Poseidon: A Source-to-Source Translator for Holistic HPC Optimization of Ocean Models on Regular Grids

Maurice Brémond[1], Hugo Brunie[2,3], Laurent Debreu[1,3], Florian Lemarié[1],
Julien Rémy[1,3], Philippe Rosales[2], Martin Schreiber[2,3], Arthur Vidard[1,3]
AIRSEA team
Centre Inria de l'Université Grenoble Alpes & Laboratoire Jean Kuntzmann
[3] Université Grenoble Alpes
Saint-Martin-d'Hères, France
[1]{first name}.{last name}@inria.fr, [2]{first name}.{last name}@univ-grenoble-alpes.fr

Rupert W. Ford †, Andrew R. Porter, Sergi Siso
Hartree Centre
Science and Technology Facilities Council
Warrington, United Kingdom
{first name}.{last name}@stfc.ac.uk

Martin Schulz, Anna Mittermair
TUM School of Computation, Information and Technology
Technical University of Munich
Munich, Germany
{first name}.{last name}@tum.de

***Abstract*—Ocean simulation models often fail to reach top performance from modern high-performance computing (HPC) architectures. Proposed solutions are often complete code rewrites which are impractical due to their cost and time requirements.**

We introduce "*Poseidon*", an HPC-oriented source-to-source translator for fluid dynamics solvers of ocean simulation models written in Fortran with a regular grid structure. Our main motivation is to perform architecture-specific optimizations by recovering from the source code and the numerics experts knowledge the high-level information and semantics.

***Index Terms*—HPC, ocean, simulation, source, optimization, fortran, transpiler**

I. Introduction and Methodology

We introduce "*Poseidon*", an HPC-oriented source-to-source translator for the dynamic core of fluid dynamics solvers for ocean simulation models with a regular grid structure written in Fortran. The primary motivation is to perform architecture-specific optimizations, by gaining back the high-level information and semantics from partial-differential equation models which got lost during the process of converting numerics to source code. *Poseidon* uses a multi-stage approach: (a) A PDE-solver-specific front-end to uplift the solver to a high-level control flow (CF) by extracting its numerics and semantics, (b) a conversion of the CF to a high-level Data Flow (hyper)Graph (DFG), (c) DFG-based modifications of the PDE solver (e.g., kernel fusion), and (d) finally scheduling and code generation in backends.

"Initiatives de recherche à Grenoble Alpes" (IRGA) 2024 for funding part of this project

II. Background and motivation

Ocean simulations on modern supercomputers often utilize only a small fraction of their peak performance, despite representing a significant share of usage [6, 5]. Many of these codes predate the integration of GPUs into supercomputers, and porting them to GPU architectures is challenging and costly due to the need for extensive rewrites of existing Fortran code.

To address the complexity of programming for massively parallel architectures, researchers have developed Domain Specific Languages (DSLs), compilers, and runtimes that abstract this complexity [7].
We focus on a source-to-source solution that maximizes HPC performance with minimal code rewrite, ensuring that numerical scientists do not need to write HPC optimizations, to tackle the performance portability challenge of ocean simulations on HPC architectures. As demonstrated in [2], parallel code can be generated directly from sequential Fortran programs.

III. Methodology details

A. Uplifting from Fortran to a Data Flow HyperGraph

Poseidon uses PSyclone [4] to parse Fortran code, extracting variables and meta-information such as type, kind, and lifespan. Computational kernels, communications, and boundary conditions nodes are identified to build a Control Flow (CF) representation, which is then transformed into a Data Flow (hyper)Graph (DFG), where edges represent data dependencies between nodes.

B. DFG-based code transformation

Following some transformation rules, computational kernels can be fused together when they are linked by a data dependency edge. The computations from the input (upflow) kernel can be substituted in the output (downflow) one, thus reducing the number of memory access and intermediate arrays being allocated. In order to avoid loop dependencies, we assign to local scalar variables and perform substitutions in expressions.

C. Scheduling and code generation

After performing static scheduling, code generation is managed by converting *Poseidon*'s intermediate representation (IR) into PSyclone IR, followed by Fortran code generation via the PSyclone backend.

