PAPER • OPEN ACCESS

Lessons from accelerating an RBF-FD phase-field model of dendritic growth on GPUs

To cite this article: Boštjan Mavrič *et al* 2024 *J. Phys.: Conf. Ser.* **2766** 012168

ECS UNITED
ECS The Electrochemical Society
Advancing solid state & electrochemical science & technology
247th ECS Meeting
Montréal, Canada
May 18-22, 2025
Palais des Congrès de Montréal
Abstracts due December 6th
Showcase your science!

Lessons from accelerating an RBF-FD phase-field model of dendritic growth on GPUs

Boštjan Mavrič[1,2], **Tadej Dobravec**[2] **and Božidar Šarler**[2,1]

[1] Institute of Metals and Technology, Lepi pot 11, 1000 Ljubljana, Slovenia
[2] Faculty of Mechanical Engineering, University of Ljubljana, Aškrčeva 6, 1000 Ljubljana, Slovenia

E-mail: `bostjan.mavric@imt.si`

Abstract. Phase-field modeling of dendritic growth presents the state of the art in the field of solidification modeling and are usually implemented using finite difference models combined with explicit time marching and accelerated by using GPUs. They are a prime candidate for such acceleration, since they require many arithmetic operations on relatively low ammount of data. We present an attempt at porting an existing RBF-FD code optimized for CPU execution to use GPU acceleration while keeping the resulting implementation portable between architectures. We discuss the acceleration achieved, scaling and implementation issues and critically discuss current landscape of GPGPU offerings.

1. Introduction

In recent years the hardware profile of the HPC hardware has moved from being focused on homogeneous compute environment based on CPUs to heterogeneous compute environment with a combination of CPUs and accelerators. Usually, most of the compute capability in terms of floating point operations is contributed by the accelerator hardware. Such accelerators are usually in the form of general purpose graphics computing units but specialized accelerator cards are becoming more and more popular in recent years driven by the requirements coming from the training of neural networks [1].

The accelerators are often used in connection with phase-field models of dendritic growth. Such models are usually implemented using finite-difference methods and run on supercomputers with great efficiency [2, 3]. Recent simulations of dendritic growth based on meshless RBF-FD (radial basis function generated finite difference) methods have shown that such formulation has an advantage over the finite-difference methods in terms of mesh-induced anisotropy [4, 5]. Artificial anisotropy is introduced into the FD-based models through the use of a structured, usually cubic, grid that is needed to formulate the spatial discretisation. The RBF-FD methods can be used on scattered isotropic nodes which helps to formulate a discretisation that does not introduce additional sources of anisotropy and can thus produce more accurate results.

The RBF-FD models have been formulated based on an original software library written in modern Fortran and did not exploit accelerators to speed-up the computation. The goal of this work is to formulate an implementation that is able to leverage the immense computing power available through accelerator hardware.

2. Governing equations

We consider the solidification of pure supercooled melt in the 2-D computational domain Ω with the boundary Γ. We study a simplified case with constant density ρ, specific heat at constant pressure c_p, and thermal conductivity k. The latent heat of melting and the melting temperature are denoted as L_m and T_m, respectively. We use the dimensionless PF model [6], where the spatial and temporal coordinates are measured in units of the PF interface thickness and the PF characteristic attachment time, respectively. The PF interface thickness is defined as

$$W_0 = d_0 \frac{1}{\alpha_1} \lambda, \tag{1}$$

where d_0 is the thermal capillary length while α_1 and λ stand for a constant and the free parameter of the PF model, respectively. The PF characteristic attachment time is given as

$$\tau_0 = \frac{d_0^2}{D_T} \frac{\alpha_2}{\alpha_1^2} \lambda^3, \tag{2}$$

where α_2 stands for a constant of the PF model and $D_T = k/(\rho c_p)$ for the thermal diffusivity. The PF constants are equal to $\alpha_1 = 0.8839$ and $\alpha_2 = 0.6267$ [6]. The selection of free parameter λ has to yield W_0 much smaller than the diffusion length of solidification to ensure a valid PF model.

