PyPOD-GP: Using PyTorch for Accelerated Chip-Level Thermal Simulation of the GPU

Neil He[a], Ming-Cheng Cheng[b], Yu Liu[b,*]

[a]*Department of Mathematics, Yale University*
[b]*Department of Electrical and Computer Engineering, Clarkson University*

Abstract

The rising demand for high-performance computing (HPC) has made full-chip dynamic thermal simulation in many-core GPUs critical for optimizing performance and extending device lifespans. Proper orthogonal decomposition (POD) with Galerkin projection (GP) has shown to offer high accuracy and massive runtime improvements over direct numerical simulation (DNS). However, previous implementations of POD-GP use MPI-based libraries like PETSc and FEniCS and face significant runtime bottlenecks. We propose a **Py**Torch-based **POD-GP** library (PyPOD-GP), a GPU-optimized library for chip-level thermal simulation. PyPOD-GP achieves over 23.4× speedup in training and over 10× speedup in inference on a GPU with over 13,000 cores, with just 1.2% error over the device layer.

Keywords: GPU Thermal Simulation, Proper Orthogonal Decomposition (POD), Finite Element Method, PyTorch

1. Motivation and significance

Advancements in chip design have significantly increased processor power density [1], leading to high temperature gradients and hot spots that degrade performance and reliability [2]. Dynamic thermal management has been implemented to mitigate these issues [3, 4], showing impressive performance gains in heterogeneous multi-core processors [5]. However, these thermal management systems still require efficient and highly accurate thermal simulation. Direct numerical simulations (DNS) provide accurate temperature solutions but are computationally expensive due to their high degrees

*Corresponding Author
Email addresses: neilhe6345@gmail.com (Neil He), mcheng@clarkson.edu (Ming-Cheng Cheng), yuliu@clarkson.edu (Yu Liu)

Nr.	Code metadata description	Metadata
C1	Current code version	v1.0
C2	Permanent link to code/repository used for this code version	`https://github.com/784956494/PyPOD-GP`
C3	None	
C4	Legal Code License	GLP
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python
C7	Compilation requirements, operating environments & dependencies	Python, PyTorch, numpy, pandas, h5py, Dolfin
C8	If available Link to developer documentation/manual	`https://github.com/784956494/PyPOD-GP/blob/main/README.md`
C9	Support email for questions	yuliu@clarkson.edu

Table 1: Code metadata

of freedom (DoF). Alternative methods [6, 7, 8, 9, 10] improve efficiency but sacrifice accuracy and/or resolution. An approach combining proper orthogonal decomposition (POD) with Galerkin projection (GP) [11] enables efficient and accurate simulations [12, 13]. The POD-GP method trains a reduced set of ODEs and POD modes, then computes temperature solutions as a linear combination of these modes weighted by ODE solutions. This approach achieves over five orders of magnitude speedup compared to DNS while maintaining high accuracy [12, 13]. However, previous implementations use CPU-based FEM libraries like FEniCS and PETSc [12, 13] and thus face runtime bottlenecks during training and inference, limiting their practicality for GPUs with many cores. To address this, we propose PyPOD-GP, a GPU-accelerated implementation leveraging PyTorch's tensor operations. Compared to CPU-based implementations, PyPOD-GP achieves over **23.4× speedup** in training and **10× speedup** in inference on an NVIDIA Tesla Volta GV100 GPU, a GPU with 13,440 cores. It also maintains an error of only 1.2% on the heating layer, demonstrating PyPOD-GP's potential for efficient and accurate thermal simulation in large-scale GPU architectures.

2. Software description

PyPOD-GP is an open source, PyTorch-based Python library for GPU-accelerated chip-level thermal simulation using the POD-GP method and its variants [12, 13]. By eliminating the need for MPI-based libraries, PyPOD-GP significantly accelerates high-resolution thermal simulations for many-

core processors with high accuracy. The library offers a user-friendly workflow managed through a single Python object, requiring only the path to training data and chip properties, making it accessible for non-programmers. The software, including installation instructions and test scripts for an AMD ATHLON II X4 610e CPU, is available on GitHub [14]. The following sections detail the software's functionality and the process for thermal simulation.

