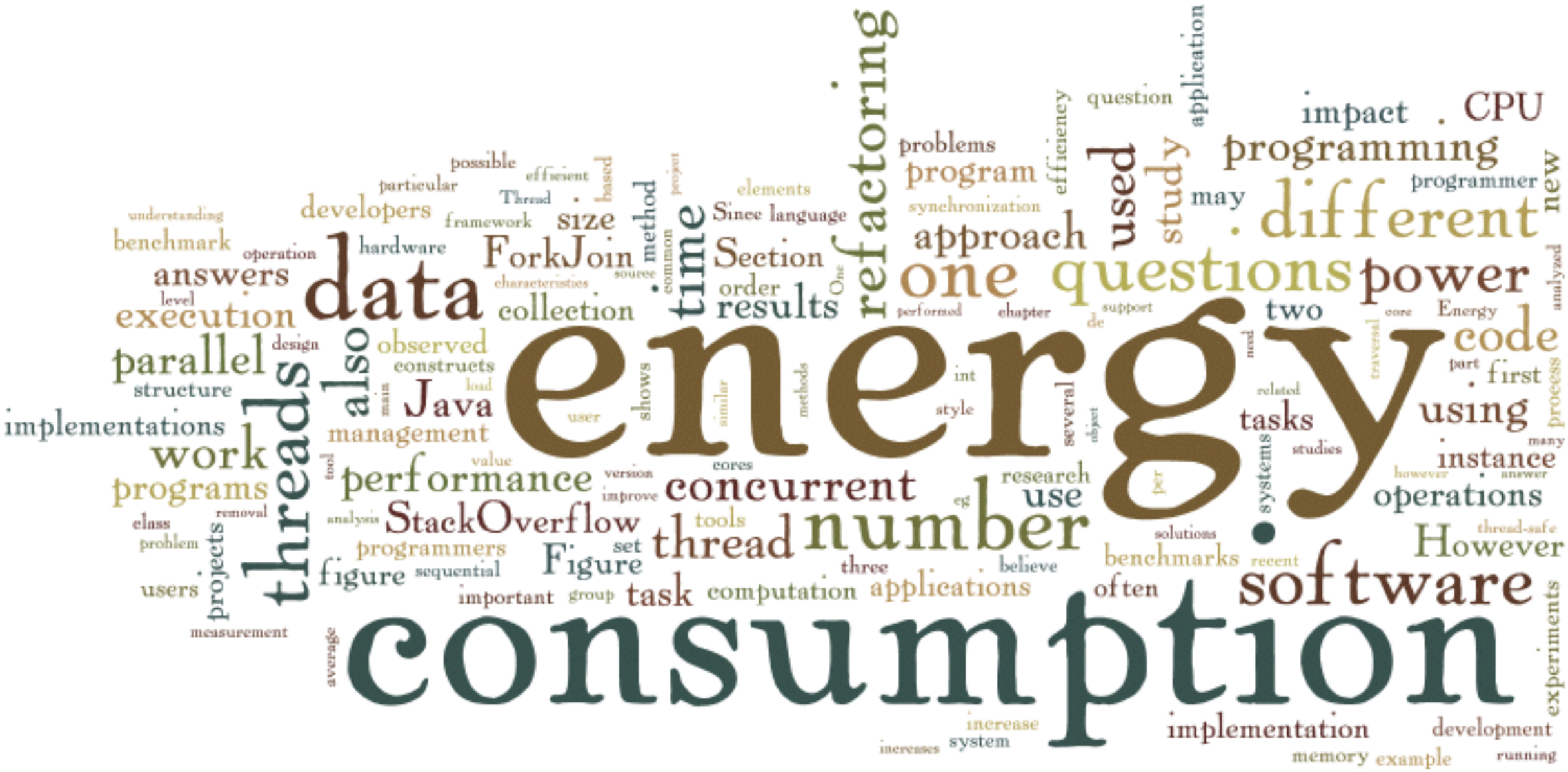
A Refactoring Approach to Improve Energy Consumption of Parallel Software Systems

Gustavo Pinto

Ph.D. Defense
Informatics Center
Federal University of Pernambuco

Recife, February/2015



Generated from 77,317 words

Motivation (1/2)

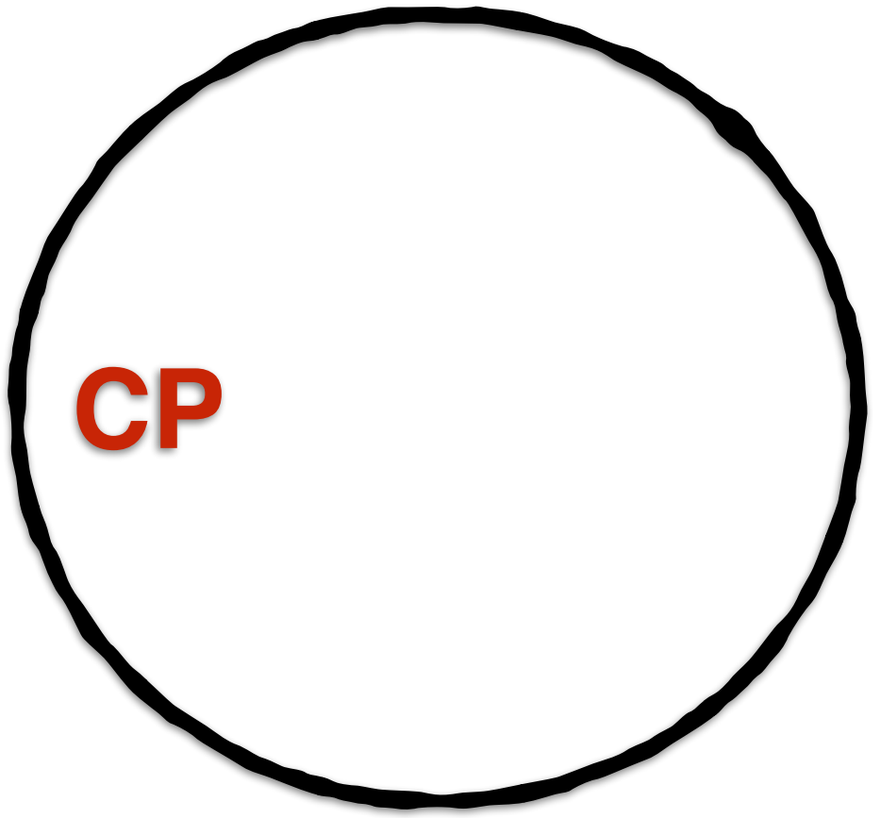
- **First**, **energy consumption** is a concern for unwired devices and also for data centers.

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

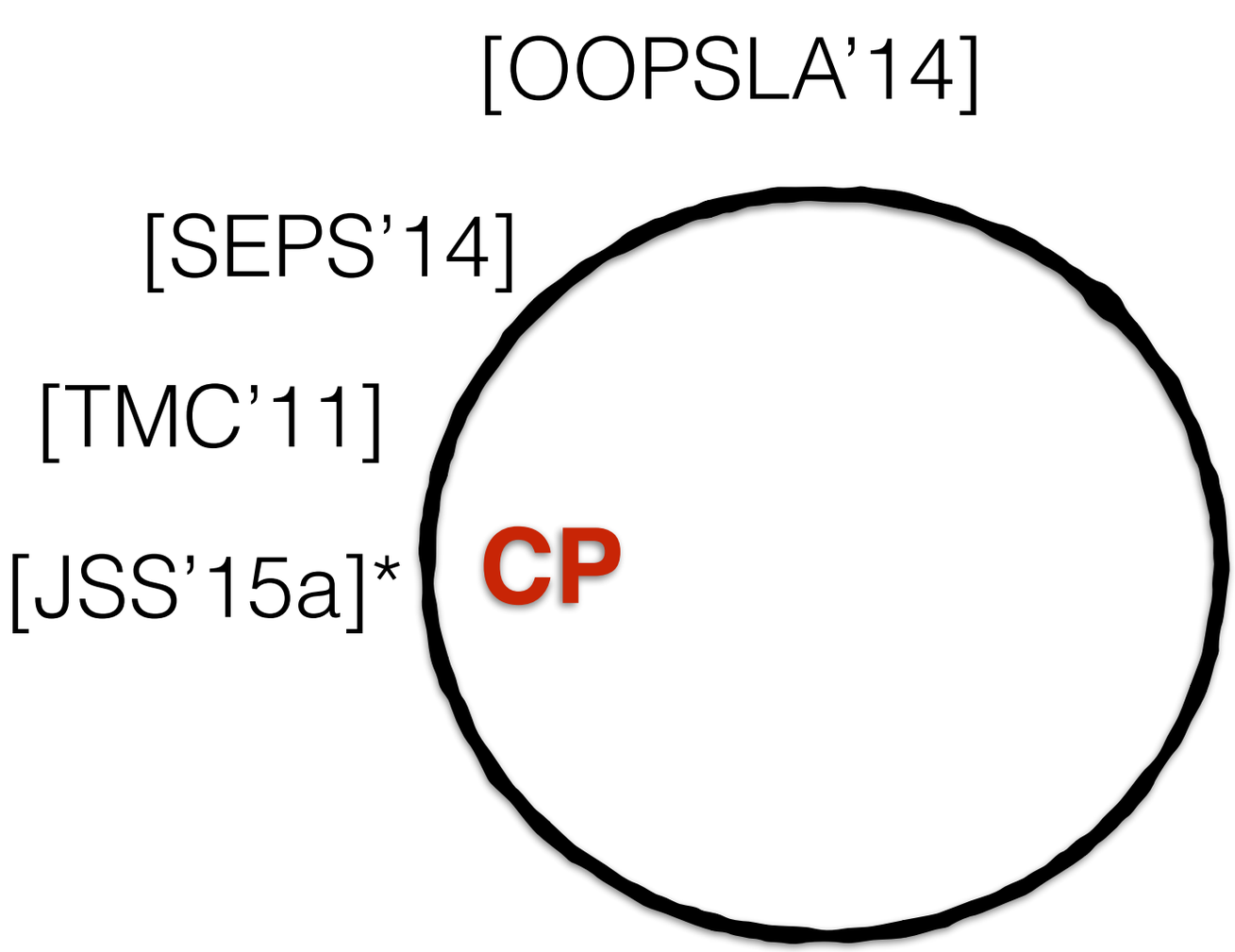
- **However**, little is known about the application level.

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

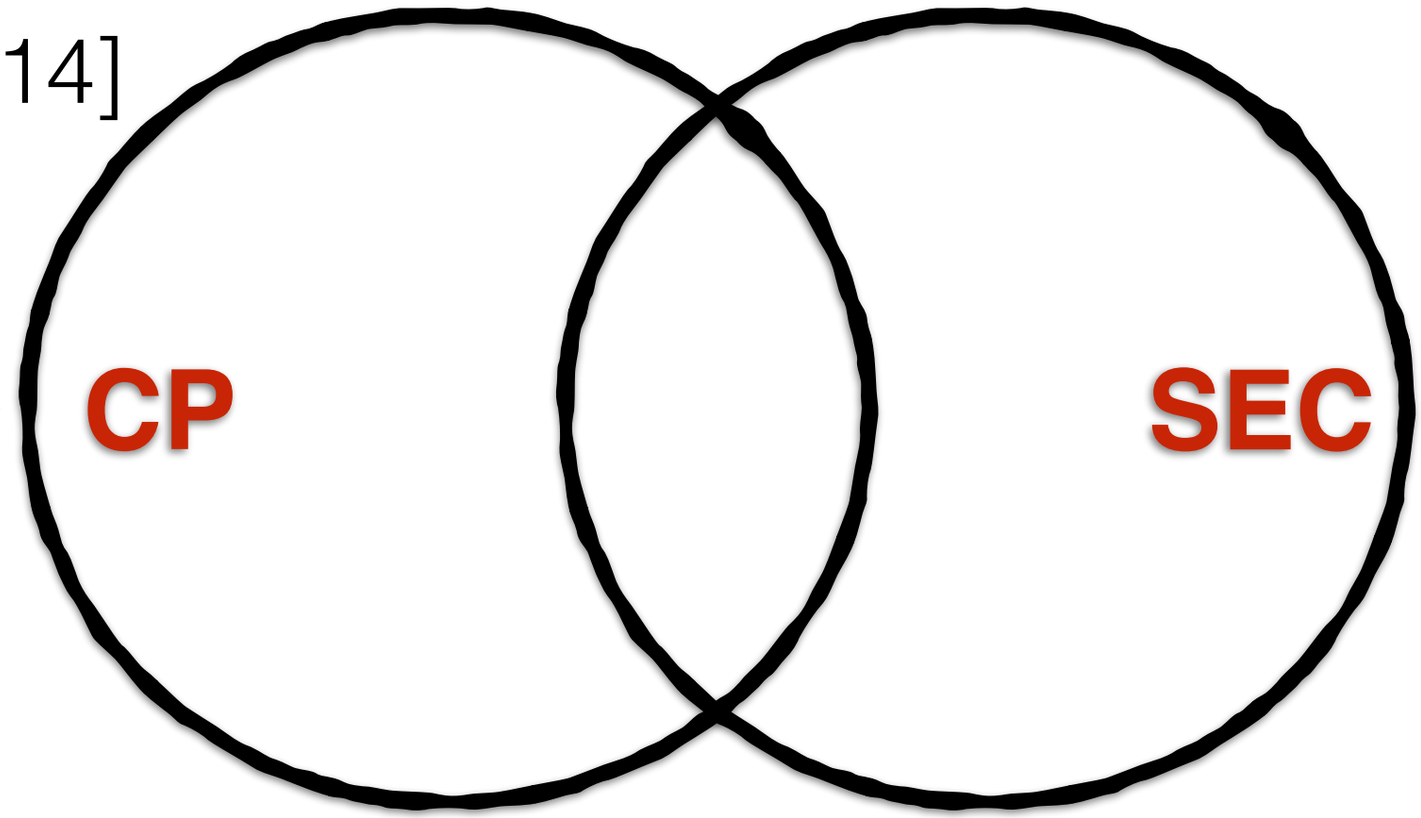
* Under submission

[OOPSLA'14]

[SEPS'14]