The PF model constrains PF values in the interval $-1 \leq \phi \leq 1$, where $\phi = 1$ and $\phi = -1$ denote solid and liquid phases, respectively. We use the preconditioned PF [7]

$$\psi = \sqrt{2} \tanh^{-1}(\phi), \tag{3}$$

to increase numerical stability for larger node spacings. The energy conservation equation in terms of dimensionless temperature $\theta = (T - T_m)/(L_m/c_p)$ reads as

$$\frac{\partial \theta}{\partial t} = D \nabla^2 \theta + \frac{1 - \phi^2}{2\sqrt{2}} \frac{\partial \psi}{\partial t} \tag{4}$$

where D stands for the dimensionless thermal diffusivity, measured in units of W_0^2/τ_0. The PF equation reads as [8]

$$\begin{aligned} a^2(\boldsymbol{n}) \frac{\partial \psi}{\partial t} &= \sqrt{2} \left(\phi - \lambda(1 - \phi)\theta\right) + 2a(\boldsymbol{n}) \nabla a(\boldsymbol{n}) \cdot \nabla \psi \\ &\quad - \sqrt{2}\phi \nabla \psi \cdot \boldsymbol{a}(\boldsymbol{n}) + \nabla \cdot \boldsymbol{a}(\boldsymbol{n}) + a^2(\boldsymbol{n}) \left(\nabla^2 \psi - \sqrt{2}\phi |\nabla \psi|^2\right), \end{aligned} \tag{5}$$

where $a(\boldsymbol{n})$ and $\boldsymbol{a}(\boldsymbol{n})$ represent the anisotropy functions. They depend on the normal to the solid-liquid interface

$$\boldsymbol{n} = (n_x, n_y) = -\frac{\nabla \psi}{|\nabla \psi|}. \tag{6}$$

We consider the cubic anisotropy of the surface energy. In this case, anisotropy functions read as

$$a(\boldsymbol{n}) = 1 - 3\epsilon_4 + 4\epsilon_4 \left(n_x^4 + n_y^4\right), \tag{7}$$

and

$$\boldsymbol{a}(\boldsymbol{n}) = 16\epsilon_4 |\nabla \psi| a(\boldsymbol{n}) \left(n_x(n_x^4 + n_y^4 - n_x^2), n_y(n_x^4 + n_y^4 - n_y^2)\right), \tag{8}$$

where ϵ_4 stands for the anisotropy strength of the interface energy.

3. Solution procedure

3.1. Discretization

The problem is solved by employing the method of lines using first order explicit time stepping method. The spatial discretization is done through meshless RBF-generated finite differences (RBF-FD, also known as local radial basis function collocation method). It is formulated on nodes x_i of a scattered node set $x_i \in X$ and works by constructing interpolants from the combination of monomials and RBFs across localized stencils, usually consisting of a small number of nearest neighbours [9, 10]. In this way we can express the value of an unknown $u(x)$ at an arbitrary point x as

$$u(x) = \sum_{i \in \Omega_k} w_{k,i}(x, \mathcal{I}) u(x_i), \tag{9}$$

where $\Omega_k \subset X$ is the stencil centered on the node closest to x. The scalars $w_{k,i}(x, \mathcal{I})$ represent the weights obtained from the solution of the local interpolation problem. The discretization weights can be determined for any linear differential operator $\mathcal{L}$ by acting with the operator on the local interpolation. In the equation above, this operator is the identity operator $\mathcal{I}$. For a general operator, we would have

$$\mathcal{L}u(x) = \sum_{i \in \Omega_k} w_{k,i}(x, \mathcal{L}) u(x_i). \tag{10}$$

From the computational point of view, it is important to note that the matrix representing $w_{k,i}(x, \mathcal{L})$ is sparse and contains n nonzero elements per row. Their positions depend on local stencils Ω_k and can be reordered into a band-limited matrix. Naturally, the weights do not depend only on k but on x as well, meaning that every evaluation point comes with unique evaluation weights, but can use the same values $u(x_i)$ given that x_k is a common closest node. Additionally, the weights $w_{k,i}(x, \mathcal{L})$ can be calculated in advance and reused during the simulation steps.

3.2. Solution steps

The goal of solution procedure is to propagate the dicretized solutions $\psi(X, t)$ and $\theta(X, t)$. With this notation we introduce the vectors of unknowns as $u(X, t) = [u(x_i, t), \forall x_i \in X]$. The solution algorithm proceeds as specified in algorithm 1.

3.3. Computer implementation

The goal of this work has been to add GPU offloading to an already existing implementation of the solution procedure listed above. When doing this, it is necessary to account for different offload paradigms and frameworks supported by specific combination of compiler toolchains and hardware. This is summarized in table 1. Since we are trying to use a vendor-agnostic solution it appears that OpenMP is the best option. It is supported by all compiler vendors, allows for CPU fallback if GPU hardware is not available and shows performance close to native implementations in CUDA or other vendor-specific languages. It is also expressive enough to function well for the problems we are trying to address. With this in mind we proceeded to use OMP offload constructs to mark the loops denoted as *Precalculation*, *Rate calculations* and *State update*. All calculations are offloaded and the solution is only copied over to host when we want to output the solution for visualisation purposes.

