# Scalable Parallelization of Stencils using MODA

Nabeeh Jumah, Julian Kunkel

Scientific Computing
Department of Informatics
University of Hamburg

$4^{th}$ International Workshop on Performance Portable Programming models for Manycore or Accelerators (P^3MA)
Frankfurt, Germany
20-06-2019

# Introduction

## Performance Demand & Technology Evolution

- Scaling on-chip capabilities reached limits
  - Frequency scaling
  - Core count
- Solutions shift towards multiple nodes

## Multi-node Scalability

- Hardware support using high-speed networks
- Software scalability to exploit hardware resaources

# Software Scalability

- Automatic parallelization is not generally applicable
  - Development is done mostly with general-purpose languages
  - Semantics to drive automatic parallelization are missing
  - New challenges arise facing software
- Manual parallelization
  - Rewriting source code
  - Consider optimal use of available nodes and network
- Portability considerations
  - The complexity of node architectures
  - The diversity of the architectures
  - Various tools and programming models
  - Communication libraries

# Project AIMES

***Advanced Computation and I/O Methods for Earth-System Simulations***

AIMES

- Enhance programmability and performance-portability
- Overcome storage limitations
- Shared benchmark for icosahedral models

Funded within the DFG priority programme SPPEXA

# Application Domain

**Earth system modeling** – applying stencils over wide grids

- Problem domain and grids
  - Dimensions
  - Structure of grids and connectivity
  - Field Localization: staggered vs. collocated grids
- Stencil variability
  - Dimensions
  - Point count
  - Shape
  - Operations

# Parallelization Challenges

## Domain Decomposition

- Distribute the workload on the used nodes
- Divide problem domain into sub-domains
- Sub-domain points are assigned to one node

## Communication

- A node may need to access data on other nodes
- Application should handle communication

## Synchronization

- Data and computation consistency is critical for correctness

# Data Access

- How to access data?
  - Explicit Memory Data Access
  - Memory-Oblivious Data Access (MODA)
- GGDML

# Data Location

## Data Access

- How is the problem domain decomposed
- Which operations need which data
- Where to find that data
- How to make data available for computation

## Explicit Memory Data Access

- Developers take care
- Application code includes necessary details
  - Map global points to local (subdomain mapping)
  - Which data on which node
  - Indices to access local memory on each node

# Our Approach – MODA

- Source code with scientific concepts
- Code unaware of hardware
  - Single vs. multiple nodes
  - Memory; shared vs. distributed, host vs. device ...
  - Processors; multi-core vs. GPU v.s VE vs. ...

## Memory-Oblivious Data Access (MODA)

- Get rid of explicit tracking of data location
  - No node location
  - No array indices
- Alternative indices
  - Scientific basis; e.g. spatial relationships
  - Unaware of underlying memory and hardware

# MODA & GGDML

- GGDML exhibits the flexibility to support MODA
  - User-defined indices
  - Spatial relationships
  - Application adaptable

## GGDML

- **GGDML**:***G**eneral **G**rid **D**efinition and **M**anipulation **L**anguage*
- Grid definition
- Field declaration
- Field data access/update
  - Iterators
  - Access operators
- Stencil operations

*GGDML: Icosahedral Models Language Extensions (Nabeeh Jumah et. al)*
*DOI: 10.15379/2410-2938.2017.04.01.01*

# Example Use of MODA using GGDML - Source Code

Access operators allow developers to refer to stencil components

- without knowing the location of data
- or whether the code is run on multiple nodes

```
// Traverse the cells of the grid
foreach c in grid{
  f_H_new[c] = f_H[c] * W1                +
               (f_H[c.east_neighbor() ] +
                f_H[c.north_neighbor()] +
                f_H[c.west_neighbor() ] +
                f_H[c.south_neighbor()]
               ) * W2;
}
```

# Definitions of Access Operators

- Definitions of access operators are provided seperately from source code, allowing tools to deal handle MODA references:

```
east_neighbor():   XD=$XD+1
north_neighbor():  YD=$YD+1
west_neighbor():   XD=$XD-1
south_neighbor():  YD=$YD-1
```