2.1. Software architecture

The POD-GP Method. The library implements the POD-GP method for thermal simulation, summarized here to clarity. Further details are available in [12]. The thermal solution is approximated as a linear combination of d POD modes η_i as $T(\vec{r},t) = \sum_{i=1}^{d} b_i \eta_i(\vec{r})$, where b_i are weighting coefficients. The POD modes maximize the mean square inner product of the thermal solution with each mode and are obtained by solving the eigenvalue problem for the time-steps correlation matrix A, whose entries are inner products of thermal data at two time steps. The coefficients b_i solve the ODE given by

$$\int_{\Omega} \left(\eta_i \frac{\partial \rho C_s T}{\partial t} + \nabla \eta_i \cdot \kappa \nabla T \right) d\Omega = \int_{\Omega} \eta_i P_d(\vec{r},t) d\Omega - \int_{S} \eta_i (-\kappa \nabla T \cdot \vec{n}) dS, \quad (1)$$

where κ, ρ, C_s are thermal conductivity, density and specific heat, respectively, P_d is the interior power density, S is the boundary surface and and $\overrightarrow{n}$ is its outward normal vector. The expression in eq. (1) can be expressed in matrix form as $C\frac{d\vec{b}}{dt} + G\vec{b} = \vec{P}$, referred to as the C, G matrices and $\vec{P}$ vector.

The PyPOD-GP Pipeline. The PyPOD-GP pipeline consists of three parts: (1) Data Preparation, (2) Model Training, and (3) Temperature Inference, as outlined in fig. 1. The library supports training a single POD-GP model but supports inference on multiple POD-GP models. The users specify parameters for the sampled temperature data, such as chip dimensions and material properties, and provide the path to temperature data in HDF5 format. The pipeline is then executed using a `PyPOD_GP` object. The `train` function trains the model, saving the POD modes and returning the C, G matrices and the $\vec{P}$ vector. When training for multiple models, these values are returned as lists. The `infer` function solves the ODEs using the trained coefficients (in

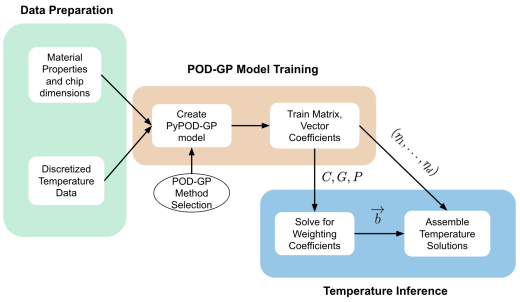

Figure 1: Flow Chart of the Pipeline for PyPOD-GP.

parallel for multiple models), and the `predict_thermal` function generates the temperature solutions.

2.2. Software Functionality

In this section, we detail the steps for thermal simulation. The entire POD-GP process is managed by a `PyPOD_GP` object.

Data Processing. After initializing the `PyPOD_GP` object with appropriate parameters and invoking the `train` function, the model processes discretized temperature data by creating mappings between cells, vertex coordinates, and their corresponding degrees of freedom (DoF). Cell properties, such as the determinant of the Jacobian transformation between global and local cell views ($\det(J_c)$), are pre-computed, and boundary surface cells are identified.

Model Training. After data processing, the model calculates the coefficients described in Sec. 2.1. Integration is performed using Gaussian quadrature, representing the integration as a weighted sum of function values at each cell scaled by $|\det(J_c)|$. Function values are interpolated at k quadrature points (user-specified quadrature degree) via the `get_interpolation` and `get_quad_points` methods (see [15]). The model assumes the data is formatted as a tetrahedron mesh as the chip can be seen as a rectangular prism. During POD-GP execution, the correlation matrix is computed using the `calc_A` function. The `get_modes` function extracts the top d normalized eigenvectors η_i, mapped to domain eigenfunctions via cell vertex mappings, and saved in matrix form U. The `calc_C`, `calc_G`, and `calc_P` functions then compute C, G, and $\vec{P}$, respectively. All computations are performed on the GPU, leveraging vectorized operations for efficiency by copying tensors from the CPU to the GPU and ensuring domain-wide vector manipulations.

Model Inference. After training, the `infer` function solves the ODEs to compute weighting coefficients $\vec{b}$ for each time step. A `POD_ODE_Solver` object solves the ODEs using a fourth-order Runge–Kutta method, returning the coefficients in matrix B, where rows index time and columns index space. The `predict_thermal` function generates predicted temperatures across the domain by computing $B[i]^T U$, where $B[i]$ is the i-th row.

3. Illustrative example

This section demonstrates chip-level thermal simulation on an NVIDIA Tesla Volta GV100 GPU with 13,440 cores. We evaluate the efficiency by comparing its runtime to previous CPU-based implementations. We also report the error across the device's heating layer to validate the accuracy of our library.