IV. Results

We tested *Poseidon* on a 2D fast barotropic solver research code involving over 20 stencil-based kernels, adapted from the CROCO ocean simulation model [5], which already leads to a high combinatorial complexity for kernel fusion. A naive autotuning method was applied to generate various versions of the Fortran code. Results show that optimal performance on an Intel® Xeon® W-2295 CPU @ 3.00GHz with 18 cores is achieved not by fusing all kernels but by selective fusion. The best kernel fusion choices yielded up to a 1.75x speedup with OpenMP parallelization, whereas full fusion increased single-threaded execution time by 10%, while optimal fusion choices still achieved a 1.29x speedup.

V. Related work

In [1], Tal Ben-Nun et al. applied DaCe to optimize FV3, an Earth System Model originally written in Fortran. Their optimizations are similar to what we aim to achieve with Poseidon, but FV3's dynamical core was entirely rewritten in Python.

Psyclone [4], a Python-based source-to-source Fortran compiler, allows us to parse Fortran code and to create a DFG-based intermediate representation in *Poseidon*.

In [2], Nick Brown et al. used the Flang LLVM MLIR compiler to extract stencil patterns from Fortran code, writing their xDSL framework as a compiler pass and leveraging existing MLIR dialects and transformations. Our approach, however, supports the use of any general-purpose compiler.

VI. Summary and conclusion

Future work will involve applying *Poseidon* to production ocean simulation models like NEMO [6] and CROCO [5] in collaboration and co-design with their developers. Co-design collaboration has already shown good results in other HPC applications [3, 8, 9].

Acknowledgments

We thank "Initiatives de recherche à Grenoble Alpes" (IRGA) 2024 for funding part of this project.

Some experiments we performed in this and other projects were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.

References

[1] Tal Ben-Nun et al. "Productive performance engineering for weather and climate modeling with Python". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '22. Dallas, Texas: IEEE Press, 2022. ISBN: 9784665454445.

[2] Nick Brown et al. "Fortran performance optimisation and auto-parallelisation by leveraging MLIR-based domain specific abstractions in Flang". In: *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. SC-W '23. Denver, CO, USA: Association for Computing Machinery, 2023, pp. 904–913. ISBN: 9798400707858. DOI: 10.1145/3624062.3624167. URL: https://doi.org/10.1145/3624062.3624167.

[3] A. Dubey et al. "A tool and a methodology to use macros for abstracting variations in code for different computational demands". In: *Future Generation Computer Systems* (2023). ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2023.07.014. URL: https://www.sciencedirect.com/science/article/pii/S0167739X23002649.

[4] Rupert Ford et al. *PSyclone*. Version 2.5.0. May 2024. DOI: 10.5281/zenodo.11190458. URL: https://doi.org/10.5281/zenodo.11190458.

[5] Swen Jullien et al. *CROCO Technical and Numerical Documentation*. Apr. 2024. DOI: 10.5281/zenodo.11036558. URL: https://doi.org/10.5281/zenodo.11036558.

[6] Gurvan Madec et al. *NEMO Ocean Engine Reference Manual*. Version v4.2.1. July 2023. DOI: 10.5281/zenodo.8167700. URL: https://doi.org/10.5281/zenodo.8167700.

[7] Sparsh Mittal and Jeffrey S. Vetter. "A Survey of CPU-GPU Heterogeneous Computing Techniques". In: *ACM Comput. Surv.* 47.4 (July 2015). ISSN: 0360-0300. DOI: 10.1145/2788396. URL: https://doi.org/10.1145/2788396.

[8] Jared O'Neal et al. "Domain-Specific Runtime tonbsp;Orchestrate Computation onnbsp;Heterogeneous Platforms". In: *Euro-Par 2021: Parallel Processing Workshops: Euro-Par 2021 International Workshops, Lisbon, Portugal, August 30-31, 2021, Revised Selected Papers*. Lisbon, Portugal: Springer-Verlag, 2021, pp. 154–165. ISBN: 978-3-031-06155-4. DOI: 10.1007/978-3-031-06156-1_13. URL: https://doi.org/10.1007/978-3-031-06156-1_13.

[9] Johann Rudi et al. *CG-Kit: Code Generation Toolkit for Performant and Maintainable Variants of Source Code Applied to Flash-X Hydrodynamics Simulations*. 2024. arXiv: 2401.03378 [cs.DC]. URL: https://arxiv.org/abs/2401.03378.