Using high-level programming tools aiming at performance portability
A short overview of some C++-based programming model(s) for performance portability

Pierre Kestener

1CEA Saclay, DSM, Maison de la Simulation

December 19, 2016
Main HPC architectures and trends: multicore, manycore, GPU, FPGA, Power8/9, NVLink, ...

What is performance portability?

A good software abstraction / programming model(s)?
  - library, framework, programming models?
  - Parallel programming patterns
  - Native language, directives, DSL?

As an example: a short overview of Kokkos: C++ library for performance portability
  Node-level parallelism, parallel pattern and data containers.

A real life example: code RamsesGPU (high-Mach number turbulent MHD) (partially) rewritten with Kokkos.
Introduction
Performance Portability
Kokkos introduction

From low-level native to high-level programming

Revisiting ways to develop software applications not only for accelerators, but multiple architectures


Find a good trade-off between ease of approach and good performance on multiple architectures.
1 Introduction

2 Performance Portability
   - Directives: OpenACC / OpenMP
   - (Active) libraries

3 Kokkos introduction
   - Kokkos basics
   - Case study: RamsesGPU on Pascal P100
   - Additionnal slides
Supercomputers architectures - TOP500

A Supercomputer is designed to be at bleeding edge of current technology. Leading technology paths (to exascale) using TOP500 ranks (Nov. 2016)

- **Multicore**: Maintain complex cores, and replicate (x86, SPARC) (#7, 10)
- **Manycore/Embedded**: Use many simpler, low power cores from embedded (IBM BlueGene) (#4, 9)
- **Manycore/Sunway** (# 1)
- **Manycore/Intel XeonPhi (1st and 2nd gen)**: Use many simpler cores with wide SIMD instructions, (# 2, 5, 6)
- **Massively Multithread/ GPU**: (# 3, 8)

**Sunway Taihulight** : programmed with MPI+OpenACC

Next year, we might have supercomputers build with ARMv8 CPU (From China, Japan, US,...), **DOE Coral machines** (NVidia GPU+IBM Power9, Intel KNL), ...
About DOE Coral next generation computing facility

- As part of **CORAL** (Next gen supercomputers): **Center for Accelerated Application Readiness**
- Provide **programming environments and tools** that enable **portability**

Two Tracks for Future Large Systems

**Many Core**
- 10’s of thousands of nodes with millions of cores
- Homogeneous cores
- Multiple levels of memory – on package, DDR, and non-volatile
- Unlike prior generations, future products are likely to be self-hosted

**Cori at NERSC**
- Self-hosted many-core system
- Intel/Cray
- 9300 single-socket nodes
- Intel® Xeon Phi™ Knights Landing (KNL)
- 16GB HBM, 64-128 GB DDR4
- Cray Aries Interconnect
- 28 PB Lustre file system @ 430 GB/s
- Target delivery date: 2016

**Aurora at ALCF**
- Self-hosted many-core system
- Intel/Cray
- Intel® Xeon Phi™ Knights Hill (KNH)
- Target delivery date: 2018

**Hybrid Multi-Core**
- CPU / GPU Hybrid systems
- Likely to have multiple CPUs and GPUs per node
- Small number of very fat nodes
- Expect data movement issues to be much easier than previous systems – coherent shared memory within a node
- Multiple levels of memory – on package, DDR, and non-volatile

**Summit at OLCF**
- Hybrid CPU/GPU system
- IBM/NVIDIA
- 3400 multi-socket nodes
- POWER9/Volta
- More than 512 GB coherent memory per node
- Mellanox EDR Interconnect
- Target delivery date: 2017
HPC architectures - Trends - Who’s driving?

- **Artificial Intelligence** applications: e.g. Japan (ABCI: a 130 single precision PetaFlops system in late 2017) for Companies (book time for a fee)
- **AI Bridging Cloud Infrastructure**: goal is 43 (FP32) GigaFlops/Watt
- **Energy efficiency**, e.g. Nvidia’s DGX-1 node server (1 Dual Xeon + 8 GPU P100) aimed at deep learning (~ 18 (FP64) GigaFlops/Watt).
- **Several new hardware solutions** to come next year and after: Intel Knights Mill (XeonPhi, 3rd gen), FPGA (?) for dedicated specific applications, ... ⇒ a good programing model!
Supercomputer node architecture

Multiples levels of hierarchy:
- Need to aggregate the computing power of several 10,000 nodes!
- Network efficiency: latency, bandwidth, topology
- Memory: on-chip (cache), out-of-chip (DRAM), IO (disk)
- Emerging hybrid programming model: MPI + X
- What is X? OpenMP, OpenACC, ..., Kokkos, RAJA, ...
- Even at node level MPI+X is required: e.g. KNL

