Application-Level Energy Accounting with Chappie

Timur Babakol, Anthony Canino, Khaled Mahmoud, Rachit Saxena, and Yu David Liu
Binghamton University,
State University of New York (SUNY),
{tabako1,acanino1,kmahou1,rsaxena3,davidl}@binghamton.edu
Application-Level Energy Management

JVM

Application (sunflow)

render()
Application-Level Energy Management

JVM

Application (sunflow)

render()
Application-Level Energy Management

JVM

Application (sunflow)

render()

Post Processing: 10

Consumes 10j
Application-Level Energy Management

JVM

Application (sunflow)

render()

Post Processing: 10

Consumes 10j

Is this accurate?
Problem #1: Multi-threading
Problem #2: Co-Running

- **render**
- **refresh**
- **download**
- **OS System Daemon**

Time

- **application**
- **system process**
- **thread**
Problem #3: Overhead

<table>
<thead>
<tr>
<th></th>
<th>Instrumentation 100%</th>
<th>Instrumentation 5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunflow</td>
<td>189x</td>
<td>164x</td>
</tr>
<tr>
<td>batik</td>
<td>118x</td>
<td>117x</td>
</tr>
<tr>
<td>avrora</td>
<td>33x</td>
<td>30x</td>
</tr>
<tr>
<td>xalan</td>
<td>30x</td>
<td>29x</td>
</tr>
<tr>
<td>h2</td>
<td>25x</td>
<td>16x</td>
</tr>
</tbody>
</table>
What is Application-Level Accounting?

Accounting Framework

- Power Domain Accounting
- Thread Accounting
- Package Accounting
- Class Accounting
- Method Accounting
- Context-Sensitive Method Accounting

JVM

App

JVM

App

OS
Fine-grained Energy Accounting of Java Applications
This Talk: **Chappie**

- A *cross-layer, concurrency-aware* design for fine-grained energy accounting of multi-threaded Java applications
- A low-overhead *sampling*-based implementation
- An evaluation on 20 benchmark applications analyzing *per-method, per-thread, and per-application* energy accounting
This Talk: **Chappie**

- A *cross-layer, concurrency-aware* design for fine-grained energy accounting of multi-threaded java applications
- A *low-overhead sampling-based* implementation
- An evaluation on 20 benchmark applications analyzing per-method, per-thread, and per-application energy accounting
Cross Layer Design

Accounting Framework

- JVM
- App
- JVM
- App
- OS
- Architecture

- Power Domain Accounting
- Thread Accounting
- Package Accounting
- Class Accounting
- Method Accounting
- Context-Sensitive Method Accounting
Cross Layer Design

Chappie Monitor

Chappie Attributor

JVM
App

JVM
App

OS
Architecture

Power Domain Accounting
Thread Accounting
Package Accounting
Class Accounting
Method Accounting
Context-Sensitive Method Accounting
Cross Layer Design

Java VM
  Thread
  Activeness
  Method Call
  Trace

JVM

App

OS

Architecture

Chappie Monitor

Chappie Attributor

Power Domain Accounting
  Thread Accounting
  Package Accounting
  Class Accounting
  Method Accounting
  Context-Sensitive
  Method Accounting
Cross Layer Design

Provide insight into application behavior
Cross Layer Design

- Java VM
  - Thread
  - Activeness
  - Method Call
  - Trace

- JVM

- App

- OS
  - Thread
  - Affinity

- Architecture

Chappie Monitor

- Power Domain Accounting
- Thread Accounting
- Package Accounting
- Class Accounting
- Method Accounting
- Context-Sensitive Method Accounting

Chappie Attributor
Cross Layer Design

Chappie Monitor

Chappie Attributor

Tracking which domain thread runs on
Cross Layer Design

Java VM
- Thread
- Activeness
- Method Call
- Trace

JVM

App

OS
- Thread
- Affinity

Architecture
- RAPL
- RAPL

Chappie Monitor

Chappie Attributor

Power Domain Accounting

Thread Accounting

Package Accounting

Class Accounting

Method Accounting

Context-Sensitive Method Accounting
Cross Layer Design

Chappie Monitor

Chappie Attributor

Finer-grained energy measurement

Java VM
- Thread
- Activeness
- Method Call
- Trace

JVM

App

OS
- Thread
- Affinity

Architecture
- RAPL

Power Domain Accounting

Thread Accounting

Package Accounting

Class Accounting

Method Accounting

Context-Sensitive
Method Accounting
Cross Layer Design

Chappie Monitor

Chappie Attributor

Power Domain Accounting
Thread Accounting
Package Accounting
Class Accounting
Method Accounting
Context-Sensitive Method Accounting
Cross Layer Design