To arrive to the implementation benchmarked in this contribution we needed to introduce some significant changes to the solver implementation. The first step was to remove most of the encapsulation present in the *Precalculation* and *Rate calculation* loops. In practice this meant removing functions implementing formulas for $a(\boldsymbol{n})$, $\boldsymbol{a}(\boldsymbol{n})$ and functions implementing

Algorithm 1: Calculation sequence of the solution procedure.

```
1  begin Initialisation
2      Position scattered nodes. ;
3      Calculate discretization weights.;
4      Initialize $\psi(X,0)$ and $\theta(X,0)$;
5  while $t < t_{end}$ do
6      begin Precalculation
7          $\nabla\psi(X,t) \leftarrow \boldsymbol{w}(\nabla)\psi(X,t)$;
8          $\boldsymbol{n}(X,t) \leftarrow \nabla\psi(X,t)/|\nabla\psi(X,t)|$;
9          $a(X,t) \leftarrow$ Calculated according to formula 7 from $\boldsymbol{n}(X,t)$. ;
10         $\boldsymbol{a}(X,t) \leftarrow$ Calculated according to formula 8 from $\boldsymbol{n}(X,t)$. ;
11     begin Rate calculations
12         $\frac{\partial\psi}{\partial t}(X,t) \leftarrow$ Calculated according to formula 5 from $\psi(X,t)$, $a(X,t)$, $\boldsymbol{a}(X,t)$ and
           $\nabla\psi(X,t)$ ;
13         $\frac{\partial\theta}{\partial t}(X,t) \leftarrow$ Calculated according to formula 4 from $\theta(X,t)$ and $\frac{\partial\psi}{\partial t}(X,t)$;
14     begin State update
15         $\psi(X,t+\Delta t) \leftarrow \psi(X,t) + \Delta t\frac{\partial\psi}{\partial t}(X,t)$;
16         $\theta(X,t+\Delta t) \leftarrow \theta(X,t) + \Delta t\frac{\partial\theta}{\partial t}(X,t)$;
17         $t \leftarrow t + \Delta t$
```

Table 1. Programming paradigm support matrix in popular Fortran compilers.

	Nvidia	Intel	AMD
GNU	OpenMP, OpenACC	OpenMP, OpenACC	OpenMP, OpenACC
ifort	/	OpenMP	/
NVHPC	OpenMP, OpenACC, CUDA	/	/
ROCm	/	/	OpenMP, CUDA

the evaluation of differential operators like equation (10). This allowed to better exploit parallelization by reducing register usage for each kernel. Secondly, related to this was the need to remove all polymorphism present in the solver since polymorphic calls are not supported in the offload sections.

3.4. Results and discussion
The results were collected on a workstation computer equipped with AMD Ryzen 9 5900X processor and NVIDIA GeForce RTX 3080 GPU. The tests were performed by compiling the code in three different modes summarized in table 2. The CPU results were obtained from parallel runs on 12 physical cores of the processor. The GPU result employed a single GPGPU. The timing results are shown in figure 1.

The timing is performed for a 2D simulation employing stencil size 13 and different number of nodes discretizing the computational domain, which is determined by the number of points along a boundary N. We chose the following values of $N = [480, 640, 960, 1280]$, while keeping the size of the computational domain fixed. The total computational work depends on the number of time steps performed as well. We fix the total simulated time and the time step is reduced in order to satisfy the von Neumann stability conditions for explicit time discretization using

Table 2. Compilers and compiler flags used to produce results for this paper.

label	compiler	version	compilation flags
GPU	`nvfortran`	23.5	`-i8 -mcmodel=medium -Mpreprocess -Mfreeform -Mrecursive -Mreentrant -mp -target=gpu -O3`
CPU nvfortran	`nvfortran`	23.5	`-i8 -mcmodel=medium -Mpreprocess -Mfreeform -Mrecursive -Mreentrant -mp -O3 -tp host`
CPU ifort	`ifort`	2023.0.0	`-reentrancy threaded -threads -parallel -O3 -march=core-avx2 -align array64byte -fma -ftz -fomit-frame-pointer`

forward Euler scheme. Overall this means that the total computational work scales as N^4.

From the timing results in figure 1 we can see that the elapsed time scales linearly with total work. Small differences between different compilers for CPU runs disappear and the problem size is increased. Interestingly, the GPU implementation seems to scale somewhat sublinearly for small problems, but eventually regresses to the linear scaling as problem size becomes large enough.

Figure 2 shows what speedup was achieved by the GPU implementation in comparison to the CPU only computation. The speedup increases as the problem size grows, but seems to reach a stationary value at around 4.5 .

We were also interested in how the balance between different parts of the solution procedures in terms of time spent evolves as the problem size in increased. This is illustrated in figure 3. There we compare the time spent in the *Rate calculations* and *Precalculations*. The former is split in two parts: *Gradient* which represents the gradient calculation in line 7 of algorithm 1 and the rest, that is lines 8 to 10 in algorithm 1, which are labeled by *Precalc. node* and are local, requiring information for a particular node only. We can see that the ratio between the parts remains constant for CPU runs. For the GPU runs it varies and more time is spent in gradient evaluation. From these we can conclude that for large problems the limiting factor is the evaluation of differential operators and not node-local calculations.

