Motivation

- PDE solvers at scale: Modeling diverse scientific phenomena through Partial Differential Equations (PDEs) on a large scale is intricate and time-consuming. The finite difference (FD) method, widely applied in practical scientific scenarios, often yields compute-intensive and memory-demanding stencil codes.

- Need for automated code generation: Crafting scalable and performance-efficient solutions for large-scale simulations can be tedious and error-prone, even for HPC specialists.

- The lack of suitable abstractions: The ability of interdisciplinary scientists to efficiently leverage HPC resources for solving real-world problems is hampered. Abstractions are key to addressing the complexities of large-scale scientific simulations and have demonstrated success in various fields, including CFD and ML.

- 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.

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, SIMD vectorization, cache-blocking, and other performance optimizations. OpenMP offloading and OpenACC are also supported for GPUs.

- 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.

- A comprehensive strong scaling cross-comparison evaluation for four wave propagator stencil kernels, used in academia and industry, scaling up to 16384 CPU cores.

Automated Distributed Memory Parallelism and MPI code generation

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

Figure 2: Different colors indicate data owned and exchanged by different ranks. Matching colors on different ranks show 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.
- halo_compute (function for Basic)
- halo_update ( Personally for Diagonal)
- MPI_Wait for halos
- halo_wait (function for basic)
- Compute stencil on domain
- halo_wait... (function for diagonal)

Full node

- Asynchronous communication
- halo_update (function for Basic)
- Compute CORE region
- halo_wait (function for Full)
- Compute the REMAINDER (R) regions

Summary of communication/computation patterns

<table>
<thead>
<tr>
<th>MPI mode</th>
<th>Communications</th>
<th>steps for message</th>
<th>#messages</th>
<th>Comp/Comm Overlap</th>
<th>Buffer allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Sync</td>
<td>Multiple</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diagonal</td>
<td>Sync</td>
<td>Single</td>
<td>25</td>
<td>Yes</td>
<td>pre-alloc (Python)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>A Sync</td>
<td>Single</td>
<td>25</td>
<td>Yes</td>
<td>pre-alloc (Python)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Different colors indicate data owned and exchanged by different ranks. Matching colors on different ranks show 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.

Strong scaling cross-comparison

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

- Isotropic Acoustic: 1024 grid, 5 fields, 290 timesteps, 2nd order in time
- Tilted Transverse Isotropic Acoustic (TTI): 1024 grid, 12 fields, 290 timesteps, 2nd order in time
- Isotropic Elastic: 1024 grid, 22 fields, 363 timesteps, 1st order in time
- ViscoElastic: 768 grid, 36 fields, 251 timesteps, 2nd order in time

This work used the ARCHER2 UK Supercomputer [4], 128 nodes of AMD Zen2 (Rome) EYPC 7742 64-core 2.25GHz processor, 128 cores per node, 8 MPI ranks per node, pinned to NUMA nodes, 16 OpenMP threads per rank, totaling 16384 CPU cores, HPE Slingshot with 200Gb/s interconnect.

References

4. https://www.archer2.ac.uk