Cooperative Multitasking for Heterogeneous Accelerators in the Linux Completely Fair Scheduler

PADERBORN CENTER FOR PARALLEL COMPUTING

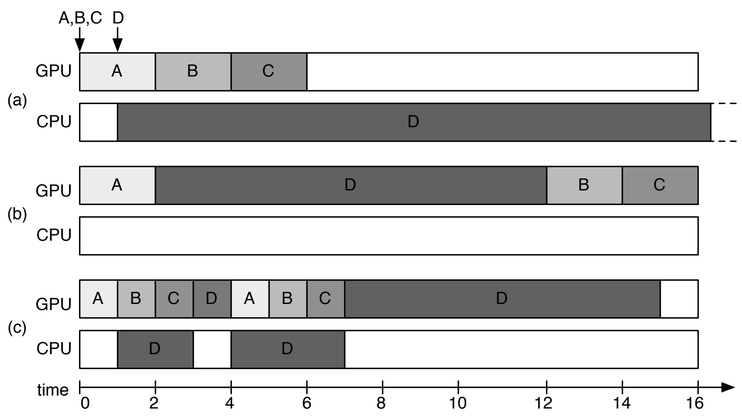
Tobias Beisel, Tobias Wiersema, Christian Plessl and André Brinkmann

ASAP'11, Santa Monica

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



A, B, C: Tasks only capable to run on the GPU
D: Task executable on CPU or GPU

Challenges

Accelerators

- Only allow run-to-completion execution model
- Do not support preemption or interrupts
- Do not allow to access full internal state

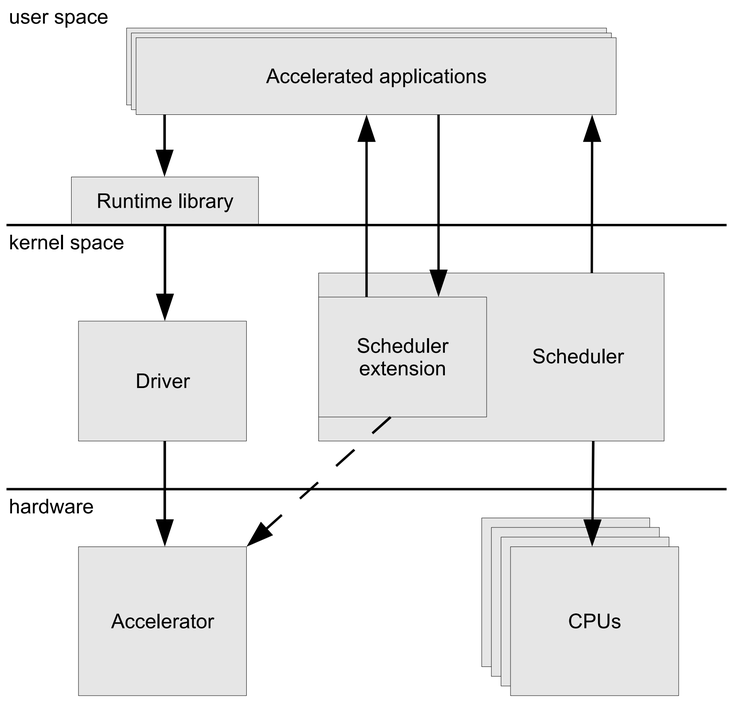
- Can usually not access shared CPU memory autonomously
- Require explicit communication of inputs and results
- Provide non-uniform communication properties

- 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



Key design decisions

Task representation as delegate threads

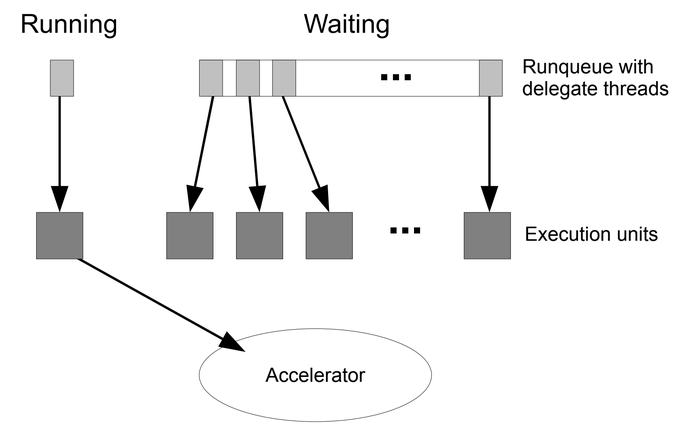
- Perform all OS interaction and communication with the accelerator
- Represent hardware task to the system
- Allow to use existing CFS scheduler