- Access operator occurences in source code allow tools to identify necessary halo exchange
- Communication code is generated based on analysis including the use of the access operators

# Example Inter-node Halo Exchange

```
int pp = mpi_rank != 0 ? mpi_rank - 1 : mpi_world_size - 1;
int np = mpi_rank != mpi_world_size - 1 ? mpi_rank + 1 : 0;

MPI_Isend(f_G[0], GRIDX + 1, MPI_FLOAT, pp, comm_tag,
    MPI_COMM_WORLD, &mpi_requests[0]);

MPI_Irecv(f_G[local_Y_Eregion], GRIDX + 1, MPI_FLOAT, np,
    comm_tag, MPI_COMM_WORLD, &mpi_requests[1]);

MPI_Waitall(2, mpi_requests, MPI_STATUSES_IGNORE);
```

# Example On-node Halo Copy

- Algorithms within tools use MODA for different purposes
  - Inter-node data communication
  - Handling boundary conditions

Example on-node data copy to handle boundary conditions

```
for (int j = 0; j < local_Y_Eregion; j++) {
    f_F[j][GRIDX] = f_F[j][0];
}
```

# Experiments

## Multi-core experiments environment

- Mistral at German Climate Computing Center (DKRZ)
- Intel(R) Xeon(R) CPU E5-2695 v4 @ 2.10GHz
- Intel C compiler (ICC 18.0.2 )
- IntelMPI (2018.1.163) library

## GPU experiments environment

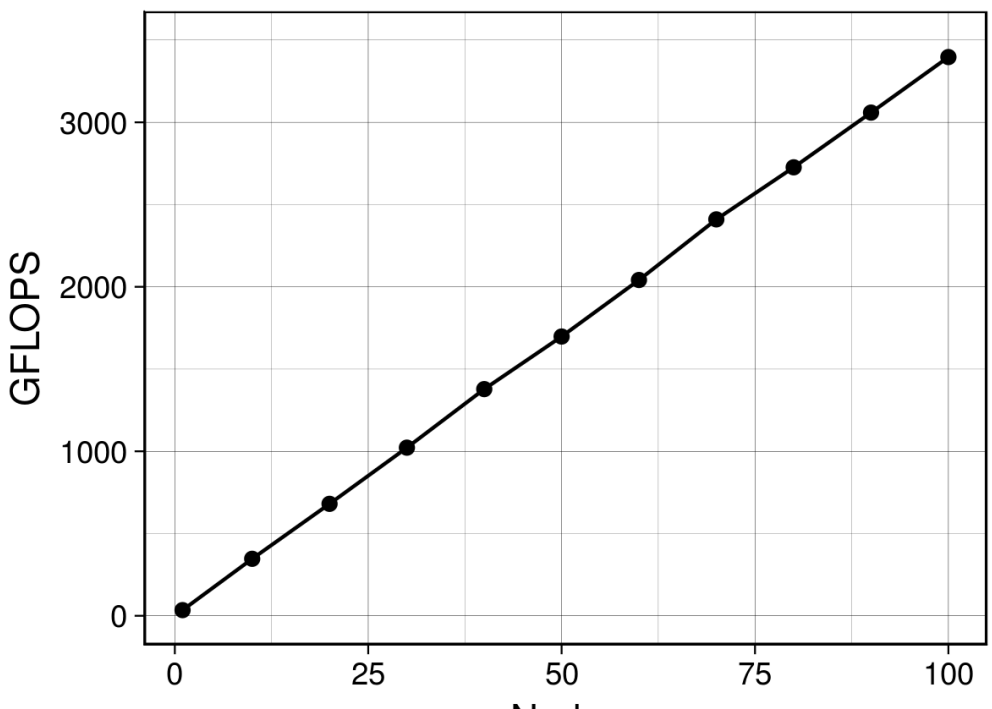
- Piz Daint at Swiss National Supercomputing Center (CSCS)
- Tesla P100 GPUs, 16 GB memory, PCIe interconnect
- PGI (17.7.0) C compiler
- MPICH (7.6.0) library

