Software Stack for an Analog Mesh Computer: The Case of a Nanophotonic PDE Accelerator

Engin Kayraklioglu
Cray, Inc.
engin@gwu.edu

Jeff Anderson
Department of Electrical and Computer Engineering
The George Washington University
jeffa@gwu.edu

Hamid Reza Imani
Department of Electrical and Computer Engineering
The George Washington University
hamidreza@gwu.edu

Volker Sorger
Department of Electrical and Computer Engineering
The George Washington University
sorger@gwu.edu

Tarek El-Ghazawi
Department of Electrical and Computer Engineering
The George Washington University
tarek@gwu.edu

ABSTRACT

The slowing of Moore's Law is forcing the computer industry to embrace domain-specific hardware, which must be coupled with general-purpose traditional systems. This architecture is most useful when large compute power is needed. Among the most compute-intensive applications is the simulation of physical sciences. To maximize productivity in this domain, a variety accelerators have been proposed; however, the analog mesh computer has consistently been proven to require the shortest time-to-solution when targeted toward the Poisson equation. Recent advances in material science have increased the flexibility of the analog mesh computer, positioning it well for future heterogeneous computing systems. However, for the analog mesh computer to gain widespread acceptance, a software stack is required to enable seamless integration with a classical computer. Here, we introduce a software stack designed for the class of analog mesh computers that efficiently generates mesh mappings of a physical problem by enabling users to describe their problem in terms of boundary conditions and mesh parameters. Experiments on a specific implementation of analog mesh computer, the nanophotonic partial differential equation accelerator, show that this stack enables problem-to-mesh scalability expected by the scientific community.

CCS CONCEPTS

• **Hardware → Emerging tools and methodologies**; • **Software and its engineering → Abstraction, modeling and modularity**; **Application specific development environments**.

KEYWORDS

analog mesh computer, emerging technology runtimes, software stack, programming stack, hardware/software abstractions

CF '20, May 11–13, 2020, Catania, Italy

ACM ISBN 978-1-4503-7956-4/20/05.
https://doi.org/10.1145/3387902.3394030

ACM Reference Format:
Engin Kayraklioglu, Jeff Anderson, Hamid Reza Imani, Volker Sorger, and Tarek El-Ghazawi. 2020. Software Stack for an Analog Mesh Computer: The Case of a Nanophotonic PDE Accelerator. In *17th ACM International Conference on Computing Frontiers (CF '20), May 11–13, 2020, Catania, Italy.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3387902.3394030

1 INTRODUCTION

Partial Differential Equations (PDEs) are a core mathematical tool in scientific computing and engineering, and are used to model physical phenomena such as fluid dynamics, electricity, and heat flow. Due to the lack of closed-form solutions for such problems, approximate solutions have been adopted by the scientific community [3]. One class of such approximations, numerical methods, maps the physical problem to a computational grid and iteratively solves for each grid point [8].

As such, the time-to-solution for numerical methods is dependent on both the grid resolution, measured as the $x \times y$ dimension of the grid, and the time to compute each grid point. Classical acceleration techniques, such as parallelism and and high-frequency operation, have been successfully used to reduce time-to-solution [10]. However, the slowing of Moore's Law has put an upper bound on the speedup that can be reasonably expected by classical techniques, resulting in the development of specialized accelerators which can be integrated into large, heterogeneous computer systems [6].

Due to the importance of PDEs to the scientific community [4], there have been many systems developed for their acceleration [5, 10, 12]. Shown in Figure 1, each system takes a different approach to accelerate PDE solutions. However, domain specialists require a common interface to enable code reuse and decrease expertise required to use the system. This is a common problem among heterogeneous systems, leading to the introduction of software stacks. Software stacks were developed to abstract away the details of a computing system, allowing users to effectively program a system without knowledge of its underlying mechanization.

