Improving Uintah’s Scalability Through the Use of Portable Kokkos-Based Data Parallel Tasks

John Holmen¹, Alan Humphrey¹, Daniel Sunderland², Martin Berzins¹
University of Utah¹
Sandia National Laboratories²
• Open source software
  • Worldwide distribution
  • Broad user base

• Applications code programming model
  • Physics routines unaware of communications

• Automatically generated abstract C++ task graph

• Adaptive execution of tasks by the runtime system
  • Asynchronous out-of-order execution,
  • work stealing,
  • overlapping of communication & computation
Uintah’s Heterogeneous Runtime System

- MPI+X schedulers support:
  - MPI + PThreads + CUDA
  - MPI + Kokkos

- Shared memory model on-node
  - 1 MPI process per node
Exascale Target Problem
DOE NNSA PSAAP II Center

- Modeling an Alstom Power 1000MWe ultra, supercritical clean coal boiler at scale with Uintah
- Supply power for 1M people
- Targeted 1mm grid resolution = $9 \times 10^{12}$ cells
- Significantly larger than largest problems solved today
Radiation Overview

- Solving energy and radiative heat transfer equations simultaneously

\[ \frac{\partial T}{\partial t} = \text{Diffusion} - \text{Convection} + \text{Source/Sinks} \]

- Need to compute the net radiative source term

- The net radiative source term consists of two terms, one of which requires integration of incoming intensity about a sphere

  - RMCRT approximates the second term using Monte-Carlo methods

\[ \int_{4\pi} I_{\text{in}} d\Omega \Rightarrow \sum_{\text{ray}=1}^{N} I_{\text{ray}} \frac{4\pi}{N} \]
Reverse Monte Carlo Ray Tracing

- Randomly cast rays to compute the incoming intensity absorbed by a given cell
- Rays are traced \textit{away from} the origin cell to compute incoming intensity \textit{backwards to} the origin cell
- When marching rays, each cell entered adds its contribution to the incoming intensity absorbed by the origin cell
- The further a ray is traced, the smaller the contribution becomes

Back path of ray from $S$ to emitter $E$, 9-cell structured mesh patch
Parallel Reverse Monte Carlo Ray Tracing

- Lends itself to scalable parallelism
  - Rays are mutually exclusive
  - Multiple rays can be traced simultaneously at any given cell and/or timestep
  - Backwards approach eliminates the need to track rays that never reach an origin cell
- Parallelize by splitting the computational domain across compute nodes
- Each node is responsible for tracing rays from within each origin cell that it owns across the entire domain
- Nodes must communicate and store geometry information and physics properties for the entire domain
Multi-Level AMR RMCRT

- Global approach involves too much communication

- Use a multilevel representation of computational domain
  - Reduces computational cost, memory usage, and MPI message volume

- Define Region of Interest (ROI), which is surrounded by successively coarser grids

- As rays travel away from ROI, the stride taken between cells becomes larger
Kokkos Performance Portability Library

- C++ library allowing developers to write portable, thread-scalable code optimized for CPU-, GPU-, and MIC-based architectures

- Kokkos provides abstractions to control:
  - how/where kernels are executed,
  - where data is allocated, and
  - how data is mapped to memory

- While Kokkos enables performance portability, the user is responsible for writing performant kernels

- Source Available at: https://github.com/kokkos/kokkos
Uintah Programming Model for Stencil Timestep

Example Stencil Task

\[ U_{\text{new}} = U_{\text{old}} + dt \cdot F(U_{\text{old}}, U_{\text{halo}}) \]

Kokkos Uintah Task

(C++11)

Declare Patch References

\[ \text{Lambda} = \text{Body} \]

Kokkos Parallel Loop

( Patch, Lambda )

Kokkos Unmanaged Views
Memory Structure
Cache, and Vectorization Friendly

Use Kokkos abstraction layer that maps loops onto machine specific data layouts and has appropriate memory abstractions
Kokkos-Based RMCRT

- CPU-, GPU-, and MIC-based RMCRT efforts have resulted in several different implementations

- Introduced RMCRT: Kokkos to consolidate implementations
  - Encapsulated “hot spots” within a Kokkos functor

