Towards cluster-based parallelization for high-dimensional Cartesian grids

Martin Schreiber, M.Schreiber@exeter.ac.uk
Outline

- (Dynamical) adaptivity everywhere
  - Application scenarios
  - Realization with space-filling curves

- **Non-parallel**: communication
  - Dependency-aware communication **programming model**
  - Realization with stacks and streams

- **Towards parallelization**: Partitioning and communication
  - Communication concepts
  - Parallelization with SFC-generated grids
  - Clustering with stack-based communication

- **Outlook & Vision**
Adaptivity everywhere

- **Sparse (vectors/) matrices/ tensors:** Linear algebra with sparse matrices. E.g. to solve problems with FEM/FD.

- **Adaptive / unpredictable workload:** Boundary condition, augmented Riemann solvers, ...

- **Particle methods:** Move particles to adopt to solution as well as generate/remove particles.

- **Adaptive grids**
  - Dynamically adaptive grids:
    - Adopt grid during computation of solution of stationary solution.
    - Adopt grid to track wave-dominated problems: E.g. areas with high gradient: storm fronts, Tsunami wave.
One-slide motivation: Tsunami

Task: compute results with higher accuracy compared to simulation on regular grid

- Save >95% cells involved in computation

Computation time reduced by one order of magnitude for getting similar results

Developed in collaboration with Alexander Breuer, TUM
Based on GeoClaw (solvers, detide scripts, etc.), GebCo datasets, Generic Mapping Tools, UCSB, Okada displ. model
http://www5.in.tum.de/sierpinski
Dynamically adaptive grids

Keep locality:

- How to optimize for memory hierarchy?
  Re-utilize data already stored in cache
- How to reduce amount of inter-rank communicated data?
  - Avoid ghost layers
  - Only communicate required parts

Intra-cell connectivity:

- How to handle inter-partition and intra-partition connectivity?
- How to store connectivity?
- How to update connectivity with changing grid?

Programmability!

- How to keep it programmable for mathematicians / scientists?
- How to make parallelization (almost) transparent?
Dynamically adaptive grids

- **Graph-based partitioning**
  - Use e.g. dual graph of grid
  - Partition optimizes for number of edge cuts of dual graph
  - High complexity
  - Memory overhead for partitioning (high/low depends on features!)
    - Well studied, see related work on ParMetis, Zoltan, etc.

- **Support for dynamically adaptive Cartesian grids?**

- **Space-filling curves (next slide)**
Use space-filling curves (SFCs)

- Recursively defined
- Inherent locality properties for grid traversals due to recursive definition
- Keeps locality despite dynamically changing grids
- Very well suited for wave-propagation phenomena
Space-filling curves

- Constant complexity for load balancing (reduced to 1D problem)
- Load-balancing reduced to 1D interval problem
- Very well suited for wave-propagation phenomena

Example: Partitioning of mesh with Hilbert curve for 7 compute nodes

Partitioning is not everything: How to solve PDEs on dynamically adaptive grids?

How to communicate?

www.exeter.ac.uk
Overview

We can

- Grid generation with SFC
- Efficient partitioning with SFCs

We need

- Solvers...
- … and therefore data exchange
One-slide Discontinuous Galerkin (DG)

- Given the following non-homogeneous PDE:
  \[ U_t + G_x(U) + H_y(U) = S(U) \]

- Galerkin approach + Gauss div. Theorem:
  \[
  \int_T \frac{dU}{dt} \varphi_i dT - \int_T F(U) \cdot \nabla \varphi_i dT + \oint_T F(U) \varphi_i \cdot ds = 0
  \]
  - Mass-term
  - Stiffness-term
  - Flux-term

- Explicit time stepping (Euler) leads to
  \[
  \tilde{U}_i(t + \Delta t) = \tilde{U}_i(t) + \Delta t M^{-1} \cdot (S(\tilde{U}(t)) + F(\tilde{U}(t), \tilde{U}^+(t)))
  \]
  - U for next time-step
  - U of previous time-step
  - Cell-local
  - Edge-communication

Hyperfaces(D): DOFs for DG simulation
Hyperface(D-1): fluxes and adaptivity information

www.exeter.ac.uk
Different data access approaches

- **Indexed grids with ghost cells:**
  - Flexible data access
  - Store indices to adjacent cells
  - Contra: Hanging hyperfaces very challenging (with Cartesian grids)
  - Contra: Synchronization of ghost layers required
  - [Partitioner approach]

- **Indexed grids without ghost cells**
  - Store indices/MPI ranks to adjacent cells
  - Communicate data
  - [See Behrens et. al. for triangular grids]