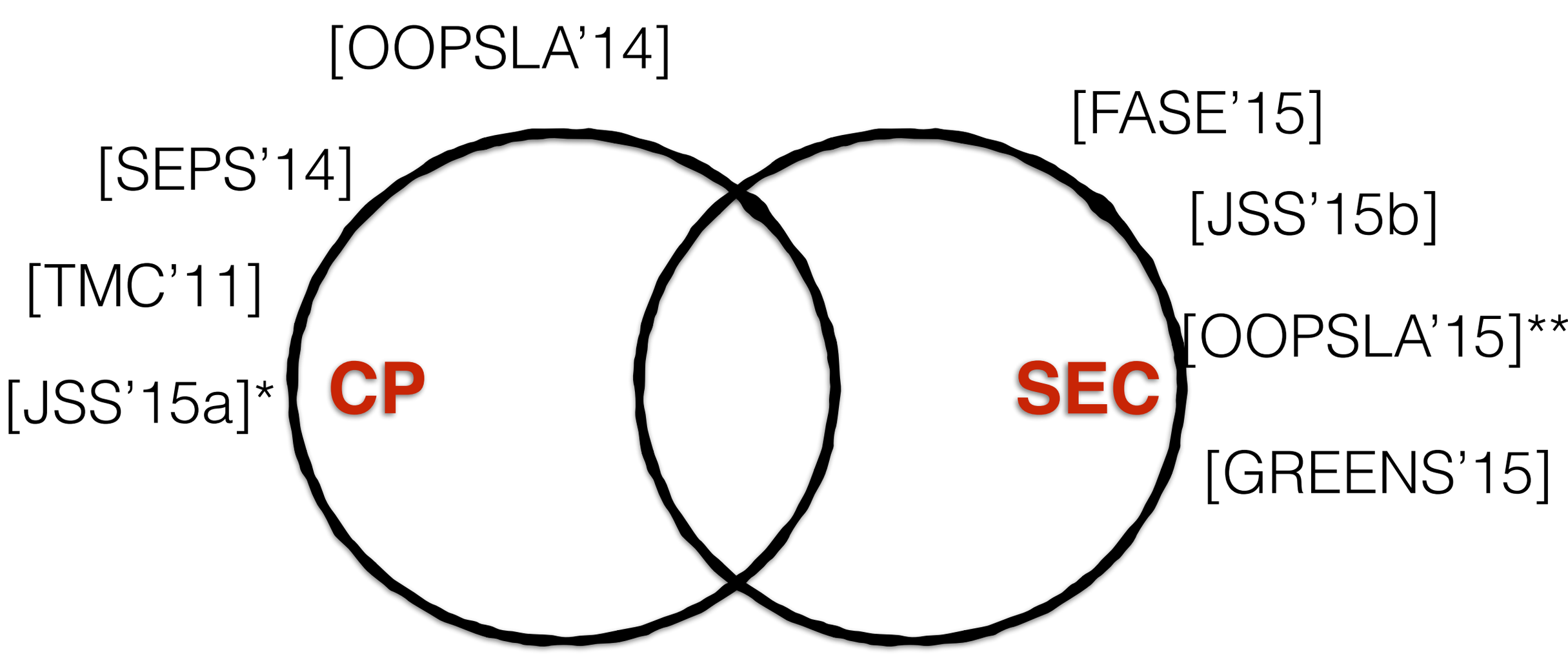
[TMC'11]

[JSS'15a]*



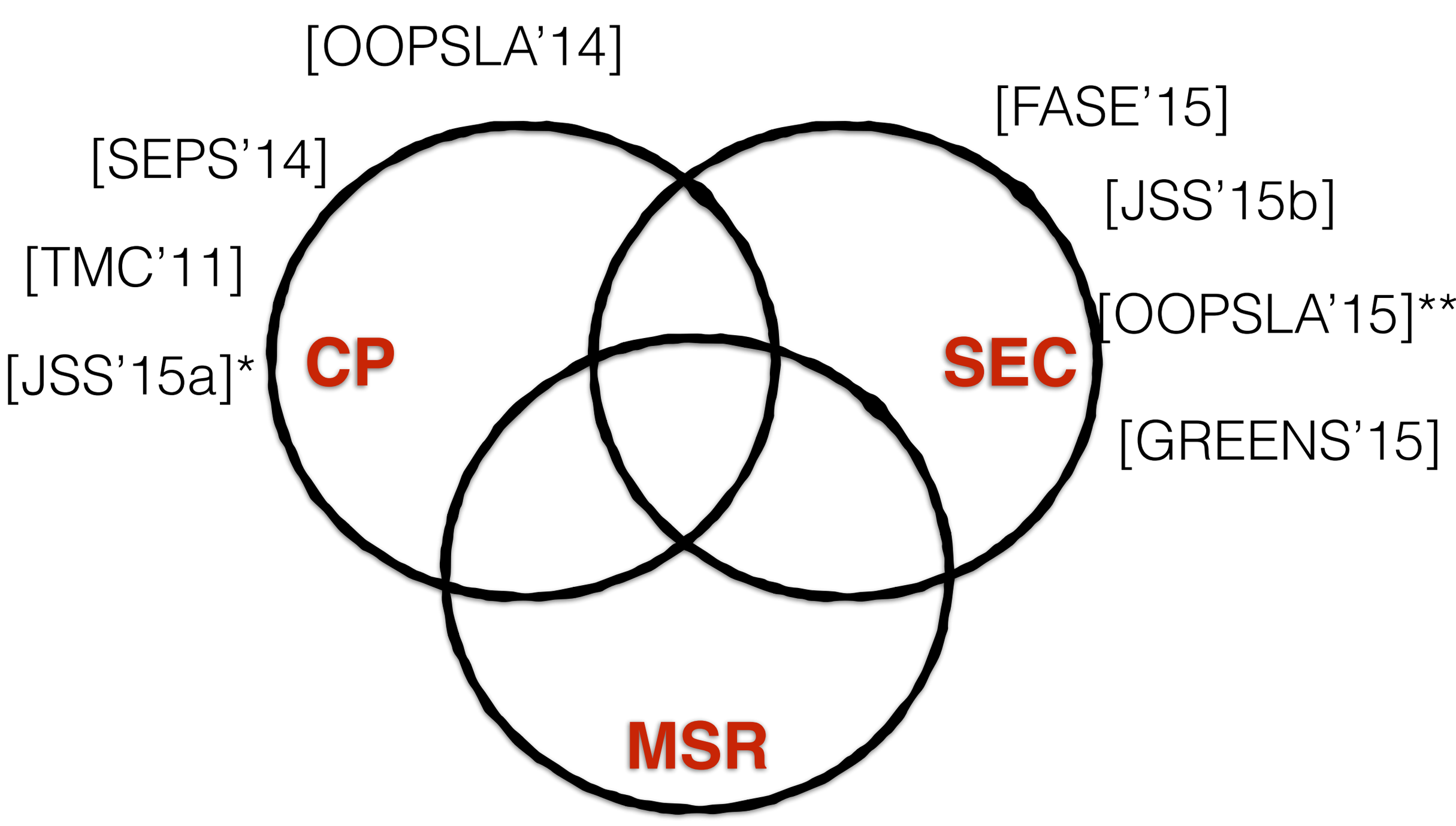
CP: Concurrent Programming
SEC: Software Energy Consumption

* Under submission



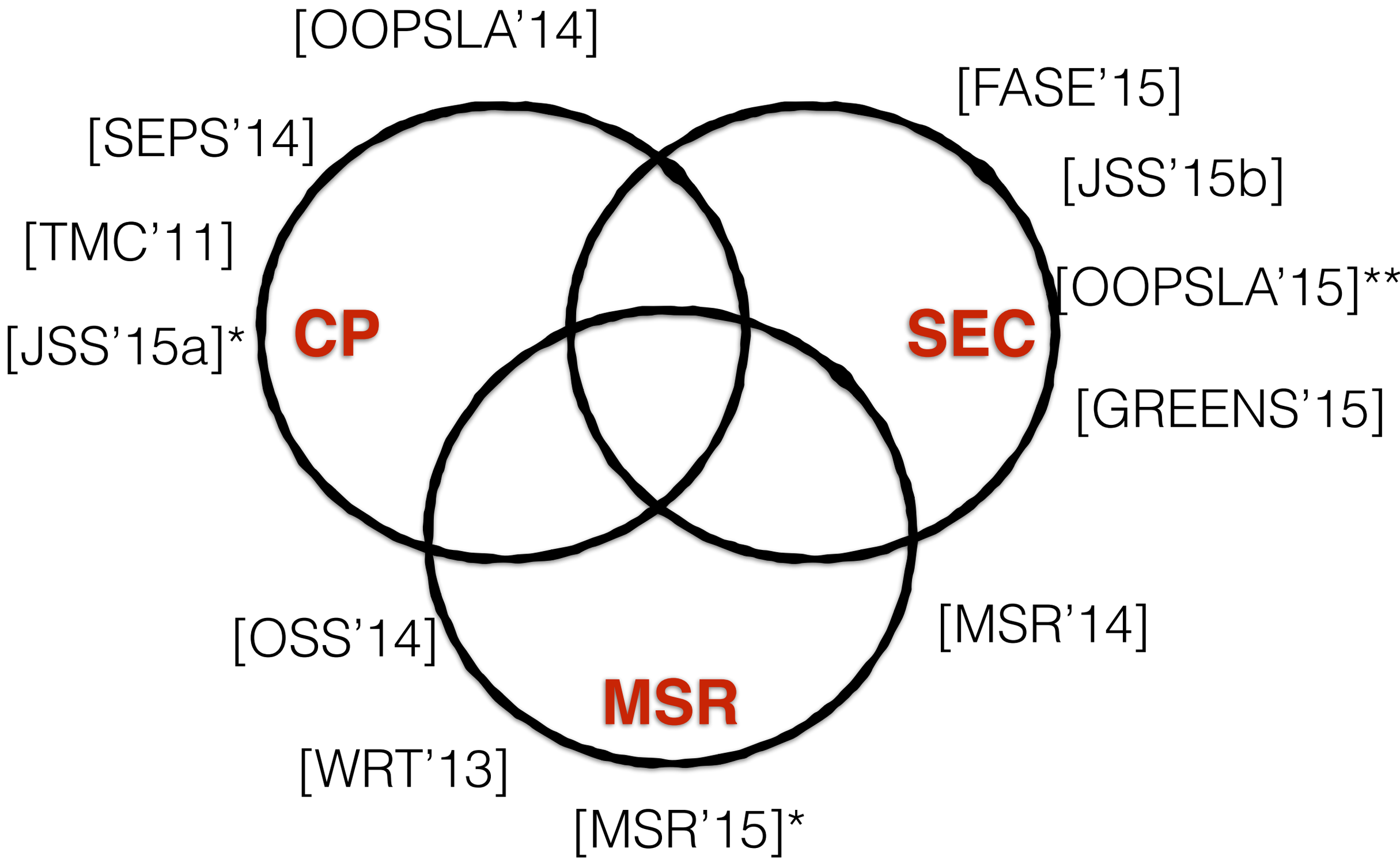
CP: Concurrent Programming
SEC: Software Energy Consumption

* Under submission



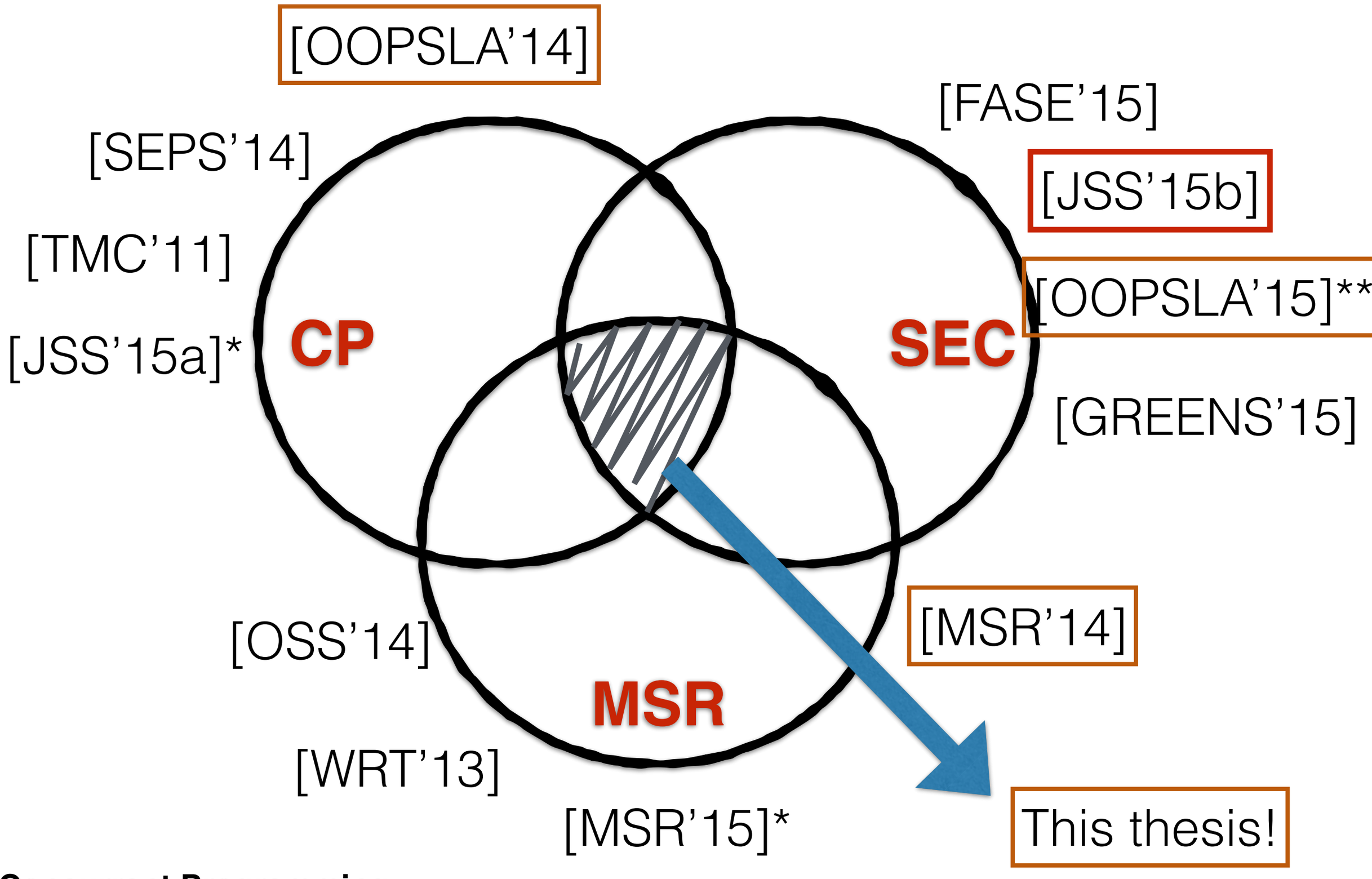
CP: Concurrent Programming
SEC: Software Energy Consumption
MSR: Mining Software Repositories

