S6212: Complex Application Proxy Implementation on the GPU

Through the Use of Kokkos and Legion

Geoff Womeldorff
GPU Technology Conference
2016/04/06
Abstract

The goal of this talk is to present research on the implementation, performance, and optimization of a complex application kernel, \texttt{dim3\_sweep} of SNAP, a neutral particle transport proxy, in CUDA through the use of the Kokkos programming model. Examples will be given of kernel performance measurements and optimization techniques enabled through the use of Kokkos. In addition, we will discuss efforts to couple the coarse-grained parallelism of SNAP, as implemented in Legion, a task-based programming model, and the fine-grained aspects, as implemented in Kokkos and CUDA, and how that coupling compares and contrasts to the native MPI+OpenMP of SNAP.
Outline

- Motivation (SNAP/Kokkos/Legion)
- Node-level Investigation (Kokkos)
- System-level Investigation (Legion)
- Future Work / Integrating Runtimes
SNAP, Kokkos, Legion
SNAP, a proxy application

- Simulates solving transport via $S_n$ (Discrete Ordinates) method
- Proxy for the PARTISN code project led by Randy Baker
- Base SNAP implementation
  - Developed by Joe Zerr and Randy Baker
  - Implemented in Fortran
  - Both a serial version and hand-tuned OpenMP
- Widely studied in the co-design community
Parallelism in SNAP

- **Outer loop**
  - Solve for fluxes over the energy domain (task parallel)
  - Coupling between energy groups

- **Inner Loop**
  - Sweep entire spatial mesh in each discrete angular direction
  - Spatial mesh is distributed (data parallel) using KBA wavefront method
  - Inner loop can be vectorized over angles (data parallel)
Kokkos Brief Overview

• Programming model in C++ that offers both abstractions for data management and parallel execution
• For data management, primary abstraction is a “View”
  • Can act like N-d data, or a subset thereof
• For parallel execution, constructs exist such as “parallel_for”, “parallel_reduce”, and “parallel_scan”
• Open source, available at: https://github.com/kokkos/kokkos
Friends of Kokkos

• S6449 - Sustainability and Performance through Kokkos: A Case Study with LAMMPS
  • Wednesday 4/6, 10:30am, Room 212AD, Christian Trott

• S6292 - Gradually Porting an In-Use Sparse Matrix Library to Use CUDA
  • Wednesday 4/6, 2:30pm, Room 212A, Mark Hoemmen

• S6257 - Kokkos Implementation of Albany: Towards Performance Portable Finite Element Code
  • Thursday 4/7, 2pm, Room 211A, Irina Demeshko

• S6145 - Kokkos Hierarchical Task-Data Parallelism for C++ HPC Applications
  • Thursday 4/7, 2:30pm, Room 211A, H. Carter Edwards
Legion: Programming Model and Runtime

- Data-aware, task-based model targeted at heterogeneous, distributed-memory machines
  - Systems have wide variability of communication latencies
- Goal: Latency tolerant and flexible
  - No magic. Task-based parallel programming is hard.
- http://legion.stanford.edu
Legion enables separation of concerns

**Tasks**
 execution model
For data management

**Regions**
data model
Describe data decomposition of computational domain

**Mapper**
Describes how tasks and regions should be mapped to the target architecture

```c
(int i) { rho(i) = … }
```

Mapper
Node Level Investigation: Kokkos
Node Level: Kokkos

• **Strategy for testing Node-Level ideas**
  • Start with (original) Fortran SNAP
  • Remove kernel, write standalone driver
  • Implement “Direct” Kokkos version of driver
  • Implement advanced Kokkos version of driver using hierarchical parallelism
  • Seek expert help for optimized version
  • Overcome GPU porting limitations
dim3_sweep
Node Level: Kokkos

- Fortran kernelized dim3_sweep, e.g.:

```fortran
pc = psi + psii(:,j,k)*mu*hi + psij(:,ic,k)*hj + psik(:,ic,j)*hk
IF ( vdelt /= zero ) pc = pc + vdelt*ptr_in(:,i,j,k)
```

Compute the numerator for the update formula
Node Level: Kokkos

• “Direct” Kokkos version
  • Tests scenario of “what if I give an application code and a threading model API to a domain scientist?”
  • Parallelism: Translate array operations into functors which are executed over array elements.
Node Level: Kokkos

- “Direct” Kokkos version

```c
// H - compute the numerator for the update formula

{ } psii_slice = subview( psii, ALL(), j, k );
psij_slice = subview( psij, ALL(), ic, k );
psik_slice = subview( psik, ALL(), ic, j );
parallel_for( nang, update_Aa_Bb_C1C2c_D1D2d_E1E2e< device_type, view_t_1d, view_t_1d_s, view_t_1d, view_t_1d_s, view_t_1d > ( pc, c0, psi, c1, psii_slice, mu, hi, psij_slice, hj, c1, psik_slice, hk, c1 ) );

if ( vdelta != c0 ) { // should be checking for a tolerance
    ptr_in_slice = subview( ptr_in, ALL(), i, j, k );
    parallel_for( nang, update<device_type,view_t_1d,view_t_1d_s>( pc, c1, ptr_in_slice, vdelta ) );
} 
```
Node Level: Kokkos

