A Fully GPU Solution Using Meshless Petrov Galerkin Local

Lucas P. Amorim[1], and Renato C. Mesquita[1].
[1]Department of Electrical Engineering, Federal University of de Minas Gerais, Belo Horizonte, Brazil, lucaspa@gmail.com

The massively parallelized solution to calculation of element contributions, assembly of the overall system (global stiffness matrix) and the solution of the resulting simultaneous equations using the Meshless Local Petrov Galerkin Method — MLPG. The main objective is to build a solution "end-to-end" running on graphics processing units — GPU, comprising the assembly and solution of linear equations system in an integrated manner. Thus, we propose the Meshless Petrov Galerkin Local Node-by-Node Method using CUDA, which takes advantage of essentially parallel nature of MLPG, the no need mesh at any stage and even the possibility of solution of systems of linear equations without the need for overall assembly of the involved matrix.

***Index Terms*—GPU, meshless, MLPG, solver.**

I. Introduction

MESHLESS methods have clear benefits for certain engineering problems, especially when the quality of the mesh can not be guaranteed. The development of the method, on the other hand, is more difficult compared to methods that have a support network [1] and this is reflected directly without computational cost.

To make it attractive also in terms of performance, an alternative as to use high-performance scientific computing techniques, such as GPUs, with its thousands of independent processing units, is gaining in importance. There are already solutions to generation of functions of formulation, numerical integration, interdependence relation between nodes and application of boundary conditions [2], however, researches that addresses the end-to-end problem by subdividing it into elements equivalent to the Finite Element Method Element per Element — EbE-FEM [3], are still in the early stages. The problem is that in addition to calculating the contributions of nodes it is also necessary to solve the system of equations even before having it completely in memory.

The objective of this work is to present a solution based on the bi-conjugate gradient method — BiCG integrated to the MLPG, which by definition inserts values in the global matrix line by line. In this way, it is expected that during the composition of the global stiffness matrix its solution will be started without the need for data transfers to CPU.

II. A Fully GPU Solution

An alternative, equivalent to the proposal for the EbE-FEM [3], is to work with the individual contributions of the nodes without building the system integrally. In MLPG this is possible due to the property that each subdomain of a node can be integrated independently of the other [1], [2]. By analyzing the formulation of the method at a more general level, the solution can be divided into the following main steps:

1) Generation of problem geometry, node distribution and generation of quadrature domains;
2) Assembly of the linear matrix system through the evaluation of the weak local form in each subdomain;
3) Solution of the resulting linear equation system;
4) Computation of results at points of interest.

The steps 2 and 3 have really relevant computational costs and therefore will be the focus of the parallelization proposed in this work. The scheme regarding of step 2, which consists of evaluating the weak form for each node of the subdomain of the problem, calculating the nodes in the support domain and interpolating the values in the Gauss points to perform the numerical integration, is presented in the Fig. 1, whereas the step 3 is represented by the Algorithm 1.

A. Stiffness Matrix Assembly

The assembly of the stiffness matrix using the MLPG is computed node-by-node independently, which means in practice that each node is in charge of a thread itself. For this calculation we apply two slightly different interpolation methods: Moving Least Squares — MLS and Radial point interpolation method with polynomial terms — RPIMp [2].

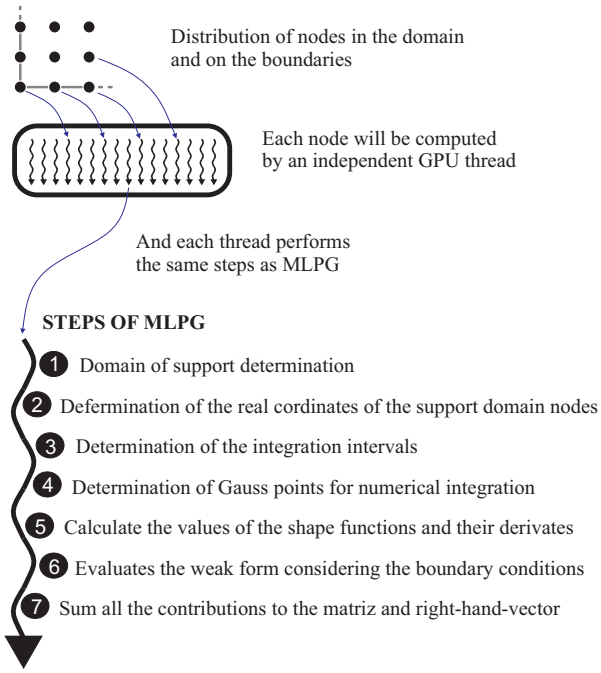

Fig. 1. Assembly of the linear matrix system through the evaluation of the weak local form in each subdomain.