** Will be submitted
* Under submission



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

** Will be submitted
* Under submission



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

** Will be submitted
* Under submission

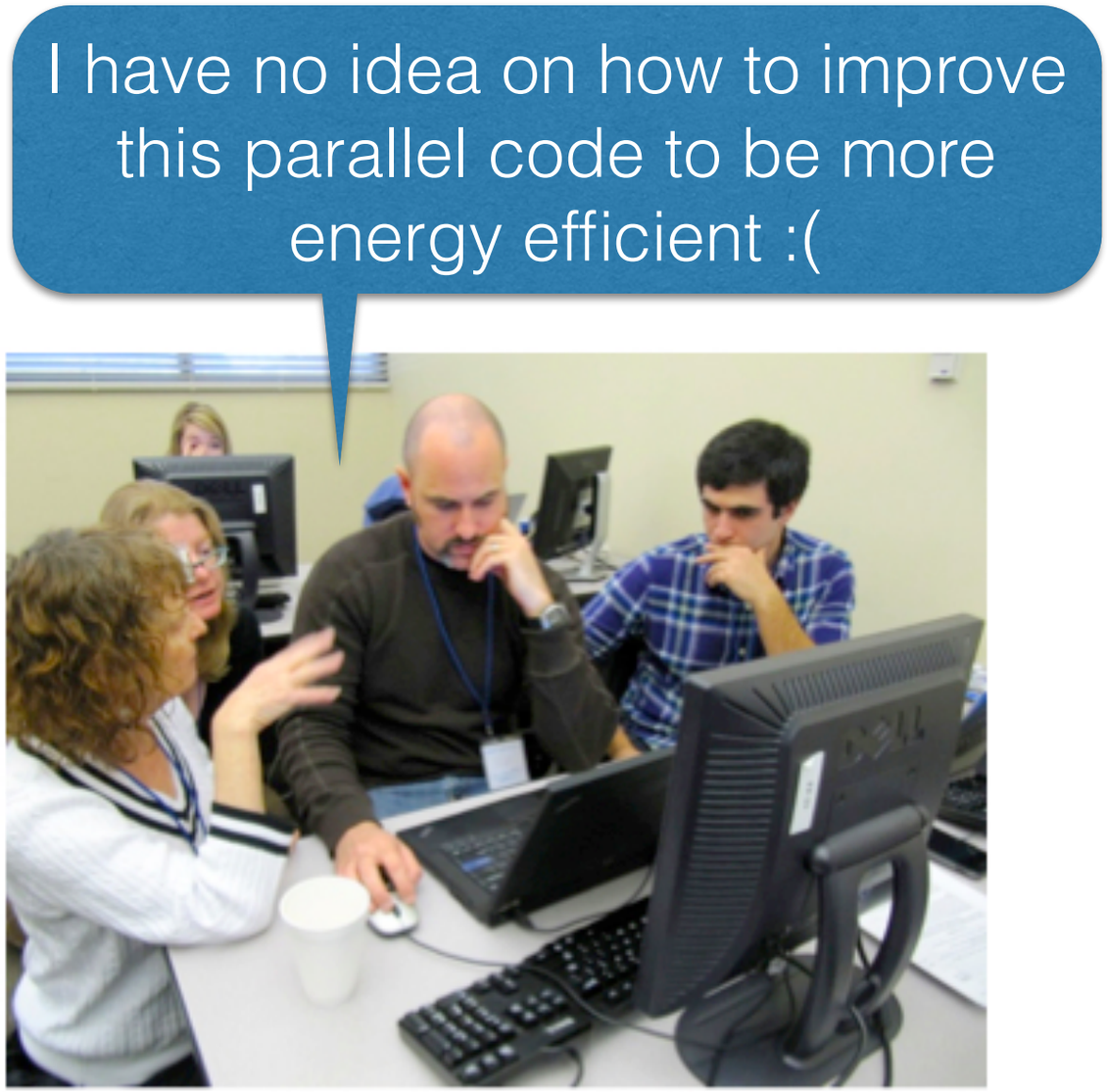
The Problem

- The lack of knowledge
- The lack of tools

The Problem

- The lack of knowledge

- The lack of tools



The Problem

- The lack of knowledge
- The lack of tools

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. <u>To understand how software developers are dealing with energy consumption issues;</u>

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

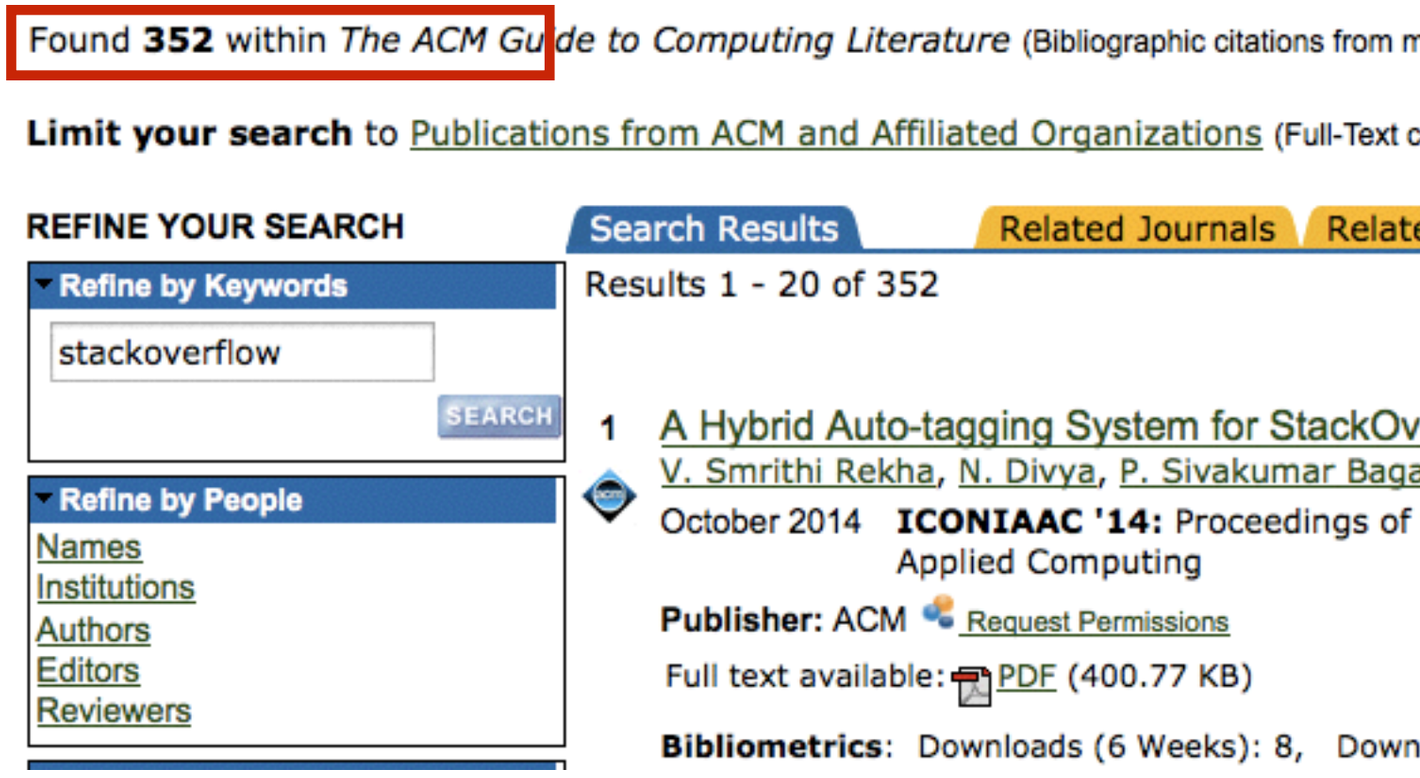
“The most used Q&A website in the world”

2M+ Users

5M+ Questions

10M+ Answers

50GB+ of data

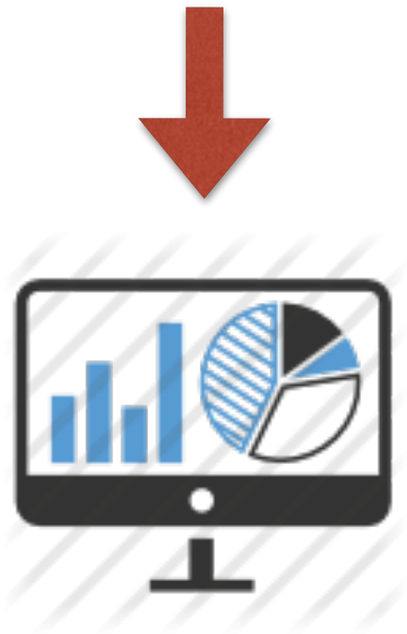


"The most used Q&A website in the world"

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

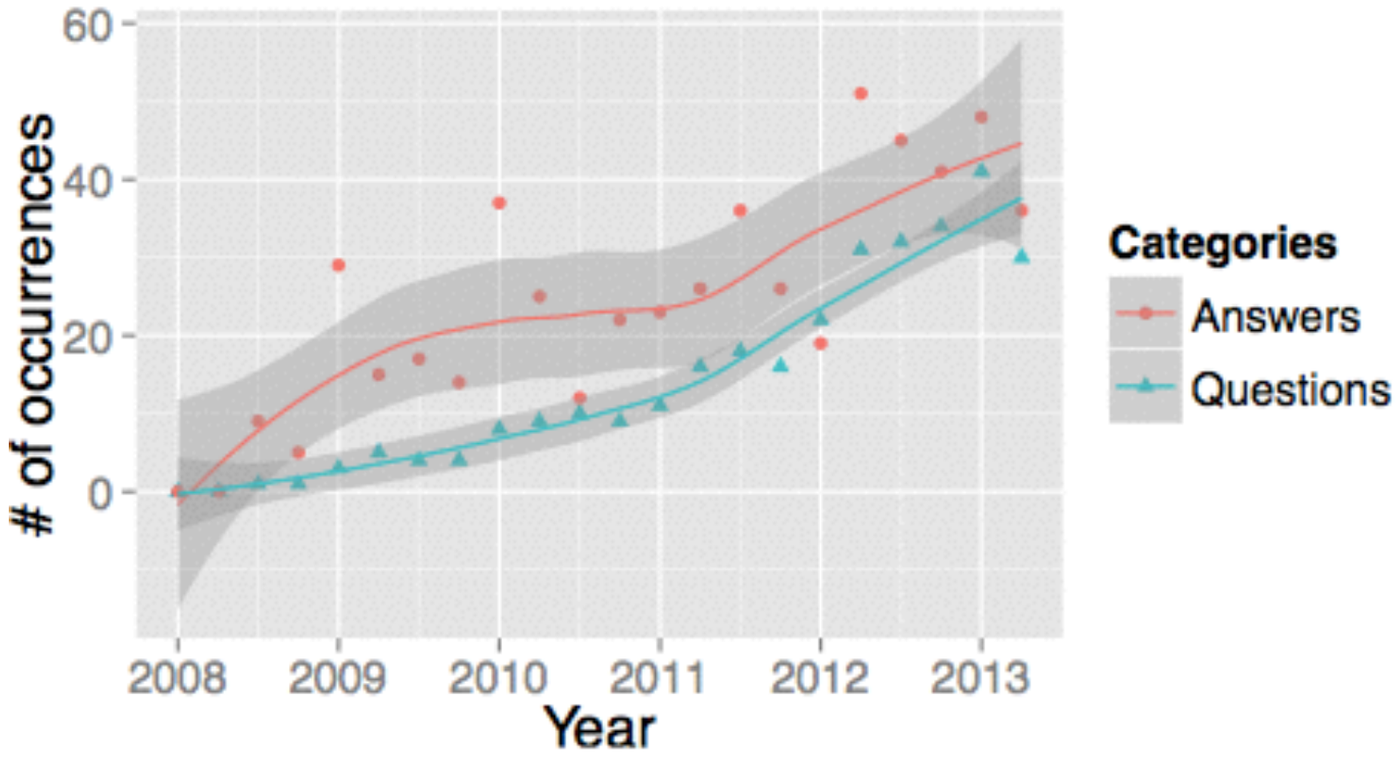
Final Data

325 Questions
558 Answers

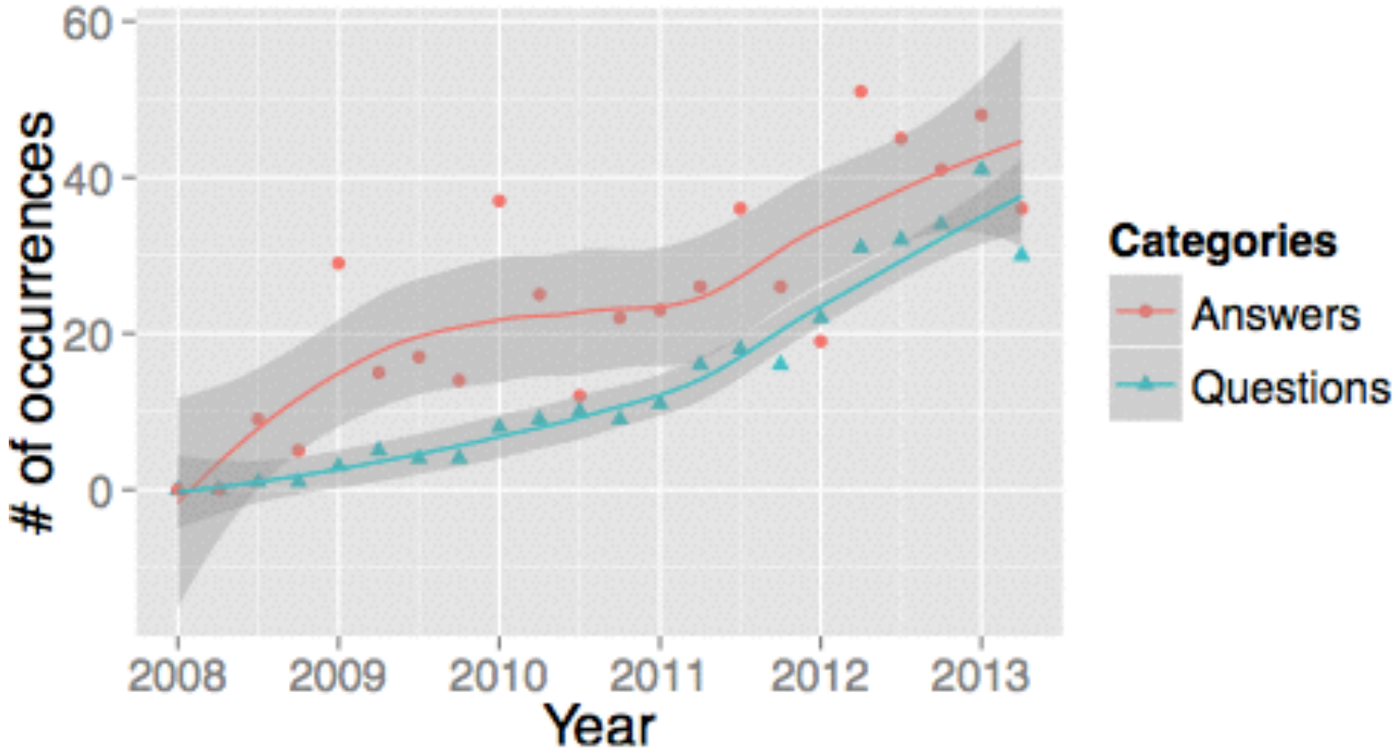
Base Group

from 2008 to 2013

Characteristics



Characteristics



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

No obvious “energy expert”

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?”

