Resource-Conscious Scheduling for Energy Efficiency on Multicore Processors

Andreas Merkel, Jan Stoess, Frank Bellosa
Memory Contention – a Problem on Multicores
Memory Contention – a Problem on Multicores
Memory Contention – a Problem on Multicores
Memory Contention – a Problem on Multicores
Memory Contention – a Problem on Multicores

- Memory
- CPU
- CPU
- CPU
- CPU
- CPU
- CPU
- CPU
Memory Contention – a Problem on Multicores

<table>
<thead>
<tr>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
</tbody>
</table>

Memory
Memory Contention Intel Core2 Quad

- Bottleneck: memory bus
- Stall cycles, increased runtime

![Graph showing normalized runtime per instance for different numbers of instances: 1 instance, 2 instances, and 4 instances on 4 cores. The x-axis represents the stream memory benchmark, and the y-axis represents the normalized runtime per instance. The graph shows increased runtime as the number of instances increases.]


Impact of Resource Contention on Energy Efficiency

- Longer time to halt
- More static power
- Increasing importance of leakage
Achieving Energy Efficiency by Scheduling

- Scheduler decides
  - When
  - Where
  - In which combination
  - At which frequency setting
to execute tasks.

- What is the most energy-efficient schedule?
Achieving Energy Efficiency via Co-Scheduling

- Combination of tasks running together determines performance and energy efficiency
- Memory-bound + memory-bound: low energy efficiency
- Avoid memory bottleneck by combining memory-bound with compute bound tasks

➔ Co-schedule tasks with different characteristics
Achieving Energy Efficiency via DVFS

- **DVFS**: Dynamic Voltage and Frequency Scaling
- Adapt processor frequency and voltage to task characteristics
  - Memory-bound tasks: low frequency/voltage
  - Compute-bound tasks: high frequency/voltage
- Multicore hardware limits options for frequency/voltage selection
  - Often shared frequency/voltage domains
  - Co-schedule similar tasks to select common best frequency and voltage

\[
\begin{align*}
\text{energy efficiency} & \quad \rightarrow \quad \text{use DVFS} \quad \rightarrow \quad \text{mem + mem}
\end{align*}
\]
Achieving Energy Efficiency

- energy efficiency
- use DVFS
- avoid contention
- mem + mem
- mem + comp
Achieving Energy Efficiency

- Use DVFS
- Avoid contention

Energy efficiency

Memory + Memory

Memory + Computation
Outline

- Analysis
  - Resource contention
  - Shared frequency/voltage domains

- Resource-conscious scheduling for energy efficiency
  - OS task scheduling
  - VM scheduling
  - Frequency selection

- Evaluation
  - Reduction of resource contention
  - Increase in energy efficiency by 10 to 20%
Analysis of Resource Contention on the Intel Core2 Quad Q6600

- Contention for shared resources reduces energy efficiency
  - Shared L2 caches (two cores)
  - Shared memory interconnect (four cores)
Resource Contention SPEC CPU 2006

The chart illustrates the normalized runtime per instance for two programs, `hmmer` and `libquantum`, under different configurations of cores and cache setups. The x-axis represents the number of instances per core, ranging from 1 to 4. The y-axis represents the normalized runtime.

- **1 instance**
- **2 instances separate caches**
- **2 instances shared caches**
- **4 instances**

The chart shows that the runtime increases as the number of instances increases, particularly noticeable for `libquantum`.

Diagram details:
- **Legend**: Blue for 1 instance, Red for 2 instances separate caches, Yellow for 2 instances shared caches, Green for 4 instances.
- **Axes**: Normalized runtime per instance on the y-axis, instances on the x-axis.
- **Graph**: Two columns for `hmmer` and `libquantum`, with bars showing the normalized runtime for each configuration.
Resource Contention SPEC CPU 2006

The chart shows normalized runtime per instance for two benchmarks, `hmmer` and `libquantum`. The vertical axis represents the normalized runtime across different configurations:

- **1 instance**
- **2 instances**
  - separate caches
  - shared caches
- **4 instances**

The benchmarks are represented by different colors:

- Blue: 1 instance
- Red: 2 instances (separate caches)
- Yellow: 2 instances (shared caches)
- Green: 4 instances

The diagram also includes a representation of a multicore processor with two L2 caches and a memory interconnect between the cores.
Resource Contention SPEC CPU 2006

- Normalized runtime per instance
- 1 instance
- 2 instances separate caches
- 2 instances shared caches
- 4 instances

1 instance: hmmer
2 instances separate caches: libquantum
2 instances shared caches: libquantum
4 instances: libquantum

Diagram showing resource contention for SPEC CPU 2006 benchmarks hmmer and libquantum across different instances and cache configurations.
Resource Contention SPEC CPU 2006

The chart shows the normalized runtime per instance for different scenarios:

- **1 instance**
- **2 instances separate caches**
- **2 instances shared caches**
- **4 instances**

The scenarios are as follows:

- **hmmr**: 1 instance, separate caches vs. shared caches vs. 4 instances.
- **libquantum**: 1 instance, separate caches vs. shared caches vs. 4 instances.

