The RAJA Encapsulation Model for Architecture Portability

DOE Centers of Excellence Performance Portability Meeting, Glendale AZ

April 19, 2016

Rich Hornung, LLNL
Jeff Keasler, LLNL
Overarching RAJA goal: enable portability with *small disruption* to application programming style

**Balance Performance:**

- Augment compiler’s ability to optimize C++ code
  - Allow work-arounds when performance is not what’s expected
- Simplify expression of various forms of fine-grained (on-node) parallelism

**And Productivity:**

- Single-source kernels
  - Do not bind an application to a particular PM technology
  - Best choice for a given platform or algorithm may not be clear
- Clear separation of responsibilities
  - **RAJA:** Encapsulate hardware and PM details and execute loops
  - **App:** Select iteration patterns and execution policies with RAJA API

**Ideal:** Developers add parallelism to code using RAJA encapsulation layer – preserve development dynamics and advantages of MPI heritage
RAJA is a low-risk way to realize latent fine-grain parallelism in existing applications

- Loop iteration and loop body are decoupled (body mostly unchanged, often untouched)
- A loop iteration is a “task” – reorder, schedule, aggregate, manage dependencies, etc.
- Explore implementation alternatives (tuning) without disrupting application source code

### C-style for-loop

```c
double* x; 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];
}
```

### RAJA-style loop

```c
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] );
} );
```

### RAJA encapsulation features
- Traversals & execution policies (loop scheduling, execution, PM backends)
- IndexSets (iteration space partition, ordering, dependencies, data placement, etc.)
- Reduction types (programming model portability)
RAJA enables systematic architecture portability and tuning for large production codes

- Design based on loop / mesh traversal patterns in LLNL ASC codes
  - A loop is the main conceptual abstraction in RAJA
  - A typical LLNL multi-physics code has $O(10^5)$ loops, but $O(10)$ loop patterns
  - App teams typically wrap RAJA in a “mini-DSL” that matches their style

- RAJA can be used selectively and adopted incrementally
  - Mapping loops to RAJA “execution policies” is key to performance
  - Important considerations: data motion, compute intensity, branch intensity, available parallelism, etc.

- Typical RAJA integration process:
  1) Achieve basic portability
     - Make code thread-safe where needed
     - Convert loops to RAJA-style
  2) Tune exec policies
     - Identify loop classes
     - Map classes to architecture features using RAJA traversal templates and execution policies
  3) Explore tuning options
     - Change iteration pattern (IndexSets)?
     - Redesign algorithm or data layout?
     - Code-specific RAJA constructs?
RAJA core abstractions can combined with application-specific implementations

Original app code

// Kernel 1
for (int i = begin; i < end; ++i) {
    Loop body 1 (stride-1)
}

// Kernel 2
for (int i = 0; i < len; ++i) {
    Loop body 2 (indirection)
}

// Kernel 3
...

// Kernel 4
...

“RAJA-fied” app code

// Kernel 1 : “stream” → low FLOP/bandwidth
RAJA::forall<stream> ( begin, end, [=] (int i) {
    Loop body 1
});

// Kernel 2 : “work” → high FLOP/bandwidth
RAJA::forall<work> (iset, [=] (int i) {
    Loop body 2 (iset = index “ranges” & “lists”)
});

Execution policies (app defined)
Arch A: stream = seq, work = omp
Arch B: stream = omp, work = gpu

RAJA supports relatively simple parameterization of most loops. Others may need customization or more severe disruption for desired performance.
RAJA IndexSets simplify thread-safe refactoring of code with data races

