Automated MPI-X code generation for scalable finite-difference solvers

DEVITO George Bisbas [1] Rhodri Nelson [1] Mathias Louboutin [2] Fabio Luporini [2]
Paul H.J. Kelly [1] Gerard Gorman [1]

[1]Imperial College London, UK [2]Devito Codes, UK

Imperial College London

EPSRC Engineering and Physical Sciences Research Council

archer2

Scan here for full paper

Motivation

- **PDE solvers at scale:** Solving Partial Differential Equations (PDEs) at scale to model diverse scientific phenomena is a complex and time-consuming task. The finite difference (FD) method, commonly used in practical scientific applications, often results in compute-intensive and memory-demanding stencil codes that can be challenging to optimize and scale.

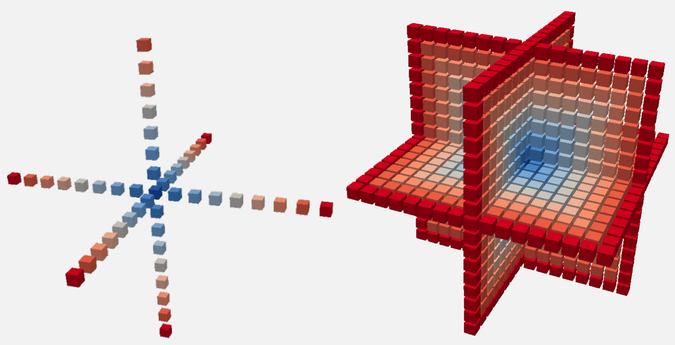

Figure 1: Stencil kernels for practical applications are not always simplistic. They may have a high number of floating point operations and memory requirements.

- **Need for automated code generation**: Developing scalable HPC solutions for large-scale simulations can be tedious and error-prone, even for HPC experts.
- **The lack of suitable abstractions**: Interdisciplinary scientists struggle to effectively utilize HPC resources for solving real-world problems due to the lack of suitable abstractions. Abstractions have proven successful in various fields, such as CFD and ML, by simplifying the complexities of large-scale scientific simulations.
- **Devito [1, 2] as a solution**: A symbolic DSL and compiler framework automates the generation of FD solvers, offering a high-level symbolic math input designed for real-world applications and an optimizing compiler framework with a primary focus on **seismic and medical imaging**.
- All contributions, code, and benchmarks are **open source** and available online.

Contributions

- A novel **end-to-end software stack** that automates and abstracts away **distributed-memory parallelism via Message-Passing Interface (MPI)** code generation within the Devito compiler framework [3].
- **Seamless integration of MPI**, with OpenMP, OpenACC, SIMD vectorization, cache-blocking, and other performance optimizations, targeting CPU and GPU clusters.
- **Flexible code-generation** for various computation and communication patterns to cater to stencil kernels with different compute and memory requirements.
- **Support for operations beyond stencils**, including sparse sources and receivers, callbacks for 3rd-party libraries (e.g., lossy compression, FFTs), essential for real-world applications.
- **Comprehensive strong scaling evaluation** for four wave propagator stencil kernels, used in academia and industry, scaling up to 16384 CPU cores and 128 A100 GPUs.

Automated Distributed Memory Parallelism and MPI code-gen

1. The Devito compiler uses a multi-step process involving multiple intermediate-representation levels, each responsible for applying optimizations for stencil kernels.
2. Devito employs domain decomposition to logically partition the grid among MPI ranks. Devito handles data access transparently, distributing data to processes according to the decomposition. The compiler analyzes data dependencies and ensures efficient halo exchanges through optimization passes, supporting various computation and communication patterns.
3. Three primary communication/computation patterns are supported: *Basic* involves multi-step synchronous exchanges. *Diagonal* uses single-step diagonal exchanges for corner points. *Full* overlaps communication and computation, splitting domain computation into CORE and REMAINDER (R) areas. The best-performing pattern depends on the computation cost, the size of the communicated halos, and cluster characteristics.