- This new implementation:
  - Required < 100 lines of new code
  - Replaces a naïve cell iterator with a Kokkos parallel loop, enabling the selection of optimal iteration schemes via Kokkos
  - Enables multi-threaded task execution via Kokkos back-ends
Node-Level Parallelism Within Uintah

- For CPU and MIC architectures, Uintah features **parallel execution of serial tasks**
  - 1 running task per thread
  - Requires at least 1 patch per thread
  - Breaks down as patches are subdivided to support more threads/cores

- Current Kokkos-based scheduler features **serial execution of data parallel tasks**
  - 1 running task per MPI process
  - Requires at least 1 patch per MPI process
  - Eliminates the need to create a new patch to run with another thread

- Next step is a Kokkos-based scheduler w/ **parallel execution of data parallel tasks**
  - We already do this for GPU but not for CPU and MIC
1-Level RMCRT - Strong Scaling
Burns and Christon Benchmark
TACC-Stampede System

RMCRT:CPU: Multi-Threaded MPI Scheduler
RMCRT:Kokkos: Kokkos MPI Scheduler
1 MPI Process and 64-256 Threads per Knights Landing
RMCRT:CPU: 1 Patch per H/W Thread
RMCRT:Kokkos: 1 Patch per MPI Process
128^3 Cells
100 Rays per Cell
Averaged over 7 Timesteps

Mean Time Per Timestep (s)

Knights Landings

Ideal
1 H/W Thread(s) per Core (RMCRT:CPU)
2 H/W Thread(s) per Core (RMCRT:CPU)
4 H/W Thread(s) per Core (RMCRT:CPU)
4 H/W Thread(s) per Core (RMCRT:Kokkos)
1-Level RMCRT - Strong Scaling
Burns and Christon Benchmark
TACC-Stampede System

Kokkos MPI Scheduler
1 MPI Process and 256 Threads per Knights Landing
1 Patch per MPI Process
100 Rays per Cell
Averaged over 7 Timesteps

Mean Time Per Timestep (s)

Knights Landings

Ideal
$512^3$
$256^3$
$128^3$
2-Level Adaptive RMCRT - Strong Scaling
Burns and Christon Benchmark
TACC-Stampede System

- RMCRT:CPU: Multi-Threaded MPI Scheduler
- RMCRT:Kokkos: Kokkos MPI Scheduler
- 1 MPI Process and 256 Threads per Knights Landing
- RMCRT:CPU: 1 Patch per H/W Thread
- RMCRT:Kokkos: 8 Patches per MPI Process
- 100 Rays per Cell
- Averaged over 7 Timesteps
- Fine-Level Halo: [4,4,4]
2-Level Adaptive RMCRT - Strong Scaling
Burns and Christon Benchmark
OLCF-Titan System
TACC-Stampede System

RMCR:CPU: Multi-Threaded MPI Scheduler
RMCR:GPU: Unified Scheduler
1 MPI Process per Node
100 Rays per Cell
Averaged over 7 Timesteps

Mean Time Per Timestep (s)

Nodes

Ideal
L-1: 512³, L-0: 128³ (Stampede:KNL)
L-1: 256³, L-0: 64³ (Titan:GPU)
L-1: 256³, L-0: 64³ (Stampede:KNL)
L-1: 128³, L-0: 32³ (Titan:GPU)
L-1: 128³, L-0: 32³ (Stampede:KNL)
Summary

- Data parallel tasks for CPU- and MIC-based architectures allow Uintah to support larger thread/core counts per node.

- Data parallel tasks offer the potential to improve microarchitecture use (e.g. per-patch work can be computed cooperatively by multiple threads sharing a cache).

- Use of Kokkos allows data parallel tasks to be introduced in a portable manner:
  - Helps avoid code divergence and architecture-specific implementations.
  - Reduces the gap between development time and our ability to run on newly introduced machines.

- Titan comparisons offer encouragement as we prepare for the Aurora Early Science Program.
Questions?

Support provided by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375.

Computing time provided by the NSF Extreme Science and Engineering Discovery Environment (XSEDE) program

Texas Advanced Computing Center resources used under Award Number(s) MCA08X004 - "Resilience and Scalability of the Uintah Software"

Thanks to TACC and those involved with the CCMSC and Uintah past and present

Uintah Download: http://www.uintah.utah.edu