Figure: Multi-core node summary, source: multicore tutorial (SC12) by G. Hager and G. Wellein
Summary

1. Introduction

2. Performance Portability
   - Directives: OpenACC / OpenMP
   - (Active) libraries

3. Kokkos introduction
   - Kokkos basics
   - Case study: RamsesGPU on Pascal P100
   - Additionnal slides
Performance portability

- Developing / maintaining a separate implementation of an application for each new hardware platform (Intel KNL, Nvidia GPU, ARMv8, ...) is less and less realistic
- Identical code will never perform optimally on all platforms
- Is it possible to have a single set of source codes that can be compiled for different hardware targets?
- Performance portability should be understood as a single source code base with
  - good performance on different architectures
  - a relatively small amount of effort required to tune app performance from one architecture to another.

source: http://www.nersc.gov/research-and-development/application-readiness-across-doe-labs

- High Developer / programmer productivity

1source: Matt Norman, WACCPD 2016
Performance portability issue: algorithmic patterns

- Is it possible to have a single set of source codes that can be compiled for different hardware targets?

  **Low-level native language:** OpenCL, CUDA, ...

  **Directive approach (code annotations)** for multicore/GPU, ...:
  - OpenMP 4.5 (Clang, GNU, PGI, ...)
  - OpenACC 2.5 (PGI, GNU, ...)

- **Other high-level library-based approaches** (mostly C++-based, à la TBB):
  - Some provide STL-like algorithmics patterns (e.g. Thrust is CUDA-based with backends for other archs, lift, arrayFire (numerical libraries, language wrappers, ...))
  - Kokkos, RAJA, ...

  **Cross-platform frameworks**
    - Chamm++: message-driven execution, task and data migration, distributed load-balancing, ...
    - hpx (heavy use of new C++ standards (11,14,17): std::future, std::launch::async, distributed parallelism, ...)

- **Use an embedded Domain Specific Language (DSL)**
  - Halide (for image processing)
  - NABLA (for HPC, developed at CEA, PDE mesh+particles apps)
Performance portability issue: memory management

- Right now **directives-based approaches** focus on algorithmic pattern, and less on memory layout (might change in the near future, at least in OpenMP).
- CPU and GPU for example require **different memory layout** for **maximum performance**:
  - vectorization on CPU
  - memory coalescence on GPU
- Some libraries like **Kokkos** promote **memory layout** as a **major concern**

### Motivation: Variety in Memory Hierarchies

<table>
<thead>
<tr>
<th>Platform</th>
<th>Memory Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Constant</td>
</tr>
<tr>
<td>Intel® Xeon® Processor</td>
<td>-</td>
</tr>
<tr>
<td>Intel® Xeon Phi™ Coprocessor</td>
<td>-</td>
</tr>
<tr>
<td>Intel® Xeon Phi™ Processor</td>
<td>-</td>
</tr>
<tr>
<td>Future System w/ 3D XPoint™ Technology</td>
<td>-</td>
</tr>
<tr>
<td>Intel® HD Graphics</td>
<td>-</td>
</tr>
<tr>
<td>Intel® Iris™ Graphics</td>
<td>-</td>
</tr>
<tr>
<td>Current Generation NVIDIA® GPU</td>
<td>✓</td>
</tr>
<tr>
<td>Future Generation NVIDIA® GPU</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.*
Programming with structured parallel patterns

- **pattern**: a basic structural entity of an algorithm
- **book** *Structured Parallel Programming: Patterns for Efficient Computation*

implementation: Intel TBB, OpenMP, OpenACC and many others

- **OpenMP/OpenAcc for GPU/XeonPhi**: pattern-based comparison: map, stencil, reduce, scan, fork-join, superscalar sequence, parallel update

reference:

*A Pattern-Based Comparison of OpenACC and OpenMP for Accelerator Computing*
Programming with structured parallel patterns

Parallel Patterns: Overview

reference: Structured Parallel Programming with Patterns, SC13 tutorial, by M. Hebenstreilt, J. reinders, A. Robison, M. McCool
Future of accelerator programming

- **passive libraries**: a collection of subroutines
- **active libraries**: take an active role in compilation (specialize algorithms, tune themselves for target architecture).