- **Example code:** accumulate element volumes to mesh nodes
- Iterations are colored into sets of independent work (IndexSet Segments)
  - Iterate over segments sequentially
  - Execute each segment in parallel
- Without reordering, requires either:
  - Contention-heavy fine-grained sync ops (atomics / critical sections)
  - Temporary arrays for accumulating sums
- RAJA reordering allows use of coarse-grained synchronization
  - Less memory contention
  - Code remains as domain expert wrote it
RAJA IndexSets and traversals also enable developers to define & schedule dependencies

```c
#pragma omp parallel for schedule(static,1)
for (int i = 0; i < is->num_seg; ++i) {
    IndexSetSegInfo* seg_info = iset.getSegmentInfo(i);
    DepGraphNode* task = seg_info->getDepGraphNode();
    while (task->semaphoreValue() != 0) {
        sched_yield();
    }

    execute<SEG_EXEC_POLICY>(seg_info, loop_body);

    if (task->semaphoreReloadValue() != 0) {
        task->semaphoreValue() = task->semaphoreReloadValue();
    }

    if (task->numDepTasks() != 0) {
        for (int j = 0; j < task->numDepTasks; ++j) {
            int seg = task->depTaskNum(j);
            DepGraphNode* dep = iset.getSegmentInfo(seg)->getDepGraphNode();
            __sync_fetch_and_sub(&(dep->semaphoreValue()), 1);
        }
    }
}
```

Segment scheduling control logic like this is hidden in a RAJA traversal template.
RAJA version of LULESH v1.0 hydro proxy app is an interesting case study

Base OpenMP implementation is memory intensive

RAJA OpenMP uses coarse-grained synchronization

Add pools for temp arrays. CPU-GPU switch in header file.
RAJA K80 GPU performance for LULESH v2.0 has improved markedly since FY15 ASC milestone

Compiled with nvcc 7.5 using compute_37
We are developing extensions and complements to RAJA that address other application needs

- Abstractions to automatically move data between DRAM and HBM or device memory – less code “clutter” than OpenMP 4 or CUDA:
  - ManagedArray objects know what data to copy
  - ResourceManager object knows whether to copy data
  - RAJA provides context to know where to copy data

- forallIN extensions for nested loops:
  - Supports loop interchange and data layout changes
  - Provides loop index types to ensure code is correct

LLNL developers are assessing these concepts in production applications today.
RAJA enables a single source code base to run with multiple forms of parallelism

- CPU-GPU portability achieved with existing PMs and standard C++11
  - Often does not require much code restructuring (incl. reductions) or multiple versions
- RAJA can help make tuning more systematic
  - Focus on loop patterns, not individual loops
- IndexSets provide a lot of flexibility
  - Code specializations generated (and optimized) at compile-time. Execution paths through specializations selected at runtime.
  - Dependencies between iteration subsets (Segments) can be defined to ensure correctness
- Multiple LLNL projects contribute to RAJA: Ares, ALE3D, Ardra, AAPS, etc.
  - See David Beckingsale policy tuning talk tomorrow
- Three LLNL production apps are integrating RAJA to prepare for Trinity and Sierra: Ares, ALE3D, Ardra
- Other codes are exploring RAJA: NIF VBL, hypre, VisIt, hydra, MARBL
  - See Matt Martineau proxy-app talk tomorrow

An open-source RAJA release with example codes is imminent and will be available at http://github.com/LLNL/RAJA -- collaborators welcome!
Software engineering alone is not enough – we need to work closely with vendors too

- Compiler vendors & PM developers have been improving support for C++ based encapsulation
  - IBM, NVIDIA, Intel, GNU, AMD (Sierra CoE, DesignForward, FastForward, van trips...)
  - OpenMP 4.0 → OpenMP 4.5 & beyond (See Arpith Jacob talk next)

- Tool support for C++ templates: HPCToolkit, Intel

- We (DOE HPC community) must tell vendors what we need

- Good solutions involve negotiations (our needs, vendor priorities, language standards, vendor extensions, etc.)

A relatively small investment in compilers and runtimes (and vigilance) – compared to HW costs – can have a huge, positive, lasting impact on our codes. Issues we identify and work to resolve also benefit the broader HPC community.