A Refactoring Approach to Improve Energy Consumption of Parallel Software Systems

Gustavo Pinto

Ph.D. Defense
Informatics Center
Federal University of Pernambuco

Recife, February/2015
Motivation (1/2)

• First, energy consumption is a concern for unwired devices and also for data centers.

• Second, there is a large body of work in hardware/architecture, OS, runtime systems.

• However, little is known about the application level.
Motivation (2/2)

• First, **multicore** CPUs are ubiquitous

• Second, performance of the existing parallel software is reasonably well-understood

• However, little is known about **energy behaviors** of multi-threaded programs on the **application and programming language** level
CP: Concurrent Programming
CP: Concurrent Programming

* Under submission
CP: Concurrent Programming
SEC: Software Energy Consumption

* Under submission
CP: Concurrent Programming
SEC: Software Energy Consumption

* Under submission
CP: Concurrent Programming
SEC: Software Energy Consumption
MSR: Mining Software Repositories

* Under submission
** Will be submitted
CP: Concurrent Programming
SEC: Software Energy Consumption
MSR: Mining Software Repositories

[FASE’15]
[JSS’15b]
[MSR’14]
[GREENS’15]

[OOPSLA’14]
[SEPS’14]
[TMC’11]
[JSS’15a]*
[OOPSLA’15]**
[MSR’15]*

** Will be submitted
* Under submission
This thesis!

** Will be submitted

* Under submission

CP: Concurrent Programming
SEC: Software Energy Consumption
MSR: Mining Software Repositories

[OOPSLA’14]
[SEPS’14]
[TMC’11]
[JSS’15a]*
[OSS’14]
[WRT’13]
[MSR’15]*
[FASE’15]
[JSS’15b]
[OOPSLA’15]**
[GREENS’15]
[MSR’14]
The Problem

• The lack of knowledge
• The lack of tools
The Problem

- The lack of knowledge
- The lack of tools

I have no idea on how to improve this parallel code to be more energy efficient :(
The Problem

- The lack of knowledge
- The lack of tools

Is there any tool that can help us to refactor our system to consume less energy?
The Contributions

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of

   1. Thread-safe collections

   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
The Contributions

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of
   1. Thread-safe collections
   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
2M+ Users

5M+ Questions

10M+ Answers

50GB+ of data

“The most used Q&A website in the world”
2M+ Users

5M+ Questions

10M+ Answers

50GB+ of data

“The most used Q&A website in the world”
5M Questions → Automatic Filter → Manual Filter → Final Data
5M Questions → Automatic Filter

615 Questions
1,197 Answers → Manual Filter

Final Data
5M Questions

Automatic Filter

Manual Filter

325 Questions
558 Answers

Base Group
from 2008 to 2013

Final Data
Characteristics
Characteristics

No obvious “energy expert”

85% of Q. have A. (45% are answered successfully)

1/4 of questions are from mobile dev.
Problems

- Measurements (59/97 — Q/A)
- General Knowledge (40/84 — Q/A)
- Code design (36/133 — Q/A)
- Context-specific (83/110 — Q/A)
- Noise (107/134 — Q/A)
“I want to measure the energy consumption of my own application (which I can modify) [...] on Windows CE 5.0 and Windows Mobile 5/6. Is there some kind of API for this?”

• Measurements (59/97 — Q/A)

• General Knowledge (40/84 — Q/A)

• Code design (36/133 — Q/A)

• Context-specific (83/110 — Q/A)

• Noise (107/134 — Q/A)
“Are there any s/w high level design considerations [...] to make the code as power efficient as possible?”

- Measurements (59/97 — Q/A)
- General Knowledge (40/84 — Q/A)
- Code design (36/133 — Q/A)
- Context-specific (83/110 — Q/A)
- Noise (107/134 — Q/A)
Problems

• Measurements (59/97 — Q/A)
• General Knowledge (40/84 — Q/A)
• Code design (36/133 — Q/A)

• Context-specific (83/110 — Q/A)
• Noise (107/134 — Q/A)

— Highest popularity
— Highest A per Q ratio
— Highest success rate
Causes

- Unnecessary resource usage (49 occurrences)
- Fault GPS behavior (42 occurrences)
- Background activities (40 occurrences)
- Excessive synchronization (32 occurrences)
- Background wallpapers (17 occurrences)
- Advertisement (11 occurrences)
“to have a background application that monitors device usage, identifies unused/idle resources, and acts appropriately”

- **Unnecessary resource usage** (49 occurrences)
- **Fault GPS behavior** (42 occurrences)
- **Background activities** (40 occurrences)

- **Excessive synchronization** (32 occurrences)
- **Background wallpapers** (17 occurrences)
- **Advertisement** (11 occurrences)
“When there are bugs that keep the GPS turned on too long they go to the top of the list to get fixed”

- Unnecessary resource usage (49 occurrences)
- Fault GPS behavior (42 occurrences)
- Background activities (40 occurrences)
- Excessive synchronization (32 occurrences)
- Background wallpapers (17 occurrences)
- Advertisement (11 occurrences)
Solutions

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)
- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)
“do not flood the output stream with null values”

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)
- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)
“Don’t transfer say 1 file, and then wait for a bit to do another transfer. Instead, transfer right after the other.”