<table>
<thead>
<tr>
<th>Library</th>
<th>CUDA</th>
<th>OpenCL</th>
<th>Other</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thrust</td>
<td>X</td>
<td></td>
<td>OMP, TBB</td>
<td>header</td>
</tr>
<tr>
<td>Bolt</td>
<td></td>
<td>X</td>
<td>TBB, DX11</td>
<td>link</td>
</tr>
<tr>
<td>VexCL</td>
<td>X</td>
<td>X</td>
<td></td>
<td>header</td>
</tr>
<tr>
<td>Boost.Compute</td>
<td>X</td>
<td></td>
<td></td>
<td>header</td>
</tr>
<tr>
<td>C++ AMP</td>
<td>X</td>
<td></td>
<td>DX11</td>
<td>compiler</td>
</tr>
<tr>
<td>SyCL</td>
<td></td>
<td></td>
<td></td>
<td>compiler</td>
</tr>
<tr>
<td>ViennaCL</td>
<td>X</td>
<td>X</td>
<td>OMP</td>
<td>header</td>
</tr>
<tr>
<td>SkePU</td>
<td>X</td>
<td>X</td>
<td>OMP, seq</td>
<td>header</td>
</tr>
<tr>
<td>SkelCL</td>
<td></td>
<td></td>
<td></td>
<td>link</td>
</tr>
<tr>
<td>HPL</td>
<td>X</td>
<td></td>
<td></td>
<td>link</td>
</tr>
<tr>
<td>CLOGS</td>
<td></td>
<td></td>
<td></td>
<td>link</td>
</tr>
<tr>
<td>ArrayFire</td>
<td>X</td>
<td>X</td>
<td></td>
<td>link</td>
</tr>
<tr>
<td>CLOGS</td>
<td></td>
<td></td>
<td></td>
<td>link</td>
</tr>
<tr>
<td>hemi</td>
<td>X</td>
<td></td>
<td></td>
<td>header</td>
</tr>
<tr>
<td>MTL4</td>
<td>X</td>
<td></td>
<td></td>
<td>header</td>
</tr>
<tr>
<td>Kokkos</td>
<td>X</td>
<td></td>
<td>OMP, PTH</td>
<td>link</td>
</tr>
<tr>
<td>Aura</td>
<td>X</td>
<td>X</td>
<td></td>
<td>header</td>
</tr>
</tbody>
</table>

reference:
The Future of Accelerator Programming in C++, S. Schaetz, May 2014
How to improve **space (memory) locality** in algorithm implementations?

*High Performance Parallelism Pearls*, **Morton order to improve memory locality**, by Kerry Evans (INTEL), chap. 28

**matrix transpose, dense matrix multiplication** on Xeon, KNC

Same feature used in some Adaptive Mesh Refinement PDE solver.
Summary

1. Introduction

2. Performance Portability
   - Directives: OpenACC / OpenMP
   - (Active) libraries

3. Kokkos introduction
   - Kokkos basics
   - Case study: RamsesGPU on Pascal P100
   - Additional slides
Kokkos: a programming model for performance portability

- **Kokkos** is a C++ library with parallel algorithmic patterns AND data containers for node-level parallelism.
- Implementation relies heavily on meta-programming to derive native low-level code (OpenMP, Pthreads, CUDA, ...) and adapt data structure memory layout at compile-time.
- Core developers at SANDIA NL (H.C. Edwards, C. Trott)
Kokkos: a programming model for performance portability

- **Open source**, [https://github.com/kokkos/kokkos](https://github.com/kokkos/kokkos)
- Primarily developed as a base building layer for **generic high-performance parallel linear algebra** in [Trilinos](https://github.com/kokkos/kokkos)
- Also used in molecular dynamics code, e.g. [LAMMPS](https://github.com/kokkos/kokkos)
- Goal: **ISO/C++ 2020 Standard** subsumes Kokkos abstractions
Kokkos: a programming model for performance portability

Kokkos abstract concepts

- **Execution patterns** (what):
  parallel_for, parallel_reduce, ...

- **Execution policy** (how):
  range iterations, teams of threads, ...

- **Execution space** (where):
  OpenMP, PThreads, CUDA, numa, ...

- **Memory space**: **data containers** with architecture adapted memory layout
  Kokkos::View, Kokkos::DualView, Kokkos::UnorderedMap, ...

- **Memory layout**: (important for vectorization, memory coalescence, ...)
  row-major, column-major, AoS, SoA, ...
  \( data(i, j, k) \) architecture aware.