The chart uses colors to differentiate between the scenarios:
- Blue for 1 instance
- Red for 2 instances separate caches
- Yellow for 2 instances shared caches
- Green for 4 instances

The y-axis represents the normalized runtime per instance, ranging from 0 to 4.5.
Resource Contention SPEC CPU 2006

- **Normalized runtime per instance**

  - **hmrer**: 1 instance, 2 instances, separate caches
  - **libquantum**: 1 instance, 2 instances, shared caches, 4 instances

- **Graph**
  - X-axis: Applications (hmrer, libquantum)
  - Y-axis: Normalized runtime per instance
  - Color Coding:
    - Blue: 1 instance
    - Red: 2 instances
    - Yellow: 2 instances, shared caches
    - Green: 4 instances

- **Diagram Notes**
  - Memory interconnect
  - Core layout: Core 0, Core 1, Core 2, Core 3
  - L2 caches for Core 0 and Core 3
Resource Contention SPEC CPU 2006

- 1 instance
- 2 instances separate caches
- 2 instances shared caches
- 4 instances

compute-bound vs. memory-bound
Resource Contention SPEC CPU 2006

- Compute-bound benchmarks
  - Little resource contention

- Memory-bound benchmarks
  - Severe slowdown caused by memory contention
  - Huge increase in memory demands since SPEC 2000
  - Cache contention of comparatively little importance
Energy Efficiency under DVFS

- Comparison of 1.6GHz to 2.4GHz
- 4 instances of benchmark
- Reducing the frequency pays off for memory intensive tasks

![Bar chart showing energy efficiency comparison between 1.6GHz and 2.4GHz for various benchmarks. The x-axis represents compute-bound and memory-bound tasks, while the y-axis shows energy consumption. The chart includes bars for time, energy, and EDP (energy-delay-product) for each benchmark.](image)
Energy Efficiency under DVFS and Resource Contention

- time
- energy
- EDP
Energy-Efficient Co-Scheduling

Energy efficiency

Avoid contention

Use DVFS

Mem + Mem

Mem + Comp
Energy-Efficient Co-Scheduling

- energy efficiency
- avoid contention
- mem + comp
Energy-Efficient Co-Scheduling

- Avoiding resource contention
  - Requires knowledge of task characteristics
  - Requires coordination of task selection across cores

- Merkel and Bellosa, EuroSys 2008
  - Task characterization
  - Execution of tasks in a defined order (runqueue sorting)
  - Used for mitigating thermal effects

- Take advantage of runqueue sorting to provide coordination with low overhead
Sorted Co-Scheduling

- Group cores in pairs
- Sort runqueues by critical resource (memory bandwidth)
- Coordinate processing of runqueues
- Co-schedule tasks with complementary resource demands

![Sorted Co-Scheduling Diagram]
Sorted Co-Scheduling

- Dealing with unequal runqueue lengths

- Example: core 0 executes one task more than core 1
  - Time needed to process runqueues does not even out
    $\rightarrow$ increase length of timeslices on core 1

```
core 1

core 0
```
Sorted Co-Scheduling

- Shift runqueues of additional cores
- Avoid running most memory intensive tasks together
Resource-Conscious Load Balancing

- Sorting requires tasks with different characteristics on each core

- Migrate task if variance among tasks in runqueue is increased
Virtual Machine Scheduling

- Leverage workload diversity of several physical machines
- Extend balancing strategy using the concept of virtualization
- Migrate entire virtual machines
- Co-scheduling of virtual machine instances
Frequency Heuristic

- Fall back to frequency scaling if workload does not allow avoiding contention

- Frequency heuristic takes effect when:
  - Too many memory-bound tasks/VMs are present
  - Sorted scheduling has to co-schedule memory-bound tasks

- Estimate if lower frequency would reduce EDP
Evaluation

- Prototype
- Modified Linux 2.6.22 kernel
  - Runqueue sorting
  - Resource-conscious load balancing
- KVM for virtualization
  - Schedule KVM instances within a physical machine like normal OS tasks
  - Use KVM migration features to move VMs between physical machines
Evaluation