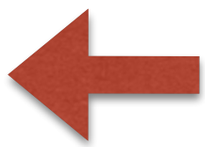
- Measurements (59/97 — Q/A)
- General Knowledge (40/84 — Q/A)
- Code design (36/133 — Q/A)
- Context-specific (83/110 — Q/A)
- Noise (107/134 — Q/A)

“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)
- ✓ Hardware Coordination (11 occurrences)
- ✓ Bulk operations (24 occurrences)
- ✗ Concurrent Programming (9 occurrences)
- ✓ Avoid polling (17 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

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

Non thread-safe

Thread-safe

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

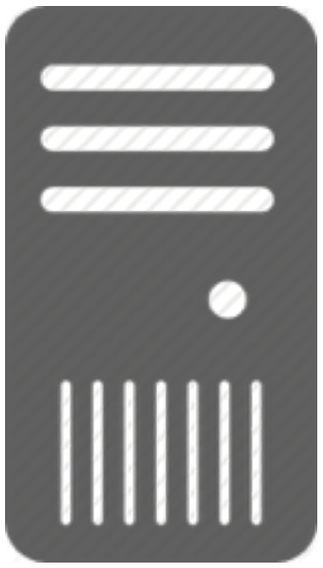
Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

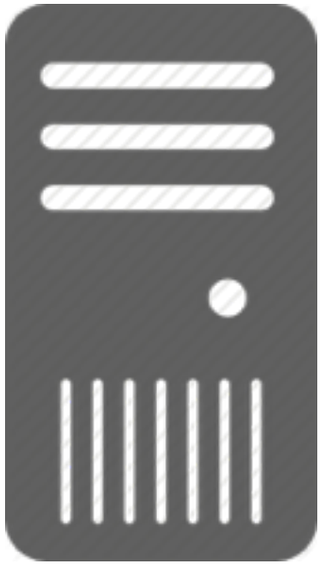
x 3 Operations

Traversal	Insertion	Removal

Experimental Environment

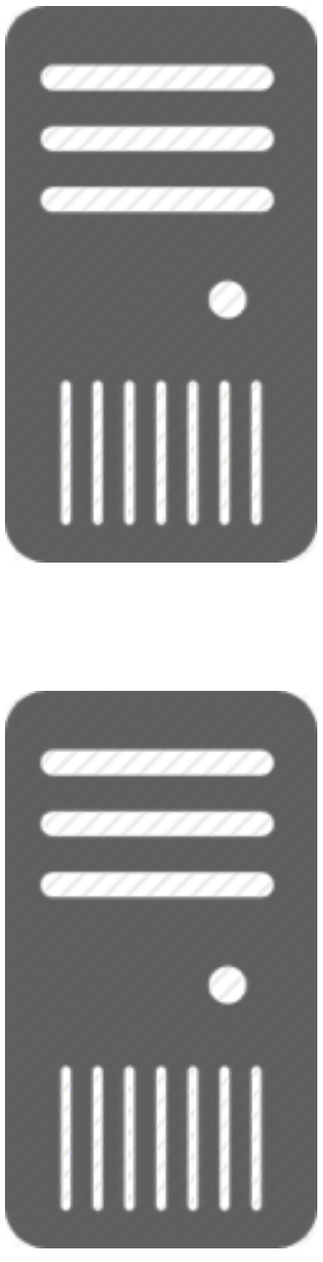


System#1: A 2×16-core **AMD CPUs**, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.



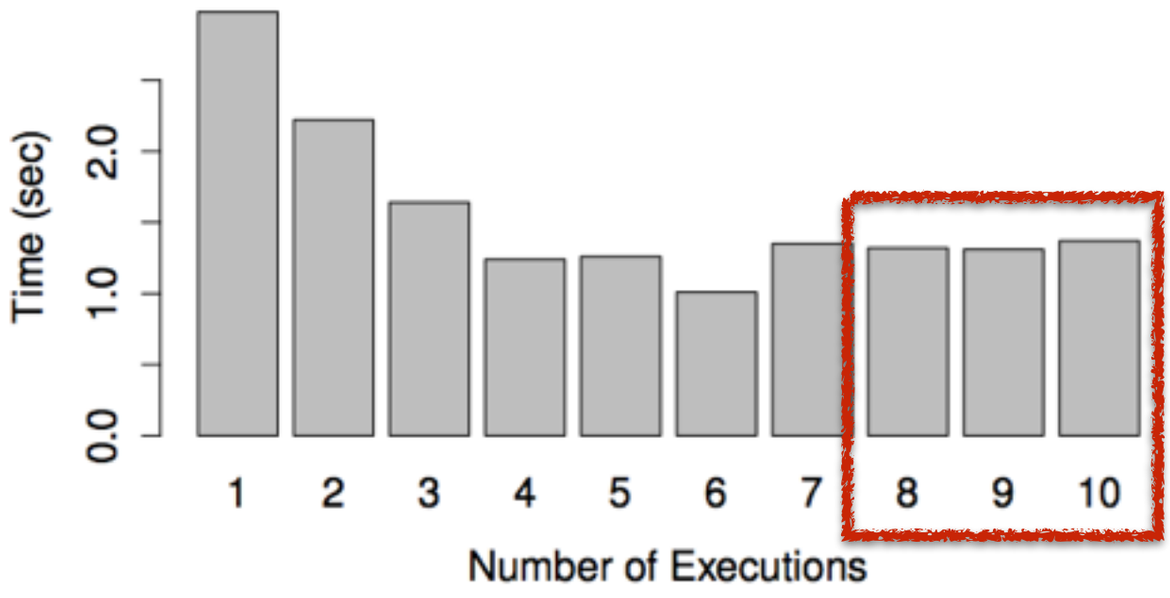
System#2: A 2×8-core (32-cores w/ hyper-threading) **Intel CPU**,running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.

Experimental Environment

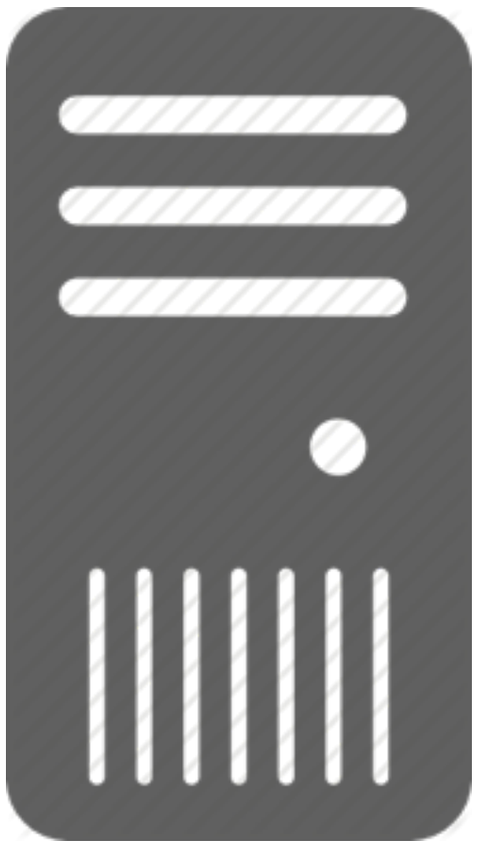


System#1: A 2×16-core **AMD CPUs**, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

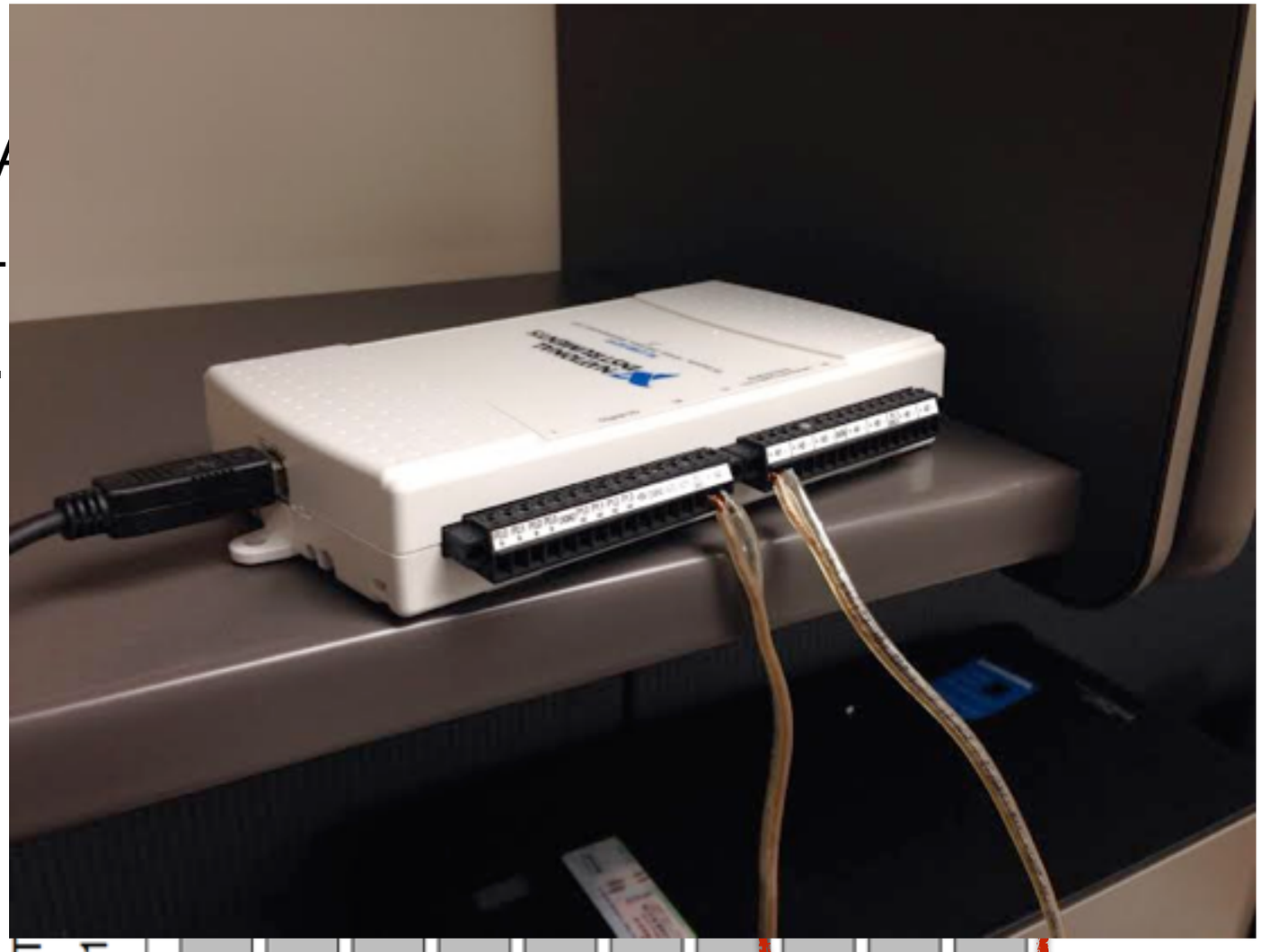
System#2: A 2×8-(threading) **Intel C**I 2.60GHz, with 64(1.7.0 71, build 14.

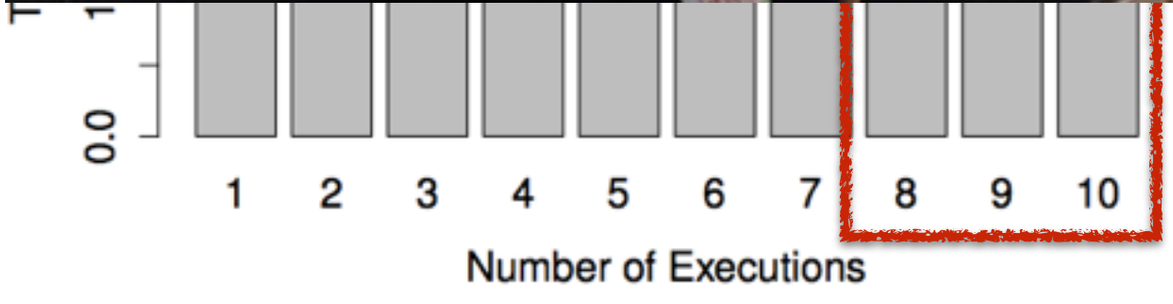


Experimental Environment

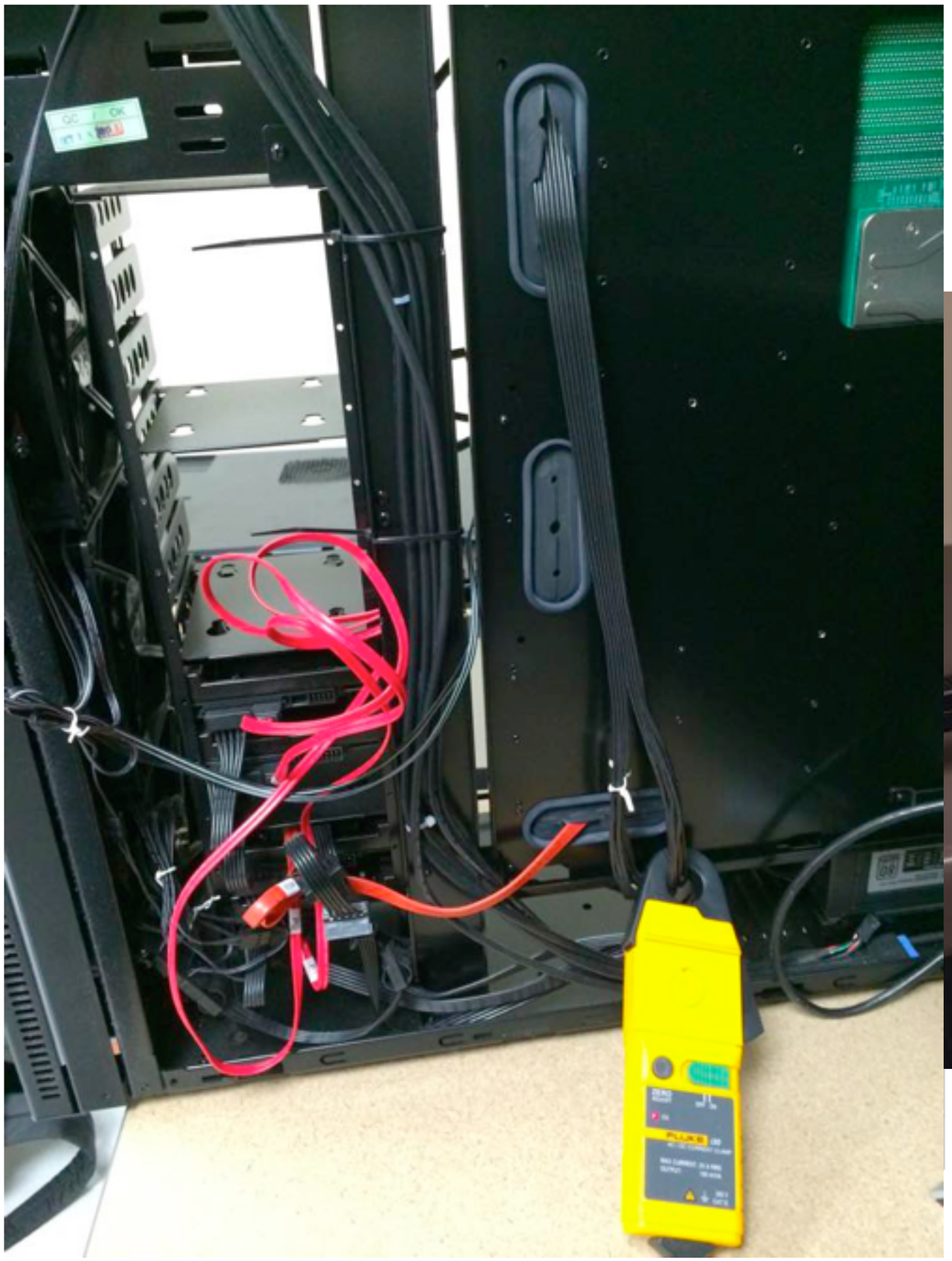


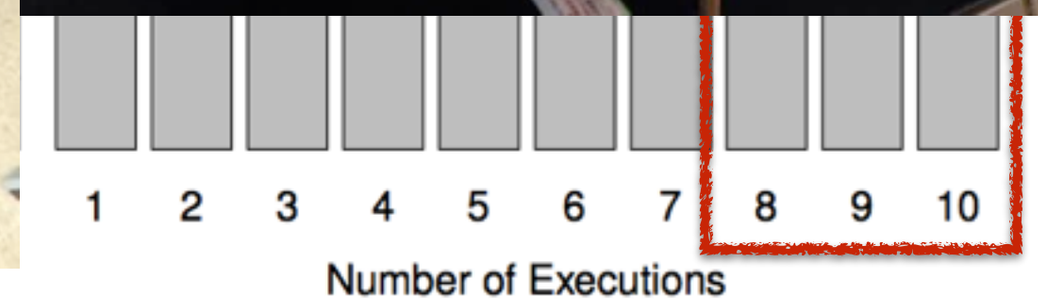
System#1: A
Debian, 2.4
version 1.7.



