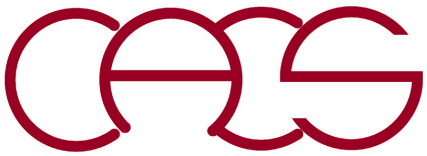
OpenMP Offload for Heterogeneous Architectures

Aiichiro Nakano

Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Chemical Engineering & Materials Science
Department of Biological Sciences
University of Southern California

Email: anakano@usc.edu

Goal: Unified high-level programming of both CPU & GPU



Exaflop/s Supercomputing

- **Diverse exaflop/s supercomputing platforms**

Summit (0.2 Exaflop/s, current)
IBM CPU/NVIDIA GPU

Aurora (Exaflop/s, 2021)
Intel CPU/Intel GPU

FRONTIER

Frontier (1.5 Exaflop/s, 2021)
AMD CPU/AMD GPU

- **Need an open programming model for heterogeneous (*e.g.* GPU-accelerated) clusters (note CUDA is a proprietary language by NVIDIA)**

See http://press3.mcs.anl.gov/atpesc/files/2019/08/ATPESC_2019_Track-1_2_7-29_845am_Parker-Theta.pdf

Open Programming Models

- **OpenCL (Open Computing Language)**

  Open standard for programming heterogeneous devices

  https://www.khronos.org/opencl/

- **OpenMP 4.5/5.0**

  Starting specification version 4.5, OpenMP allows offloading the execution of the code & data to heterogeneous devices

  https://www.openmp.org/specifications/

OpenMP Offload

- **Latest version of OpenMP allows one to maintain one version of a code, which can run on either a general-purpose central processing unit (CPU) or an accelerator (*e.g.* graphic processing unit, GPU; tensor processing unit, TPU; digital signal processor, DSP; field-programmable gate array, FPGA)**
- **Objective is to execute parts of the program on a heterogeneous *accelerator device* (or *target deice*), *i.e.*, dedicated computer hardware outside CPU (which will be called *host device*) to execute certain functions faster than CPU.**
- **In OpenMP, program execution begins on the host, which offloads the execution of parts of the code & data to accelerator.**

See “OpenMP 4.5 Target Offload” (NASA Ames)
http://cacs.usc.edu/education/cs653/OpenMP4.5_3-20-19.pdf

OpenMP Target Construct

- **Simple example**

```
main() {
  float a[1000],b[1000],c,d;
  ...
  #pragma omp target map(a,b,c,d)
  {
    int i;
    #pragma omp parallel for
    for (i=0; i<N; i++)
      a[i] = b[i]*c+d;
  }
  ...
}
```

- **When a host thread encounter the `#pragma omp target` directive, the target region specified it will be executed by a new thread running on an accelerator.**
- **Before the new thread starts executing the target region, the variable in the `map()` clause are mapped onto accelerator memory, which often is disjunct from the host memory.**
- **The offloaded code is usually a data-parallel structured block, which can be handled by multiple threads on accelerator using standard OpenMP constructs like `#pragma parallel for`.**

Asynchronous Offload

```
main() {
  float a[1000],b[1000],c,d;
  ...
  #pragma omp target nowait map(a,b,c,d)
  {
    int i;
    #pragma omp parallel for
    for (i=0; i<N; i++)
      a[i] = b[i]*c+d;
  }
  func(b);  // perform computation independent of device output
  #pragma omp taskwait
  func(a);  // perform computation dependent on device output
}
```

- **By default, the thread that encounters a device construct waits for the construct to complete before executing the next line.**
- **When a `nowait` clause is added to the device construct, the encountering thread does not wait but instead continues executing the code passed the construct.**
- **The `taskwait` constructs lets the original thread wait for the completion of the target task generated by it before continuing to the next line.**