One Intel Core2 Quad, no virtualization
Workload: 8 SPEC benchmarks

- **gamess**, **hmmer**, **namd**, **lbm**, **libquantum**, **mcf**, **soplex**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time</th>
<th>EDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GameSS</td>
<td>0.84</td>
<td>1.01</td>
</tr>
<tr>
<td>Gromacs</td>
<td>0.96</td>
<td>1.00</td>
</tr>
<tr>
<td>Namd</td>
<td>0.92</td>
<td>1.00</td>
</tr>
<tr>
<td>LBM</td>
<td>0.95</td>
<td>1.00</td>
</tr>
<tr>
<td>Libquantum</td>
<td>0.98</td>
<td>1.00</td>
</tr>
<tr>
<td>MCF</td>
<td>1.01</td>
<td>1.00</td>
</tr>
<tr>
<td>Soplex</td>
<td>0.97</td>
<td>1.00</td>
</tr>
</tbody>
</table>

*Linux* standard
Evaluation

Two Intel Core2 Quads
Workload: 8 SPEC benchmarks, each in a separate VM
Worst case: 4 memory-bound benchmarks on one physical machine

![Graph showing relative runtime and EDP for different benchmarks with and without balancing.](image)
Conclusion

- Cross-effects lead to low energy efficiency in multicores
  - Resource contention
  - Shared voltage domains
- Analysis: contention avoidance more important than common optimal frequency/voltage
- Approach: co-scheduling by sorting memory intensity in different directions
  - Resource-conscious load balancing
  - VM scheduling and migration
  - Frequency scaling as fallback
- Result: reduction of EDP by 10 to 20%
Energy Efficiency under DVFS

- Task specific optimal processor frequency/voltage
  - Memory-bound task $\rightarrow$ low frequency
  - Compute-bound task $\rightarrow$ high frequency
Resource Contention

- Tasks compete for shared chip resources
  - e.g., caches, memory (CMP)

- Impact on
  - Runtime
  - Energy efficiency
New Challenges for OS Scheduling

- Scheduler determines task execution
  - When
  - Where
  - What combination

- Scheduling decisions have impact on
  - Energy efficiency
  - Resource contention

=> Information about task characteristics is crucial!
Resource Contention vs. Frequency Selection

- Reducing contention has much greater potential for increasing energy efficiency than DVFS

➔ Schedule tasks in a way that avoids contention, even if some tasks have to run at the “wrong” frequency
New Challenges for OS Scheduling

- Task characterization in today's general purpose OS schedulers
  - User-specified priorities
  - I/O-intensive vs. CPU-intensive
- No indicators for energy efficiency, or resource contention
Task Characterization

- Task activity vectors
  - Characterize tasks by their resource utilization
    - (e.g., functional unit, cache, memory interconnect, ...)
  - Provide information to smart schedulers

- Resource utilization: versatile indicator for
  - Temperature
  - Optimal frequency
  - Contention

Task Activity Vectors: A New Metric for Temperature-Aware Scheduling
Andreas Merkel and Frank Bellosa
Third ACM SIGOPS EuroSys Conference, 2008
Task Activity Vectors

- Vector with \( n \) components
  - Each component represents a resource
  - Component value: utilization of resource while task is running
    - Inferred on-line from performance monitoring counters

\[
\mathbf{v} = \begin{pmatrix}
  v_1 \\
v_2 \\
  \vdots \\
v_n
\end{pmatrix}
\]
Vector-Based Scheduling for Energy Efficiency

- Multiprocessor schedulers make decisions independently for each processor
  - Arbitrary combinations of tasks running together
    - Disregarding of interference
    - Disregarding of task-specific optimal frequency

→ Resource contention
→ Prolonged task runtimes
→ Inefficient use of energy
Energy-Efficient Co-scheduling

- energy efficiency
  - use DVFS
  - avoid contention
EDP Estimation

- Linear interpolation
  - f(1): EDP factor of completely memory-bound microbenchmark
  - f(0): EDP factor of completely compute-bound microbenchmark
  - Estimation for EDP factor of task with memory bus utilization x:
    \[ f(x) = x \times f(1) + (1-x) \times f(0) \]
Resource-Conscious Scheduling for Energy Efficiency on Multicore Processors

Evaluation

Modified Linux 2.6.22 kernel,Intel Core 2 Quad

Workload: eight SPEC benchmarks

Moscibroda and Mutlu

*Memory performance attacks: denial of memory service in multicore systems*

USENIX Security Symposium 2007

- **gamess**
- **gromacs**
- **namd**
- **lbm**
- **libquantum**
- **mcf**
- **soplex**

