Openacc 2.0++ and OpenMP 4.0++

Directive based programing on “accelerators” today and into the future

James Beyer, Ph.D

What is an “accelerator”?

- **MIC/PHI**
  - Intel says co-processor
  - Why?
- **GPU**
  - Nvidia say accelerator
  - Why?
- **NUMA node**
  - Did you think of this one?
- **DSP**
  - This is why TI co-chairs the OpenMP subcommittee!
- **The main processor**
- **Network controller**
- **Storage controller**
- **…**

Background -- OpenMP

- **FORTRAN version 1.0 - (October 1997)**
- **Accelerator additions**
  - Subcommittee formed Aug 2009
  - Initial proposal submitted Dec 2009
- **Cray OpenMP for Accelerators nears release**
- **Fall 2010 several members form OpenACC working group**
  - No mechanism with OpenMP for OpenACC to be released quickly
- **OpenMP TR1 - Technical Report on Directives for Attached Accelerators (November 2012)**
- **OpenMP 4.0 (July 2013)**

Background -- OpenACC

- PGI released accelerator directives
- CAPS released HMPP
- Cray prepares to release OpenMP for accelerators directives
- Fall 2010 several members form OpenACC working group
- OpenACC 1.0 (Nov 2010)
- OpenACC 2.0 (June 2013)
- OpenACC 2.0a (Aug 2013)

OpenMP and OpenACC today a comparison

- **Differences**
  - Parallelism
  - Present_or_*
  - Scalars
  - Loops
  - Calls (separate compilation units)

- **Parallelism**
  - OpenACC
    - “Off-load” and parallel startup tied together
      - acc parallel
      - acc kernels
  - OpenMP
    - “Off-load” and parallel startup disconnected
      - omp target
        - omp parallel
        - omp teams
      - omp target teams – closely nested or not going to work

OpenMP teams vs parallel

- **Why two different "parallel" mechanisms**
- **Teams**
  - Independent collision domains
    - There used to only be one of these
  - Same behavior as OpenACC gangs
  - Only select directives allowed
- **Parallel**
  - A single collision domain
  - Default if neither is present
  - All non-accelerator OpenMP directives allowed

Loops

- **OpenACC**
  - One construct “loop”
  - Multiple parallelism types
  - “nested” parallelism implicit
  - Three levels available
    - Gang
    - Worker
    - Vector
- **OpenMP**
  - Three constructs
    - Distribute
    - Do/for
    - Simd
  - Nested parallelism explicit
- **Implict parallelism?**
  - OpenACC
    - !$acc loop worker
  - OpenMP
    - !$omp parallel do

Calls

- **OpenACC**
  - Routine
  - Only one type of parallelism allowed
    - Gang
    - Worker
    - Vector
    - Seq
  - Hard on user
  - Easy for implementer
- **OpenMP**
  - Declare
    - Type of parallelism ignored
  - Easy on user
- Hard for implementer

Nested parallelism

- **OpenACC**
  - Added in 2.0
  - Currently no full implementations
    - Why?
- **OpenMP**
  - Parallel inside of teams is allowed
  - Teams inside of teams is not allowed.

OpenACC and OpenMP tomorrow

- **OpenACC**
  - Tools interfaces
    - Call back based
    - Host captures state of device
  - Better user defined type support
    - Structures, classes, derived types
  - Better performance portability across implementations
    - Requires agreement about what the spec means!
- **OpenMP**
  - Unstructured data constructs
  - Declare target deferred_map (OpenACC link)
  - Interoperability with accelerated libraries
  - Multiple devices
  - User defined type support
  - Asynchronous support
  - map(<direction>:update: list)
  - Better memory hierarchy support

OpenACC and OpenMP on future architectures

- **Upcoming "interesting" architectures**
  - Intel PHI
  - Nvidia
  - ???
- **Commonalities**
  - Vectorization
  - A lot of threads
    - Groups of threads, numa on a chip
    - 60+ (120+) depends what you count
      - This number jumps if you count vector lanes
    - 2048 per multiprocessor
      - 15 multiprocessors
      - This number goes down if you consider warps as vectors
  - Interesting memory structure
- **Separate memories no longer an "issue"**
  - Or are they?

Vectorization

- **Does we need anything new here?**
- **OpenMP**
  - SIMD
  - Workshare
- **OpenACC**
  - Loop vector
  - Autovector
  - Kernels
- **Programmers just need to learn to write vector loops**
  - Questions
    - Is a vector loop a parallel loop?
      - yes
    - Is a parallel loop a vector loop?
      - No
- **Conclusion**
  - We just need better programmers not directives

Threads

- **Is the number of threads an issue?**
  - How many codes do you see that need 10000 OpenMP threads?
  - What does a barrier look like if 10000+ threads need to checkin?
    - What about 100 threads?
    - Answer, barriers are going to get more complex
  - Does threadprivate make sense anymore
    - Did it ever really make sense
    - Fortran programmers say yes
    - C and C++ programmers tend to say no
  - Thread affinity
    - Intel
    - Nvidia

Memory architecture

- **Intel and Nvidia are both making it easier to access data where ever it is in memory!**
- **Is this a good solution?**
- **At least two memories**
  - Bandwidth optimized memory
  - Latency optimize memory
- **Cache**
  - Do we expose these in the programming model?
  - OpenACC uses cache directive
    - This is targeted at nvidia shared memory
    - Is it sufficient or do we need something more?
- **Do we need something new when memory motion is no purely an optimization?**
  - Compiler can get something right
    - Is this enough?

Conclusions

- **OpenACC is nearing mid-life**
- **OpenMP is working hard to catch up with OpenACC**
- **All this work is not a waste of effort**
  - Memory structures are just going to get more complex
  - Parallel loops will always be important
  - Vector loops are important again and will remain so
    - Remember vector loops are parallel loops
- **Location location location**
  - Not just a realtors buzz word anymore!