Accelerators for HPC: Programming Models

**GPUs**
- Massively Data Parallel Computation Engines, capable of delivering TFLOPs level throughputs.
- Organized as a set of Streaming Multiprocessors (SMs), each of which can be viewed as a very wide SIMD unit
- Typically programmed using CUDA, OpenCL, which are extensions of C/C++
- Steep learning curve and Portability Issues

**StreamIt on GPUs**
- Provides a natural way of implementing DSP applications on GPUs
- Challenges:
  - Determining the optimal execution configuration for each filter
  - Harnessing the data parallelism within each SM
  - Orchestrating the execution across SMs to exploit task and pipeline parallelism
  - Register constraints

**Our Approach**
- Good execution configuration determined by using profiling – Identify near-optimal number of concurrent thread instances per filter.
  - Takes into consideration register constraints
- Formulate work scheduling and processor (SM) assignment as a unified Integer Linear Program problem.
  - Takes into account communication bandwidth restrictions
- Efficient buffer layout scheme to ensure all accesses to GPU memory are coalesced.
  - Ongoing Work: Stateful filters are assigned to CPUs – Synergistic execution across CPUs and GPUs

**StreamIt**
- A High level programming model where nodes and arcs represent computation and communication respectively
- Exposes Task, Data and Pipeline Parallelism
- Example: A Two-Band Equalizer

**The Process**
- Generate Code for Profiling
- Profile Execution and Configuration Selection
- Constraint Solver (CPLAS)
- Profile Data
- Constraint Selection
- Software Pipelined Scheduling
- Constraint Resolution
- Schedule for Each Filter
- Execute Profiles
- Configuration Selection
- Schedule and Execute

**The Results**
- Speedups of up to 33.82X over a single threaded CPU execution
- Software Pipelined scheme outperforms a naive serial SAS schedule in most cases
- Coalesced Accesses result in huge performance gains, as evident from the poor performance of SWPNC

Further details in: “Software Pipelined Execution of Stream Programs on GPUs” (CGO 2009)

**The PLASMA Framework**
- “CPLASMA”, a prototype high-level assembly language
- Prototype PLASMA IR Compiler
- Currently Supported Targets:
  - C (Scalar), SSE3, CUDA (NVIDIA GPUs)
- Future Targets:
  - Cell, ATI, ARM Neon, ...
- Compiler Optimizations for “Vector” IR

**Enter PLASMA...**
- Operator
  - Add, Mul, ...
- Vector
  - 1-D bulk data type of base types
    - E.g. <1, 2, 3, 4, 5>
- Distributor
  - Distributes operator over vector
    - Example: par add <1,2,3,4,5>
    - <10,15,20,25,30> returns <11, 17, 23, 29, 35>
- Vector composition
  - Concat, slice, gather, scatter, ...

**Initial Performance Results**

**What a Solution Needs to Provide**
- Rich abstractions for Functionality
  - Not a lowest common denominator
- Independence from any single architecture
- Portability without compromises on efficiency
  - Don’t forget high-performance goals of the ISA
- Scalability – Both ways, from micro engines to massively parallel devices
  - Single core embedded processor to multi-core workstation
- Take advantage of Accelerators (GPU, Cell, etc.)
- Transparent Access to Distributed Memory
- Management & load balance across heterogeneous devices