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)
- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)
Do researchers agree?

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)
- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)
Do researchers agree?

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)

- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)
The Goal

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of

   1. Thread-safe collections
   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
The Goal

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of
   1. Thread-safe collections
   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
# 16 Collections

<table>
<thead>
<tr>
<th>List</th>
<th>Set</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList</td>
<td>LinkedHashSet</td>
<td>LinkedHashMap</td>
</tr>
<tr>
<td>Vector</td>
<td>Collections.syncSet()</td>
<td>Hashtable</td>
</tr>
<tr>
<td>Collections.syncList()</td>
<td>CopyOnWriteArraySet</td>
<td>Collections.syncMap()</td>
</tr>
<tr>
<td>CopyOnWriteArrayList</td>
<td>ConcurrentSkipListSet</td>
<td>ConcurrentSkipListMap</td>
</tr>
<tr>
<td></td>
<td>ConcurrentHashSet</td>
<td>ConcurrentHashSet</td>
</tr>
<tr>
<td></td>
<td>ConcurrentHashSetV8</td>
<td>ConcurrentHashMapV8</td>
</tr>
</tbody>
</table>
16 Collections

<table>
<thead>
<tr>
<th>List</th>
<th>Set</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList</td>
<td>LinkedHashMap</td>
<td>LinkedHashMap</td>
</tr>
<tr>
<td>Vector</td>
<td>LinkedHashSet</td>
<td>Hashtable</td>
</tr>
<tr>
<td>Collections.syncList()</td>
<td>Collections.syncSet()</td>
<td>Collections.syncMap()</td>
</tr>
<tr>
<td>CopyOnWriteArrayList</td>
<td>CopyOnWriteArraySet</td>
<td>ConcurrentSkipListMap</td>
</tr>
<tr>
<td></td>
<td>ConcurrentSkipListSet</td>
<td>ConcurrentHashSet</td>
</tr>
<tr>
<td></td>
<td>ConcurrentHashSetV8</td>
<td>ConcurrentHashMap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConcurrentHashMapV8</td>
</tr>
</tbody>
</table>

Non thread-safe

Thread-safe
# 16 Collections

<table>
<thead>
<tr>
<th>List</th>
<th>Set</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList</td>
<td>LinkedList</td>
<td>LinkedHashMap</td>
</tr>
<tr>
<td>Vector</td>
<td>Collections.syncSet()</td>
<td>Hashtable</td>
</tr>
<tr>
<td>Collections.syncList()</td>
<td>CopyOnWriteArraySet</td>
<td>Collections.syncMap()</td>
</tr>
<tr>
<td>CopyOnWriteArrayList</td>
<td>ConcurrentSkipListSet</td>
<td>ConcurrentHashMap</td>
</tr>
<tr>
<td></td>
<td>ConcurrentHashSet</td>
<td>ConcurrentHashMapV8</td>
</tr>
<tr>
<td></td>
<td>ConcurrentHashSetV8</td>
<td></td>
</tr>
</tbody>
</table>

## x 3 Operations

<table>
<thead>
<tr>
<th>Traversal</th>
<th>Insertion</th>
<th>Removal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental Environment

System#1: A 2×16-core AMD CPUs, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

System#2: A 2×8-core (32-cores w/ hyper-threading) Intel CPU, running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.
Experimental Environment

System#1: A 2×16-core AMD CPUs, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

System#2: A 2×8-core (32-cores w/ hyper-threading) Intel CPU, running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.
Experimental Environment

System#1: A 2 × 16-core AMD CPUs, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.
Experimental Environment

A 2×16-core AMD CPUs, running Debian Linux, 64GB of DDR3 1600 memory, and JDK version 1.7.0 11, build 21.
Experimental Environment

System#2: A 2×8-core (32-cores w/ hyper-threading) **Intel CPU**, running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0_71, build 14.

**jRAPL – A framework for profiling energy consumption of Java programs**

**What is jRAPL?**

jRAPL is a framework for profiling Java programs running on CPUs with Running Average Power Limit (RAPL) support.

