Review Report

July 3, 2019

The present review report concerns the paper entitled "*Optimization of a discontinus finite element solver with OpenCL and StarPU*", submitted in June 2019 to the International Journal on Finite Volumes by Bérenger Bramas, Philippe Helluy, Laura Mendoza and Bruno Weber.

This paper is concerned with the OpenCL implementation of a discontinuous Galerkin (dG) method for the numerical simulation of linear electromagnetic waves. All the difficulties of data distribution and memory transfers for parallel execution, which can be very cumbersome in a pure OpenCL context, are removed by the use of the StarPU runtime system. Only the computational kernels, known as "*codelets*", are implemented in OpenCL. Indeed, within the StarPU framework, the dG method is decomposed into a stack of tasks to be accomplished at the end of the simulation. The declaration of the tasks is done in C99 and this allows StarPU to generate on the fly a dependency graph between the tasks, based on their input- and output-data predeclared memory access. Then the StarPU scheduler, given a computational architecture, handles the distribution of the codelets and their associated data to the available computational units (CPU or GPU). Finally, coding the computational kernels under OpenCL offers the possibility of using the same language for all the targeted processing units, even though different versions may be implemented for "*incremental optimization*" toward different types of accelerators.

In a light introductory section, the authors presents the different ingredients of the current paper: the discontinuous Galerkin method, OpenCL and StarPU. The second section is dedicated to a clean mathematical definition of the model and its Runge-Kutta/dG discretization. Section 3 presents the StarPU runtime system; first globally, then more specifically in the context presented in section two: the different tasks needed to be executed per Runge-Kutta sub-time-step are detailed. Section four justifies a specific choice in the data ordering in memory. The authors claim that storing all the conserved variables per degree of freedom is more efficient than storing all the degrees of freedom of the subcell per conserved variable. It is however a pity that such an argument is not clearly demonstrated by the numerical results. Section five is dedicated to numerical experiments on a single test case: the propagation of a linear eletromagnetic wave. First the efficiency of the distribution algorithm is assessed on CPUs, then the GPU are shown to greatly accelerate the computation, and finally, the hybrid CPU+GPU scheduling is claimed to always improve the pure GPU execution. A sixth very short conclusive section closes the speech.

The paper is globally well-written and its reading is enjoyable. The complex mathematical and computer-science contexts are rather well-explained and the manuscript is definitely worth publishing. However, at the end of the reading I stay a little bit hungry and I think the paper would not suffer a few more pages, in particular in the numerical result section and in the conclusion. Here a some remarks, questions and suggestions, which, I hope, will help developing the overall analysis:

1. For me, it is not fully clear why you finally aim at writing all the codelets in OpenCL. I don't find your introductory paragraph on OpenCL at the top of page 2 very clear and I

suggest you should rewrite it to this purpose. Since you can write as many codelets in as many languages you wish, why sticking with OpenCL?

2. Also, if you run a simulation on CPUs only, with both C99 and OpenCL kernels, are the OpenCL kernels always choosen for all the tasks?

3. You can claim the discontinuous Galerkin method as a finite element method, but you can claim it as an enriched finite volume method as well. In the context of the "*International Journal of Finite Volume*", I suggest you rather refer to a finite volume method or just to a discontinuous Galerkin method, especially in the title.

4. The model used is fully linear and this is written nowhere. However, I believe this is helping your discretization and its numerical implementation a lot. Frankly, given the complexity of the overall implementation, I fully understand your choice of solving such a linear model, but by honesty I would underline it somewhere. For example, your conserved variables can freely move within $\mathbb{R}^m$, which is seldom the case in non-linear systems: you would then need some form of limiter to prevent unphysical states. Also, the linearity of the flux is implicitely used when going from equation (2.4) to equation (2.6). See also, the remark on time-stepping below.