Here, we propose a software stack for an analog mesh computer which abstracts the mesh technology from the user. It is our belief that support of a software stack will enable code reuse between

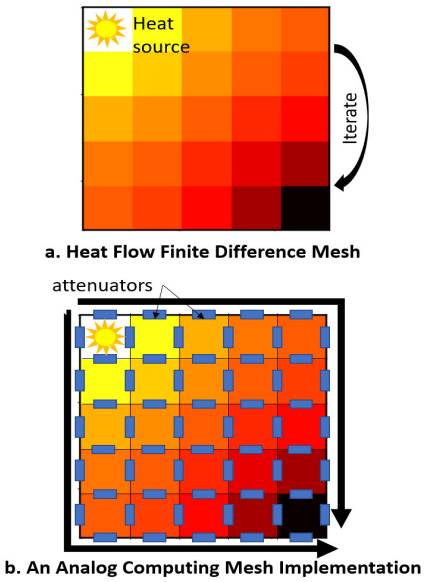

Figure 1: Numerical solutions solve for computational grid points iteratively, while analog mesh computers use a mesh of attenuating elements to solve for grid points in one shot, minimizing time-to-solution.

various analog mesh computers, leading to their mainstream acceptance in heterogeneous computing systems.

2 BACKGROUND AND RELATED WORK

A PDE is a mathematical equation that consists of variables and their partial derivatives. PDEs are widely used for describing a wide variety of physical phenomena such as sound, heat, fluid dynamics, and quantum mechanics. While this makes them invaluable in physical science simulations, PDEs require much work and time to solve. Due to the non-existence of closed form solutions, approximations such as numerical methods have been developed for PDE computation [3].

Numerical methods are the most common approach for solving PDEs and are widely used in engineering. They discretize the problem into small sections and then solve for each section. The three most widely used numerical methods to solve PDEs are the finite element method (FEM), finite volume methods (FVM) and finite difference methods (FDM) [8]. These techniques discretize the problem into a computational grid and solve for each grid point iteratively. For these techniques, grid point spacing is inversely proportional to the accuracy of the solution. However, an increase in the number of grid points results in an increased time-to-solution, which can be unacceptable.

To obtain a PDE solution with both acceptable accuracy and time-to-solution, a variety of accelerators have been proposed [9, 10, 12]. Digital accelerators based on graphics processor units (GPU) and central processing units (CPU) [10, 12], obtain speedups from clock-frequency scaling and enhanced parallelism. However, limitations with the parallel scalability of a computational grid require accelerators with alternative architectures. Analog accelerators, in particular, have shown the ability to minimize time-to-solution in PDE computation.

Introduced in [9], the resistance network analogue was developed to solve the Poisson equation, which characterizes heat transfer and oscillatory flow. The resistance network analogue computes a solution using a mesh of resistors. This architecture allows the resistance network analogue to compute a solution in one-shot, as opposed to iteratively, resulting in minimal time-to-solution. However, the proposed design is a static network and does not support the programmability that is expected by the scientific community. To overcome this defect, *analog mesh computers* with configurable attenuating elements can be used. Equivalent to the resistance network analogue, [1, 5] and [11] are examples of the analog mesh computers that can be programmed by reconfiguring their attenuating elements. The transistor-based analog mesh computer in [11] uses subthreshold CMOS devices as attenuators and [1, 5] uses optical attenuators in place of resistors and diffusion currents, or optical intensity, in place of electrical current.

Due to the varied nature of PDE accelerators, frameworks must be developed to enable code reuse and eliminate the requirement for domain expertise in a particular accelerator. Traditionally, software frameworks have been used to abstract away the underlying mechanization of a system, making it more accessible to users who need only apply the system. For example, UG is a framework for simulation of time-dependent, nonlinear PDEs suitable for execution on distributed systems [2]. Blaze is middleware for accelerator deployments in datacenters, and uses a set of standardized system calls to communicate with varied hardware [7].

PDEs are very important for scientists and engineers in various domains, and analog mesh computers can eliminate several problems in solving PDEs. Reconfigurable analog mesh computers can be used and programmed as accelerators in heterogeneous systems. However, analog mesh computers lack a software framework, hindering their adoption by the scientific computing community. For such computers to be integrated into heterogeneous systems, a framework with a standard interface must be developed. This framework should comprise different layers of abstraction which support various methods of problem description.

