Abstract

OpenACC is a directive-based language extension for Fortran, C, and C++, that facilitates the simple and effective use of accelerators (e.g., GPUs) without sacrificing portability for non-accelerator systems. The Oak Ridge Leadership Computing Facility (OLCF) has made a strategic investment in OpenACC for the Titan system and applications are starting to use it. However OpenACC is a very young specification. Application scientists at ORNL have already identified a number of extensions to OpenACC that would significantly enhance its expressiveness and usability in their applications. Looking further forward, towards exascale computers, we see trends towards node-level environments with heterogeneous compute resources, and more complex memory environments. Extending OpenACC to support such environments, with task-based execution, the ability to control placement of data in memory, and interoperability with other prominent node-level programming models will smooth the path for today’s applications to make the transition to new exascale architectures, as well as preparing them to for the jump to next-generation programming models and languages. We will develop key extensions to OpenACC to support current ORNL applications and demonstrate their efficacy with prototype implementations.

Challenges of Programming Accelerators

- Heterogeneous systems multiply the challenge of programming
  - Multiple processor architectures
  - Multiple memory systems
- Programmer must decide best placement of work
- Applications need to run on a wide range of systems
  - And evolve toward exascale-friendly programming approaches
- Programming for accelerators is young and immature

OpenACC in a Nutshell

```c
#pragma acc kernels loop
for( i = 0; i < nrows; ++i ){
  double val = 0.0;
  int nstart = rowindex[i];
  int nend = rowindex[i+1];
  #pragma acc loop vector reduction(+:val)
  for( n = nstart; n < nend; ++n )
    val += m[n] * v[colndx[n]];
  r[i] = val;
}
```

Supporting Tasks

- Asynchronous task-based execution models central to exascale programming
  - Expose parallelism
  - Avoid synchronization
  - Process efficiently
- Facilitates programming of heterogeneous nodes
  - Use all compute resources
  - Load balancing and work sharing
- Leverage existing “streams” concept
  - Add nested parallelism, dependencies

Deep Copy and C++

Support for Complex Data Structures in OpenACC

- Complex, non-contiguous data structures are a challenge
  - What really needs to be present on the accelerator?
  - What needs to be updated on the host?
  - Want to minimize data movement
  - Particularly in C++
    - OD features, like encapsulation
    - Multiple fields, pointers, inheritance aliases, etc.
    - Usage changes in different contexts
  - Proposed solution
    - Use policies to describe:
      - Different types of direction/visibility for the different members
      - Default behavior for data members
    - Apply policies to data regions, updates

Challenges: Tasks on Multiple Accelerators

- Hybrid CPU/GPU tasks on multiple accelerators
  - Multiple Wave-fronts at once.
  - Multiple energy levels with different workloads.
  - Can tasks be scheduled on both CPU and GPUs?
  - Decomposition study task and fine grain parallelism
  - Different architectures may need different task granularities
    (Xeon Phi, vs Multiple GPUs) on node
  - Can we create multiple levels of tasks and
    to adapt granularity?
  - Need a queuing model to balance work across multiple accelerators
  - Explicit vs. Implicit queuing model.

Deep Copy


Challenges: Supporting Tasks

- Dealing with the “this” pointer in C++