5. When going from equation (2.1) to (2.4), you make two simplifications, one of them being very important: you integrate numerically volume and surface integrals by $(d+1)$ Gauss-Lobatto quadrature points in each direction (2D or 3D. From now on I will only speak of number of quadrature points and order of approximation per space direction). This is exact for the dG volume term $\int_L \mathbf{f}(\mathbf{w}_L, \nabla\varphi_i)$, because the integrand is a polynomial of order $2d-1$, for which the $(d+1)$ GL quadrature is exact. This is no more true for the nodal numerical fluxes $\int_{\partial L} \mathbf{f}(\mathbf{w}_L, \mathbf{w}_R, \mathbf{n})\,\varphi_i$, but this simplification is not too bad, because the numerical order is kept in the mean: the update of the mean value per cell involves only the integration of a polynomial of order d per faces. However, stricly speaking you are not solving equation (2.1) exactly.
   But the biggest simplification comes when approximating the integral with the time-derivative. Here, you integrate a $2d$ polynomial with a $2d-1$ exact quadrature. It is well known that the basis made of Lagrange polynomials at Gauss-Lobatto quadrature points is not orthogonal and you should have a mass matrix per subcell to invert. Here you have clearly performed a mass-lumping and the accuracy of the method should be reduced compared to the expected $h^{(d+1)}$. Could you comment on that?

6. Speaking of accuracy, the paper does never show that the method has been correctly implemented and that the accelerated results are physically relevant. For me the best way to do that is to provide convergence curves which show that the theoretical convergence rates have been attained, particularly in this purely linear context where the exact solution is obvious.

7. For the sake of consistency, even though one can not mix up $\omega_{L,i}$ with ω_i, I would write the quadrature weights on the reference element with a hat.

8. What do the macrocell and the interface data contain? Frow what I have understood, the macrocell data contain three "*fields*" (the conserved variables at the degrees of freedom, the numerical fluxes and the volume fluxes) and the interface data two (the conserved variables on both sides of the interface, one being possibly external to the domain Ω). If I am correct, then I don't understand the necessity of considering tasks (2) and (3) as separate. Boundary states could be filled in tasks (1) and (2) solves for the interface

numerical fluxes, whatever the nature of the neighbor. Could you make your data structure clearer to enlight your choices in the tasks separation?

9. Page 8, you claim that "*the wait for task completion is done at the end of the algorithm and not inside each iteration*". This implies that the time-step is kept constant during the entire simulation and that you know the number of time-steps to be performed in advance. This is once more closely related to the fact that the considered model is linear. Otherwise, by the stability constraint of the explicit Runge-Kutta procedure, you would have to compute this time-step specifically during each time step, and ask for a global synchronization of the data before updating them. Also, checking if the final time has been reached is a major bottleneck in the graph of tasks. Could you comment on that and make adequate modification in the paper to make this clearer for the reader?

10. If you generate the graph of tasks all at once, how many time steps do you perform? How many tasks does it represent? What is the size of the graph?

11. Section 4 is a bit confusing. The two solutions (4.1) and (4.2) you propose for the memory storage are interesting but your choice of using (4.1) comes with no argumentative data. When you claim that the *volume kernels* and *surface kernels* are the most time consuming, what are the CPU-time percentage of each task? Have you run simulations with storage solution (4.2)? If not, I am not sure mentionning it is helping your speech.

12. There is a $t_{\text{CPU+GPU}}$ at the bottom of page 10. Is this a mistake?

13. Pages 11 and 12 are two occurences of *peak* and *peek*, which, I guess, are both mispelling of the word *pick*.

14. I don't completely agree with your final statement that "*in all the situations, StarPU is able te get an additional gain from the CPU cores*". When looking at WS8, adding CPUs to a GPU does not speed up the computation at all (29.8s), when adding a GPU helps (24.3s) and adding CPUs to the two GPUs still helps (18.1s). What is happening in your simulation with 1GPU and the CPU cores? How are the tasks distributed? Is this related to you LAHeteroprio scheduler? Does dmda helps?