In-kernel task handling

- CFS: preemtive multitasking
- Cooperative multitasking allows voluntarily release accelerator
- Accelerator specific queues

Kernel Space Integration

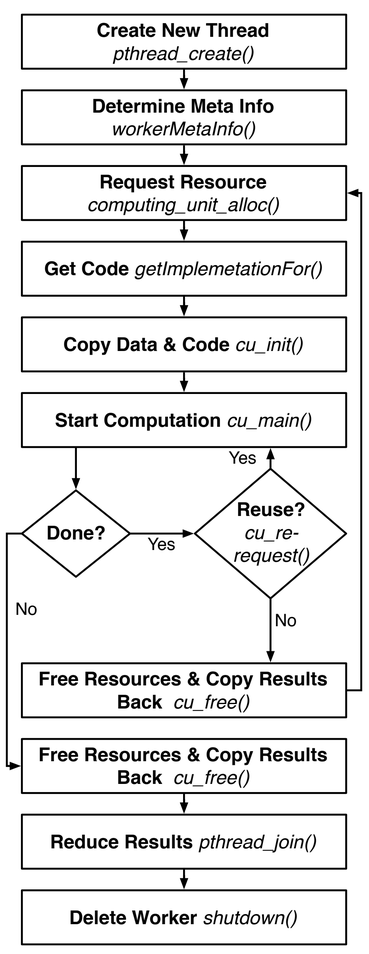
- Allows global decision
- Hides scheduling principles from user
- Requires inputs from user space



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

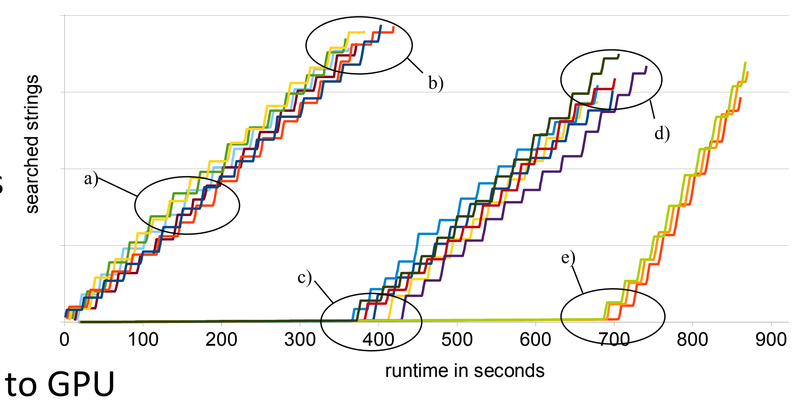


Results

Cooperative Multitasking on Heterogeneous Systems is possible

- Example:
  - 15 GPU affine MD5 cracking instances
  - Using GPU and CPUs

a) Stop + resume tasks
b) + d) Refill GPU queue
c) + e) Migrate CPU jobs to GPU

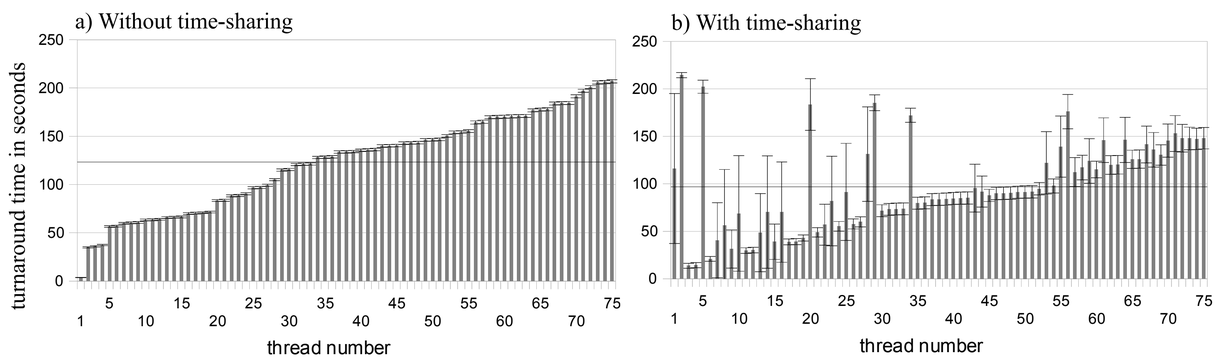


→ *Example: cooperative scheduler prefers best suited hardware (GPU)*

Average turnaround times can be reduced significantly

- 25 MD5 cracking and 50 prime factorization threads
- 4s time slices

→ *Example: Average turnaround-time reduced by approx. 22 %*



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