GPGPU support in Chapel with the Radeon Open Compute Platform

Michael L. Chu, Ashwin M. Aji, Daniel Lowell, Khaled Hamidouche
AMD Research
{Mike.Chu, Ashwin.Aji, Daniel.Lowell, Khaled.Hamidouche}@amd.com

In this talk, we provide a description of our continuing effort to develop generalized GPGPU support within the Chapel language. As GPUs are increasingly important for accelerating computation, adding a method for users to develop, target and execute code on them is of utmost importance for the future of the language.

In our previous CHIUW 2016 talk, we described the modifications to expose a GPU through hierarchical locales and the runtime changes to execute specific data-parallel constructs, namely reductions. This talk will be a progress report of our development over the past year in generalizing our technique for more language constructs, and detail our plans for the future. So far, our focus has been to first enable Chapel GPU execution with as minimal language changes as necessary. As we look towards the future, language extensions for Chapel will likely be required to fully take advantage and optimize code to execute on a GPU.

This talk will also introduce AMD's Radeon™ Open Compute Platform (ROCm). ROCm is AMD's current evolution of the Heterogenous Systems Architecture (HSA) tool chain, with a strong focus in the HPC area. ROCm includes a compiler, runtime and tools which help developers target programming models for AMD GPUs. ROCm serves as the underlying layer in our Chapel implementation for creating and launching kernels for execution.

Our current implementation supports executing the following Chapel code constructs on an AMD APU:

```
proc run() {
  var A: [D] int;
  var B: [D] int;
  var C: [D] int;
  on (Locales[0]:LocaleModel).GPU do {
    // 1. Reduction
    var sum = + reduce A;
    // 2. for-all expression
    [i in D] A(i) = i;
    // 3. for-all statement
    forall i in 1..n {
      A[i] = A[i] + 1164;
    }
    // 4. Array assignment
    B = A;
    // 5. Function promotion
    C = square(B);
    // 6. for-all with filtering
    [i in D] if i % 2 == 1 then C[i] = 0;
  }
}
```

These Chapel constructs are executed via a Chapel-to-OpenCL™ pass in the compiler that runs alongside the current Chapel-to-C code generation. Our compiler takes order-independent CForLoops, and generates a GPUForLoop block. During runtime, this code block is conditionally executed on the GPU if requested by the Chapel developer. We plan to discuss the details of how the Chapel compiler and runtime was modified to generate the GPU kernels and launch them during runtime. This talk will then highlight on our future plans for the GPU enablement and areas we are interested to study. Topics include: interoperability of Chapel with

existing GPU languages, Chapel language change proposals for optimized GPU acceleration, support for discrete GPU cards, and creation of task graphs via possible integration with Chapel futures.