3.1. Local Ensemble POD

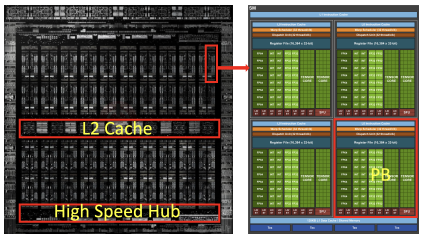

Figure 2: GPU Floor Plan

A localized method similar to [13] was used in this example. The chip was divided into smaller units called heat source blocks (HSBs). Since the temperature induced by an HSB becomes negligible beyond a few thermal lengths, data was sampled from a truncated domain around each HSB. Separate POD-GP processes were trained for each HSB, and the results were assembled to simulate the chip's overall thermal behavior. For the NVIDIA Tesla Volta GV100 GPU, there are 80 streaming multiprocessors (SMs). Each SM comprises four texture units and four processing blocks (PBs). Fig. 2 shows a floor plan of the chip. A total of 404 HSBs were selected, each representing individual PBs, groups of four texture units within an SM, L2 cache, high-speed hub, or one of the two memory interfaces. As the chip's has many repetitive units, a single POD-GP model can be trained to represent multiple identical HSBs. As a result, only 16 POD-GP models were needed to simulate the thermal effects of all 404 HSBs. For each model, data was sampled over 2000 time steps with a discretization of over 4 million cells per HSB.

3.2. Runtime Analysis

To verify the faster runtime of our implementation compared to FEniCS- and PETSc-based methods, PyPOD-GP was tested using the experimental setup described above, running on an NVIDIA Tesla Volta GV100 48GB GPU. The baseline POD-GP process was executed on an Intel(R) Xeon(R) Gold 6130 CPU with 20 MPI processes, utilizing FEniCS for training and PETSc for inference, as implemented in [12]. Although only 16 POD-GP models were trained, the total computation time for 404 HSBs was estimated by scaling the runtime for HSBs of the same size. Comparison for training runtime is shown in table 2a and for inference in table 2b. PyPOD-GP achieves over **23.4× speedup** during training and over **10×** during inference with three or more modes, highlighting its efficiency. Both the GPU and CPU used for the experiment were released in 2017, ensuring a fair runtime comparison.

3.3. Accuracy Evaluation

To validate the accuracy of PyPOD-GP, thermal simulation results on the chip's top layer (device layer) were compared to the sampled temperature data. The least squares (LS) error was computed as

$$\text{Err} = \sqrt{\frac{\sum_{i=1}^{N_t} \int_\Omega e^2(\vec{r}, t_i)\, d\Omega}{\sum_{i=1}^{N_t} \int_\Omega \left[T(\vec{r}, t_i) - T_{\text{amb}}\right]^2\, d\Omega}}, \tag{2}$$

Task	Method	Runtime(s)	Improvement
A Matrix	FEniCS(CPU)	5.75×10^7	-
A Matrix	**PyPOD-GP**	$\mathbf{2.46 \times 10^6}$	**23.4X**
C Matrix (50 Modes)	FEniCS(CPU)	1.39×10^4	-
C Matrix (50 Modes)	**PyPOD-GP**	$\mathbf{7.82 \times 10}$	**177.7X**
G Matrix (50 Modes)	FEniCS(CPU)	1.37×10^4	-
G Matrix (50 Modes)	**PyPOD-GP**	$\mathbf{1.29 \times 10^2}$	**106.2X**
P Vector (20 Modes)	FEniCS(CPU)	7.68×10^3	-
P Vector (20 Modes)	**PyPOD-GP**	$\mathbf{2.18 \times 10^2}$	**35.2X**

(a) Training time

#Modes	Method	Runtime(s)	Improvement
2 Modes	PETSc(CPU)	78.6	-
2 Modes	**PyPOD-GP**	**8.9**	**8.7X**
3 Modes	PETSc(CPU)	92.9	-
3 Modes	**PyPOD-GP**	**9.2**	**10.1X**
4 Modes	PETSc(CPU)	93.1	-
4 Modes	**PyPOD-GP**	**8.9**	**10.5X**
5 Modes	PETSc(CPU)	97.0	-
5 Modes	**PyPOD-GP**	**9.6**	**10.1X**

(b) Inference time

Table 2: Runtime comparison for PyPOD-GP and baselines on 404 HSBs, in seconds

where N_t is the number of time steps, $T(\vec{r}, t_i)$ is the sampled temperature at time step i, T_{amb} is the ambient temperature, and $e^2(\vec{r}, t_i)$ is the squared temperature difference between PyPOD-GP and the sampled temperature at time i. The results, shown in fig. 3, indicate that using 5 or more POD modes achieves less than **2%** LS error across the device layer, with the error reducing to **1.2%** when 7 or more modes are used.