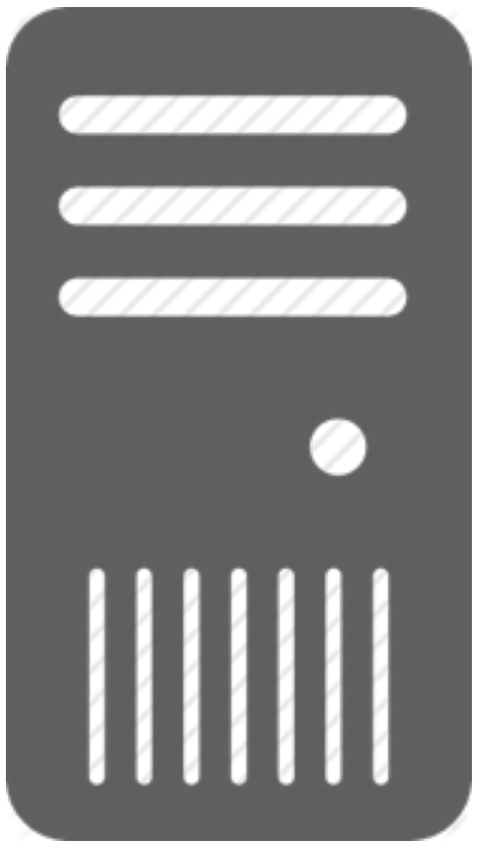


l Environment





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

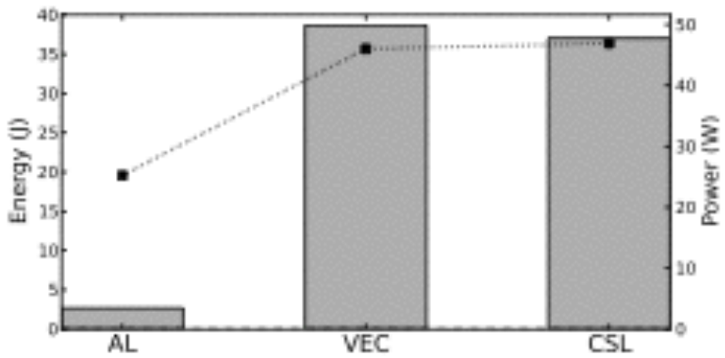
But, what is RAPL?

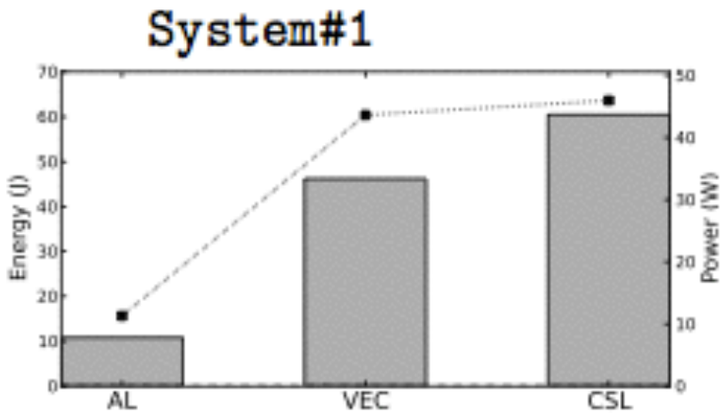
RAPL is a set of low-level interfaces with the ability to monitor, control, and get notifications of energy and power consumptio

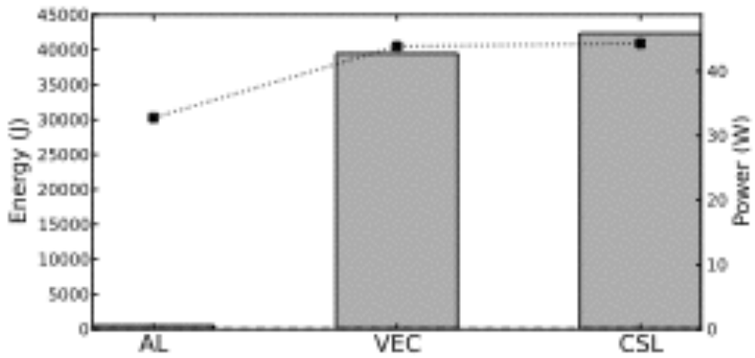
Originally designed by Intel for enabling chip-level power management, RAPL is widely supported in today's Intel architecture popular i5 and i7.