• Hierarchical parallelism Kokkos version
  • Tests scenario of “what if I give an application code and a threading model API to a computational scientist?”
  • Parallelism:
    • N groups -> oversubscribed to node cores (SM)
    • Diagonals (>M) -> statically scheduled to M hyperthreads (warps)
    • L length arrays mapped to vector operations (threads in a warp)
Node Level: Kokkos

- Hierarchical parallelism Kokkos version, e.g.:

  - \( N \)
    ```
    const team_policy_t policy( N, hyper_threads, vector_lanes );
    ```

  - \( M \)
    ```
    parallel_for( Kokkos::TeamThreadLoop( team_member, M ), [=]( const int& mm ) { // diag loop
    ```

  - \( L \)
    ```
    // H - compute the numerator for the update formula
    //-------------------------------------------------------------------------------
    parallel_for( Kokkos::ThreadVectorLoop( team_member, L ), [=]( const int& ll ) {
        pc(ll,mm) = psi(ll,mm) + psi_i(ll,j,k,g)*mu(ll)*hi + psi_j(ll,ic,k,g)*hj(ll) + psi_k(ll,ic,j,g)*hk(ll);
        if ( vdel(t) != c0 ) { // should be checking for a tolerance
            pc(ll,mm) = pc(ll,mm) + vdel(t) * ptr_in(ll,i,j,k,oct,g);
        }
    }); // ThreadVectorLoop H
    ```
Node Level: Kokkos

- In producing optimized (OPT) version
  - In porting a Fortran code, Kokkos::LayoutLeft
  - Making use of thread specific scratch space
    - Both performance and correctness
  - Thread specific copies of prior parallel level indices (GNU specific)
- In producing Kokkos::CUDA compatible version
  - Flatten scheduling data structures which rely on STL containers
  - Replace math functions with CUDA compatible ones
  - Reduce shared memory pressure (undo a few CPU optimizations)
Node Level: Kokkos

- **Results which test code versions:**
  - Fortran with and without OpenMP (F O and F NO)
  - Direct Kokkos Serial, OpenMP, Pthreads (DK S, DK O, DK T)
  - HP Kokkos Serial, OpenMP, Pthreads (KHP S, KHP O, KHP T)
  - Optimized Kokkos Serial, OpenMP, Pthreads (OPT S, OPT O, OPT T)

- **With problem configurations (A, B, C):**
  - Spatial (nx x ny x nz): 8x8x8, 16x16x16, 32x32x32
  - Groups: 40, 80, 120
  - Angles: (240, 360, 480) x 8 (3-D)
Node Level: Kokkos
Node Level: Kokkos

- Results which test code versions:
  - Fortran with and without OpenMP (F O and F NO)
  - Optimized Kokkos Serial, OpenMP (OPT S, OPT O)
  - Two Kokkos CUDA-UVM versions (OPT C, a baseline, and, OPT C2)

- With problem configurations (A, B, C):
  - Spatial (nx x ny x nz): 8x8x8, 8x8x8, 8x8x8
  - Groups: 32, 64, 128
  - Angles: (240, 360, 480) x 8 (3-D)
Node Level: Kokkos
Node Level: Kokkos

C C2, 20x over C F O
System-Level Investigation: Legion
System Level: Legion

• Separate tasks for each function call and node
• Multiple tasks for all groups for each node / octant combination
• Use data access patterns to control execution (runtime schedule vs human schedule)
• Ability to interleave convergence tests with computation
• Ability to schedule multiple octant sweeps simultaneously based on data requirements.
System Level: Legion

- Implemented an abstraction layer on top of Legion
- Verbosity significantly reduced
- Simplified partitioning interface
- “Dragon” abstraction layer:

```cpp
LRWrapper a;
a.create(ctx, runtime, {10, 10, 10}, 0, int())
Print3D check(10, 10, 10);
auto check_add = genIndexKernel(check, ctx, runtime, c);
runtime->execute_index_space(ctx, check_add);
```
System Level: Legion
System Level: Legion
Future Work / Integrating Runtimes
Future Work / Integrating Runtimes

• From Kokkos, new profiling interface, KokkosP, for kernel-level profiling statistics

• From Legion, there is an “OpenMP Processor” at the low-level (Realm), and work continues to make it accessible from the high-level runtime
  • Having the OpenMP Processor would allow for integration of the fine and coarse grained expressions of parallelism, here, to be integrated.

• In addition, we are also exploring options of various complexity related to integrating Kokkos in Fortran applications.
  • Use standard F <-> C language operability!
  • Decision points – who allocates memory?
    • Good news, proof of concepts for both F allocation, and C++ allocation.
Thank you for listening! :) womeld@lanl.gov