3 SOFTWARE STACK FOR AN ANALOG MESH COMPUTER

Providing ease-of-use for domain specialists while using highly-specialized hardware requires an end-to-end design for the software stack, shown in Figure 2. In this section we first describe the multiple front-ends of our modular software stack, then discuss its multipass implementation for translating high-level problem descriptions into hardware configurations. Currently, hardware configurations created by the software stack are limited to netlist descriptions that can be exported to electrical and optical circuit simulators.

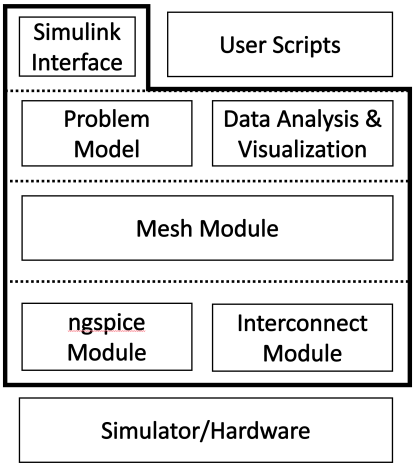

Figure 2: Modular software stack takes user input in the form of problem size and boundary conditions, along with mesh physical parameters, and generates an appropriate bit-file which can be downloaded to the analog mesh computer. User interaction is in the form of an API or Simulink GUI.

```
# create a problem description
prob = Problem(size=10,
               sources=((0,0,1,1),),
               sink=((9,9,1,1),))
# create a mesh model, load the problem
mesh = ROCModel(size=4)
mesh.load_problem(prob)

# run electrical solver
mesh.run_solver(kind='electrical')

# plot the mesh data as a heatmap
plot_heatmap(mesh.final_grid)
```

Listing 1: An Example User Script

3.1 Front-end: Python and Simulink Interfaces

A high-level overview of the software stack is shown in Figure 2. Implemented in Python, the software stack has a user-facing application programming interface (API) for creating problem descriptions, analog mesh hardware, running simulations, and analyzing results. Listing 1 shows a short snippet that runs a 10x10 problem on a 4x4 analog mesh computer.

For cases where the dimension of the problem does not equal the dimension of the mesh, scaling is required. Up-scaling maps a small problem to a large mesh by mapping the boundary conditions of the problem to corresponding points on the mesh. Unmapped boundary points are left unset and automatically solved for by the mesh fabric. Down-scaling is required when a large problem is mapped to a small mesh, and is accomplished by mapping the boundary points of the problem to corresponding points on the mesh, thus removing any extra points. While scaling is done automatically, users can adjust scaling targets by manually setting the required mesh dimension or providing an accuracy requirement and allowing the stack to select an appropriate mesh dimension.

Our stack also has a Simulink interface for a graphical user interface (GUI). This interface includes a Simulink model whose implementation in Matlab calls the Python library using Matlab's interoperability support. Arguably, this interface is more limiting than the Python interface, however, Simulink is a very common tool in science and engineering workflows and this interface makes using an analog mesh computer even simpler for those who do not have the expertise or time budget to program in Python.

3.2 Back-end: Multipass Translation

The back-end translates a high-level problem description into a low-level hardware description. To enable portability between different hardware technologies, this translation happens in two passes and uses an intermediate representation.

First, the problem description is translated into a mesh-like data structure. Analogous to an abstract syntax tree commonly found in compilers, this data structure represents nodes and links in the analog mesh. With this representation, characteristics of links (e.g. thermal conductance for a heat transfer problem) and nodes (e.g. whether it is a boundary condition) are stored. For simulation purposes, the stack also supports some hardware characteristics (e.g. parasitic capacitance, leakage, component variability).

Second, this data structure is translated into a netlist that can be used with different circuit simulators. Currently, we support two netlist flavors: (1) an electrical circuit simulation using ngspice and (2) an optical circuit simulation using Lumerical Interconnect.