- **“Search” adjacent cells:**
  - Flexible data access
  - Synchronization of ghost cells required (?)
  - [See e.g. p4est approach]

- **Stack-based communication with SFCs**
  - More restrictive data communication
  - Efficient communication scheme
  - Avoid ghost layers
  - Allows efficient cluster-based parallelization [Schreiber et. al., so far only for triangular 2D grids]
Data access and approaches

Programmability
Terminology for 2D adaptive Cartesian grids

- Hyperface D0
  - Node
  - Vertex

- Hyperface D
  - Cell
  - Hypercell

- Hyperface D-1
  - Edge

- Hanging hyperface
  - Hanging node
  - Adjoint
  - T-joint
  - The devil in Cartesian grids
Terminology for 3D & nD adaptive Cartesian grids

- Hyperface D0
- Hyperface D-2
- Hyperface D-1
- Hanging hyperface
- Hypercube
- Hyperface D
Application-driven *proposal* for dataflow solver formulation

Example 1, Matrix-free linear algebra: Stencil computations (iterative solvers, etc.)

Cell-centered stencil:

Node-centered stencil:

Communication hyperfaces
Application-driven proposal for dependency-aware solver formulation

Example 2, Discontinuous Galerkin
Explicit timestep & flux computations

Discontinuous Galerkin (explicit Timesteping):

Interpolate

Communication hyperfaces and flux computations
Dataflow language: Communicate & computate

1) Transfer operations
   (HypA → HypB): “SEND”
   Single sender, single receiver
   (HypB → HypA): “RECV”
   Single sender, single receiver
   (so far neglecting hanging hyperfaces)

2) Reduce operations
   (HypA → HypB): “First, Middle & last touch policy” [1,2,3,4]
   Multiple senders, one receiver
   (HypB → HypA): “Recv:
   One sender, multiple receivers

A, B: Dimension of hyperface involved in communication
Concrete example: SWE

DG shallow water simulation (pseudo code):

```plaintext
send(D, D-1) {...};  // Face DoFs
compute(D-1) {...};  // Flux
reduce_interpolate(D-1){...}{...};
sync();
userFun();         // timestep size
recvAll(D-1) {...};  // dt update
```

“Force” developers/mathematicians to create a dependency graph

=> Outlook: Optimizations possible with the help of compiler writers / language designers (embedded language, etc.)

=> Outlook: Dependency graphs helps to develop different JIT compiler backends (GPUs, XeonPhi, etc.)

=> send/recv avoids communication of halo cells (see next slides)
Stack-based communication (1/2)

See [1,2,3,4]

Step 1)
Color hyperfaces for communication

Step 2)
Annotate with new/old:

- New: Adjacent cell is visited during traversal => PUSH data
- Old: Adjacent cell was already visited => PULL data

Coloring has to support stack-based communication

Here, we focus on communication via hyperface dimension D-1 (edges/faces), see [1,2,3,4] for communication via hyperface dimensions < D-1
Stack-based communication results in cache-aware enumeration for accessing communication interfaces.

Enumeration can be also used to store persistent data on hyperfaces with dimensionality \(< D-1\) (parallelization with clustering becomes trivial).

1-4: push operations onto the stack
5-8: pull operations:

Edge 1: push(data)
Edge 2: push(data)
Edge 3: push(data)
Edge 4: push(data)

Edge 5: pull(data)
Edge 6: pull(data)
Edge 7: pull(data)
Edge 8: pull(data)
Handling of hanging hyperfaces

- Improve scalability for frequently changing dynamically adaptive grids:
  - Allow 1:N balanced grids with N>2 or N>3 (otherwise, this requires hierarchical / iterative method)
  - Traverse grid cells only (hierarchy would require subtree of SFC trees)
  - Efficient handling of **hanging nodes**:
    Use stack-feature!
  - Hanging hyperfaces **consecutively stored** on communication stack

- Determine hanging nodes on cell level and execute reduce/interpolate operations
  - => no recursion
  - => no search for adjacent cells
Overview

We can

- Specify mathematical problem with dataflow “language”
- Communicate via stacks and traverse the grid on leaves
- Efficiently handle hanging nodes

We need...