# Experiments

- Test code
  - Shallow water equations
  - Structured grid
  - Explicit time stepping scheme
  - Finite difference method
  - Eight kernels
    - Flux components
    - Tendencies of the two velocity components
    - Surface level tendency
    - Velocity components
    - Surface level
- Code was run on different numbers of nodes
  - 1, 10, 20, 30 .. 100

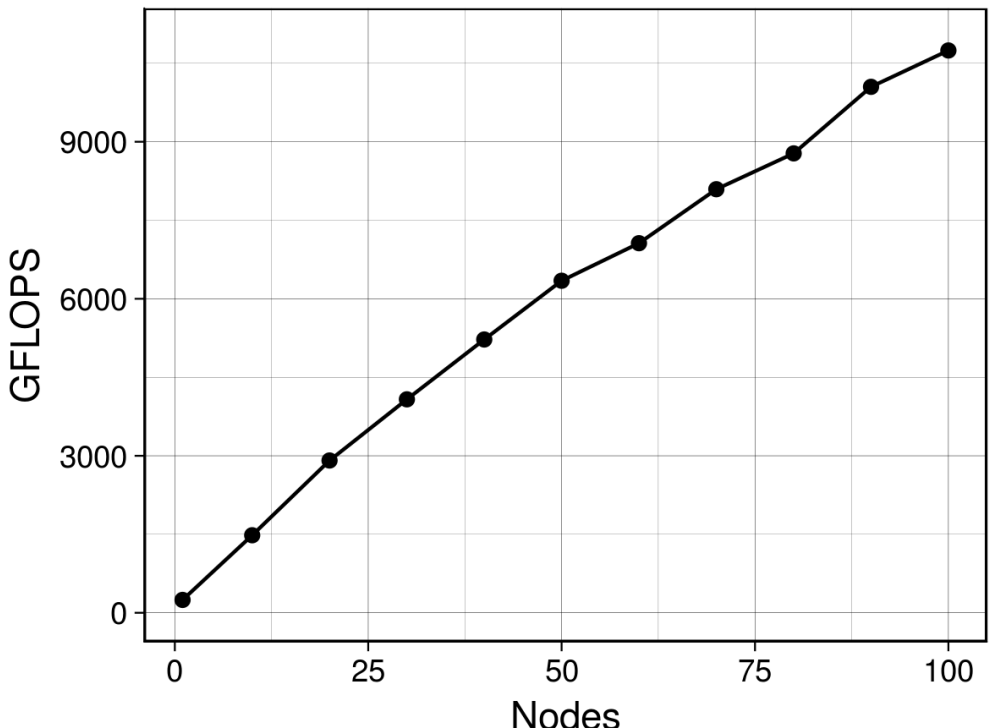
# Results on Broadwell Multi-core Processor

- On-node performance is nearly optimal
  - Code is memory bound
  - Achieved throughput is ~80% of max. memory bandwidth
- Code is scaling efficiently over all tested cases
  - Communication time is small in comparison to application time



# Results on P100 GPUs

- On-node performance is nearly optimal again (Achieved throughput is ~80% of max. memory bandwidth)
- Code is scaling efficiently over all tested cases
  - Linear performance with node count
  - Except for single node (where no data exchange between host and device memories)



# Conclusion

- MODA allows avoiding tracking data location
  - No need to know which data is on which node
  - No need to know location in local memory
- Scientific concepts are used to refer to data
  - Spatial relationships were used
- GGDML access operators support MODA concept
- The information extracted from MODA references allow to identify necessary halo exchange
- The experiments show the success to generate scalable code
- Portability is achieved (multi-core and GPUs are shown)

# Future Work

- Continue working on alternative communication solutions
  - MPI (already started)
  - GASPI (already started)
  - ?
- Explore additional domain decomposition alternative strategies in coupled earth system models

# Acknowledgement


- DFG (German Research Foundation)
- Swiss National Supercomputing Center (CSCS)
- German Climate Computing Center (DKRZ)
- Prof. John Thuburn, University of Exeter