4. Impact

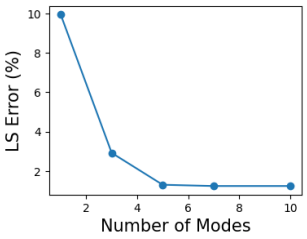

Figure 3: Error at device layer by number of modes from PyPOD-GP

As discussed in the introduction, effective cooling of modern chips is critical for device performance and lifespan. Modern dynamic cooling systems require thermal monitoring that is accurate, high-resolution, and efficient. While POD-GP offers an efficient and accurate solution, previous CPU-based FEM implementations suffered from significant runtime bottlenecks. Our GPU-optimized implementation, PyPOD-GP, achieves remarkable speedups, representing a significant advancement toward real-time thermal monitoring. The speedup provided by PyPOD-GP offers several practical benefits. Faster training enables the use of finer meshes for higher quality sampled temperature data, which leads to notably improved accuracy [12]. Additionally, the speedup makes thermal prediction across multiple chips more feasible, such as during multi-GPU operations for training multi-head large language models [16]. Finally, PyPOD-GP contributes to the research community by offering a faster tool that enables more advanced thermal simulation studies and a framework for applying POD methods to other physics problems.

5. Conclusions

We present PyPOD-GP, a library implementing POD-GP methods and variants with GPU-optimized tensor operations. Compared to previous CPU-based FEM implementations using FEniCS and PETSc, PyPOD-GP achieves significant speedups in both training and inference. This enables more practical use of the POD method for real-time thermal management, particularly with higher-quality training data or multi-device predictions.

Acknowledgments

This project is supported by National Science Foundation Research Experience for Undergraduates (REU) site "HPC with Engineering Applications" under Grant No. OAC-2244049.

References

[1] S. I. Guggari, Analysis of thermal performance metrics application to cpu cooling in hpc servers, IEEE Trans Compon Packag Manuf Technol (2021).

[2] A. Heinig, R. Fischbach, M. Dittrich, Thermal analysis and optimization of 2.5 d and 3d integrated systems with wide i/o memory, in: ITHERM, IEEE, 2019, p. 86–91.

[3] S. Alzemiro, I. Weber, A. L. del Mestre Martins, F. Gehm Moraes, Dynamic thermal management in many-core systems leveraged by abstract modeling, in: ISCAS, 2021, pp. 1–5.

[4] M. S. Mohammed, A. K. Al-Dhamari, N. Rahman, Ab Al-Hadi Ab amd Paraman, A. A. M. Al-Kubati†, M. N. Marsono, Temperature-aware task scheduling for dark silicon many-core system-on-chip, in: ICMSAO, IEEE, 2019, pp. 1–5.

[5] Y. G. Kim, M. Kim, J. Kong, S. W. Chung, An adaptive thermal management framework for heterogeneous multi-core processors, in: Transactions On Computers, Vol. 69, IEEE, 2020, pp. 894–906.

[6] W. Huang, K. Sankaranarayanan, R. J. Ribando, M. R. Stan, K. Skadron, An improved block-based thermal model in hotspot 4.0 with granularity considerations, in: WDDD, 2007.

[7] Z. Yuan, P. Shukla, S. S. Chetoui, Nemtzow, S. Reda, A. K. Coskun, Pact: An extensible parallel thermal simulator for emerging integration and cooling technologies, in: Trans. CAD ICs Syst, Vol. 41, IEEE, 2021, pp. 1048–1061.

[8] A. Ziabari, J.-H. Park, E. K. Ardestani, S.-M. Renau, Jose Kang, A. Shakouri, Power blurring: Fast static and transient thermal analysis method for packaged integrated circuits and power devices, in: Trans. VLSI Syst, Vol. 22, IEEE, 2014, pp. 2366 – 2379.

[9] K. Zhang, A. Guliani, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, Machine learning-based temperature prediction for runtime thermal management across system components, in: Trans. Parallel Distrib. Sys, Vol. 29, IEEE, 2017, pp. 405–419.

[10] H. Sultan, A. Chauhan, S. R. Sarangi, A survey of chip-level thermal simulators, in: ACM Comput. Surv., Vol. 52, 2019, pp. 1–35.

[11] G. Berkooz, P. Holmes, J. Lumley, The proper orthogonal decomposition in the analysis of turbulent flows, in: Annu. Rev. Fluid Mech, Vol. 25, 1993, pp. 539–575.

[12] L. Jiang, A. Dowling, M.-C. Cheng, Y. Liu, Podtherm-gp: A physics-based data-driven approach for effective architecture-level thermal simulation of multi-core cpus, in: Transaction on Computers, Vol. 72, IEEE, 2023, pp. 2951–2962.

[13] L. Jiang, A. Dowling, Y. Liu, M.-C. Cheng, Ensemble learning model for effective thermal simulation of multi-core cpus, in: Integration, Vol. 97, Elsevier, 2024, p. 102201.

[14] PyPOD-GP GitHub Repository, https://github.com/784956494/PyPOD-GP, Last Visited: Nov. 26, 2024 (2024).

[15] M. G. Larson, F. Bengzon, the finite element method: theory, implementation, and applications, Springer, 2013.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: 31st Conference on Neural Information Processing Systems, 2017.