**relative runtime, EDP**

- **time**
- **EDP**

**compute-bound**

**memory-bound**

standard linux
New Processor Topologies

- On-chip thread-level parallelism
  - simultaneous multithreading (SMT)
  - chip multiprocessors (CMP)
- shared resources
- shared power management
Old Scheduling Policies

- Schedulers designed for traditional SMP systems
- Independent scheduling decisions for each processor
  - combination of tasks running at a time is arbitrary
  - is this optimal for SMT/CMP?
  - what about resource contention?
  - what about power management features like frequency scaling?
- Assumption: a set of unrelated, single-treaded processes is running
  - no communication
Power Management

- Frequency selection
  - SMP: independently for each processor
  - SMT: affects all logical threads of a processor
  - CMP: per-core selection possible at the price of hardware complexity, but often only per-chip

- Some tasks run more efficiently at a certain frequency than others
  - memory-bound tasks: lower frequencies
  - compute-bound tasks: higher frequencies
Multiprocessor Architectures

- Classical SMP
  - physically different chips
  - interference via memory bus (shared bus, cache coherency)

- SMT
  - multiple logical threads on one chip
  - heavy contention for almost all resources

- CMP
  - multiple processors on one chip
  - interference via memory access logic, memory bus
  - sometimes shared caches
Experiments

- Intel Core2 Quad
  - resource contention
    - L2 cache shared between 2 cores
    - memory access infrastructure shared by all 4 cores
  - frequency selection
    - frequency shared by two cores
    - voltage scaling only for entire chip

- Microbenchmarks
- SPEC CPU 2006 benchmarks
Discussion

- Lower frequency is beneficial if all cores execute memory-intensive tasks
- But: Overhead in terms of time and energy if all cores execute memory intensive tasks
- Do the benefits outweigh the overhead?

No:
Contention causes runtime to increase by up to factor 2 to 4
Frequency scaling reduces energy by factor 0.7 at best
=> avoiding contention central issue for energy efficiency
Example Scenario

- 4x hmmer (compute-intensive)
- 4x soplex (memory-intensive)

<table>
<thead>
<tr>
<th></th>
<th>hmmer, 4 instances</th>
<th>soplex, 4 instances</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>energy 24</td>
<td>86427.92</td>
<td>70741.44</td>
<td>157169.36</td>
</tr>
<tr>
<td>time 24</td>
<td>1210</td>
<td>1230</td>
<td>2440</td>
</tr>
<tr>
<td>energy 16</td>
<td>90885.56</td>
<td>57048.03</td>
<td>147933.59</td>
</tr>
<tr>
<td>time 16</td>
<td>1817.5</td>
<td>1282.5</td>
<td>3100</td>
</tr>
<tr>
<td>energy 24/16</td>
<td></td>
<td></td>
<td>143475.96</td>
</tr>
<tr>
<td>time 24/16</td>
<td></td>
<td></td>
<td>2492.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2 hmmer (with 2 soplex)</th>
<th>2 soplex (with 2 hmmer)</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>energy 24</td>
<td>79308.38</td>
<td>51718.62</td>
<td>131026.99</td>
</tr>
<tr>
<td>time 24</td>
<td>1230</td>
<td>805</td>
<td>2035</td>
</tr>
<tr>
<td>energy 16</td>
<td>86645.61</td>
<td>42120.7</td>
<td>128766.32</td>
</tr>
<tr>
<td>time 16</td>
<td>1840</td>
<td>899</td>
<td>2739</td>
</tr>
</tbody>
</table>
Goals

- Design scheduling policy that is optimal for the new architectures
- Use the resource CPU as efficiently as possible in terms of
  - energy
  - time
- Sometimes controversial goals
  - compromise: EDP = energy * delay
Goals

- Run tasks in combinations that cause no interference
- Run each task at its optimal frequency
  - combination matters, if frequency selection affects multiple CPUs
- => we need to be able to determine what tasks run simultaneously
Mechanisms

- Task migrations
- Coordination of scheduling decisions (sort of gang scheduling)
Result

- Run memory-intensive tasks parallel to compute-intensive tasks at highest frequency
- Only lower the frequency if nothing but memory-intensive tasks are available for execution
Evaluation Sorting (Dual Core)

runtime

compute-bound memory-bound

hmmer namd libquantum mcf

- standard linux
- balance
- balance+sort
Evaluation Frequency Heuristic

- Execution of 4 x hmmer and 4 x lbm
- normalized to 2.4 GHz
Evaluation: discussion

- Improved runtime and EDP by avoiding contention
  - Reduction of EDP by reduction of runtime

- Frequency scaling only beneficial if scheduling cannot avoid contention
  - Reduction of EDP by reduction of power