**But, what is RAPL?**

RAPL is a set of low-level interfaces with the ability to monitor, control, and get notifications of energy and power consumption. Originally designed by Intel for enabling chip-level power management, RAPL is widely supported in today’s Intel architectures, including popular i5 and i7.

Lists

Traversal

Insertion

Removal
Lists
Lists
Lists

**COW: -46x**

**COW: +152x**
Maps
Maps
Maps
Do Maps Scale?

System#2

Energy

Time

Traversal

Insertion

Removal
Do Maps Scale?
Do Maps Scale?

GAP

GAP

GAP
Do Maps Scale?

If you are in doubt, go for CHMV8!
The Goal

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of
   1. Thread-safe collections
   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
The Goal

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of
   1. Thread-safe collections
   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
Thread management constructs

• **Explicit threading (the Thread-style):** Using the `java.lang.Thread` class

• **Thread pooling (the Executor-style):** Using the `java.util.concurrent.Executor*` framework

• **Working Stealing (the ForkJoin-style):** Using the `java.util.concurrent.ForkJoin*` framework
Benchmarks

• **Embarrassingly parallel**: spectralnorm, sunflow, n-queens

• **Leaning parallel**: xalan, knucleotide, tomcat

• **Leaning serial**: mandelbrot, largestImage

• **Embarrassingly serial**: h2
Benchmarks

- **Embarrassingly parallel:** spectralnorm, sunflow, n-queens
- **Leaning parallel:** xalan, knucleotide, tomcat
- **Leaning serial:** mandelbrot, largestImage
- **Embarrassingly serial:** h2

Micro-benchmarks

DaCapo benchmarks
Experimental Environment

A 2×16-core **AMD CPUs**, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.
Energy Consumption When Varying the Number of Threads

![Graphs showing energy consumption for Sunflow, Mandelbrot, Xalan, and H2 with varying number of threads.](image)
The Λ Curve
The Λ Curve
The Λ Curve
The \( \Lambda \) Curve

- More cores idle
- Frequency at a lower level
The Λ Curve

- More cores idle
- Frequency at a lower level
- More threads used, performance increase
- The greater the ratio between speedup and power, the steeper the \[ Λ \] curve
Which programming style should I use?
Overpopulating Cores with Threads

**Sunflow**

**Mandelbrot**

**Xalan**

**H2**
Overpopulating Cores with Threads

- Sunflow
- Mandelbrot
- Xalan
- H2

Graphs showing energy consumption with different numbers of threads (64, 128) for each application.
Faster ≠ Greener
Data Locality

public void compute() {
    NQueensSolver[] tasks = new NQueensSolver[size];
    for (int i = 0; i < tasks.length; i++) {
        int[] newElements = new int[depth + 1];
        System.arraycopy(currentElements, 0, newElements, 0, depth);
        tasks[i] = new NQueensSolver(newElements);
        tasks[i].fork();
    }
    for (int i = 0; i < tasks.length; i++) {
        if (tasks[i] != null) tasks[i].join();
    }
}

public void compute() {
    List<NQueensSolver> tasks = new ArrayList<>(size);
    for (int i = 0; i < tasks.size(); i++) {
        int[] newElements = new int[depth + 1];
        System.arraycopy(currentElements, 0, newElements, 0, depth);
        tasks.add(new NQueensSolver(newElements));
    }
    invokeAll(tasks);
}
Data Locality

Copy

Fork/Join

$\pm 10\%$ of energy savings!
Copying vs Sharing

```java
import static Arrays.*;
class Task extends RecursiveAction{
    public Task (User[] u) {}  
    protected void compute() {  
        if (u.length < N) { local(u); }
        else {
            int split = u.length / 2;

            User[] u1 = copyOfRange(u, 0, split);
            User[] u2 = copyOfRange(u, split, u.length);

            invokeAll(new Task(u1),
                      new Task(u2));
        }
    }
}
```

```java
class Task extends RecursiveAction{
    public Task (User[] u, int from, int to) {}  
    protected void compute() {
        if (to - from < N)
            local(u, from, to);
        else {
            int split = (from + to) / 2;

            invokeAll(  
                new Task(u, from, split),
                new Task(u, from + split, to)
            );
        }
    }
}
```
Copying vs Sharing

```java
import static Arrays.*;
class Task extends RecursiveAction{
    public Task (User[] u) {}
    protected void compute() {
        if (u.length < N) { local(u); }
        else {
            int split = u.length / 2;

            User[] u1 = copyOfRange(u, 0, split);
            User[] u2 = copyOfRange(u, split, u.length);

            invokeAll(new Task(u1),
                    new Task(u2));
        }
    }
}
```

```java
class Task extends RecursiveAction{
    public Task (User[] u, int from, int to) {}
    protected void compute() {
        if (to - from < N)
            local(u, from, to);
        else {
            int split = (from + to) / 2;

            invokeAll(new Task(u, from, split),
                      new Task(u, from + split, to));
        }
    }
}
```