Provide total system usage picture

Chappie Monitor

Chappie Attributor

Java VM
- Thread Activeness
- Method Call Trace

Java VM
- Thread Activeness
- Method Call Trace

OS
- Thread Affinity
- Process Jiffies

Architecture
- RAPL
- RAPL

Power Domain Accounting
- Thread Accounting
- Package Accounting
- Class Accounting
- Method Accounting
- Context-Sensitive Method Accounting
This Talk: Chappie

- A cross-layer, concurrency aware design for fine-grained energy accounting of multi-threaded java applications
- A low-overhead sampling-based implementation
- An evaluation on 20 benchmark applications analyzing per-method, per-thread, and per-application energy attribution
Addressing Problem #1: Multi-threading
Concurrency-Aware Attribution

Chappie Runtime

A1

A2

E

JVM

Application

Thread 1

Thread 2

OS

RAPL
Concurrence-Aware Attribution

Chappie Runtime

JVM

Application

Thread 1

Thread 2

OS

RAPL

Query Activeness

At timestamp: 1
Concurrency-Aware Attribution

At timestamp: 1

Record Activeness sample
Concurrent-Aware Attribution

At timestamp: 1
Concurrenty-Aware Attribution

At timestamp: 1

Record energy for timestamp
At timestamp: 2

Thread 2 becomes inactive
Concurency-Aware Attribution

At timestamp: 2

Record activeness and energy for timestamp

Chappie Runtime

A1
A2
E

10 5

JVM
Application
Thread 1
Thread 2

OS
RAPL
Concurrency-Aware Attribution

At timestamp: 5
Concurrency-Aware Attribution

Attribute 5j to each Thread at T1

At timestamp: 5
Concurreny-Aware Attribution

Attribute 5j to each Thread 1 at T2

At timestamp: 5
Concurrency-Aware Attribution

Attribute no joules to either Thread at T3

At timestamp: 5
Concurreny-Aware Attribution

Thread 1: Attributed = 20j
Thread 2: Attributed = 10j

At timestamp: 5
Concurrenty-Aware Attribution

Thread 1: Attributed = 20j
Thread 2: Attributed = 10j

Thread 1: Unattributed = 17.5
Thread 2: Unattributed = 17.5

At timestamp: 5
Aligning Attribution to Methods

At timestamp: 1

Sample for current executing method
Aligning Attribution to Methods

Chappie Runtime

A1
A2
E

10 5 5 5 10

Thread 1 :: foo()

foo : 5 j

foo gets executing thread attributed energy

At timestamp: 1

Thread 1
Thread 2
Aligning Attribution to Methods

Chappie Runtime

A1

A2

E

10 5 5 5 10

Thread 1 :: bar()

At timestamp: 2

JVM

Application

Thread 1

Thread 2

foo : 5 j

bar : 5 j
Aligning Attribution to Methods

At timestamp: 5

Thread 2 :: foo()

foo : 10 j
bar : 5 j
Addressing Problem #2: Co-Running Applications

- foo
- bar
- foo
- bar
- OS System Daemon

Time

- application
- system process
- thread
Co-Running Attribution

Chappie Runtime

JVM
Application 1
Thread 1
Thread 2

JVM
Application 2
Thread 3
Thread 4

OS
RAPL
TID 1
TID 2
TID 3
TID 4
Co-Running Attribution

Query Application Jiffies
Co-Running Attribution

<table>
<thead>
<tr>
<th>Chappie Runtime</th>
<th>JVM</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>Application 1</td>
<td>RAPL</td>
</tr>
<tr>
<td>J1</td>
<td>Thread 1</td>
<td>TID 1</td>
</tr>
<tr>
<td>J1</td>
<td>Thread 2</td>
<td>TID 2</td>
</tr>
<tr>
<td>J2</td>
<td>Application 2</td>
<td>TID 3</td>
</tr>
<tr>
<td>J2</td>
<td>Thread 3</td>
<td>TID 4</td>
</tr>
<tr>
<td>J2</td>
<td>Thread 4</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Application 1</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Thread 1</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Thread 2</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Application 2</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Thread 3</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Thread 4</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Application 1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Thread 1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Thread 2</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Application 2</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Thread 3</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Thread 4</td>
<td></td>
</tr>
</tbody>
</table>

Numbers in boxes represent values or statistics related to the components.
Co-Running Attribution