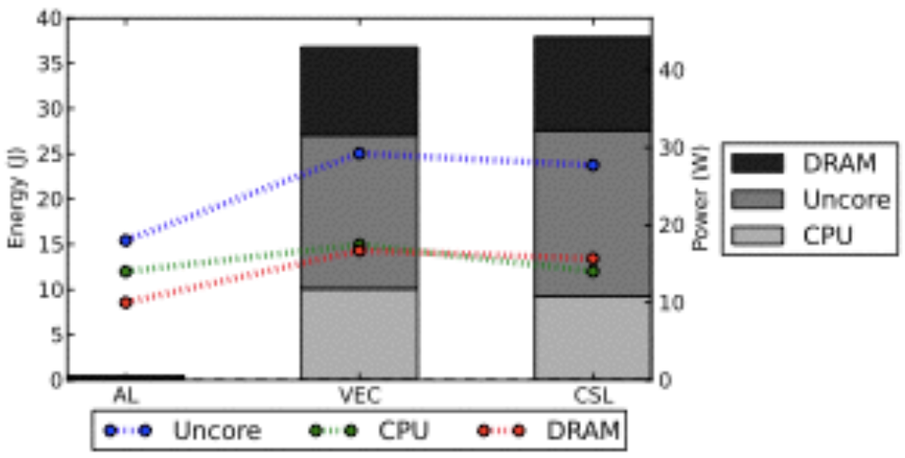
http://kliu20.github.io/jRAPL/

Lists

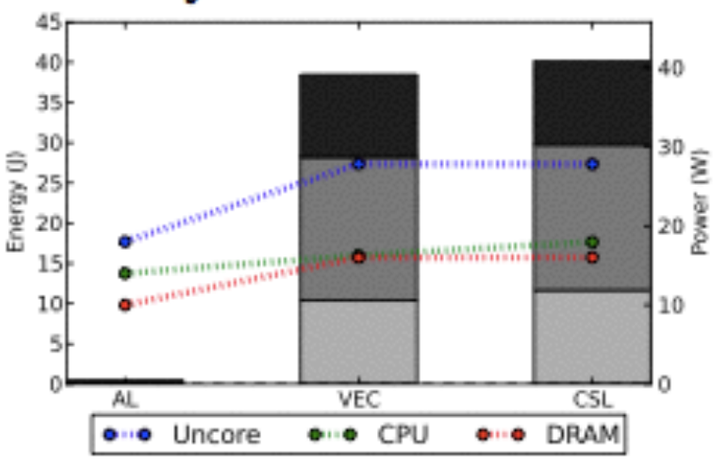




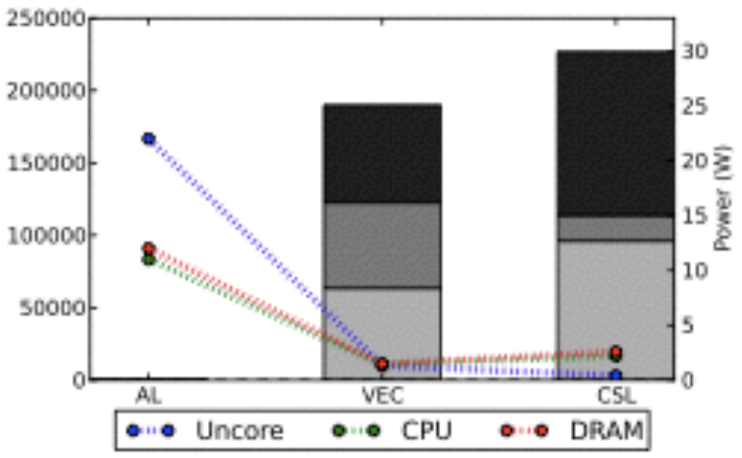




Traversal

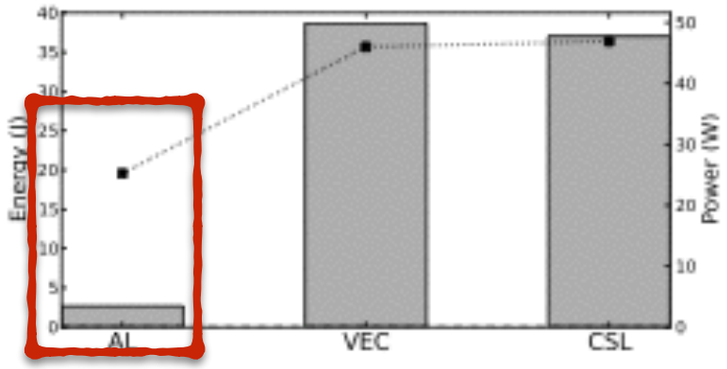


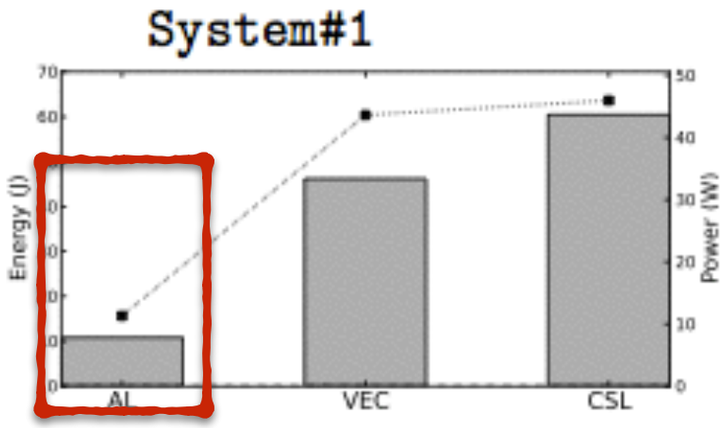
Insertion

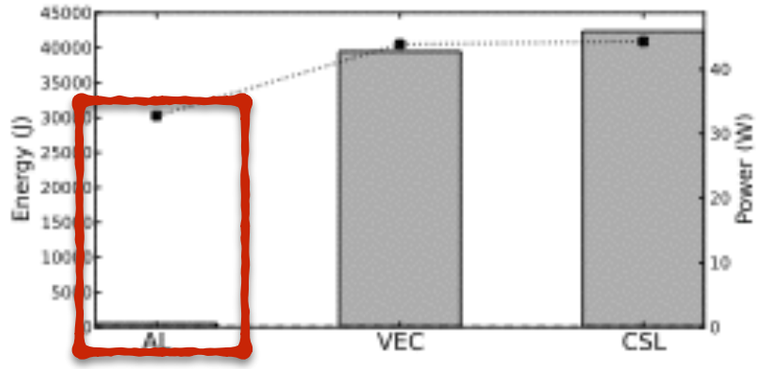


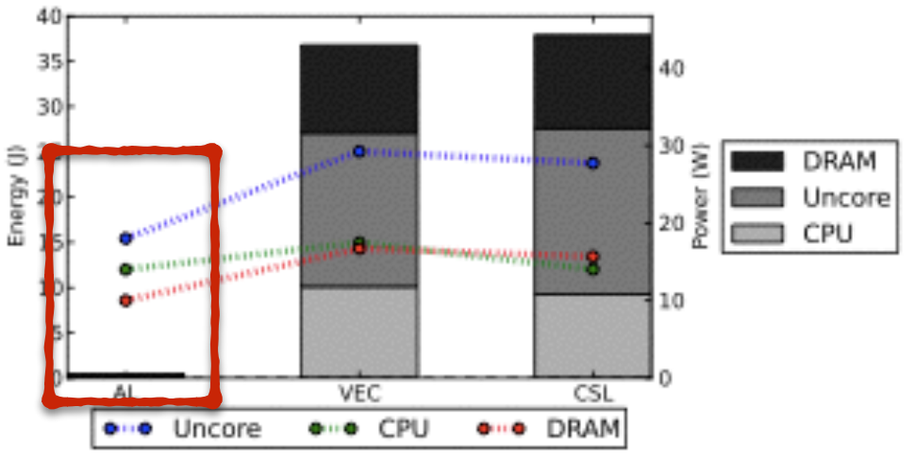
Removal

Lists

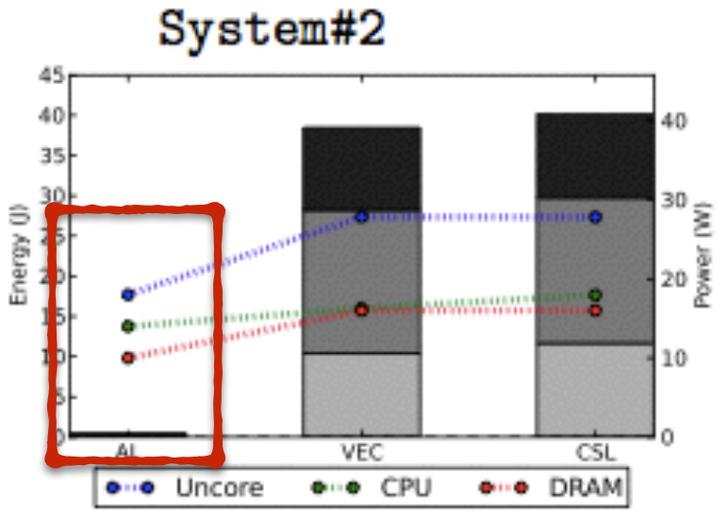




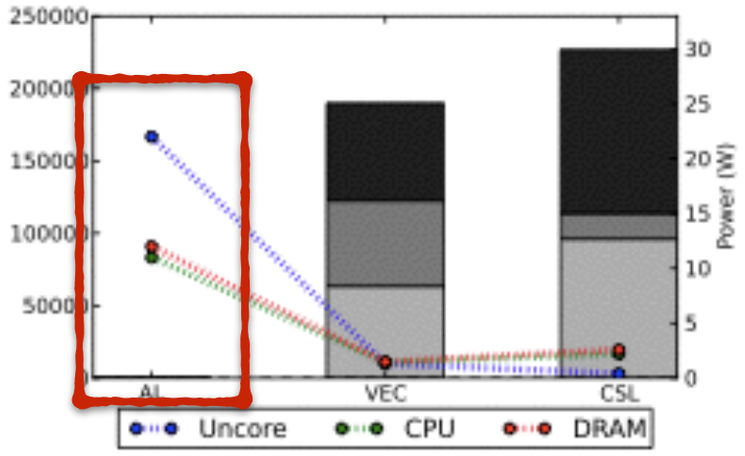




Traversal

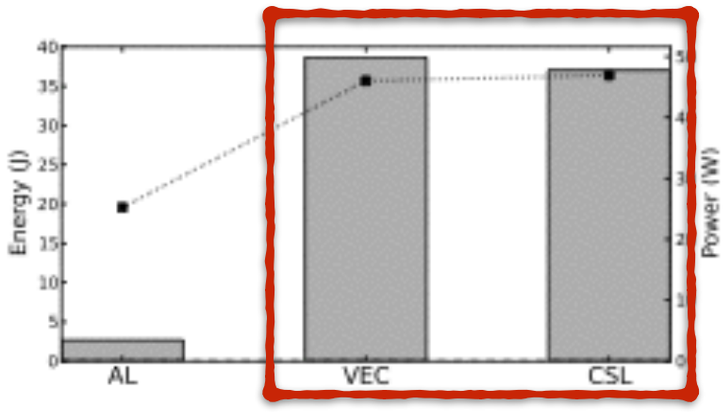


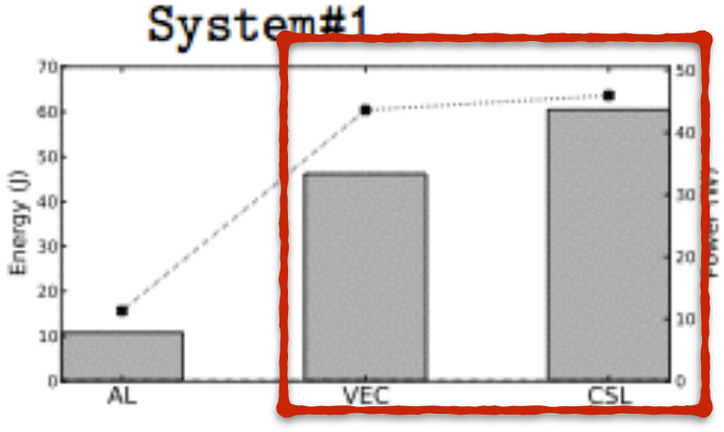
Insertion

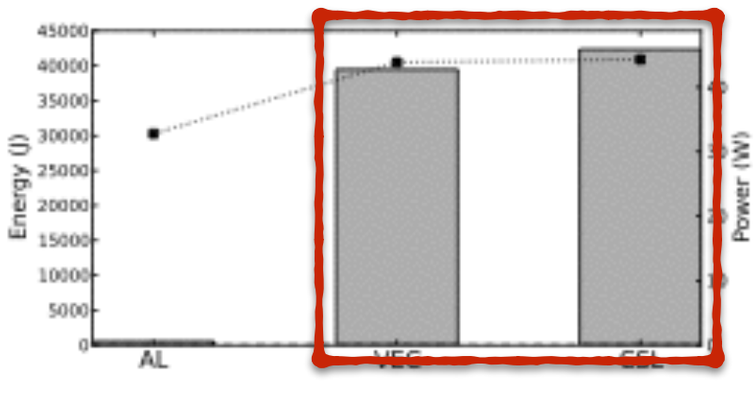


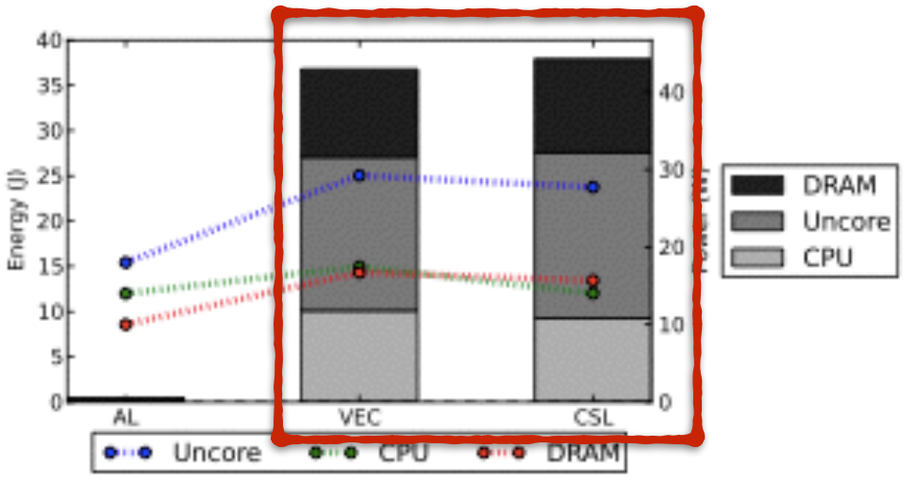
Removal

Lists

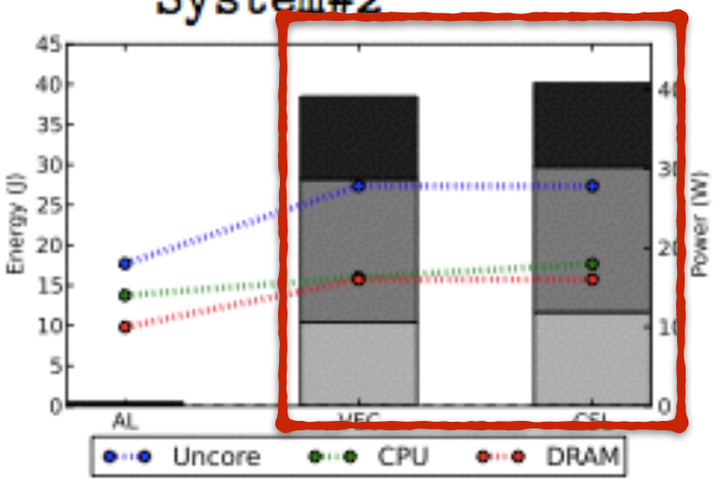




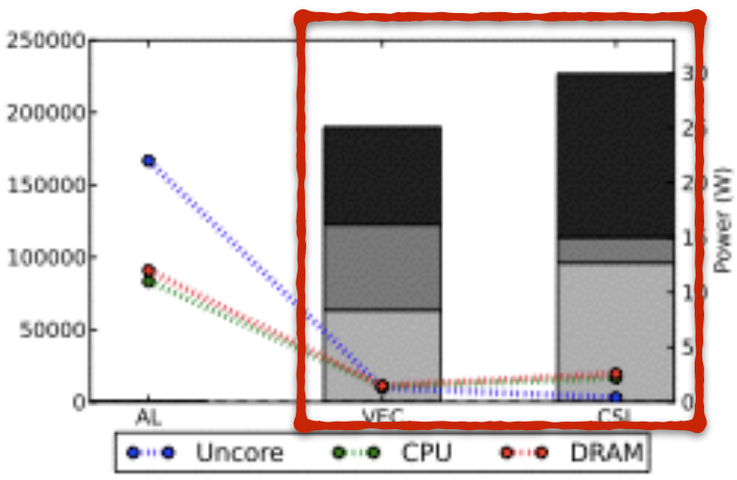




