes, the more acceptable this enforcement is. Estimated data size copied on GPU is also shown in Fig. 5.

Support size	Estimated $N_{5.5\,\mathrm{MB}}$	N speedup peak
12	30556	31085
15	25463	26601
20	19928	19527
30	13889	14313
45	9549	8991
60	7275	7707

TABLE 1. ESTIMATED AND OBSERVED PEAK PERFORMANCE AS FUNCTION OF NUMBER OF NODES N.

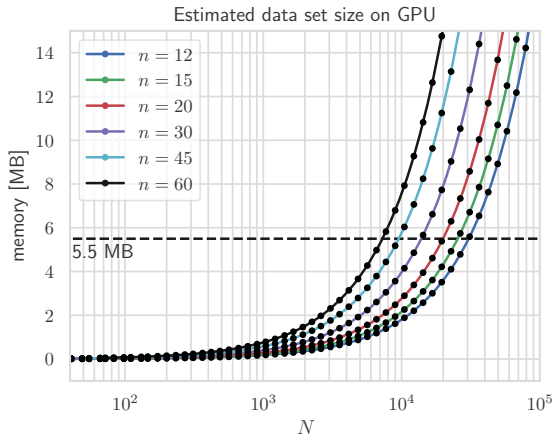

Figure 5. Estimated data set size copied to GPU.

Table 1 shows how observed speedup peaks are close to the estimated calculations. We therefore conclude that the drop of speedup performance is closely related to the size of L2 cache.

VI. CONCLUSIONS

The execution performance of solution of a Poisson problem on 2D scattered nodes with Dirichlet boundary conditions was analyzed in this paper. We measured how changing support sizes, a crucial parameter in using high order RBF-FD, and total number of discretization nodes affects the parallel execution performance. We observed the speedup peaks, which can be explained by measuring the data set size dumped to the GPU and comparing it with the available L2 cache memory size.

In this paper only the execution of time loop was considered as performance important, however some researchers already reported on using GPUs to calculate the shapes. The next step could therefore be to dump even more computation to the GPU. Additional extension could also be by solving the Poisson problem implicitly or by comparing the computational performance using mulitple CPU and GPU devices.

ACKNOWLEDGMENTS

The authors would like to acknowledge the financial support of the ARRS research core funding No. P2-0095 and the Young Researcher program PR-08346.

REFERENCES

[1] Bengt Fornberg and Natasha Flyer. Solving pdes with radial basis functions. *Acta Numerica*, 24:215–258, 2015.

[2] Gregor Kosec. A local numerical solution of a fluid-flow problem on an irregular domain. *Adv. Eng. Software*, 120:36–44, 2018.

[3] Eko Prasetya Budiana et al. Meshless numerical model based on radial basis function (rbf) method to simulate the rayleigh–taylor instability (rti). *Computers & Fluids*, page 104472, 2020.

[4] Jia-Le Zhang, Zhi-Hua Ma, Hong-Quan Chen, and Cheng Cao. A gpu-accelerated implicit meshless method for compressible flows. *J. Comput. Phys.*, 360:39–56, 2018.

[5] Jamal Amani Rad, Kourosh Parand, and Saeid Abbasbandy. Pricing european and american options using a very fast and accurate scheme: The meshless local petrov–galerkin method. *Proceedings of the National Academy of Sciences, India Section A: Physical Sciences*, 85(3):337–351, 2015.

[6] Jure Slak and Gregor Kosec. On generation of node distributions for meshless PDE discretizations. *SIAM Journal on Scientific Computing*, 41(5):A3202–A3229, jan 2019.

[7] T. Belytschko, Y. Y. Lu, and L. Gu. Element-free galerkin methods. *Int. J. Numer. Methods Eng.*, 37(2):229–256, 1994.

[8] B. Nayroles, G. Touzot, and P. Villon. Generalizing the finite element method: Diffuse approximation and diffuse elements. *Comput. Mech.*, 10(5):307–318, Sep 1992.

[9] J. M. Melenk and I. Babuška. The partition of unity finite element method: Basic theory and applications. *Comput. Methods Appl. Mech. Eng.*, 139(1):289 – 314, 1996.

[10] Satya N. Atluri and Tulong Zhu. A new meshless local petrov-galerkin (mlpg) approach in computational mechanics. *Comput. Mech.*, 22(2):117–127, 1998.

[11] C. Armando Duarte and J. Tinsley Oden. H-p clouds—an h-p meshless method. *Numerical Methods for Partial Differential Equations: An International Journal*, 12(6):673–705, 1996.

[12] A. I. Tolstykh and D. A. Shirobokov. On using radial basis functions in a "finite difference mode" with applications to elasticity problems. *Comput. Mech.*, 33(1):68–79, 2003.

[13] Natasha Flyer, Bengt Fornberg, Victor Bayona, and Gregory A. Barnett. On the role of polynomials in RBF-FD approximations: I. Interpolation and accuracy. *J. Comput. Phys.*, 321:21–38, 2016.

[14] Victor Bayona, Natasha Flyer, Bengt Fornberg, and Gregory A. Barnett. On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs. *J. Comput. Phys.*, 332:257–273, 2017.

[15] Roman Trobec and Gregor Kosec. *Parallel scientific computing: theory, algorithms, and applications of mesh based and meshless methods*. Springer, 2015.

[16] Jesse M. Kelly, Eduardo A. Divo, and Alain J. Kassab. Numerical solution of the two-phase incompressible navier–stokes equations using a gpu-accelerated meshless method. *Eng. Anal. Boundary Elem.*, 40:36–49, 2014.

[17] G. Kosec and P. Zinterhof. Local strong form meshless method on multiple Graphics Processing Units. *CMES: Computer Modeling in Engineering and Sciences*, 91(5):377–396, 2013.

[18] Evan F. Bollig, Natasha Flyer, and Gordon Erlebacher. Solution to pdes using radial basis function finite-differences (rbf-fd) on multiple gpus. *J. Comput. Phys.*, 231(21):7133–7151, 2012.

[19] Jure Slak, Blaž Stojanovič, and Gregor Kosec. High order RBF-FD approximations with application to a scattering problem. In *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–5. IEEE, 2019.

[20] Jure Slak and Gregor Kosec. Medusa: A C++ library for solving pdes using strong form mesh-free methods, 2019.

[21] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[22] Mitja Jančič, Jure Slak, and Gregor Kosec. Analysis of high order dimension independent RBF-FD solution of Poisson's equation, 2019.

[23] William Harmon. Nvidia geforce rtx 2080 ti graphics card review, 2019.