Finally, an appropriate simulator module loads the generated netlist in the simulator, executes the simulation, parses the output generated by the simulator and populates the mesh data structure with the collected data. The raw data is exposed to the user as a 2D numpy array, allowing the user to do custom data processing. However, the stack also provides an analysis module that can help with some basic operations as well as visualization via matplotlib.

4 EVALUATION METHODOLOGY

We created an evaluation methodology for the software stack comprising three test cases. Test cases were chosen as to represent the spectrum of anticipated needs: 1) problem size < mesh size, 2) problem size = mesh size and 3) problem size > mesh size. We described three steady-state heat transfer configurations, each with different boundary conditions.

Each configuration was based on a 64×64 square surface with a consistent resistance throughout. The first configuration consists of a single source and single sink residing on opposite edges of the surface. The second configuration consists of a single source on one edge of the surface, and three sinks on the remaining edges. The third configuration consists of a point source in the center of the surface and sinks on each edge.

Each configuration was created with two options: 1) manual dimension-setting of the abstract mesh by the user, and 2) automatic dimension-setting of the abstract mesh with a user-provided accuracy requirement. These options were chosen to exercise scaling over multiple levels of abstraction. For this experiment, we set the accuracy requirement to 70% and the manually set mesh

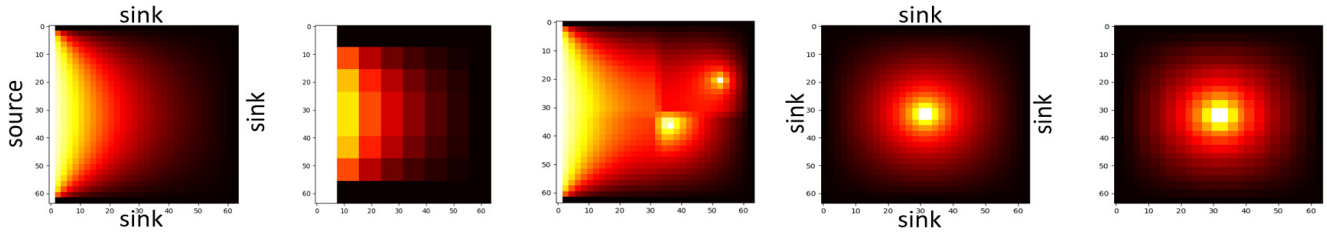

Figure 3: Mesh computation results from 1 x source & 3 x sink configuration (two leftmost) and center-source & edge-sink configuration (two rightmost). Among the configurations, the problem size = mesh size testcase (left) and problem size > mesh size testcase (right) are shown. Note the difference in resolution due to inequalities between the problem and mesh dimensions. The problem size < mesh size testcase (center) is illustrated by solving multiple configurations on a single mesh.

Table 1: Accuracy Results for Generated Meshes

Test Configuration	User Settings	Accuracy	Mesh Size
1 x source, 1 x sink	Mesh Size: 32	98.41%	32
	Accuracy: 70%	91.66%	8
1 x source, 3 x sink	Mesh Size: 32	95.78%	32
	Accuracy: 70%	74.66%	8
center-source	Mesh Size: 32	80.41%	32
	Accuracy: 70%	70.10%	24

dimension to 32×32. All configurations were used to generate a nanophotonic mesh-based accelerator, as described in [1].

5 RESULTS

Figure 3 shows heatmaps generated from various test cases. The differing result accuracy from like configurations are due to inequalities between mesh resolution and problem size. The center test case in the figure illustrates simultaneous computations taking place when the problem is smaller than the mesh. This is accomplished by isolating sub-meshes by setting a high-attenuation region at their boundaries.

Table 1 shows accuracy results for each test configuration. Mesh size was either explicitly set by the user, or automatically generated by the stack to satisfy a minimum accuracy requirement. Note that dissimilar configurations required different mesh sizes to meet the accuracy requirement. We suspect that the stack could be augmented with different optimization strategies to ensure the most efficient mesh is selected for a given configuration.

