Asymmetry-Aware Work-Stealing Runtimes

Christopher Torng, Moyang Wang, and Christopher Batten

School of Electrical and Computer Engineering
Cornell University

43rd Int’l Symp. on Computer Architecture, June 2016
How can we use asymmetry awareness to improve the performance and energy efficiency of a work-stealing runtime?
Work-Stealing Runtimes

Task Queues

Task B

Work in Progress

Task A

Core 0

Core 1

Core 2

Core 3

Spawn Task B
Work-Stealing Runtimes

<table>
<thead>
<tr>
<th>Task Queues</th>
<th>Task Queues</th>
<th>Task Queues</th>
<th>Task Queues</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dequeue Task B

Work in Progress

<table>
<thead>
<tr>
<th>Task B</th>
<th>Task B</th>
<th>Task B</th>
<th>Task B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
<td>Core 1</td>
<td>Core 2</td>
<td>Core 3</td>
</tr>
</tbody>
</table>

Motivation
First-Order Modeling
Asymmetry-Aware Work-Stealing Runtimes
Evaluation

Work in Progress

Cornell University
Christopher Torng
Work-Stealing Runtimes

Task Queues

Core 0

Core 1

Core 2

Core 3

Task C

Work in Progress

Task B

Spawn Task C
Work-Stealing Runtimes

Task Queues

- Task C
- Task D

Work in Progress

- Task B

Core 0  Core 1  Core 2  Core 3

Spawn Task D
Work-Stealing Runtimes

Task Queues

Task B
Core 0

Task D
Core 1

Task C
Core 2

Core 3

Work in Progress

Steal Task D

Steal Task C
Work-Stealing Runtimes

Task Queues

Work in Progress

Core 0
Core 1
Core 2
Core 3

Task E
Task D
Task F
Task C

Spawn Task E
Spawn Task F
Work stealing has good performance, space requirements, and communication overheads in both theory and practice.

Supported in many popular concurrency platforms including: Intel’s Cilk Plus, Intel’s C++ TBB, Microsoft’s .NET Task Parallel Library, Java’s Fork/Join Framework, and OpenMP.
Static Asymmetry vs. Dynamic Asymmetry

Samsung Exynos Octa Mobile Processor

Little ARM Cores
A7  A7
A7  A7
L2$

Big ARM Cores
A15  A15
A15  A15
L2$

Test Chip with Four Integrated Voltage Regulators

How can we use asymmetry awareness to improve the performance and energy efficiency of a work-stealing runtime?

- **Bender et al.** "Online Scheduling of Parallel Programs on Heterogeneous Sys ..." SPAA 2002
Talk Outline

Motivation

First-Order Modeling

Asymmetry-Aware Work-Stealing Runtimes

Evaluation
Building Intuition by Exploring a 1 Big 1 Little System

Motivation
- First-Order Modeling
- Asymmetry-Aware Work-Stealing Runtimes Evaluation

Normalized Power
Normalized IPS

System with 1 big 1 little

Four-Way Big Core

(2.0, 6.0)

Little Core

(1.0, 1.0)

Power
IPS

B
L

(2.0, 6.0)

(1.0, 1.0)
Building Intuition by Exploring a 1 Big 1 Little System

Motivation
• First-Order Modeling
• Asymmetry-Aware Work-Stealing Runtimes Evaluation

Normalized Power
Normalized IPS
0.5 1.0 1.5 2.0 2.5 3.0

System with 1 big 1 little

Four-Way Big Core

Little Core

(2.0, 6.0)

(1.0, 1.0)

Little Core

Big Core

(2.0, 6.0)

(1.0, 1.0)
Building Intuition by Exploring a 1 Big 1 Little System

Motivation
- First-Order Modeling
- Asymmetry-Aware Work-Stealing Runtimes Evaluation

Building Intuition by Exploring a 1 Big 1 Little System

Normalized Power vs Normalized IPS

System with 1 big 1 little
- Big Core: (2.0, 6.0)
- Little Core: (1.0, 1.0)

10% Performance Increase
Same Power

Normalized Power
Normalized IPS
0.5 1.0 1.5 2.0 2.5 3.0
8
7
6
5
4
3
2
1
LB

Little Core

Four-Way Big Core

10% Performance Increase
The Law of Equi-Marginal Utility

British Economist
Alfred Marshall (1824 - 1924)

"Other things being equal, a consumer gets maximum satisfaction when he allocates his limited income to the purchase of different goods in such a way that the Marginal Utility derived from the last unit of money spent on each item of expenditure tend to be equal."