- … to see if it's really working
- Benchmarks and further optimization
Support for high-dimensional dynamically adaptive grids

3D cubes with Z curve

4D hypercube with Z curve
(Only first 3 dimensions rendered)
Data exchange

Stack-based communication variants:

1) Forward DoFs: Increases number of read/writes into cache
2) Communicate pointers: less read/writes, but stack-index-computation overhead

Performance comparison:
- 2D communication test
- Varying number of DoFs per cell used for communication via hyperface
- Runtime variable of number of DoFs
- Measure change in performance

=> Forwarding pointers faster in most of the cases (the plot is just one example!)
Benchmark results: Different SFCs, index vs. stack

- Benchmark based on 2D regular grid
- SWE finite volume simulation
- Indexed versions slower than stack-based ones
- Focus on stack-based communication:
  - Peano curve has highest throughput (See drawback on upcoming slide)
  - Overheads for stack-based communication with Morton order
Bi- or Trisection & Higher-dimensional traversals

- Assume given solution for 5D Poisson problem
  \[ \nabla \cdot (2 \sin(\pi x_0) \sin(\pi x_1) \cos(2\pi x_2/3) \sinh(\sqrt{2}\pi x_4) \cosh(2\pi x_3/3))/32 = 0 \]
- How many grid cells (workload) is required?

Adaptive grids:
- > 60% cells saved in average
- bisection (Z curve) requires in most cases less cells than trisection (Peano curve)
Parallelization concept with clustering
Outlook: Cluster-based Parallelization

Data exchange via partitioning with Peano/Hilbert SFC and D-1 communication

“Traditional approach”:
- Can be used to generate ordering of communicated data for MPI parallelization [see Behrens et. al [5] with Sierpinski SFC]
- Repartitioning: Send/Recv of MPI ranks for each communication element

Challanges:
- Hybrid parallelization necessary for large scale + XeonPhi (MPI+OpenMP/TBB)
- Support for node-based communication and node-based data

Clustering:
- Multiple partitions per memory context
- Use replicated data scheme for data shared among partitions
- Efficient data exchange for partitions in the same memory context

Dynamic clustering:
- Support split and merging clusters
Outlook: RLE connectivity data

Properties of stack-based communication:

- Assures **unique 1:1 relation** for inter-partition shared hyperfaces D-1
- Allows **run-length encoding** of connectivity information
- Communication data is **compactly stored** on the communication stacks => block-wise communication

**Tuple representation:**

( number of shared hyperfaces,  
  x cluster id  
  (x MPI rank) )
Outlook: possibilities with clustering (2D triangles without hanging nodes)

- Implemented
- Possible future work

- **MPI** data migration
- Skipping of grid conformity traversals
- Work stealing on shared domain
- Unpredictable computations of flux solvers
- **Shared**- and **distributed**-memory parallelization
- **Hybrid** parallelization
- Cluster-based local timestepping
- Field of view aware rendering
- Local residual traversals
- Work stealing on distributed memory

Advantages of clustering approach

... and many more ...

[Image: University of Exeter logo and website URL: www.exeter.ac.uk]
Results with Sierpinski SFC
(2D triangles without hanging nodes)

- Clustering with shared-memory parallelization
- SWE radial dam break
Results with Sierpinski SFC (2D triangles without hanging nodes)

- MPI parallelization on MAC Cluster (TUM)
SWE on the sphere

Million cells per second

MPI ranks

0 32 64 96 128 160 192 224 256

www.exeter.ac.uk
Strong scaling on SuperMUC

Parallel efficiency

Number of cores

2690 Mio. Cells (strong scaling)
673 Mio. Cells (strong scaling)
168 Mio. Cells (strong scaling)
168 Mio. Cells (~weak scaling)
Clustering vs. alternative approaches

**Clustering:**
- Efficient shared- and distributed-memory parallelization on partition level
- Pro: Avoid synchronization of ghost layers
- Cons: However, synchronization of replicated hyperfaces follows particular SFC traversal order
- Cluster-based optimizations (local-residual corrections for iterative solvers, spontaneous load imbalances, adaptivity traversals, etc.)
- ... any many more possible benefits

**Alternative concepts** (partitioners, p4est)
- Pro: Support for larger stencils (?)
- Pro: Well established / supported
- Cons: Synchronization of ghost layers required (?)
- Cons: repartitioning expensive
Outlook for application scenarios

(Possible) applications:

- **Galerkin** (classical FEM / DG)
- Simulations in **phase-space** (6D and higher)
- **Big data**: storing (of approximation) & processing
- Ensemble runs with **adaptivity in fourth** dimension
- **Visualization** of nD datasets

- **HEVI** for climate/weather
- **Exascale**: Fault-tolerance (via cluster independence features)
- **Parallelization in time**

*What are your ideas?*
Thank you for your (adaptive) attention

Don't get lost on a space-filling curve

Any questions?

Contact: M.Schreiber@exeter.ac.uk
http://emps.exeter.ac.uk/mathematics/staff/ms698

Related work (Cartesian grids):

Related work on clustering (two-dimensional Sierpinski-SFC generated triangular grids):