±15% of energy savings!
Copying vs Sharing

Copying

Sharing

±15% of energy savings!
The Goal

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of

   1. Thread-safe collections

   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
The Goal

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of
   1. Thread-safe collections
   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
Copying vs Sharing

```java
import static Arrays.*;

class Task extends RecursiveAction{
    public Task (User[] u) {}  
    protected void compute() {
        if (u.length < N) { local(u); }
        else {
            int split = u.length / 2;
            User[] u1 = Arrays.copyOfRange(u, 0, split);
            User[] u2 = Arrays.copyOfRange(u, split, u.length);
            invokeAll(new Task(u1), new Task(u2));
        }
    }
}
```

```java
class Task extends RecursiveAction{
    public Task (User[] u, int from, int to) {}  
    protected void compute() {
        if (to - from < N)
            local(u, from, to);
        else {
            int split = (from + to) / 2;
            invokeAll(new Task(u, from, split), new Task(u, from + split, to));
        }
    }
}
```
Copying vs Sharing

import static Arrays.*;
class Task extends RecursiveAction{
    public Task (User[] u) {}
    protected void compute() {
        if (u.length < N) { local(u); }
        else {
            int split = u.length / 2;
            User[] u1 = copyOfRange(u, 0, split);
            User[] u2 = copyOfRange(u, split, u.length);
            invokeAll(new Task(u1),
                      new Task(u2));
        }
    }
}

class Task extends RecursiveAction{
    public Task (User[] u, int from, int to) {}
    protected void compute() {
        if (to - from < N)
            local(u, from, to);
        else {
            int split = (from + to) / 2;
            invokeAll(
                        new Task(u, from, split),
                        new Task(u, from + split, to));
        }
    }
}
Solution

1. Add field variable

```java
import static Arrays.*;
class Task extends RecursiveAction {
    private int from, to;
    public Task (User[] u) { ... }
    protected void compute() {
        if (u.length < N) { local(u); }
        else {
            int split = u.length / 2;

            User[] u1 = copyOfRange(u, 0, split);
            User[] u2 = copyOfRange(u, split, u.length);

            invokeAll(new Task(u1), new Task(u2));
        }
    }
}
```
Solution

1. Add field variable

2. Add new constructor and update its usage

```java
import static Arrays.*;

class Task extends RecursiveAction {
    private int from, to;

    public Task (User[] u) { ... }

    protected void compute()
        if (u.length < N) { local(u); }
        else {
            int split = u.length / 2;

            User[] u1 = copyOfRange(u, 0, split);
            User[] u2 = copyOfRange(u, split, u.length);

            invokeAll(
                new Task(u, from, split),
                new Task(u, from + split, to)
            );
        }
    }
```
Solution

1. Add field variable
2. Add new constructor and update its usage
3. Modify threshold management policy

```java
import static Arrays.*;

class Task extends RecursiveAction {
    private int from, to;
    public Task (User[] u) { ... }
    private Task (User[] u, int from, int to) { ... }
    protected void compute() {
        if (to - from < N) { local(u, from, to); }
        else {
            int split = (from + to) / 2;
            User[] u1 = copyOfRange(u, 0, split);
            User[] u2 = copyOfRange(u, split, u.length);

            invokeAll(
                new Task(u, from, split),
                new Task(u, from + split, to));
        }
    }
}
```
Solution

1. Add field variable

2. Add new constructor and update its usage

3. Modify threshold management policy

4. Remove copy statements
GitHub is the largest code host on the planet with over 19.9 million repositories.

4Mi+ Users

19Mi+ Repositories

“GitHub is the largest code host on the planet with over 19.9 million repositories.”

https://github.com/features
4Mi+ Users

19Mi+ Repositories

“GitHub is the largest code host on the planet with over 19.9 mi repositories.”
https://github.com/features
4Mi+ Users
19Mi+ Repositories

"GitHub is the largest code host on the planet with over 19.9 mi repositories."
https://github.com/features
Experimental Environment

A 2×8-core (32-cores w/ hyper-threading) Intel CPU, running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.
Results
Results

