Uintah+Hedgehog: Combining Parallelism Models for End-to-End Large-Scale Simulation Performance

Timothy J. Blattner, John K. Holmen, Damodar Sahasrabudhe, Alexandre Bardakoff, Walid Keyrouz, Martin Berzins
Disclaimer

No approval or endorsement of any commercial product by NIST is intended or implied. Certain commercial software, products, and systems are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST, nor does it imply that the software and products identified are necessarily the best available for the purpose.
Outline

• Context
  • Parallel Programming Models
  • Asynchronous Many-Task Runtimes
  • Exascale Challenges
  • Target Codes
• Hedgehog
• Uintah
• Approach
Parallel Programming Models

• Increasing node-level parallelism and heterogeneity **complicates** programming efforts

• Simplified by parallel programming models
  • DPC++, Kokkos, OpenMP, RAJA, TBB,
    • Hedgehog

• Helps manage increasing concurrency and architectural diversity

• Some offer abstractions mapped to multiple programming models
  • e.g., performance portability layers, model portability

• **Less** duplication with a **smaller** learning curve
Asynchronous Many-Task Runtimes

• Increasing HPC system complexity complicates efficient utilization

• Simplified by asynchronous many-task runtime systems
  • e.g., Charm++, HPX, Legion, PaRSEC, StarPU, Uintah

• Helps manage increasing concurrency, deep memory hierarchies, etc

• Applications overdecomposed into many tasks

• Runtime manages execution details

• More node-level parallelism with a smaller learning curve
Exascale Challenges

• Solutions independently show **promise**

• Interoperability of solutions **unclear**

• Where and how do you add parallelism?

• How do you schedule diverse tasks on heterogeneous systems?

• Code refactors may be substantial

• Solutions may be developed at differing rates
Target Codes

- **Hedgehog**
  - Parallel programming model
  - Open-source C++ header only library
  - Emphasizes *node-level* performance

- **Uintah Computational Framework**
  - Asynchronous many-task runtime system
  - Fluid-structure interaction problems
  - Emphasizes *large-scale* performance

- Direct collaboration between University of Utah and NIST
Hedgehog Overview

- Parallel library for **coarse grain parallelism**
- Based on a **data-flow** graph model
- Using **data pipelining** to get performance
- **No scheduler**
  - Relies only on OS to manage threads
  - Execution based only on the presence of data
- **Header only** library
  - V.1 C++17
  - V.2 C++20
Hedgehog Execution Model
Data-flow graph

• Directed graph (cycles accepted!)
• 1 entry and 1 exit point (source and sink)
• Component:
  • Nodes: computation or state management
  • Edges: directed information flow
• If graph = node → Composability!

• Coarse grain decomposition
  • Node = execution kernel
    • Fine grain parallelism with specialized library
Hedgehog Performance model
Data-Pipelining

Stage starts as soon as data become available
• Asynchronous behavior
• Without any scheduler added!

Data Pipelining representation
Hedgehog Methodology

Methodology used in Hedgehog
Hedgehog State Management

- Local state management between set of nodes in a graph
- Use a state manager and state, both customizable
- A state can be shared amongst different state managers

Communication between a state manager and a state:
1) lock the state,
2) send data to the state,
3) do the state execution,
4) get output data from the state,
5) unlock the state,
6) send output data.

Hedgehog node, e.g. a task, state manager, graph
Hedgehog state manager
Hedgehog state
Antecedent nodes
Hedgehog graph
Successor nodes
...
Example of Single-node Performance with HH

- Hedgehog implementation of General Matrix Multiplication (GEMM)
- Scaling from 1-4 GPUs
  - Achieves >95% of peak using 4 GPUs
- Hardware:
  - 2x 16-core Xeon Silver 4216 @ 2.1 GHz
  - 4x Tesla V100-PCie w/ 32 GB HBM2
Uintah

- Open source computational framework
  - App developers isolated from runtime
- Auto-generated abstract C++ task graph
- Adaptive execution of tasks by runtime
  - Asynchronous out-of-order execution
  - Work stealing
  - Overlapping of communication & computation
- Adaptive mesh refinement
- Performance portability
Uintah’s Runtime System

- MPI+Kokkos using:
  - 1 MPI Process per Device
  - n Running Tasks per Process
  - n Threads per Task

- Per-process task parallelism with Kokkos Partitioning

- Task executors execute individual tasks

- Task executor logic coordinates actions surrounding executors
Research Goals

• Extend Hedgehog to support multiple compute nodes
  • Draw upon strengths of Uintah

• Improve Uintah’s node-level performance
  • Draw upon strengths of Hedgehog

• Generalize the problem space supported by Uintah
  • Heavy focus on time-stepped stencil calculations

• Identify common needs to simplify API of each

• Understand the necessary abstractions for local/global state management
  • Conceptually similar frameworks

• Evaluate the performance and compare with other approaches
Feasibility Studies

• Two approaches taken to explore different communication patterns
  • Uintah + Hedgehog for Stencil Calculations
  • MPI + Hedgehog for DGEMM (Cannon’s Algorithm)

• First explores how Uintah and Hedgehog API interoperates
  • Ensures interoperability
  • Adopted Hedgehog in Uintah with <100 lines of code

• Second explores how Hedgehog API and MPI interoperates
  • Avoids complexity of Uintah
  • Achieved ~60% of peak across 4 V100 GPUs
Future Directions

• Implement a hybrid Uintah + Hedgehog task scheduler

• Possible scheduling approach:
  • One Hedgehog graph per node
  • Hedgehog manages per node local state
  • Uintah manages global state across nodes and coordinates MPI

• Explore other communication patterns

• Generalize core Uintah infrastructure to support a broader set of problems
Summary

• AMT and parallel programming models offer promise for exascale challenges

• Successful prototypes show promise for:
  • Uintah and Hedgehog interoperability,
  • Comparing models to generalize API, and
  • Improving capabilities of each
Thank you

Question?

Hedgehog: https://github.com/usnistgov/hedgehog /
Uintah: http://uintah.utah.edu/