Programming Accelerators using Directives

Credits: Introduction to OpenACC and toolkit – Jeff Larkin, Nvidia – Oct 2015
Heterogeneous Parallel Computers

• Composed of
  – CPU(s)
    • Low-latency processor optimized for sequential execution
    • large memory size and deep memory hierarchy
  – Accelerator(s)
    • high throughput SIMD or MIMD processors optimized for data-parallel execution
    • high performance limited memory size and small depth memory hierarchy

• Example
  – Multisocket compute server
    • Host: two-socket 20 – 40 Intel Xeon cores with 128 – 512 GB CC-NUMA shared memory
    • Accelerators: 1-8 accelerators (e.g. Nvidia Cuda cards, Intel Xeon Phi cards) connected via PCIe x16 interfaces (16GB/s)
      – host controls data to/from accelerator memory
Basic Programming Models

• Offload model
  – idea: offload computational kernels
    • send data
    • call kernel(s)
    • retrieve data
  – accelerator-specific compiler support
    • Cuda compiler (nvcc)
    • Intel vectorizing compiler (icc -mmic)
      – #pragma offload target(mic:n) in(...) out(...) inout(...)
  – accelerator-neutral OpenCL
    • Cuda-like notation
    • OpenCL compiler can target Nvidia or Intel Xeon Phi
Emerging Programming Models

- **directive model**
  - idea: identify sections of code to be compiled for accelerator(s)
    - data transfer and kernel invocation generated by compiler

- **accelerator-neutral efforts**
  - **OpenACC**
    - \#pragma acc parallel loop
      - for (...) { ... }
    - gang, worker, vector (threadblock, warp, warp in SIMT lockstep)
    - PGI, Cray, CAPS, Nvidia compilers
  - **OpenMP 4.0**
    - similar directives to (but more general than) OpenACC
    - implemented by gcc 4.9 and icc compiler

- **accelerator-specific compiler support**
  - Intel Cilk Plus and C++ compilers for Intel Xeon Phi