PERFORMANCE PORTABILITY

TIM WILLIAMS
Deputy Director of Science
ALCF

Argonne
NATIONAL LABORATORY
## TWO ARCHITECTURAL FOR PRE-EXASCALE SYSTEMS

<table>
<thead>
<tr>
<th>MANY CORE</th>
<th>HYBRID MULTI-CORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Tens of thousands of nodes</td>
<td>▪ Few thousand nodes</td>
</tr>
<tr>
<td>▪ Millions of homogeneous cores</td>
<td>▪ CPU + multiple GPUs</td>
</tr>
<tr>
<td>▪ Multiple memory levels:</td>
<td>▪ Coherent shared memory on node</td>
</tr>
<tr>
<td>– HBM (on-package), DDR, NVM</td>
<td>▪ Multiple memory levels</td>
</tr>
<tr>
<td></td>
<td>– On-package, DDR, NVM</td>
</tr>
</tbody>
</table>
## TWO ARCHITECTURAL FOR PRE-EXASCALE SYSTEMS

### MANY CORE
- Tens of thousands of nodes
- Millions of homogeneous cores
- Multiple memory levels: HBM (on-package), DDR, NVM
  
**DOE:**

- Maintain architectural diversity

### HYBRID MULTI-CORE
- Few thousand nodes
- CPU + multiple GPUs
- Coherent shared memory on node
- Multiple memory levels
  - On-package, DDR, NVM
## TWO ARCHITECTURAL FOR PRE-EXASCALE SYSTEMS

<table>
<thead>
<tr>
<th>MANY CORE</th>
<th>HYBRID MULTI-CORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tens of thousands of nodes</td>
<td>Few thousand nodes</td>
</tr>
<tr>
<td>Millions of homogeneous cores</td>
<td>CPU + multiple GPUs</td>
</tr>
<tr>
<td>Multiple memory levels:</td>
<td>Coherent shared memory on node</td>
</tr>
<tr>
<td>– HBM (on-package), DDR, NVM</td>
<td>multiple memory levels</td>
</tr>
<tr>
<td>DOI:</td>
<td>On-package, DDR, NVM</td>
</tr>
<tr>
<td>Maintain architectural diversity</td>
<td></td>
</tr>
</tbody>
</table>

**DOE:**
- Maintain architectural diversity

...and make applications portable
<table>
<thead>
<tr>
<th>MANY CORE</th>
<th>HYBRID MULTI-CORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Tens of thousands of nodes</td>
<td>▪ Few thousand nodes</td>
</tr>
<tr>
<td>▪ Millions of homogeneous cores</td>
<td>▪ CPU + multiple GPUs</td>
</tr>
<tr>
<td>▪ Multiple memory levels: – HBM (on-package), DDR, NVM</td>
<td>▪ Coherent shared memory on node</td>
</tr>
</tbody>
</table>

**DOE:**
Maintain architectural diversity

...and make applications portable

...with high performance!
ALCF, OLCF, NERSC, LLNL, SNL, LANL, LBN, JLAB, IBM, INTEL, CRAY, NVIDIA

- Application Experiences
- Performance Portable Abstractions
- Managing Memory Hierarchy
- Experience with OpenMP
- Tools
- DSLs
- ...

DOE Centers of Excellence Performance Portability Meeting
April 19–21, 2016
Glendale, Arizona

https://asc.llnl.gov/DOE-COE-Mtg-2016/
PERFORMANCE PORTABILITY: WHAT IS IT?
PERFORMANCE PORTABILITY: WHAT IS IT?

- 2014 FASTMath meeting:
  - *Same piece of code (from the user perspective) runs on different architectures with ‘good’ performance*
  - *A relatively small amount of effort is needed to make a change to get good performance within advertised (algorithmic or performance) tolerances across both current and future architectures*

- Kokkos
  - *The amount of user code which can be compiled for diverse manycore architectures and obtain the same, or nearly the same, performance as an architecture specialized version of that code.*

- A Metric for Performance Portability (Pennycook et al.)
  - *A measurement of an application’s performance efficiency for a given problem that can be executed correctly on all platforms in a given set*
PERFORMANCE PORTABILITY STRATEGIES

1. Conditional compilation
2. Directives
3. Libraries
4. Frameworks
5. General-purpose high-level programming languages
6. DSLs (Domain specific languages)
7. Common HPC development environment
8. Co-design hardware and software using scientific apps
CONDITATIONAL COMPILATION: XGC

Gyrokinetic particle-in-cell tokamak plasma simulation

- Particles (ions & electrons) move continuously through toroidal spatial grid
- Accumulate charge and current densities from particles to grid points
- Solve Maxwell’s equations on the grid
- Gather electromagnetic field values from grid to particle positions
- Move particles via Lorentz force of EM fields on charged particle

http://epsi.pppl.gov/
CONJONCTIONAL COMPILATION: XGC

**Gyrokinetic particle-in-cell tokamak plasma simulation**

- Particles (ions & electrons) move continuously through toroidal spatial grid
- Accumulate charge and current densities from particles to grid points
- Solve Maxwell’s equations on the grid
- Gather electromagnetic field values from grid to particle positions
- Move particles via Lorentz force of EM fields on charged particle
CONDITIONAL COMPILATION: XGC

Gyrokinetic particle-in-cell tokamak plasma simulation

- Particles (ions & electrons) move continuously through toroidal spatial grid
- Accumulate charge and current densities from particles to grid points
- Solve Maxwell’s equations on the grid
- Gather electromagnetic field values from grid to particle positions
- Move particles via Lorentz force of EM fields on charged particle
### CONDITIONAL COMPILATION: XGC

**Gyrokinetic particle-in-cell tokamak plasma simulation**

- Particles (ions & electrons) move continuously through toroidal spatial grid
- Accumulate charge and current densities from particles to grid points
- Solve Maxwell’s equations on the grid
- Gather electromagnetic field values from grid to particle positions
- Move particles via Lorentz force of EM fields on charged particle

#### MIRA, THETA (MANY CORE)

- MPI + OpenMP

#### TITAN (CPU + GPU)

- Accumulation + field solve: MPI + OpenMP on CPU
- Push: CUDA Fortran on GPU
CONDITIONAL COMPILATION: HACC

Hardware Accelerated Cosmology Code

• Underlying calculation is the gravitational force.
• Brute force method $O(N_p^2)$. Number of particles $N_p \sim 10^{12}$.

1) Long-range component.
   o Particle mesh (PM) method
     – Global FFT across all MPI ranks.

2) Short-range component.
   o Specifics depend on architecture:
     – GPU: Direct pairwise force
     – CPU: Tree computation
   o Bulk of computational time spent here.
   o Requires only rank-local shared memory.

---

DIRECTIVES

- Can MPI + OpenMP 4+ give us performance portability?

double x[128], y[128];
#pragma omp target data map(to:x[0:64])
    map(tofrom:y[0;64])
{
    #pragma omp target
    {
        // y computed on device
    }
}

double x[128], y[128];
#pragma omp for simd aligned(x, y: 32)
for (int i=0; i<128; i++) {
    // thread’s iterates \rightarrow SIMD Lanes
}

#pragma omp target data map(to:x) {
    #pragma omp target map(tofrom:y)
    #pragma omp teams
    #pragma omp distribute
    #pragma omp parallel for
    for (int i = 0; i < n; ++i) {
        y[i] = a*x[i] + y[i];
    }
}
DIRECTIVES

- OpenMP 5.x – proposed memory management support (TR 5)
  - Memory space: value of memory traits define characteristics of a space:
    - distance = near, far
    - bandwidth = highest, lowest
    - latency = highest, lowest
    - location = core, socket, device
    - optimized = bandwidth, latency, capacity, none

Lots of ongoing work trying MPI+OpenMP 4+ on multiple architectures. Comparisons with OpenACC 2+, other non-directives approaches.

## LIBRARIES / FRAMEWORKS

<table>
<thead>
<tr>
<th>SOLVERS/MATHEMATICAL</th>
<th>PARALLEL ALGORITHMS/STRUCTURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ PETSc🔗</td>
<td>▪ Thrust🔗</td>
</tr>
<tr>
<td>▪ Trilinos🔗</td>
<td>– Containers, iterators</td>
</tr>
<tr>
<td>▪ ESSL🔗</td>
<td>– Algorithms</td>
</tr>
<tr>
<td>▪ ScaLAPACK🔗</td>
<td>• Transforms w/functors</td>
</tr>
<tr>
<td>▪ MKL🔗</td>
<td>• Parallel prefix-sums</td>
</tr>
<tr>
<td>▪ MAGMA🔗</td>
<td>▪ ADLB🔗</td>
</tr>
<tr>
<td>▪ FFTW🔗</td>
<td>▪ Global Arrays🔗</td>
</tr>
<tr>
<td>▪ HYPRE🔗</td>
<td>▪ Legion - data centric🔗</td>
</tr>
<tr>
<td>▪ BoxLib🔗</td>
<td>▪ HPX🔗</td>
</tr>
<tr>
<td></td>
<td>▪ Kokkos🔗</td>
</tr>
<tr>
<td></td>
<td>▪ RAJA🔗</td>
</tr>
</tbody>
</table>
WHY ALL THIS C++?

