Lucky scheduling for energy-efficient heterogeneous multi-core systems

Vinicius Petrucci (UFF, Brazil)
Orlando Loques (UFF, Brazil)
Daniel Mosse’ (PITT, USA)

2012 Workshop on Power-Aware Computing and Systems (HotPower '12)
Multi-core system evolution

Traditional homogenous multi-core system

Asymmetric/heterogeneous multi-core system
Heterogeneous platform

- Thread
- Thread
- Thread
  ...  
- Thread

Compute-intensive vs memory/IO-intensive threads

Single-ISA high-performance "big" vs low-power "small" types of cores
Some challenges

1. Optimized thread assignment on different core types
   – enabling individual thread scheduling within each type of cores
2. Workload characterization of threads running on the different core types (speed-up, energy efficiency, etc)
   – CPU-bound or memory-bound? Or something else altogether?
State-of-the-art *bias* scheduling

- Determine CPU or memory intensity (IPC or stalls/LLC miss rate) as a *bias* to guide thread scheduling
  - *highest* (*lowest*) bias threads scheduled on *small* (*big*) cores

- **Some issues:**
  1. inherently *unfair* thread scheduling may cause severe performance loss (big core monopoly)
  2. *single bias metric* cannot clearly characterize some threads and schedule them to the right core type
  3. unawareness of core *power usage* might allow sub-optimal energy-efficient decisions
State-of-the-art *bias* scheduling

- Determine CPU or memory intensity (IPC or stalls/LLC miss rate) as a *bias* to guide thread scheduling
  - *highest* (*lowest*) bias threads scheduled on *small* (*big*) cores

---

**Our solution:** schedule threads to big/small cores in a fair and energy-efficient way, taking into account both core execution and memory/stalls measures!
Performance-asymmetric multi-core system

Quad-core x86_64 processor

Shared cache (L3)

Big/fast core (3.2Ghz)

Small core (800Mhz)
Small core (800Mhz)
Small core (800Mhz)

Memory

<table>
<thead>
<tr>
<th>Core</th>
<th>Peak power</th>
<th>Idle power</th>
<th>Avg. power</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>big</td>
<td>18.75 W</td>
<td>9.625 W</td>
<td>15.63 W</td>
<td>6,307 MIPS</td>
</tr>
<tr>
<td>small</td>
<td>2.15 W</td>
<td>0.7 W</td>
<td>1.6 W</td>
<td>1,592 MIPS</td>
</tr>
</tbody>
</table>

Estimation of power consumption ("Web Search Using Mobile Cores" ISCA’10): Big core (Intel Xeon), Small core (Intel Atom)
Thread/core performance measurement

- **Hardware performance counters**
  - Linux *perf* measurement subsystem (kernel 2.6.34)
  - per-core/thread run-time monitoring over a fixed sampling interval (one thread per core)

- **Instruction execution rate (MIPS)**
  - counter for number of committed instructions

- **Memory access rate (LLCMS)**
  - counter for LLC (Last Level Cache) misses
  - each LLC miss represents a memory access
1) Effect of unfair scheduling decisions

Issue → Lowest bias (LLCM) threads can monopolize the few big cores available, hindering progress of other compute-intensive threads

Simple policy that provides equally fair sharing of the big core among all threads

Workload speedup of $1.47x$!
2) Thread bias/characterization (SPEC benchmark)

Some observations:
(1) Both MIPS and LLCM can be increased, such as *milc* (64M LLCM, 2K MIPS) when compared to *mcf* (18M LLCM, 0.4K MIPS)

(2) Very similar MIPS can lead to very different LLCM, such as *lbm* (48M LLCM, 2.4K MIPS) and *cactusADM* (8M LLCM, 2.3K MIPS)
3) Considering core power consumption

**OBSERVATION #1:** Some threads with similar bias measures should run energy efficiently on different core types

- bias (LLCM) \( \approx 13K \)
- bias (LLCM) \( \approx 14K \)

![Energy-efficiency (MIPS^2/Watt) on big vs small core](chart.png)
3) Considering core power consumption

**OBSERVATION #2**: *bwaves* may run on a big core type for improved energy efficiency, despite being also memory-intensive (small core bias)

bias (LLCM) \( \approx 29K \)
Our scheduling solution

• Mapping of threads to the right core type
  – given threads' computational and memory demands
  – considering individual power consumption of core types
  – providing *fairness* via lottery-based scheduling

• Optimization of both energy and performance (EDP, energy-delay product) in the multi-core system
Heterogeneous multi-core model

• N = set of core types, each core of type i in N:
  o $C_i$ = computational capacity (MIPS)
  o $B_i$ = peak/busy, $I_i$ = static/idle, power consumption (Watts)

• K = set of threads, each thread k in K running on a core of type i in N:
  o $\text{MIPS}_{i,k}$ = instruction execution rate
  o $\text{LLCMS}_{i,k}$ = memory access rate
Heterogeneous multi-core model

• Power/energy modeling (estimation)

\[
P_{i,k} = (B_i - I_i) \cdot (\frac{MIPS_{i,k}}{C_i}) + I_i
\]

\[
E = \sum_{k \in K} P_{i,k} \cdot S
\]

energy efficiency(i, k) = \frac{MIPS_{i,k}^2}{P_{i,k}} ~ Energy-Delay product

Given a scheduling interval S and assignment of thread k to core type i

But... Wait... what is the energy efficiency of a thread currently running on a given core type when assigned to run on a different core type?
Thread performance model

Performance prediction approach:

(instead of direct measurement on both core types, because of known overhead, load imbalance issues…)