Traversal



Insertion

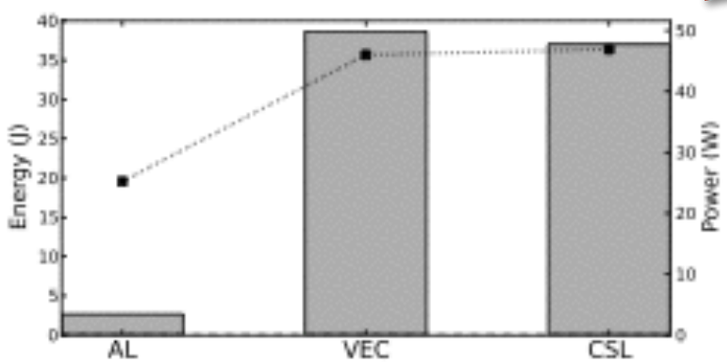


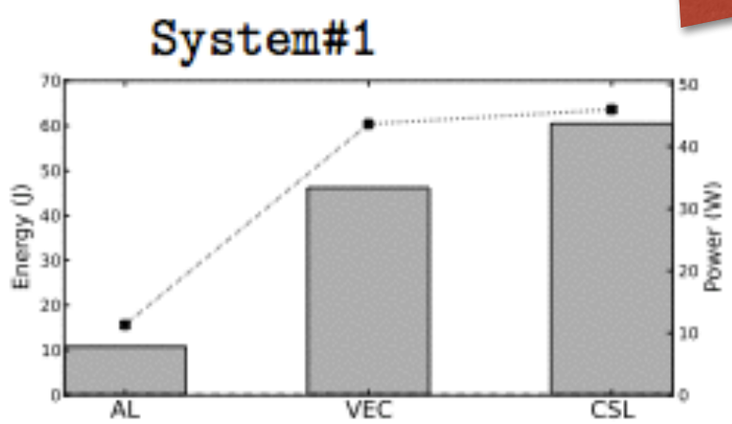
Removal

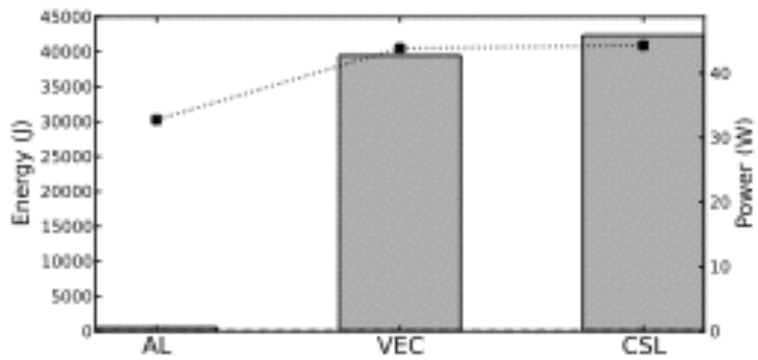
Lists

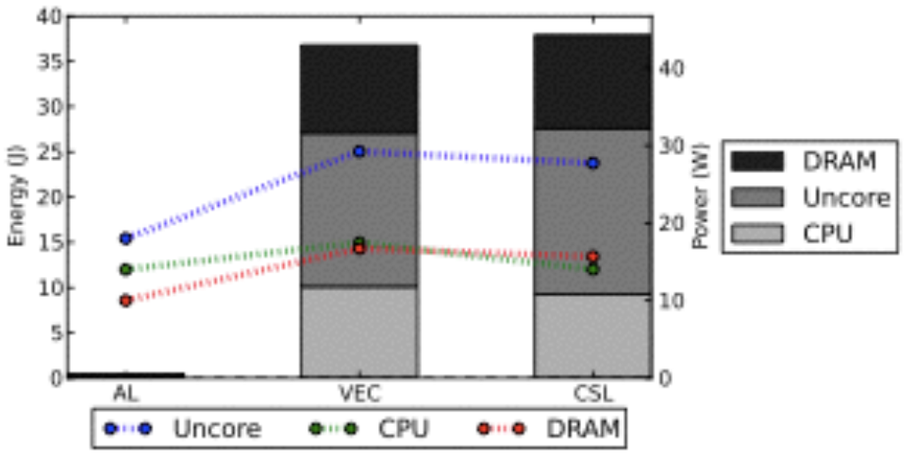
COW: -46x

COW: +152x

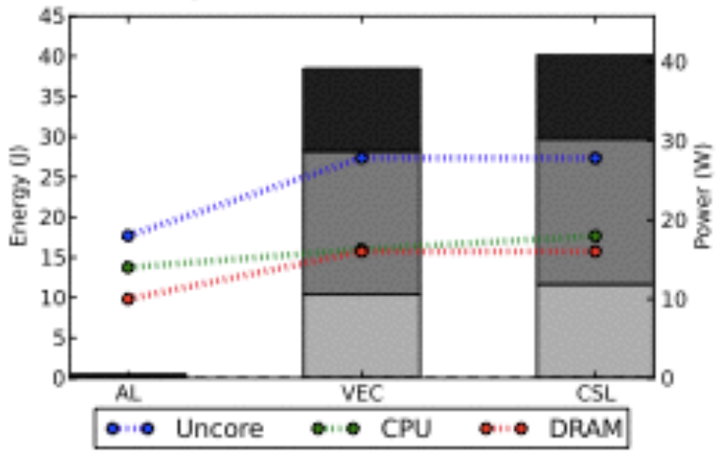




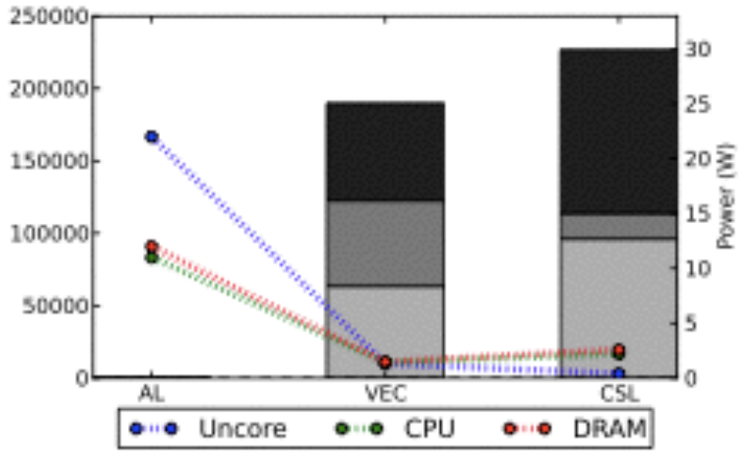




Traversal

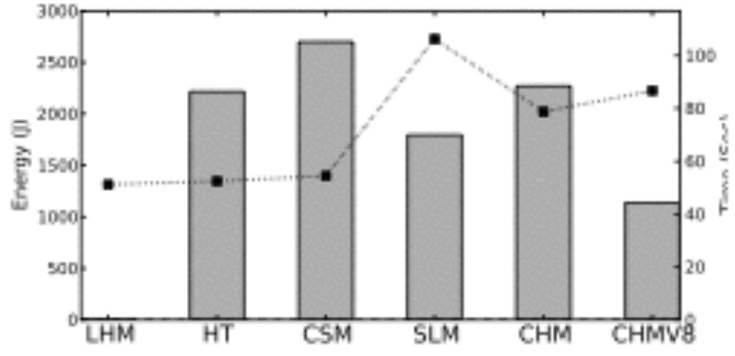


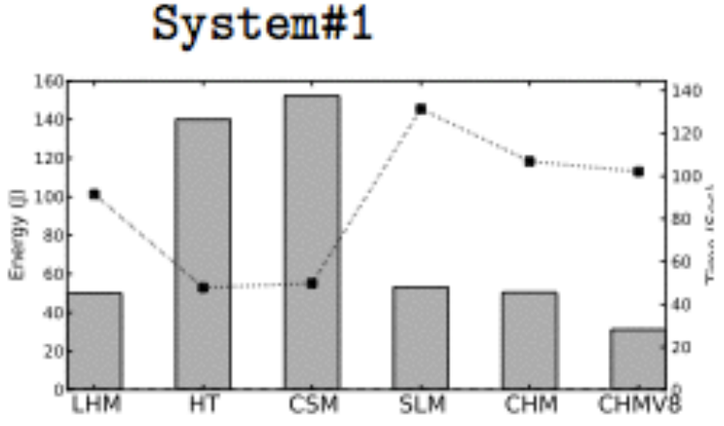
Insertion

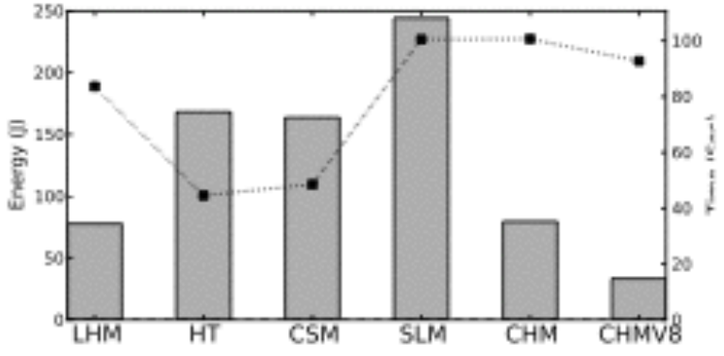


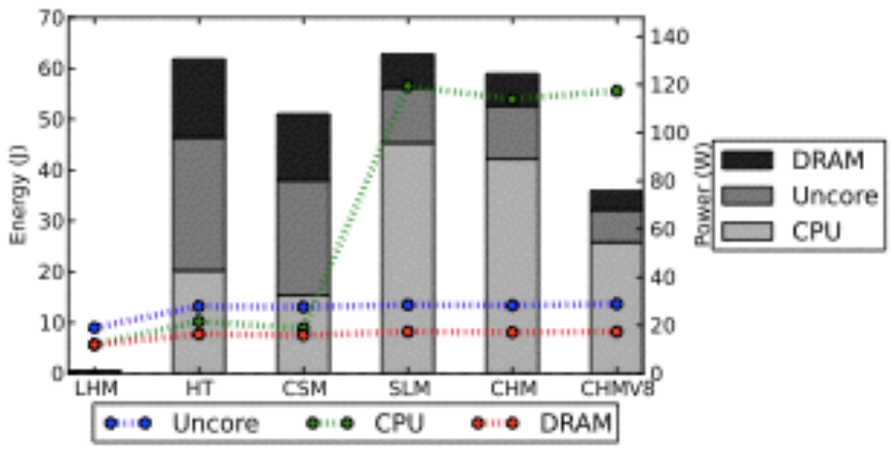
Removal

Maps

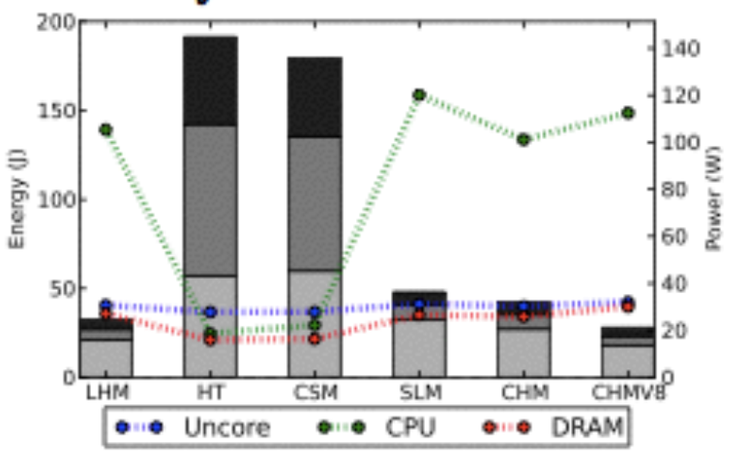




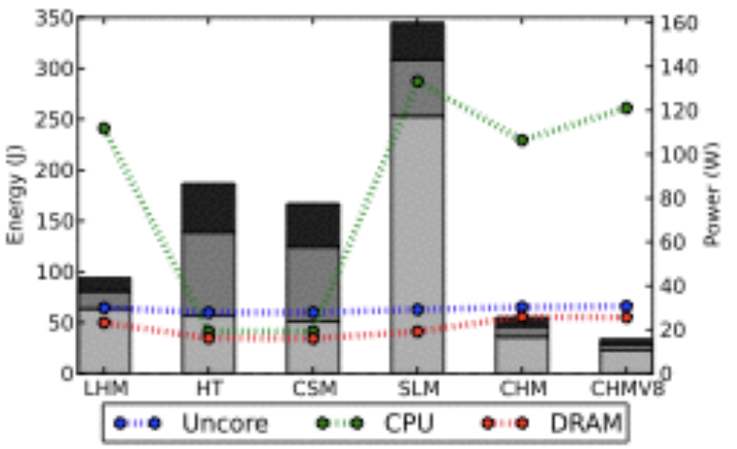




Traversal

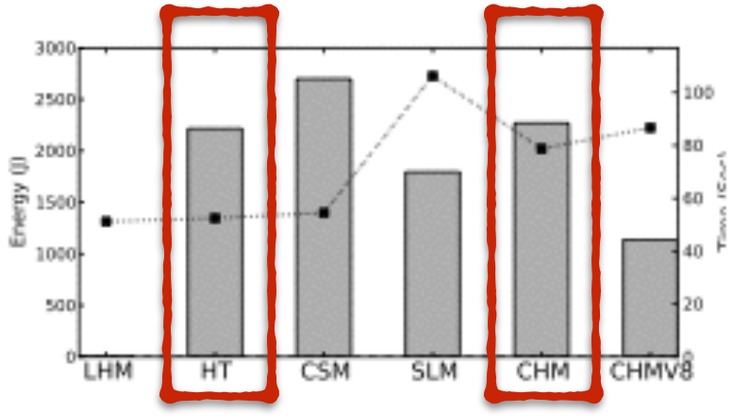


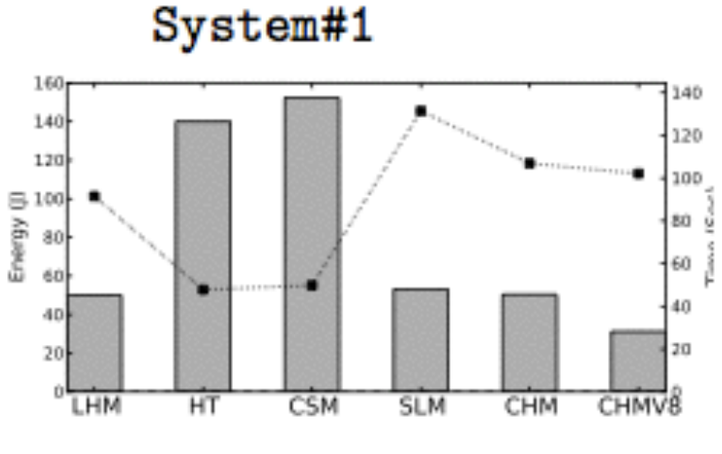
Insertion

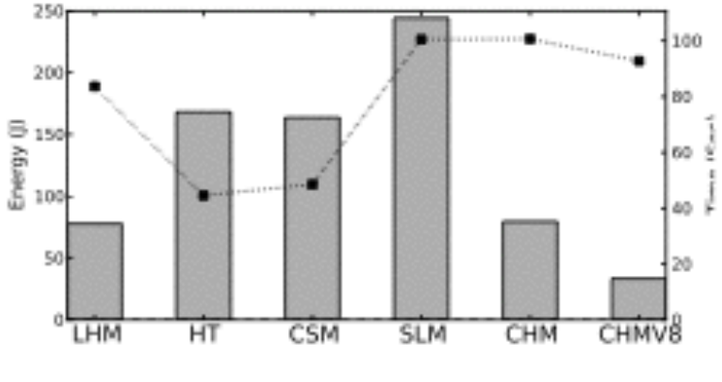