9/15 crossed the 10% energy saving!
# Patches

<table>
<thead>
<tr>
<th>Status</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td>Improving ForkJoin usage for better performance</td>
<td>czbabi/itemupdown</td>
</tr>
<tr>
<td>Merged</td>
<td>Improving ForkJoin usage for better performance</td>
<td>toby1984/jAcer</td>
</tr>
<tr>
<td>Proposed</td>
<td>Improving ForkJoin usage for better performance</td>
<td>nikkrichko/Educational</td>
</tr>
<tr>
<td>Merged</td>
<td>Improving ForkJoin usage for better performance</td>
<td>mayukh42/scalatuts</td>
</tr>
<tr>
<td>Merged</td>
<td>Improving ForkJoin usage for better performance</td>
<td>tfreese/knn</td>
</tr>
<tr>
<td>Closed</td>
<td>Improving ForkJoin usage for better performance</td>
<td>TarantulaTechnology/netflixoss</td>
</tr>
<tr>
<td>Proposed</td>
<td>Improving ForkJoin usage for better performance</td>
<td>statsbiblioteket/doms-transformers</td>
</tr>
<tr>
<td>Merged</td>
<td>Improving ForkJoin usage for better performance and energy</td>
<td>cjlaro/MagicSquares</td>
</tr>
<tr>
<td></td>
<td>on Dec 5, 2014</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2sbsbsb/ForkAndJoinUtility</td>
<td></td>
</tr>
</tbody>
</table>
## Patches

<table>
<thead>
<tr>
<th>Status</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td>#1 Improving ForkJoin usage for better performance</td>
<td>czbabl/itementupdown</td>
</tr>
<tr>
<td>Merged</td>
<td>#1 Improving ForkJoin usage for better performance</td>
<td>toby1984/jAcer</td>
</tr>
<tr>
<td>Proposed</td>
<td>#1 Improving ForkJoin usage for better performance</td>
<td>nikkrichko/Educational</td>
</tr>
<tr>
<td>Merged</td>
<td>#1 Improving ForkJoin usage for better performance</td>
<td>mayukh42/scalatuts</td>
</tr>
<tr>
<td>Merged</td>
<td>#1 Improving ForkJoin usage for better performance</td>
<td>tfreese/knn</td>
</tr>
<tr>
<td>Closed</td>
<td>#1 Improving ForkJoin usage for better performance</td>
<td>TarantulaTechnology/netflixoss</td>
</tr>
<tr>
<td>Proposed</td>
<td>#1 Improving ForkJoin usage for better performance</td>
<td>statsbiblioteket/doms-transformers</td>
</tr>
<tr>
<td>Merged</td>
<td>#1 Improving ForkJoin usage for better performance</td>
<td>2sbsbsb/ForkAndJoinUtility</td>
</tr>
</tbody>
</table>

7/9 of projects that replied have accepted
Patches

7/9 of projects that replied have accepted
Patches

7/9 of projects that replied have accepted
The Goal

1. To understand how software developers are dealing with energy consumption issues;

2. To characterize the energy-consumption behavior of

   1. Thread-safe collections

   2. Thread management techniques

3. To derive a refactoring to (1) identify and (2) refactor one energy-consumption anti-pattern;
Conclusions

There is a “brave new world” for Refactoring for Energy Efficiency.

Review 3

This paper has the main goal of identifying opportunities, and challenges in the context of (application level) refactoring for energy efficiency. In this line, the authors review a number of conferences in order to find research articles related to energy and power. They select 16 papers. Six categories are created from those papers, and for each, problems, opportunities, and challenges are presented.

Overall, I found the paper interesting, with a varied list of opportunities, and a competent analysis of expected challenges. I am personally sure refactoring for energy efficiency will be the next hot topic in Green computing.
Conclusions

However, the questions is: When to refactor?
Conclusions

However, the question is: When to refactor?

Threads

ForkJoin
Conclusions

However, the question is: When to refactor?

Threads

ForkJoin
Conclusions

However, the questions is: When to refactor?

Non Thread-Safe
Data Structures

Thread-Safe
Data Structures
Conclusions

However, the question is: When to refactor?

Non Thread-Safe Data Structures

Thread-Safe Data Structures
Conclusions

• This thesis just scratched the surface

• More research is indeed needed

• We welcome you to join us!
Conclusions

• This thesis just scratched the surface

• More research is indeed needed
Conclusions

• This thesis just scratched the surface
• More research is indeed needed
• We welcome you to join us!
Conclusions

- This thesis just scratched the surface
- More research is indeed needed
- We welcome you to join us!
A Refactoring Approach to Improve Energy Consumption of Parallel Software Systems

Gustavo Pinto

Ph.D. Defense
Informatics Center
Federal University of Pernambuco

Recife, February/2015