1. Collect performance data from a representative set of workloads (running each thread individually on each core type) \(\rightarrow\) CPU SPEC 2006 benchmark used

2. Solve the following linear regression model:

\[
\begin{align*}
\text{MIPS}_{\text{big}} &= w_1 \times \text{MIPS}_{\text{small}} + w_2 \times \text{LLCMS}_{\text{small}} + w_3 \\
\text{MIPS}_{\text{small}} &= w_4 \times \text{MIPS}_{\text{big}} + w_5 \times \text{LLCMS}_{\text{big}} + w_6
\end{align*}
\]

(very good correlation coefficient of \(\sim 98\%\) )

Such a performance characterization needs to be done once at design stage.
Lottery scheduling

• Scheduling approach to address the fairness problem
• **Tickets**: represent the share of a resource that a thread/process should receive
• On each scheduling interval (time slice)
  – a ticket is randomly picked
  – thread with winning ticket uses the resource

**Obs. #1:** The probability of a thread winning a lottery is proportional to the number of tickets it holds

**Obs. #2:** The allocated number of tickets (share of a thread) can be adjusted dynamically to meet time-varying workloads
1. Measure $MIPS_{i,k}$ and $LLCMS_{i,k}$ of each thread $k \in K$ running on a core of type $i \in N$.

2. Predict $MIPS_{j,k}$ on the other core of type $j \in N - \{i\}$.

3. Evaluate $\text{big\_core\_benefit}(k) = \frac{\text{energy\_efficiency}(b,k)}{\text{energy\_efficiency}(s,k)}$ for each thread $k$, big core $b$, and small core $s$. 
4. Generate a number of \( tickets(k) = 100 \cdot \text{big_core_benefit}(k) \) to assign for each thread \( k \in K \)

5. **Determine** thread \( T \) that holds the \( \text{winning_ticket} \) given by a random number uniformly distributed between \([0, \text{total_tickets}]\) where \( \text{total_tickets} = \sum_{k \in K} tickets(k) \)

6. **Swap** thread \( T \) with a thread \( T' \) in case \( T \) is not running on a big core, considering that \( T' \) has the minimum number of tickets and is running on the least-loaded big core.
Implementation remarks

• Initial thread assignment given by the OS (Linux)

• Lucky scheduling binds threads to set of cores of the same type (sched_setaffinity system call)
  – underlying OS carries out thread-to-core scheduling
  – reassignment interval of 200ms (load balancing)

• Swapping threads between different core types help preserve load balancing

• Lottery-based scheduling complexity $O(#\text{threads})$
  – given #coretypes small and fixed
  – heap/tree implementation can reduce to $O(\log #\text{threads})$
Evaluation: system workload

4-thread combinations of the SPEC CPU2006 benchmark (#cores = #threads)

<table>
<thead>
<tr>
<th>Workload</th>
<th>Set of programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1CI–3MI</td>
<td>sjeng lbm milc soplex</td>
</tr>
<tr>
<td>2CI–2MI</td>
<td>bwaves tonto soplex mcf</td>
</tr>
<tr>
<td>3CI–1MI</td>
<td>povray sjeng bwaves soplex</td>
</tr>
<tr>
<td>4CI</td>
<td>calculix povray namd tonto</td>
</tr>
<tr>
<td>4MI</td>
<td>lbm milc mcf soplex</td>
</tr>
<tr>
<td>4P</td>
<td>astar bzip2 leslie3d milc</td>
</tr>
<tr>
<td>4R1</td>
<td>namd mcf astar bwaves</td>
</tr>
<tr>
<td>4R2</td>
<td>lbm bzip2 calculix GemsFDTD</td>
</tr>
</tbody>
</table>

- Linux gettimeofday() measure workload execution time (wall clock)
- Long-lived workload: threads are restarted until longest thread finishes
Preliminary results

- A simple **big core fair policy** provides EDP gains of **16% over bias scheduling**

- **Lucky** scheduling outperforms big core fair policy in EDP by **12% (avg.) and 20% (max.)**.

- **Lucky** scheduling achieved better EDP when compared to **bias scheduling** over all workloads executions (**avg. 39% and max. 51%**).
Conclusions

• **Lucky**: Proportional-share scheduling strategy for heterogeneous multi-cores
  – leverages lottery/ticket mechanisms (fairness)
  – optimizes for combined performance and energy savings

• Preliminary results show energy/performance improvements over state-of-the-art thread assignment schemes
Future directions

• Incorporate real-time performance guarantees
  – latency-sensitive apps (web search, media streaming)

• Thread/workload consolidation (few cores as possible)
  – Core on/off: energy savings vs. performance degradation (resource contention, data movements)

• Multithreaded / HPC apps
  – explore big cores to speed-up parallel execution “bottlenecks” (critical path) of applications
Thank you!

Questions?

Special thanks to Rami Melhem (PITT), Neven Abou Gazala and Sameh Gobriel (Intel Labs) for their constructive feedback on the early stage of this work.
Performance prediction analysis

Prediction error (normalized root mean square deviation) for different benchmarks

Max. prediction error of 8% (*astar*, small-to-big) and 10% (*bwaves*, big-to-small)

Average prediction error of less than 3%
Performance prediction analysis

*aStar* SPEC benchmark

*bwaves* SPEC benchmark

Performance data collected from a small core to predict the performance on a big core
Energy efficiency in big vs small cores
Energy efficiency in big vs small cores

Switch to small core
Energy efficiency in big vs small cores

Switch to BIG core
Energy efficiency in big vs small cores

Switch to small core