Balance the ratio of utility (IPS) to cost (power)
The Law of Equi-Marginal Utility

British Economist
Alfred Marshall (1824 - 1924)

"Other things being equal, a consumer gets maximum satisfaction when he allocates his limited income to the purchase of different goods in such a way that the Marginal Utility derived from the last unit of money spent on each item of expenditure tend to be equal."

Balance the ratio of utility (IPS) to cost (power)

Arbitrage
"Buy Low, Sell High"
Systematic Approach for Balancing Marginal Utility

Motivation
- First-Order Modeling
- Asymmetry-Aware Work-Stealing Runtimes Evaluation

Normalized Energy Efficiency
\[ \frac{E_{\text{normalized}}}{E_{\text{baseline}}} \]

Normalized IPS
\[ \frac{IPS_{\text{normalized}}}{IPS_{\text{baseline}}} \]

Normalized IPS
0.6 0.8 1.0 1.2 1.4

Normalized Energy Efficiency
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4

Assumptions
- 1 Big 1 Little System at Nominal voltage
- Individual \((V_B, V_L)\) pair

Perfectly parallel application
Ideal load balancing

Energy efficiency at expense of performance

Performance at expense of energy efficiency

isopower

Pareto-Optimal Frontier
Systematic Approach for Balancing Marginal Utility

Motivation
- First-Order Modeling
- Asymmetry-Aware Work-Stealing Runtimes Evaluation

Normalized Energy Efficiency vs. Normalized IPS

Pareto-Optimal Frontier

- Improve both performance and energy efficiency

Assumptions
- 1 Big 1 Little System at Nominal voltage
- Individual \((V_B, V_L)\) pair

Perfectly parallel application
Ideal load balancing
Motivation

- First-Order Modeling
- Asymmetry-Aware Work-Stealing Runtimes Evaluation

Systematic Approach for Balancing Marginal Utility

Pareto-Optimal Frontier

- 1 Big 1 Little System at Nominal voltage
- Individual (V_B, V_L) pair

Assumptions

- Perfectly parallel application
- Ideal load balancing

Marginal Utility-Based Optimization Problem

- Constraint: isopower line
- Objective: maximize performance
- Solved numerically

Normalized Energy Efficiency
Normalized IPS

0.6 0.8 1.0 1.2 1.4
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
isopower
Talk Outline

Motivation

First-Order Modeling

Asymmetry-Aware Work-Stealing Runtimes

Evaluation
Work-Pacing: Building Intuition

Balance performance/power across cores in the high-parallel (HP) region

System with both big cores active and both little cores active
Work-Pacing, Work-Sprinting, and Work-Mugging

**Work-Pacing**
Balance performance/power across cores in the high-parallel (HP) region.

**Work-Sprinting**
Rest cores in the steal loop to the lowest voltage.

With additional power slack, balance performance/power across busy cores in the low-parallel (LP) region.

**Work-Mugging**
Move work from slow little cores to fast big cores in the low-parallel (LP) region.
Work-Pacing and Work-Sprinting Mechanisms

Which cores are stealing? Big or little?

Task Queues: Big Big Little Little

Work in Progress: Task A Task B Task C

Activity Pattern:

<table>
<thead>
<tr>
<th>Task</th>
<th>Big</th>
<th>Little</th>
<th>Stealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>0.91V</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>S</td>
<td>0.98V</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>S</td>
<td>1.03V</td>
</tr>
<tr>
<td>A</td>
<td>S</td>
<td>A</td>
<td>1.04V</td>
</tr>
<tr>
<td>A</td>
<td>S</td>
<td>A</td>
<td>1.13V</td>
</tr>
<tr>
<td>A</td>
<td>S</td>
<td>S</td>
<td>1.21V</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>A</td>
<td>0.70V</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>A</td>
<td>0.70V</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>0.70V</td>
</tr>
</tbody>
</table>

Activity Pattern:

<table>
<thead>
<tr>
<th>Task</th>
<th>Big</th>
<th>Little</th>
<th>Stealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>1.30V</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>S</td>
<td>1.30V</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>S</td>
<td>1.30V</td>
</tr>
<tr>
<td>A</td>
<td>S</td>
<td>A</td>
<td>1.30V</td>
</tr>
<tr>
<td>A</td>
<td>S</td>
<td>A</td>
<td>1.30V</td>
</tr>
<tr>
<td>A</td>
<td>S</td>
<td>S</td>
<td>0.70V</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>A</td>
<td>0.70V</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>A</td>
<td>0.70V</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>0.70V</td>
</tr>
</tbody>
</table>

Vdd = Active, S = Stealing

On-Chip Interconnect

L1$ L1$
B L
Voltage
Regulators

DVFS Controller

Hints

Shared L2$ Cache Banks

DRAM Memory Controller

A = Active, S = Stealing
Work-Mugging Mechanisms

Mug Instruction
- Thread ID to mug
- Address of thread-swapping handler

User-Level Interrupt Network
- Simple, low-bandwidth inter-core network
- Latency on order of 20 cycles

Thread Context Swap
- Threads store architectural state to separate locations in shared memory
- Both threads sync
- Threads load architectural state from other location
Talk Outline

Motivation
First-Order Modeling
Asymmetry-Aware Work-Stealing Runtimes
Evaluation
Evaluation Methodology: Modeling

Work-Stealing Runtime
- State-of-the-art Intel TBB-inspired work-stealing scheduler
- Chase-Lev task queues with occupancy-based victim selection
- Instrumented with activity hints

Cycle-Level Modeling
- Heterogeneous system modeled in gem5 cycle-approximate simulator
- Support for scaling per-core frequencies + central DVFS Controller

Energy Modeling
- Event-based energy modeling based on detailed RTL/gate-level sims (Synopsys ASIC toolflow, TSMC LP, 65 nm 1.0 V)
- Carefully selected subset of McPAT results tuned to our μarchitecture
Work-Pacing in *cilk-sort*

**No AAWS Techniques**

<table>
<thead>
<tr>
<th>Little</th>
<th>Big</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Activity Bar" /></td>
<td><img src="image" alt="Activity Bar" /></td>
</tr>
</tbody>
</table>

**DVFS Controller Decision**

- 1.30 V
- 1.24 V
- 1.04 V
- 0.90 V
- 0.70 V

**Graph**

- **Speedup 1.11x**
- **Energy Efficiency 1.11x**

**Evaluation**

- Work-Pacing
- Speedup of 1.11x
- Normalized Energy Efficiency
- Isopower
Motivation
First-Order Modeling
Asymmetry-Aware Work-Stealing Runtimes

Evaluation

Work-Sprinting in *quicksort*

**No AAWS Techniques**

**Activity Bar**
- Busy
- Steal Loop

**DVFS Controller Decision**
- 1.30 V
- 1.24 V
- 1.00 V
- 1.10 V
- 0.70 V

**Graph**
- Normalized Energy Efficiency
- Performance
- Speedup 1.34x
- Energy Efficiency 1.16x
Work-Mugging in *radix sort*

**Evaluation**

- No AAWS Techniques
- Work-Mugging

**Activity Bar**
- Busy
- Steal Loop

**DVFS Controller Decision**
- 1.30 V
- 1.00 V
- 1.24 V
- 0.90 V
- 1.04 V
- 0.70 V

**Graph**
- Speedup 1.17x
- Energy Efficiency 1.40x

**Legend**
- Speedup 1.17x
- Energy Efficiency 1.40x
Evaluation of Complete AAWS Runtime

Application Kernels
- pbbs-bfs
- pbbs-quicksort
- pbbs-samplesort
- pbbs-dictionary
- pbbs-convex-hull
- pbbs-radix-sort
- pbbs-knn
- pbbs-max-independent-set
- pbbs-nbody
- pbbs-remove_duplicates
- pbbs-suffix-array
- pbbs-spanning-tree
- cilk-cholesky
- cilk-cilksort
- cilk-heat
- cilk-knapsack
- cilk-matrix-multiply
- parsec-blackscholes
- unbalanced-tree-search

Performance
- Median: 1.10 x
- Max: 1.32 x

Energy Efficiency
- Median: 1.11 x
- Max: 1.53 x
Take-Away Point

Holistically combining

• work-stealing runtimes
• static asymmetry
• dynamic asymmetry

through the use of

• work-pacing
• work-sprinting
• work-mugging

can improve both performance and energy efficiency in future multicore systems

This work was partially supported by the NSF, AFOSR, and donations from Intel and Synopsys