DG-Legion: Approaches to Scaling Unstructured Applications

Legion Retreat 2022

Kihiro Bando
Outline

1. A brief introduction to discontinuous Galerkin methods
2. The limits of the implicit exact ghost approach: space-time inefficiency
3. Experience with implicit approximated ghost
4. The semi-explicit ghost formulation: combining the behavior of explicit ghost with automation
1. The discontinuous Galerkin method

- Typical model for many engineering configurations: the compressible Navier-Stokes equations
  \[ \partial_t \mathbf{U} + \nabla \cdot \mathbf{F}_c = \nabla \cdot \mathbf{F}_d \]

- Key ideas of DG
  - Represent the solution in each element as a polynomial of arbitrary order ($hp$)
  - Compatible with unstructured meshes (nearest-neighbor stencil)
  - Higher $p$ $\rightarrow$ higher AI & higher convergence rate
1. The discontinuous Galerkin method
1. The discontinuous Galerkin method

➢ The algorithmic view

\[ d_t \tilde{U} = \text{Volume} + (\text{Private faces}) + (\text{Shared faces}) \]

- Really local!
  - No stencil needed to get the values at the faces
  - Structured evaluation instead of unstructured reconstruction
- Depends on face-sharing neighbors

➢ Many implementations are possible

➢ Focus is on the performance of the distributed execution

➢ Key challenges

  ➢ Sparsity of the ghost data
  ➢ No small upper bound to the number of neighbors per rank
1. The discontinuous Galerkin method
1. The discontinuous Galerkin method

➢ Implementation
   1. Evaluate the values at the faces
   2. Compute the volume and private faces contribution
   3. Compute the shared faces contribution
   4. Update the degrees of freedom

➢ Shape of the relevant region

\[ N_{\text{face/elem}} \times N_{\text{qf}} \times N_{U} \times N_{e} \]

➢ Reduced problem: 2D index space that can theoretically be partitioned in both dimensions
2. Limits of the implicit exact ghost

- Consider a 2D index space of size 18x2

- Blue (red) arrow indicate exchange of the first (second) point along the second dimension
- Ghost of rank 0: (7,1), (8,2), (13,1)
- Note that elements are ordered according to their owner rank
2. Limits of the implicit exact ghost

- We can construct the exact (disjoint) ghost partition
  - G0 = (7,1), (8,2), (13,1)
  - G1 = (5,1), (6,2), (14,1)
  - G2 = (6,1), (8,1)
- We can ask for G0 on rank 0. What does the instance look like?

  Reasons: “normal instances” are easily accessible
  - Points are accessed with their global index
  - Bounding-box instances allow O(1) access using point arithmetic without indirection
2. Limits of the implicit exact ghost

- Space inefficiency
- Hard bottleneck
- Time inefficiency
- Perf bottleneck

- Sparse ghost sub-regions trigger generic packing kernels before data exchange
- As of today, implemented as multi-D d-to-d mem copies

“Compact sparse instances” allow for optimal memory usage at the cost of logarithmic access time.
- The indirection has to be recomputed in every task
- The overhead for highly irregular access patterns is unknown
2. Limits of the implicit exact ghost

- Construct a single index space based on a mesh entity
- Construct multiple views of the data with multiple partitions
- Ask for sub-regions of specific partitions in tasks
- Guaranteed correctness

- Manage multiple logically aliased instances explicitly
- Introduce explicit synchronization
- Introduce explicit communication
- Guaranteed performance when tuned to a machine
2. Limits of the implicit exact ghost

- How sparse can it get in practice? $10^3$ mesh partitioned into 6 pieces.
3. Approximate the ghost: introduce structure

- Forget about the 2\textsuperscript{nd} dimension for now
- What the ghost of rank 0 along dimension 1?
  - \((7,8)\ (13,18)\ ()\ ()\)
- New ordering
  - private: \((2,3,4)\ (9)\ (15,16,17)\ ()\)
  - shared: \((5,1/6)\ (7,11/8,10)\ (13,18,12,14/)\ (19/)\)
- Approximated ghost of rank 0
  - \((5,1/6)\ (7,11/8,10)\ (13,18,12,14/)\ (19/)\)

Approximation 1: points \((10,:))\ are not needed by rank 0

Approximation 2: keep forgetting about the 2\textsuperscript{nd} dim

- At most 2 dense rectangles per pair of connected ranks (per field)
- Sparsity along the 2\textsuperscript{nd} dimension is eliminated
- Cost: significant increase in the size of the payloads
3. Approximate the ghost: introduce structure
3. Approximate the ghost: introduce structure

- $40^3$ elements partitioned into 18 pieces
- For my app, additional sparsity comes from the 2\textsuperscript{nd} dimension
3. Approximate the ghost: introduce structure

\[ N = 81^3, \ p = 4 \]

Strong scaling on Summit (6 GPUs/node)

CUDA IPC goodness: does not use the network

Still plagued by the space inefficiency which forces us to use zero-copy memory
3. Approximate the ghost: introduce structure
4. Semi-explicit formulation

➢ “Semi-explicit” ghost formulation for unstructured applications

- Ordering by consumer, then by producer
- 2-level disjoint complete partitions
- 2D producer-side packing kernel launch
- 1D consumer-side kernel launch
  ghost data is accessed via a static indirection
4. Semi-explicit formulation

➢ "Semi-explicit" ghost formulation for unstructured applications

- Application-specific packing kernels (desired)
- All copies are region-to-region automatic copies
- 2-level disjoint partitioning for efficient dependency analysis

local ghost instance  per neighbor "send" buffers
4. Semi-explicit formulation

➢ Practical considerations
  ➢ How many hops?
  ➢ Current Legion/Kokkos interop forces a rank per GPU
    ➢ Exposing a lot of hardware threads for threads not under direct control of the runtime was found beneficial
  ➢ Profile!
    ➢ -cuda:legacysync 1 flag
    ➢ GASNet credits
    ➢ Experiments with different one-sided RDMA models lead to significant improvements
4. Semi-explicit formulation

\[ N = 91^3, p = 4 \]

\[ N = 195^3, p = 4 \]
Summary

➢ Standard implicit ghost implementations are not compatible with fully GPU resident execution
  ➢ Space-inefficiency: bloated ghost instances
  ➢ Time-inefficiency: can be addressed to some extent by approximating the ghost
➢ The semi-explicit formulation introduces an intermediate region allowing to replicate the behavior of explicit codes
  ➢ No explicit synchronization
  ➢ No explicit communication
➢ Initial scaling results show that this can be competitive with MPI for simple programs
  ➢ More benefits can be expected for more complicated programs