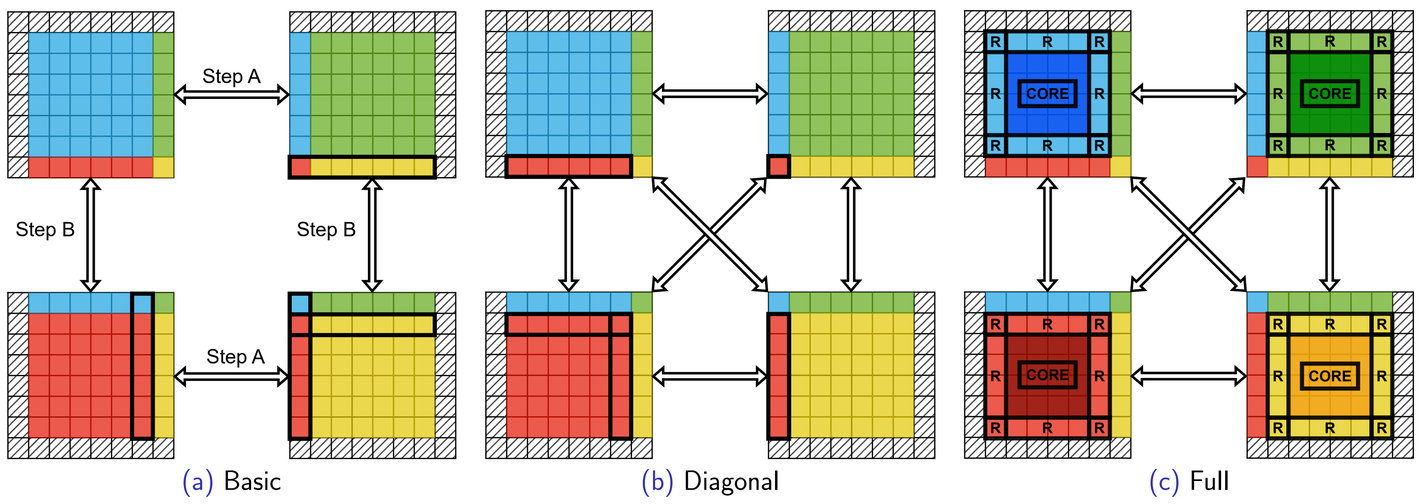

Figure 2: Different colors indicate data owned and exchanged by different ranks. Matching colors on different ranks shows the data updated from neighbors. *Basic* mode communicates exchanges in a multi-step synchronous manner. *Diagonal* performs additional diagonal communications. *Full* performs communication/computation overlap. The domain is split into CORE and REMAINDER (R) areas. REMAINDER areas are communicated asynchronously with the CORE computation.

```
BASIC/DIAGONAL

# Synchronous non-blocking send/
  receive to update the domain
# (Multi-step for Basic)/(Single-
  step for Diagonal)
halo_update()

# MPI_Wait for halos
halo_wait()

# Compute stencil on domain
compute()
```

```
FULL mode

# Asynchronous communication
halo_update()

# Compute CORE region
compute_core()

# Wait for halos
halo_wait()

# Compute the REMAINDER (R) regions
compute_remainder()
```

Summary of communication/computation patterns

MPI mode	Target	Communication	Message batches	#msgs (3D)	Buffer allocation
Basic	CPU, GPU	Sync, No comp overlap	Multi-step	6	runtime (C/C++)
Diagonal	CPU	Sync, No comp overlap	Single-step	26	pre-alloc (Python)
Full	CPU	ASync, comp overlap	Single-step	26	pre-alloc (Python)

Strong scaling cross-comparison

Models were approximated using an 8^{th} order accurate discretization in space.

- Isotropic Acoustic: 5 fields, 290 timesteps, 2^{nd} order in time
- Tilted Transverse Isotropic Acoustic (TTI): 12 fields, 290 timesteps, 2^{nd} order in time
- Isotropic Elastic: 22 fields, 363 timesteps, 1^{st} order in time
- ViscoElastic: 36 fields, 251 timesteps, 2^{nd} order in time

This work used 128 nodes of AMD EPYC 7742 on ARCHER2 UK Supercomputer [4] and 128 A100 GPUs on Tursa [5]. More details on the setup are available in the full paper [3].

Figure 3: Numbers on the ideal line show the percentage of the achieved ideal efficiency (Gpts/s for N nodes)/((Gpts/s for 1 node) * N).

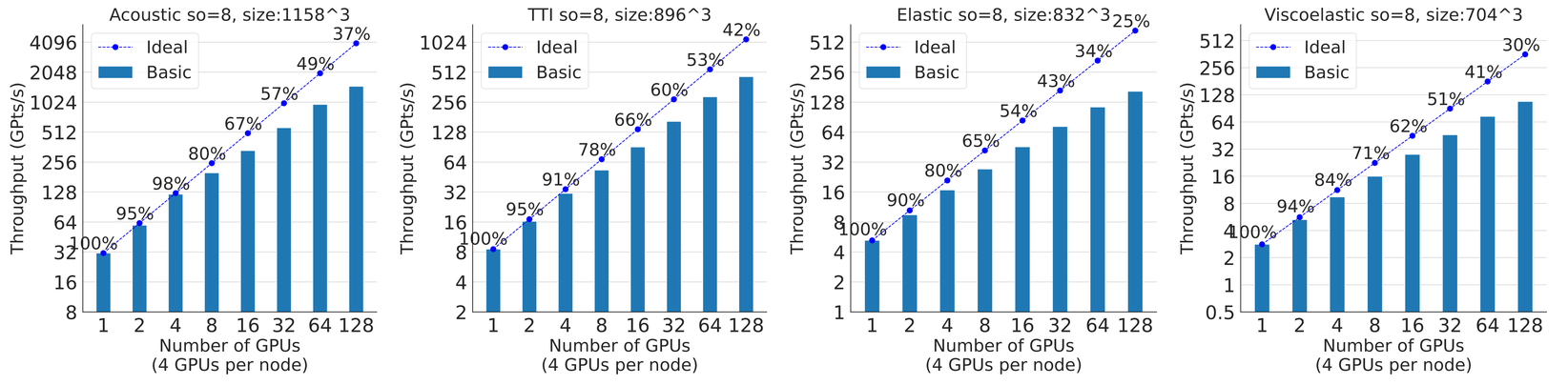

Figure 4: Numbers on the ideal line show the percentage of the achieved ideal efficiency (Gpts/s for N GPUs)/((Gpts/s for 1 GPU) * N).

References

1. Luporini, F., et al. "Architecture and performance of Devito, a system for automated stencil computation." ACM Transactions on Mathematical Software (TOMS) 46.1 (2020): 1-28.
2. Louboutin, M., et al. "Devito (v3. 1.0): an embedded domain-specific language for finite differences and geophysical exploration." Geoscientific Model Development 12.3 (2019): 1165-1187." (2018).
3. Bisbas, G., et al. "Automated MPI code generation for scalable finite-difference solvers." arXiv preprint arXiv:2312.13094 (2023).
4. https://www.archer2.ac.uk
5. https://www.epcc.ed.ac.uk/hpc-services/dirac-tursa-gpu