6 CONCLUSION

Due to the importance of minimizing time-to-solution in physics simulations, various PDE solvers have been proposed. One specific class of PDE solver, the analog mesh computer, minimizes time-to-solution by solving Poisson equations in a one-shot fashion. Recent advances in material science have addressed the shortcomings inherent in traditional analog mesh computers, thus enabling their integration into modern heterogeneous architectures [1]. However, for these architectures to gain widespread acceptance, infrastructure must be developed to enable intuitive interaction between domain experts in the physical sciences and mesh computers.

We have implemented a software stack which supports integration of classical and analog mesh computers. This stack enables users to define a PDE in terms of computational mesh parameters and boundary conditions, and generates an equivalent mesh representation which is then mapped to available analog mesh hardware. Our results show successful mappings for various configurations of steady-state heat transfer. Future work will consider application of this stack as an enabler for abstractions such as virtualization and automated hardware synthesis flows.

ACKNOWLEDGMENTS

This work is funded by the NSF RAISE program as award number 1748294 under the NSF EPMD-ElectroPhotonic Mag Devices, CSR-Computer Systems Research, Networking Technology and Systems.

REFERENCES

[1] Jeff Anderson, Engin Kayraklioglu, Shuai Sun, Joseph Crandall, Yousra Alkabani, Vikram Narayana, Volker Sorger, and Tarek El-Ghazawi. 2020. ROC: A Reconfigurable Optical Computer for Simulating Physical Processes. *ACM Transactions on Parallel Computing* 7 (2020). Issue 1.

[2] P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neu Rentz-Reichert, and C. Wieners. 1997. UG – A flexible software toolbox for solving partial differential equations. *Computing and Visualization in Science* 1 (1997). Issue 1.

[3] Johannes Hendrikus Maria Thije Boonkkamp, Robert M. M. Mattheij, and Sjoerd Willem Rienstra. 2005. *Partial Differential Equations: Modeling, Analysis, Computation.* Society for Industrial and Applied Mathematics.

[4] Jack Dongarra. [n.d.]. *Current Trends in High Performance Computing and Challenges for the Future.* Retrieved February 7, 2017 from https://www.acm.org/binaries/content/assets/education/lc-monthly-bulletins/january2017.html

[5] Tarek El-Ghazawi, Volker J. Sorger, Shuai Sun, Abdel-Hameed A. Badawy, and Vikram K. Narayana. 2019. Reconfigurable optical computer. Patent No. US10318680B2, Filed December 5th., 2017, Issued June. 8th., 2019.

[6] Adi Fuchs and David Wentzlaff. 2019. The Accelerator Wall: Limits of Chip Specialization. In *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA '19).* IEEE.

[7] Muhuan Huang, Di Wu, Cody Hao Yu, Zhenman Fang, Matteo Interlandi, Tyson Condie, and Jason Cong. 2016. Programming and Runtime Support to Blaze FPGA Accelerator Deployment at Datacenter Scale. In *Proceedings of the Seventh ACM Symposium on Cloud Computing (SOCC '16).* ACM.

[8] H.J. Lee and William Schiesser. 2003. *Ordinary and Partial Differential Equation Routines in C, C++, Fortran, Java, Maple and MATLAB.* CRC Press, Boca Raton, FL, USA.

[9] George Liebmann. 1950. Solution of Partial Differential Equations with a Resistance Network Analogue. *BRITISH JOURNAL OF APPLIED PHYSICS* (1950).

[10] Luis Pinuel, I. Martin, and Francisco Tirado. 1998. A special-purpose parallel computer for solving partial differential equations. In *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing (PDP '98).* IEEE.

[11] J. Ramirez-Angulo and Mark R. DeYong. 2000. Digitally-configurable analog VLSI chip and method for real-time solution of partial differential equations. Patent No. US6141676, Filed July 22, 1998, Issued October 31, 2000.

[12] Y. Zhao. 2008. Lattice Boltzmann based PDE solver on the GPU. *The Visual Computer* 24 (2008). Issue 5.