Towards Performance-Portable Applications through Kokkos:

A Case Study with LAMMPS

Christian Trott
Carter Edwards
Simon Hammond
The challenge – Node parallelism

<table>
<thead>
<tr>
<th>CPU 2001</th>
<th>CPU Now</th>
<th>MIC</th>
<th>APU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>256</td>
<td>~2,000</td>
<td>~5,000</td>
<td>~50,000</td>
</tr>
</tbody>
</table>

MPI-Only will not work anymore
! Domains get to small!
We need threading.
The challenge – Memory Access

Memory systems get more complex.

We need to use special hardware capabilities to achieve good performance.
What do we want?

- Single code base
- Support for all current (and future) hardware
- Flexible run configurations
  - MPI-Only
  - MPI + Threads
  - MPI + GPU
  - MPI + GPU + Threads
- Close to optimal performance (i.e. performance of a specialized code)
- Possibility for code specialisation
- Use vendor compilers
- Simple code

Eierlegende Wollmilchsau (egg-laying wool-milk-sow)
Kokkos as a solution

A programming model with two major components:

Data access abstraction
- Change data layout at compile time without changing access syntax
  => Optimal access pattern for each device
- Data padding and alignment is transparent
- Access traits for portable support of hardware specific load/store units

Parallel dispatch
- Express algorithms with \texttt{parallel_for}, \texttt{parallel_reduce} etc.
- Using functor concept
- Transparently mapped onto back-end languages (e.g. OpenMP, CUDA)

Goal: Separate science code from hardware details
What is Kokkos?

- C++ template library => almost everything is headers
- Developed as node level parallelism layer for Trilinos
  Trilinos is an Open-Source solver library, development led by Sandia
  www.trilinos.org
- Open-Source
- Standalone (no required dependencies)
- Lead developer: Carter Edwards, SNL
- Will be integrated into Trilinos during 2014

Pre print: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns
H. Carter Edwards, Christian R. Trott; submitted to JPDC
How does it work

Multidimensional Arrays:
View<\textbf{int**}[8][3], \textbf{LayoutRight}, \textbf{DeviceType}> \ a(“A”,N,M);
- 4D array NxMx8x3
- RowMajor data storage (i.e. 4\textsuperscript{th} index is stride-one access)
- allocated in memory space of \textbf{DeviceType}
- access: \texttt{double tmp = a(i,j,k,l)};

View<\textbf{const int**}[8][3], \textbf{LayoutRight}, \textbf{Device}, \textbf{RandomRead}> \ b = a;
- b is a const view of the same data as a
- \texttt{const + RandomRead} => use Texture fetches on GPUs

Parallel dispatch:
\begin{verbatim}
struct \textbf{AXPYFunctor} {
  typedef Kokkos::Cuda \textbf{device\_type};
  \textbf{ViewType} a,b;
  \textbf{AXPYFunctor} (\textbf{ViewType} A, \textbf{ViewType} B): a(A),b(B) {}
  void operator() (\texttt{const int \&i}) \textbf{const} { a(i) += b(i); }
}

parallel_for(n, \textbf{AXPYFunctor}(a,b));
\end{verbatim}
Performance Portability with Mantevo: MiniFE

Finite element code miniApp in Mantevo (*mantevo.org*)

*Heat conduction, Matrix assembly, CG solve*

Most variants of any miniApp in Mantevo

*more than 20 implementations in Mantevo repository; 8 in Mantevo 2.0 release*

Models aspects of Sandia’s mechanical engineering codes

**MiniFE CG-Solve time**

200x200x200 cells, 200 iterations

44-57s
Performance Portability with Mantevo: MiniMD

MiniMD Strongscaling
2M atoms; Standard Lennard Jones

Molecular Dynamics application simplified LAMMPS

Variants:
Reference (SNL)
OpenCL (SNL)
Kokkos (SNL)
Intel Xeon Phi intrinsics (Intel)
OpenACC (AMD)
Chapel (Cray)
Intel intrinsics (Warwick/Intel)
Qthreads (SNL)
LAMMPS Prototype

Exploration of Kokkos for use in LAMMPS (lammps.sandia.gov)
replace specialized packages => reduce code redundancy 3x
enable thread scalability throughout code base

Leverage algorithmic exploration from MiniMD
transferring thread-scalable algorithms

Get some simple simulations to run well
Implement framework (data management, device management)
Get all parts of a simulation run with Kokkos
First Goal: MiniMD run
LAMMPS Strongscaling

1M atoms; Standard Lennard Jones

- Xeon - Kokkos
- Xeon - OpenMP
- Xeon Phi - Kokkos
- Xeon Phi - OpenMP
- Kepler - Kokkos
- Kepler - Cuda
- Xeon Phi - MPI-only Kokkos

# Devices x Time in sec

# Devices

1  2  4  8  16  32
A side note: Performance on Xeon Phi

Per Gather: 2 Flops, 2 Loads   8 Flops, 1 Load   4 Flops, 0.5 Loads

Gather out of cache appears to be inefficient on Xeon Phi.
Features of Kokkos

Backends:
- Pthreads
- OpenMP
- CUDA (UVM support in the plans)

Parallel execution:
- parallel_for
- parallel_reduce (for arbitrary types)
- parallel_scan

2 level threading:
- teams of threads
- primitives (team_scan, team_barrier)
- shared memory

Data abstractions:
- 8-dimensional arrays
- View semantics
  - (no hidden data transfers)
- Compile-time data-layouts
- Access traits (random, stream* ...
- Data padding, alignment

Higher Level Libraries:
- Container classes
  - “std::vector”, dual-view, map
- Sparse linear algebra
  - CRS-Matrix, MatVec, ...
Conclusions

Kokkos: Research stable since September (keeping backward compatibility)

**Portable:** \textit{one code for CPUs, MIC, GPUs, ...}

**Performance:** \textit{>90\% of native implementations}

**Extensible:** \textit{use new back-ends without changing code}

Look for: Mantevo 2.0 release here at SC13 and at \texttt{mantevo.org}

=> \textit{get the MiniAPPs}

Kokkos included in Trilinos at \texttt{trilinos.org}

LAMMPS downloads at \texttt{lammps.sandia.gov}
Questions and further discussion: crtrott@sandia.gov