Application 1: \( \frac{20}{50} \times 20j \)
Application 2: \( \frac{30}{50} \times 20j \)
Co-Running Attribution

Application 1: 8j
Application 2: 12j
Putting Two Tiers Together

- Higher rate intra-application accounting
- Lower sampling rate for inter-application accounting
This Talk: **Chappie**

- A cross-layer, concurrency aware design for fine-grained energy accounting of multi-threaded Java applications
- A low-overhead sampling-based implementation
- An evaluation on 20 benchmark applications analyzing per-method, per-thread, and per-application energy attribution
Experimental Setup

- Experiments performed on dual socket Intel E5-2623 2.20 GHz CPU, 10 cores per socket, 64 DDR RAM, Debian 4.9 OS, default linux power governor
- VM Sampling run at 4ms, OS Samping at 40ms
- Benchmarks from Dacapo, Graphchi, and OLTPBench
- Experiments run 20 times for Dacapo, 10 times for Graphchi and OLTPBench
Method Accounting

Top 10 consuming methods for twitter

- FloatingDecimal$BinaryToASCIIBuffer.dtoa
- ReadAheadInputStream.read
- SingleByteCharsetConverter.<clinit>
- Results.writeAllCSVAbsoluteTiming
- MysqlIO.nextRowFast
- BenchmarkState.getState
- ThreadBench.runRateLimitedMultiPhase
- UTF_8$Encoder.encodeArrayLoop
- NonRegisteringDriver.connect
- Worker.run

Normalized Energy/Time Consumed Per Method

**twitter**
Method Accounting

Top 10 consuming methods for twitter

Important for energy debugging and program understanding
Method Accounting

- FloatingDecimal$BinaryToAsciiBuffer.dtoa
- ReadAheadInputStream.read
- SingleByteCharsetConverter.<clinit>
- Results.writeAllCSVAbsoluteTiming
- MysqlIO.nextRowFast
- BenchmarkState.getState
- ThreadBench.runRateLimitedMultiPhase
- UTF_8$Encoder.encodeArrayLoop
- NonRegisteringDriver.connect
- Worker.run

Normalized Energy/Time Consumed Per Method

Difference between energy and time
Method Accounting

Normalized Energy/Time Consumed Per Method

- FloatingDecimal$BinaryToASCIIBuffer.dtoa
- ReadAheadInputStream.read
- SingleByteCharsetConverter.<clinit>
- Results.writeAllCSVAbsoluteTiming
- MysqlIO.nextRowFast
- BenchmarkState.getState
- ThreadBench.runRateLimitedMultiPhase
- UTF_8$Encoder.encodeArrayLoop
- NonRegisteringDriver.connect
- Worker.run

Difference between energy and time

Important for fine-grained "logical" power analysis
Method Accounting

Top 10 consuming methods for h2

- BaseIndex.compareRows
- Row.getValue
- ScanIndex.getNextRow
- BaseIndex.compareValues
- Select.queryFlat
- MultiVersionCursor.next
- TPCC.calculateSumDB
- Comparison.getValue
- TreeIndex.next
- JdbcConnection.checkClosed

Normalized Energy/Time Consumed Per Method

h2
Method Accounting

Top 10 consuming methods for h2

Known optimization for h2

Normalized Energy/Time Consumed Per Method

- BaseIndex.compareRows
- Row.getValue
- ScanIndex getNextRow
- BaseIndex.compareValues
- Select.queryFlat
- MultiVersionCursor.next
- TPCC.calculateSumDB
- Comparison.getValue
- TreeIndex.next
- JdbcConnection.checkClosed

Energy
Time
Class Accounting

Top 10 consuming classes for h2
Context-Sensitive Method Accounting

BaseIndex.compareRows
- [compareRows,next]
- [compareRows,compare]
- [removeIfExists,add]
- [next,step]
- [next,removeIfExists]
- [add,add]
- [add,remove]
- [findFirstNode,find]
- [findFirstNode,remove]

Row.getValue
- [compareRows,compareRows]
- [compareRows,next]
- [compareRows,add]
- [compareRows,findFirstNode]
- [getSearchRow,find]
- [validateConvertUpdateSequence,insertRows]
- [validateConvertUpdateSequence,update]
- [getValue,getValue]
- [update,update]

Percent Energy Per Calling Context for Method

2-CFA Context

h2
Analyzing Co-Running Applications
Analyzing Co-Running Applications

1. compareRows
2. getValue
3. getNextRow
4. compareValues
Analyzing Co-Running Applications

