Dynamic inter-core scheduling in Barrelfish

avoiding contention with malleable domains

Georgios Varisteas, Mats Brorsson, Karl-Filip Faxén

November 25, 2011
Outline

• Introduction
• Scheduling & Programming models
• Malleability
• Future work
Barrelfish: a multi-kernel OS

- message based communication
- replication and consistency
- heterogeneity

- assumes no shared-memory
- provides no system-wide resource management
Overall Goals

• Allow for shared memory based parallel programming models
  – OpenMP, Wool, Cilk++
  – take advantage of the underlying hardware architecture.

• … while exploiting the message passing nature of Barreelfish
  – scalability
  – portability
This project...

- Perform resource management in order to increase throughput and minimize contention in Barrelfish
  - Inter-core scheduling
    - system-wide load balancing
  - Dynamic scheduling
    - malleable resource allocation
  - In a multiprogrammed context
Motivation

• Current parallel programming models:
  – focus on running in isolation
  – minimal operating system support
  – can be wasteful in a multiprogrammed context

• Many real-life applications:
  – exhibit fluctuating parallelism throughout their execution
  – are not that parallel from the start
Scheduling

- Split into two cooperating levels
  - System level,
    - aware of the global state and the availability of diverse resources
  - User level,
    - aware of the parallelism in the application
System scheduler

- Accept feedback on process efficiency
- Modify the allotment of cores (domain) of each process for maximum resource utilization
- Distributed service
  - multiple instances
  - overlook distinct segments
  - processes can span multiple segments
User-level scheduler

- Integrated into the application run time
- Schedules a process' threads in its domain
- Provides feedback on per core **efficiency**, to the system scheduler [1]
  - metric: **wasted cycles**
    “cycles spent while not having work”

User level scheduler, cont'd

- Capture average & worst thread efficiency
- Over a fixed interval classify on two criteria:
  - inefficient or efficient: utilization of workers
  - satisfied or deprived: system contention
  - inefficient: overestimation, desire decreased
  - efficient & satisfied: underestimation, desire increased
  - efficient & deprived: balanced, desire unchanged
- Forward new desire and classification
Shared memory programming models (OpenMP, Wool, Cilk++)

- Focusing on the task-based paradigm
  - work-stealing models scale easily

- Wool already ported
  - application state in the stack

- Cilk++ requires a custom compiler
  - application state in the heap
Over a fixed interval each instance will:
- increase allotment for its segment's "efficient and satisfied" processes
- extra cores are either idle or taken from its segment's "inefficient" processes
- if needed broadcast a request to other scheduler instances
- result to time-sharing if not enough "inefficient" processes exist

Which worker to suspend?
Time sharing not always avoided

- Joining a task requires simultaneous execution of the workers involved

- Phase-lock gang scheduling [1]
  - Efficient gang scheduling for barrelfish

Scalability and Portability?

Inter-Core communication
Intra-Core communication

Process 1
- Worker
- Task
- Task
- Task

System-wide scheduler
Monitor
CPU driver
Core 0

Process 2
- Worker
- Task
- Task
- Task

System-wide scheduler
Monitor
CPU driver
Core 2
Core 3

Section 1
Section 2
Malleable domains

- Load balance the system by modifying the domain of each process
  - unwanted worker-threads are suspended
  - or new ones are added

- Worker-thread suspension tricky, depends on the run-time in use
  - lazy-suspension
  - immediate-suspension

1) Continuation-passing-style: Shared memory is used instead of the CStack.
Immediate suspension
Lazy suspension
- Intelligently migrate processes to avoid contention
- Allot processing resources according to runtime's efficiency & app's parallelism
Future Work

- Evaluation in comparison to other OSs
- Locality aware allotment of cores
- Use core attributes as criteria on heterogeneous systems
- Handle the absence of shared-memory support in the architecture
THANK YOU

Q & A