4. Conclusions

The implementation of the solution procedure presented in this paper leveraged OpenMP offloading to employ computational resources available on the GPU. This produced a portable implementation that can run either on CPU or GPU hardware. The portability achieved comes, however, at a cost that is paid in terms of code flexibility and extensibility. These are introduced by the limitations that exist on what language features are allowed in the offloaded sections and the fact that employing encapsulation in offloaded sections leads to degraded performance. This could of course change with new compiler versions.

The speedups achieved show promising results, especially for larger problems. There are further open avenues of research in terms of data structures employed to store the simulation state and the development of algorithms specialized for evaluation of differential operators. Because we are relying on a framework to produce GPU code, the balance has to be struck between sophisticated algorithms that are hard for compiler to optimize and naive algorithms that the compiler can understand well.

In the future we plan to further explore the interaction between the method parameters and multiplication algorithms. We also plan to test the code on more GPU hardware to see what

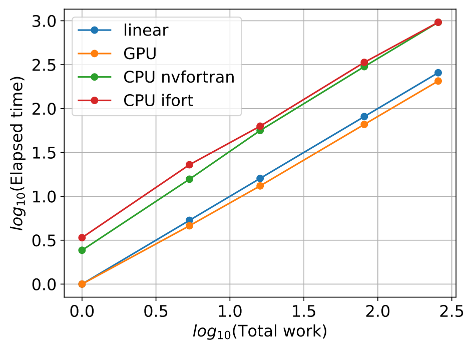

Figure 1. Time spent in the time stepping loop. Time is reported relative to the duration of the smallest case run on GPU which took 21s. The data labeled *linear* represents linear scaling.

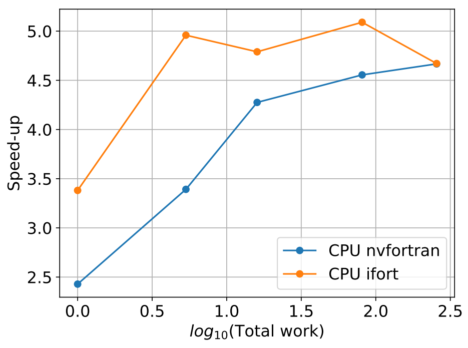

Figure 2. Speed-up achieved with the GPU offloaded version.

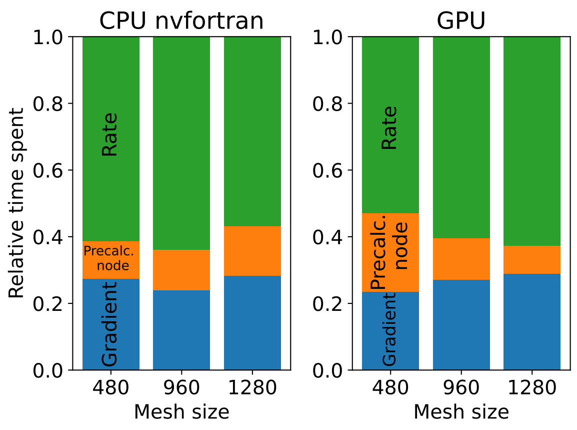

Figure 3. Relative time spent for particular task of the solution procedure.

speed-ups can be offered by other manufacturers in this space.

Acknowledgements
The research has been supported by the ARIS projects Z2-4479, Z2-2640 and ARIS program financing P2-0162 .

References

[1] Peccerillo B, Mannino M, Mondelli A and Bartolini S 2022 *Journal of Systems Architecture* **129** 102561
[2] Sakane S, Takaki T and Aoki T 2022 *Materials Theory* **6** 3
[3] Takaki T 2023 *IOP Conference Series: Materials Science and Engineering* **1274** 012009
[4] Dobravec T, Mavrič B and Šarler B 2020 *Computational Materials Science* **172** 109166
[5] Dobravec T, Mavrič B and Šarler B 2022 *Computers & Mathematics with Applications* **126** 77–99
[6] Karma A and Rappel W J 1998 *Physical Review E* **57** 4323
[7] Glasner K 2001 *Journal of Computational Physics* **174** 695–711
[8] Boukellal A K, Rouby M and Debierre J M 2021 *Computational Materials Science* **186** 110051
[9] Šarler B and Vertnik R 2006 *Computers & Mathematics with Applications* **51** 1269–1282
[10] Flyer N, Fornberg B, Bayona V and Barnett G A 2016 *Journal of Computational Physics* **321** 21–38