Baseline serial version

```cpp
for ( int i = 0 ; i < nrow ; ++i ) {
    for ( int j = irow[i] ; j < irow[i+1] ; ++j )
        y[i] += A[j] * x[ jcol[j] ];
}
```

Simple Kokkos parallel version

```cpp
parallel_for( nrow , KOKKOS_LAMBDA( int i ) {
    for ( int j = irow[i] ; j < irow[i+1] ; ++j )
        y[i] += A[j] * x[ jcol[j] ];
});
```

- Execution pattern: **parallel for**
- Execution policy: **range iteration**
- Execution space: default (defined at compiled time)

**Work to do can be**

- A *Lambda anonymous function*, convenient for short loop bodies
- A *C++ class function*, maximum flexibility
MiniMD used to bench thread-scalable algorithm before integrating them in LAMMPS (2014)

**MiniMD Performance**
Lennard Jones force model using atom neighbor list

- Solve Newton’s equations for $N$ particles
- Simple Lennard Jones force model: $F_i = \sum_{j, r_{ij} < r_{cut}} 6\epsilon \left[ \left( \frac{\sigma}{r_{ij}} \right)^7 - 2 \left( \frac{\sigma}{r_{ij}} \right)^{13} \right]$
- Use atom neighbor list to avoid $N^2$ computations
  ```c
  pos_i = pos(i);
  for( jj = 0; jj < num_neighbors(i); jj++) {
    j = neighbors(i,jj);
    r_ij = pos_i - pos(j); //random read 3 floats
    if ( |r_ij| < r_cut )
      f_i += 6*epsilon*( (s/r_ij)^7 - 2*(s/r_ij)^13 )
  }
  f(i) = f_i;
  ```
- Moderately compute bound computational kernel
- On average 77 neighbors with 55 inside of the cutoff radius

**source:** [http://lammps.sandia.gov/bench.html](http://lammps.sandia.gov/bench.html)
**LAMMPS** Accelerator benchmarks for CPU, GPU, KNL Oct 2016
Future of accelerator programming: Kokkos among other

MiniMD used to bench thread-scalable algorithm before integrating them in LAMMPS (2014)


LAMMPS Accelerator benchmarks for CPU, GPU, KNL Oct 2016
RamsesGpu with Kokkos

- **RamsesGPU** is a C++/Cuda code for Compressible MHD application, developed since 2009-2010 (ran on Titane Tesla S1070, sm_13).

- Several numerical schemes variants, ~ 70 CUDA kernels.

- **MagnetoRotational Turbulence in accretion disks;** 576-nodes jobs (K20) on NCSA BlueWaters, colleagues from CEA and Univ. Illinois: Ryan, Gammie, Fromang, P.K.

- Recently, rewrite core kernel application using **Kokkos**.
**RamsesGpu with Kokkos**

### Directionally splitted Hydro - 3D performances

#### Number of $10^6$ cell updates / second versus domain size

<table>
<thead>
<tr>
<th>taille</th>
<th>M2090 SP/fast</th>
<th>SP</th>
<th>DP</th>
<th>K20 SP/fast</th>
<th>SP</th>
<th>DP</th>
</tr>
</thead>
<tbody>
<tr>
<td>32x32x32</td>
<td>18.8</td>
<td>26.6 (+41%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64x64x64</td>
<td>83.4</td>
<td>95.2 (+14%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96x96x96</td>
<td>100.7</td>
<td>175.1 (+73%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128x128x128</td>
<td>114.7</td>
<td>32.1</td>
<td>9.2</td>
<td>178.7 (+55%)</td>
<td>72.4</td>
<td>34.9</td>
</tr>
<tr>
<td>192x192x192</td>
<td>133.0</td>
<td></td>
<td></td>
<td>226.7 (+70%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>225x225x225</td>
<td>137.2</td>
<td>39.5</td>
<td>11.1</td>
<td>210.5 (+53%)</td>
<td>97.4</td>
<td>40.6</td>
</tr>
</tbody>
</table>

- **Architecture Kepler K20 versus Fermi M2090**
- Rebuild application with CUDA 5.5 toolchain for architecture 3.5 and tune flags
- Tune max register count for Kepler, and care about *read-only data cache* => **No more register spilling, DP perf is optimal!**
- in DP: **Kepler is ~ 3.5× faster** than M2090

- With Kokkos, from **Kepler K80 ⇒ Pascal P100, performance scaling almost perfect** (~ ×3.0); no tuning required; 360 Mcell-update/s
- With hand-written CUDA, tuning is required to recover this perf scaling
- Number of lines of codes divided by 2-3
- Get for free an efficient OpenMP implementation
Want to know more?

- A free 3-days training on **performance portability**:
  
  Performance portability for GPU applications using high-level programming approaches
  
  [https://events.prace-ri.eu/event/568/](https://events.prace-ri.eu/event/568/)

- Themes: OpenACC / Kokkos
- Dates: **16-18 January 2017**
- Location: **IDRIS** Computing center, Orsay
- Hardware platform: Ouessant (IBM Power8 + Nvidia P100)
Additionnal links

- [https://asc.llnl.gov/CORAL-benchmarks/](https://asc.llnl.gov/CORAL-benchmarks/): CORAL Benchmark codes
An interesting research compiler multi-platform

source: [http://ft.ornl.gov/research/openarc](http://ft.ornl.gov/research/openarc)