1. compareRows
2. getValue
3. getNextRow
4. compareValues

1. compareRows
2. getValue
3. compareValues
4. getNextRow
Analyzing Co-Running Applications

1. `compareRows`
2. `getValue`
3. `getNextRow`
4. `compareValues`

Pearson Correlation Coefficient (PCC)

PCC > 0.7 indicates strong correlation

1. `compareRows`
2. `getValue`
3. `compareValues`
4. `getNextRow`
Analyzing Co-Running Applications

1. compareRows
2. getValue
3. getNextRow
4. compareValues

Pearson Correlation Coefficient (PCC)

PCC > 0.7 indicates strong correlation

Energy Accounting Isolation

JVM
h2
Thread 1
Thread 2

JVM
jython
Thread 1
Thread 2

JVM
h2
Thread 1
Thread 2

1. compareRows
2. getValue
3. compareValues
4. getNextRow
Co-Running Applications

Each pair represent co-running apps

Energy (J)

<table>
<thead>
<tr>
<th>App A</th>
<th>App B</th>
</tr>
</thead>
<tbody>
<tr>
<td>.8690</td>
<td>.9093</td>
</tr>
<tr>
<td>.9798</td>
<td>.9093</td>
</tr>
<tr>
<td>.9999</td>
<td>.9093</td>
</tr>
<tr>
<td>.9897</td>
<td>.9093</td>
</tr>
<tr>
<td>.9694</td>
<td>.9093</td>
</tr>
<tr>
<td>.9890</td>
<td>.9093</td>
</tr>
<tr>
<td>.8797</td>
<td>.9093</td>
</tr>
<tr>
<td>.8898</td>
<td>.9093</td>
</tr>
<tr>
<td>.9896</td>
<td>.9093</td>
</tr>
<tr>
<td>.9896</td>
<td>.9093</td>
</tr>
<tr>
<td>.9999</td>
<td>.9093</td>
</tr>
<tr>
<td>.9696</td>
<td>.9093</td>
</tr>
</tbody>
</table>
Co-Running Applications

Each pair represent co-running apps

Energy consistent across loads
Co-Running Applications

Benchmarks

PCC shows method ranking stable

Each pair represent co-running apps
Application with Foreign Load

Benchmark run with PARSEC Ferret
Application with Foreign Load

Benchmark run with PARSEC Ferret

Energy consistent, PCC stable
# Chappie Overhead

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Total Threads</th>
<th>Time Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>avrora</td>
<td>11</td>
<td>4.6%</td>
</tr>
<tr>
<td>batik</td>
<td>12</td>
<td>12.3%</td>
</tr>
<tr>
<td>eclipse</td>
<td>23</td>
<td>8.8%</td>
</tr>
<tr>
<td>h2</td>
<td>46</td>
<td>4.0%</td>
</tr>
<tr>
<td>GraphChi-ALS</td>
<td>47</td>
<td>15.1%</td>
</tr>
<tr>
<td>Twitter</td>
<td>17</td>
<td>0.4%</td>
</tr>
</tbody>
</table>
Application-Level Energy Accounting

Benchmarks

Energy consumed per app as attributed by chapple
Energy consumed per app as read by RAPL
Summary

Application-Level Energy Accounting

Cross Layer Design

Energy consumed per app as attributed by chappie

Energy consumed per app as read by RAPL
Summary

Application-Level Energy Accounting

![Graph showing energy consumption per app]

Cross Layer Design

![Diagram illustrating cross-layer design]

Co-Running Attribution

![Table showing co-running attribution]

Application 1: 20/50 * 20j
Application 2: 30/50 * 20j
Summary

Application-Level Energy Accounting

Cross Layer Design

Co-Running Attribution

Co-Running Applications
Questions?
System Accounting

Energy Attributed Per-Domain

OS schedules threads consistently across sockets
Related Work

- **Hardware and OS Energy Accounting**
  - iCount (IPSN '08)

- **Application-level Profiling and Energy Management**
  - JouleTracke (DAC '11)
  - PowerScope (WMCSA '99)

- **Energy Analysis**
  - Tiwari (VLSI '96)
  - Greec (SCOPES '15)
  - Jayaseelen (RTAS '06)
  - Hao (GREENS '12)

- **Runtime-Centric or Cross-Layer Approaches**
  - Krishnan (Perf. Eval. Rev. ‘11)
  - Bertran (Future Generation Computer System ‘12)
  - JouleGuard (SOSP '15)
  - ARO (MobiSys ‘11)