Cooperative Multitasking for Heterogeneous Accelerators in the Linux Completely Fair Scheduler

Tobias Beisel, Tobias Wiersema, Christian Plessl and André Brinkmann

Problem definition

**Heterogeneous Scheduling**
- State-of-the-art: Mapping tasks to homogeneous CPUs
- Heterogeneous Systems: Mapping tasks to multiple accelerators
  - Tasks may be capable to run on different available architectures
  - Scheduling: Choosing the best architecture for a specific objective
    - Performance (example)
    - Energy efficiency
    - Utilization

**Acceleration**
- Only allow run-to-completion execution model
- Do not support preemption or interrupts
- Do not allow to access full internal state
- Can usually not access shared CPU memory autonomously
- Require explicit communication of inputs and results
- Provide non-uniform communication properties
- Use completely different APIs and ISAs
- Require dedicated binaries
- Do not allow tasks to be migrated easily

Contribution

**Linux Completely Fair Scheduler (CFS) Extension**
- Provides awareness of installed accelerators
- Enables scheduling of specific tasks to accelerators
- Allows time-sharing with preemption at application provided checkpoints
- Uses proprietary runtime libraries and drivers to allow communication

Using the Kernel Space Scheduler

**Programming model / Application provided information:**
- Application spawns threads
- Application uses system calls to
  1) Allocate a compute unit
  2) Re-Request a compute unit
  3) Free a compute unit
- Application provides
  1) Meta information about threads
    - E.g. accelerator affinities, data sizes
  2) A Checkpoint for cooperative multitasking
  3) Functions pointers for each supported hardware architecture for:
    - Accelerator initialization
    - Computation between 2 checkpoints
    - Free function to release accelerator

**Cooperative Multitasking on Heterogeneous Systems is possible**
- Example:
  - 15 GPU affine MDS cracking instances
  - Using GPU and CPUs
  - a) Stop + resume tasks
  - b) + d) Refill GPU queue
  - c) + e) Migrate CPU jobs to GPU
- Example: cooperative scheduler prefers best suited hardware (GPU)

Average turnaround times can be reduced significantly
- 25 MDS cracking and 50 prime factorization threads
- 4s time slices

**Outlook**
- Compare kernel space scheduler with user space approach
- Simplify programming model by automatic checkpoint detection and extraction
- Use further example applications also incorporating FPGAs

Contact
- Tobias Beisel, tbeisel@uni-paderborn.de
- Paderborn Center for Parallel Computing, University of Paderborn, Germany