- Templates
  - `template <typename Policy> class A;`
    - `Policy` specifies _how_ class A implements functions
    - Write tuned specializations for different Policy types
    - Functions are non-virtual and can be inlined for speed
  - Operator overloading (+, -, ...) 
  - Parenthesis “()” overloading
  - Compile-time template metaprogramming
  - Lambda expressions
  - Traits classes – useful properties of a type wrapped in a class
    - C++11: include `<type_traits>`
      - `is_array`, `is_class`, `is_pointer`, `alignment_of`, ...
RAJA

- Traversals & execution policies (loop scheduling, execution)
- IndexSets (iteration space partitioning, ordering, dependencies, placement, etc.)
- Reduction types (programming model portability)

forall<exec_policy>(iset, [&] (Index_type i) {
    y[i] += a * x[i];
});

How loop iterations are scheduled to hardware: OpenMP, sequential, CUDA
double* x; double* y; double* y;
double a, tsum = 0, tmin = MYMAX;
...
for (int i = begin; i < end; ++i) {
y[i] += a * x[i];
tsum += y[i];
if (y[i] < tmin) tmin = y[i];
}

double* x; double* y; double a;
RAJA::SumReduction<reduce_policy, double> tsum(0);
RAJA::MinReduction<reduce_policy, double> tmin(MYMAX);
...
RAJA::forall<exec_policy>(IndexSet, [=] (int i) {
    y[i] += a * x[i];
tsum += y[i];
tmin.min(y[i]);
});
KOKKOS

- Data Structures
  - View – Multidimensional Array
    - View<double**, MemoryTraits<Atomic>> > a_atomic = a;
  - Kokkos Containers: DualView<type,device>, Vector<t,d>, UnorderedMap<Key,Value,Device>

- Parallel Execution
  - parallel_for, parallel_reduce, parallel_scan

- KokkosKernels (under dev COEPP)
  - BLAS, Sparse, Graph, Tensor kernels
Performance Portability through Abstraction

Separating of Concerns for Future Systems...

Kokkos

Data Structures

Memory Spaces (“Where”)
- Multiple-Levels
- Logical Space (think UVM vs explicit)

Memory Layouts (“How”)
- Architecture dependent index-maps
- Also needed for subviews

Memory Traits
- Access Intent: Stream, Random, …
- Access Behavior: Atomic
- Enables special load paths: i.e. texture

Parallel Execution

Execution Spaces (“Where”)
- N-Level
- Support Heterogeneous Execution

Execution Patterns (“How”)
- parallel_for/reduce/scan, task spawn
- Enable nesting

Execution Policies
- Range, Team, Task-Dag
- Dynamic / Static Scheduling
- Support non-persistent scratch-pads

GENERAL-PURPOSE HIGH-LEVEL LANGUAGES

- Charm++
- UPC
- X10

- Coarray Fortran (Fortran 2008)
- HPF
- Chapel
DSLs (DOMAIN SPECIFIC LANGUAGES)

- “Natural” language of the scientific/mathematical domain
- Compact, unambiguous

- Example: NMODL DSL
  - Domain: computational neuroscience

```plaintext
NEURON {
  SUFFIX leak
  NONSPECIFIC_CURRENT I
  RANGE i, e, g
}

PARAMETER {
  g = 0.001  (siemens/cm2)  < 0, 1e9 >
  e = -65  (millivolt)
}

ASSIGNED {
  i  (milliamp/cm2)
  v  (millivolt)
}
```

DSL FOR PERFORMANCE PORTABILITY

- CoreNEURON brain tissue simulation

```
NMODL Description

NEURON {
  SUFFIX leak
  PARAMETER {
    g = 0.001  (siemens/cm²) < 0, 1e9 >
    e = -65  (millivolt)
  }
  ASSIGNED {
    i (milliamp/cm²)
    v (millivolt)
  }
}

Code generator

Modeling Kernels
C/C++/CUDA/
OpenMP/OpenCL...

Compiler

Framework
(C/C++)

Executable

- KNL
- GPU
- FPGA
- Neuromorphic

http://www.alcf.anl.gov/projects/large-scale-simulation-brain-tissue-blue-brain-project-epfl
FREE ADVICE

- Establish performance targets/bounds
- Detailed characterization of performance
- Extract kernels/mini-apps
- Encapsulate portability challenges
  - Modularity
  - High-level abstractions
  - Libraries
  - …good software engineering practices

- Avoid architecture specific models:
  - Intel Thread Building Blocks
  - NVIDIA CUDA
  - If necessary, encapsulate

- Good coding practices
  - Parameters for thread counts and thread placements
  - Data structures flexibly allocatable to different memory spaces
  - Task level flexibility so work can be allocated on different compute elements (GPU & CPU)
Another one in 2017

Concrete portability studies:

- ALCF
  - NekBone

- NERSC
  - BoxLib MG solvers

- OLCF
  - DSL-based library for MD

Developing best practices guide