Algorithm 1: Overview of the algorithm that addresses the solution of the linear equations system assembled in the device (GPU).

```
(...)
//Initialization of the residue sequence
r̃^(0) = r_q^(0) = b_q        x_q^(0) = 0
for i = 1, 2, ... do                          //BiCG iterations
    //Application of the preconditioner in z_q e z̃_q
    resolva M z = r_q^(i-1)        resolva M z̃_q = r̃^(i-1)
    //Calculation of ρ from residue sequences
    ρ_i = z^(i)^T r̃^(i-1)
    if ρ_i = 0 or ξ_i = 0 then (failure);
    //Update of the search direction sequence p
      and p̃ from the iteration residue sequences
      and the last values of ρ
    if i = 1 then
        p_q = z^(i)        p̃ = z̃_q^(i)
    else
        β_i = ρ_i / ρ_{i-1}
        p_q = z^(i) + β_i p_q        p̃ = z̃_q^(i) + β_i p̃
    end if
    //Updating the sequence that defines the step
      to be given, k e k̃
    k = A_q p_q        k̃_q = A_q^T p̃
    //Step direction update, α
    α_i = ρ_i / p̃^(i)^T k^(i)
    //Response sequence updating, x, based on step
      direction and size
    x_q = x_q + α_i p_q
    //Updating the iteration residue sequence, r
      and r̃, subtracting from each value the step
      k and k̃ respectively in the direction α
    r_q = r_q - α_i k        r̃ = r̃ - α_i k̃_q
    //Check accuracy of approach and continue if
      necessary
end for
//Composition of x from all x_q
//Interpolation of the values of interest
```

The first task is the determination of the local node support domain, made by scanning in the neighborhood of the node to find the nodes that form the support domain. In sequence, the real (Cartesian) coordinates are obtained. Then the boundary of the subdomain is calculated to perform the numerical integration by the Gauss method, and the integration points are determined. In these points the values of ϕ and $d\phi/dn$ are evaluated, that is, the value of the functions of form and of its derivative with respect to normal to the border of the subdomain at the points of integration is calculated.

The weak form is then evaluated, according to the form function adopted, and all numerical integrations are performed. In this step, the boundary conditions must be considered when using the penalty method (MLS case). The MLPG5 [6] has a particularity that makes it interesting for parallelization: the contribution of each node to the assembly of the system happens in a single line of the stiffness matrix.

Finalizing the individual node computation, its contribution is summed in the resulting system of linear equations and then the solution process starts immediately from the individual contribution of the node, without necessarily having a synchronization barrier separating the assembly (step 2) and solution (step 3) and no data traffic return to the CPU other than those of the response to the treated problem.

B. Solver Step

The BiCG iterations are essentially sequential, since all values used by any i iteration were computed by the iteration $i-1, \forall i \neq 0$. However, within each iteration we have operations performed line by line of the system, in addition to completely independent operations that can be performed concurrently. It is from these operations that we will explore the paralelelism and, above all, create approaches so that CPU intervention is not necessary until the iterations sufficient to reach the predefined acceptable error are realized. The various details of this algorithm will be explained later.

After the system solution process, the interpolation of the values of interest, objective of the method, is then possible. At this point only the resulting processing data is passed to the CPU.

III. Conclusion

The solution being built has become feasible due to the dynamic paralelelism now available in the CUDA language. This feature allows the CUDA kernel to create and synchronize directly on the GPU, anywhere in the program. Therefore, we are doing an end-to-end formulation of the solution, enforced completely in GPU: no data traffic to the CPU during processing, except for the delivery of the values already computed and ready for the interpolation of the values of interest.

References

[1] G. R. Liu. Mesh Free Methods: Moving Beyond the Finite Element Method. *CRC Press*, 1 edition, July 2002.

[2] B. C. Correa, R. C. Mesquita, and L. P. Amorim. “CUDA Approach for Meshless Local Petrov Galerkin Method”. *CEFC*, vol. 51, no. 3, pp. 3-6, 2014.

[3] I. Kiss, S. Gyimothy, Z. Badics, and J. Pavo. “Parallel realization of the element-by-element FEM technique by CUDA”. *IEEE Trans. Magn.*, vol. 48, no. 2, pp. 507-510, 2012.

[4] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. DerVorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods (Miscellaneous Titles in Applied Mathematics Series No 43). *Society for Industrial and Applied Mathematics*, 1 edition, January 1987.

[5] E. Mller, X. Guo, R. Scheichl, and S. Shi. “Matrix-free gpu implementation of a preconditioned conjugate gradient solver for anisotropic elliptic pdes”. *Comput. Vis. Sci.*, vol. 16, no. 2, pp. 41-58, 2013.

[6] Atluri, S. N. and Zhu, T. A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. Computational Mechanics, no. 22, pp. 117-127, 1998.