Programming Heterogeneous Systems

Our Goal
To understand how programming models, compilation techniques, and runtime systems can help ease the burden associated with programming heterogeneous systems.

Stepping into the Heterogeneous
Heterogeneous systems are becoming more popular

- Several appearances on the Top500 and Green500 lists
- Can be effective for small research clusters (greater performance per dollar)

Benefits of heterogeneous systems
- High flop rates per dollar
- High flop rates per watt

Nature of the Charm++ programming model
- Asynchronous tasks with an arbitrary execution model
- Well defined working sets
- Can execute on either the host or an attached accelerator (based on a load balancing scheme)
- Associated callback function (general)

Why Charm++?
Nature of the Charm++ programming model
- Asynchronous tasks ( unlike threads in MPI)
- Migratable chare objects allow data and computation to be passed between different cores
- Asynchronous tasks with an arbitrary execution model

Benefits of heterogeneous systems
- Associated callback function (general)
- Typing and array information (requirements/limitations)

Accelerated Entry Methods
Asynchronous tasks with an arbitrary execution model
- Well defined working sets
- Can execute on either the host or an attached accelerator (based on a load balancing scheme)
- Accelerated function body (limited)
- Associated callback function (general)

Heterogeneous Execution
Program model provides:
- Clearly defined communication boundaries
- Typing and array information (requirements/limitations)
- The runtime system handles tedious (but necessary) tasks related to heterogeneous execution, such as:
- Interoperability between different "flavors" of the runtime system within a single application execution
- Real-time manipulation of application data between cores (e.g., big vs. little endian encodings)

Performance on a Heterogeneous System
Heterogeneous applications may scale better on heterogeneous clusters (compared to a homogenous cluster)
- We demonstrate a simple MD program that scales better using a mixture of x86 and Cell processors (compared to just using Cells)
- Reaches 19.8% of peak using one dual-core x86 processor, four PS3 Cells, and four Intel blade Cells
- Does not include any architecture specific code or code to translate data between architectures
- Makes use of three different core types (x86, PPE, SPE), three different SIMD extensions, two different memory schemes, and two endian schemes (little and big endian)

Continuing & Future Work
Dynamic Load Balancing
- Dividing a homogeneous workload across a set of heterogeneous cores to minimize the time-to-solution
- Dividing a heterogeneous workload across a set of heterogeneous cores, matching sub-computations to the appropriate cores
- Accounting for the non-peer relationships (or topology) of the cores (i.e., host cores and accelerators are not peers to one another)

Improved Support for GPUs
The runtime tightens the execution model of sub-computations when it is beneficial
- Start with an arbitrary execution model (asynchronous tasks)
- Take advantage of patterns (broadcasts, stencil patterns, n-body spatial decompositions, etc.)

Granularity
The compiler assists with automatic granularity adjustments
- Distinguish between communication and computation granularity
- Computation granularity largely based on architecture characteristics
- Compilers focus on what they are good at (small well-defined tasks)
- Programmers focus on what they are good at (identifying high-level parallelism within their applications)

Grain

Related Publications: