Chapel-on-HSA: Towards Seamless Acceleration of Chapel Code using the Heterogeneous System Architecture

Abhisek Pan, Michael Chu
Advanced Micro Devices
Austin, TX, USA
{ Abhisek.Pan, Mike.Chu }@amd.com

In this talk, we report our continuing work towards supporting execution of GPU kernels through Chapel with minimal modifications to the core language specification. Specifically, we have integrated the Heterogeneous System Architecture (HSA) framework within the Chapel programming language in order to provide support for GPU-execution of the data-parallel constructs already present in Chapel. GPUs are exposed within a hierarchical locale to the programmer, who can explicitly request GPU execution for certain data-parallel constructs (such as foralls, reductions, and scans) by specifying these constructs within the GPU sublocale.

Among these data-parallel constructs, we have first provided support for GPU execution of the predefined reduction operations in Chapel. Chapel provides a number of predefined reduction operators that can be used to reduce aggregate expressions (such as arrays, sequences, or any iteratable expression) to a single result value. Chapel also provides a mechanism to support user-defined reductions where the programmer can implement custom operators for the reductions. We have modified the Chapel compiler and runtime to allow execution of predefined reductions on GPUs, if the reduction operation is executed on a GPU sublocale. For example, the following code snippet will be executed on a GPU.

```
on Locales[0] do {
    var A: [1..3] int = (1,2,3);
    on (Locales[0]:LocaleModel).GPU do {
        var sum = + reduce A;
   }
}
```

Currently the predefined reductions are implemented through Chapel classes defined in the internal module ChapelReduce. Each predefined operator is backed by a separate class that implements a specific interface. The Chapel parser replaces a reduction expression with a call to a new function that returns the result of the reduction. This new function implements the reduction by calling the methods of the backend operator-class in a loop.

For GPU acceleration of reductions, we store the pre-compiled version of an OpenCL kernel for every combination of reduction operator and data-type that Chapel supports. We implement a C-function inside the Chapel runtime that constructs the Architected Queuing Language (AQL) packets and enqueues the appropriate kernels depending on the operator and data-type of reduction. The function also waits for completion of kernel execution and returns the final result. We then modify the Chapel compiler so that if the reduction is executed on a GPU sublocale, the compiler replaces the reduction operator with a call to this C-function instead. Note that the C-function has access to the GPU device, the HSA command queue, and the symbols for the predefined kernels since GPU device discovery and HSA initialization happens during the initialization of the GPU sublocale, before the execution of any reduction. The HSA framework provides a shared address-space abstraction to the application, so that data allocated on the CPU can be accessed from the GPU without any explicit data-movement operations.

This talk will discuss the details of our current Chapel/HSA prototype implementation and highlight how we hope to further extend the use of GPUs within the Chapel language.