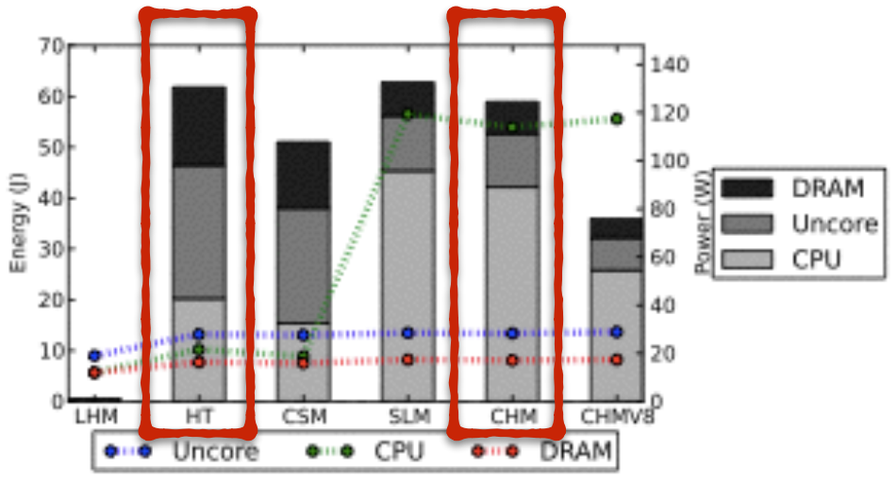
Removal

Maps

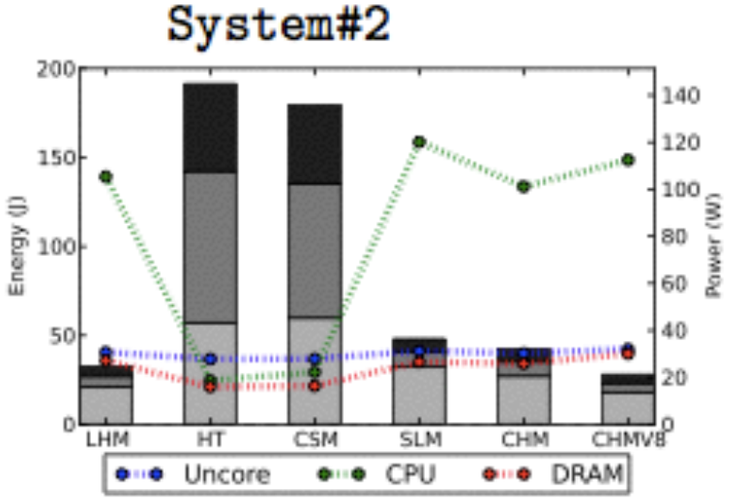




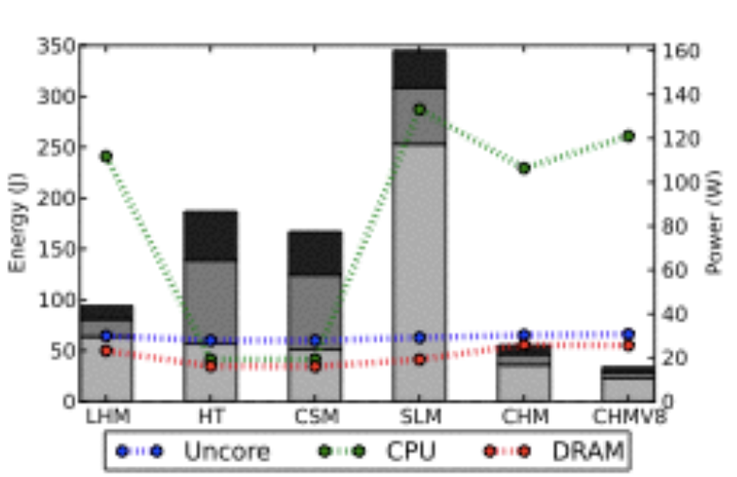




Traversal

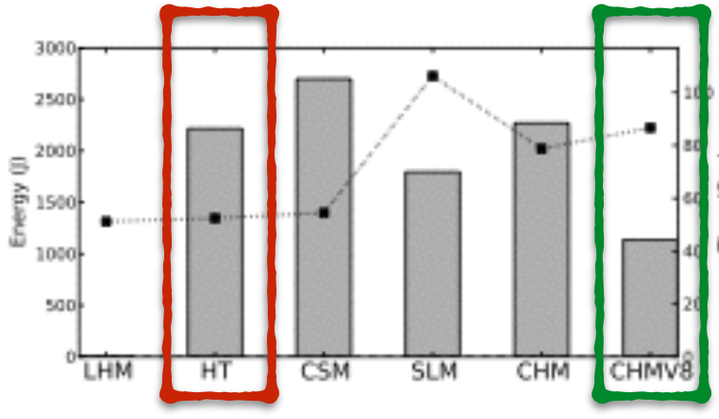


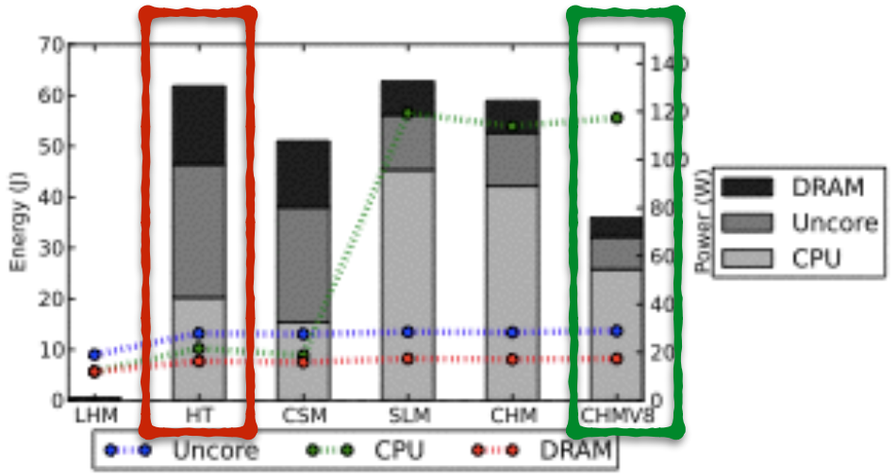
Insertion



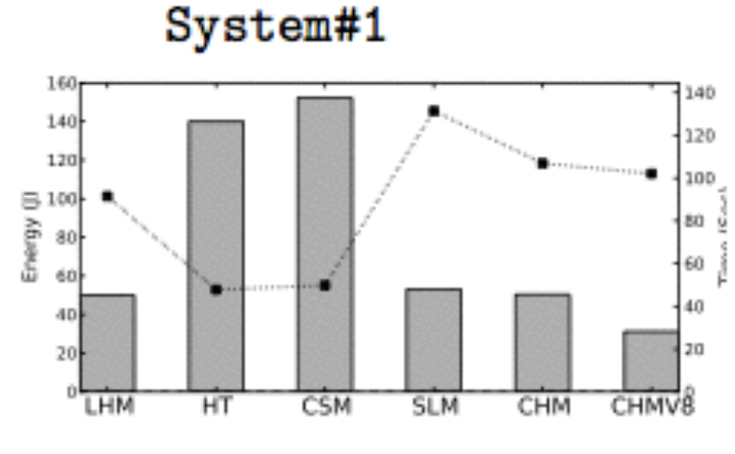
Removal

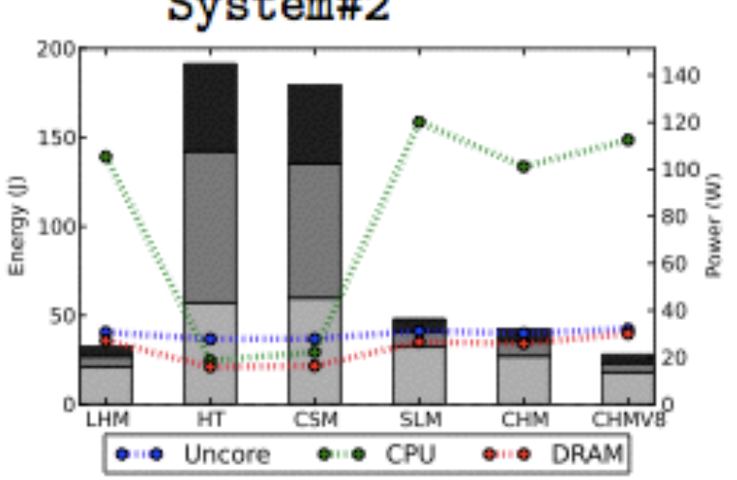
Maps



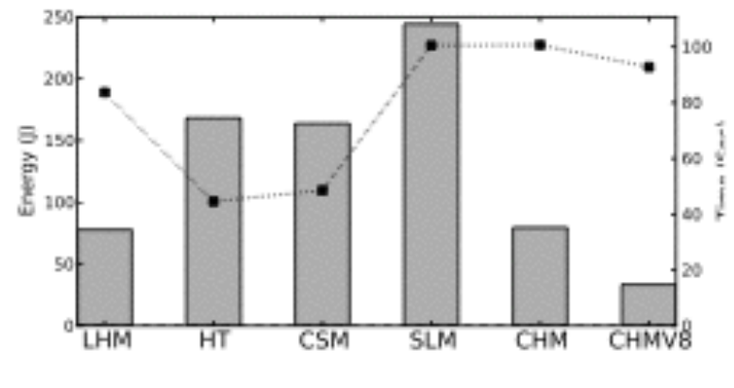


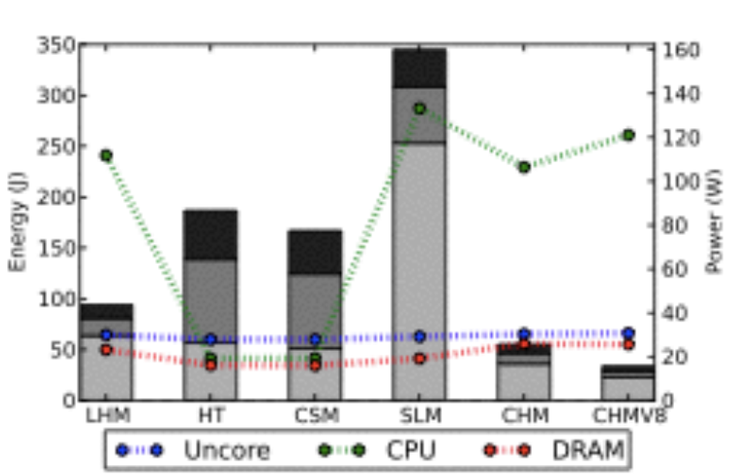
Traversal





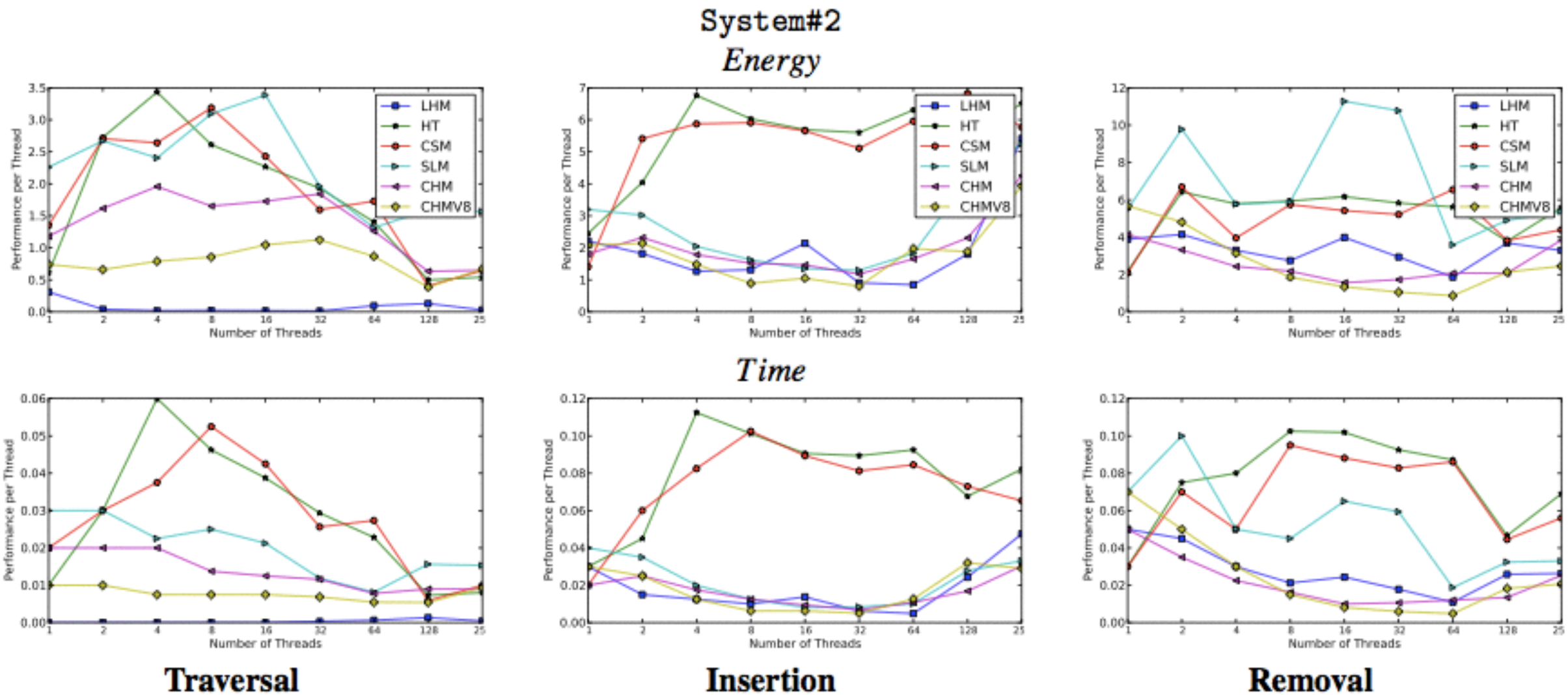
Insertion



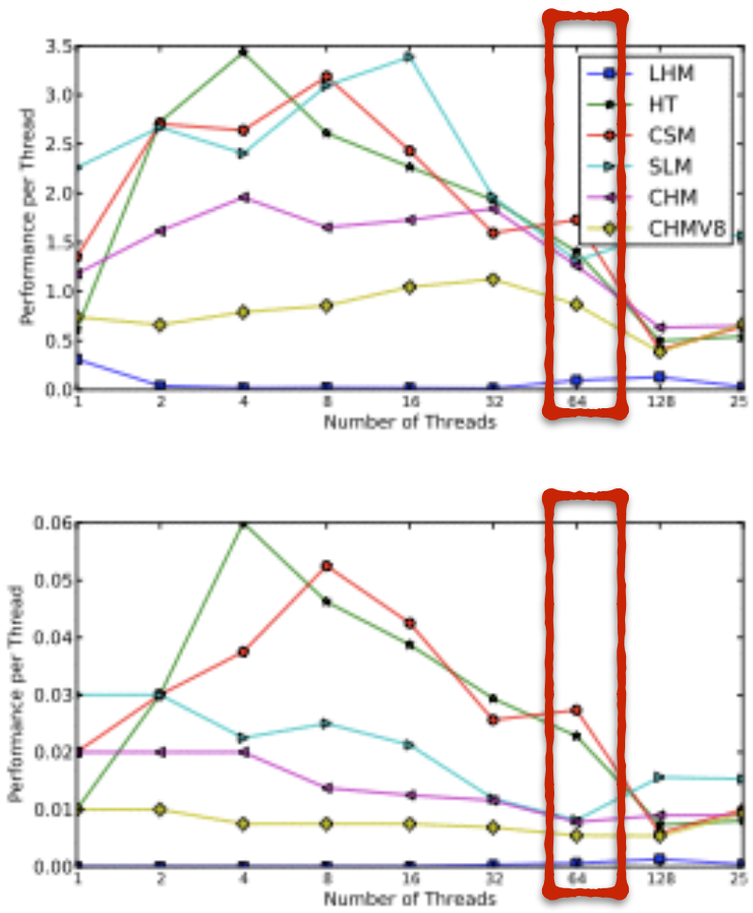


Removal

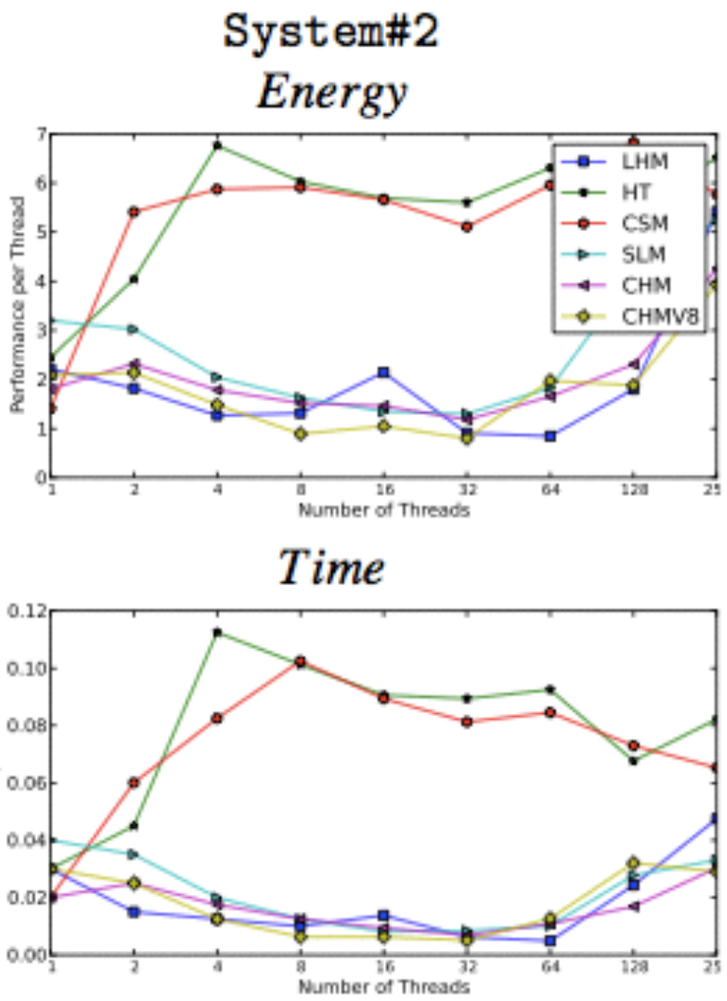
Do Maps Scale?



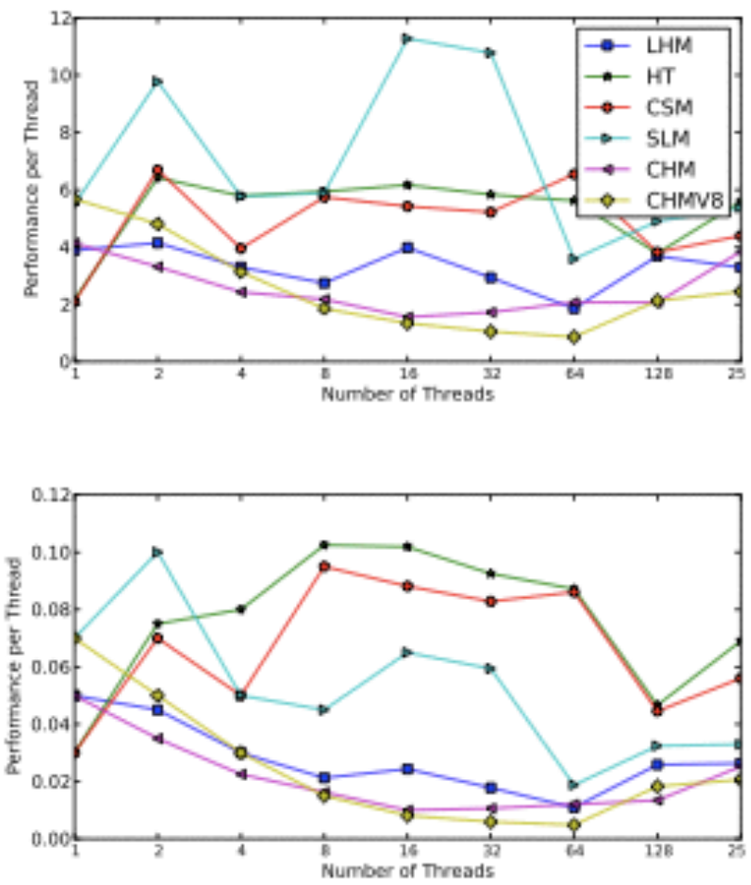
Do Maps Scale?



Traversal



Insertion

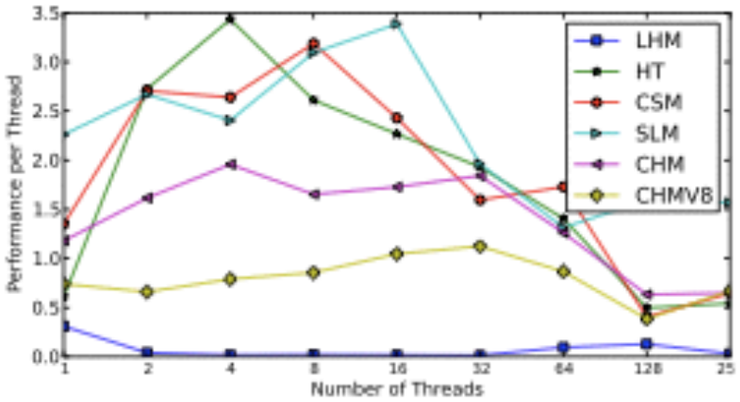


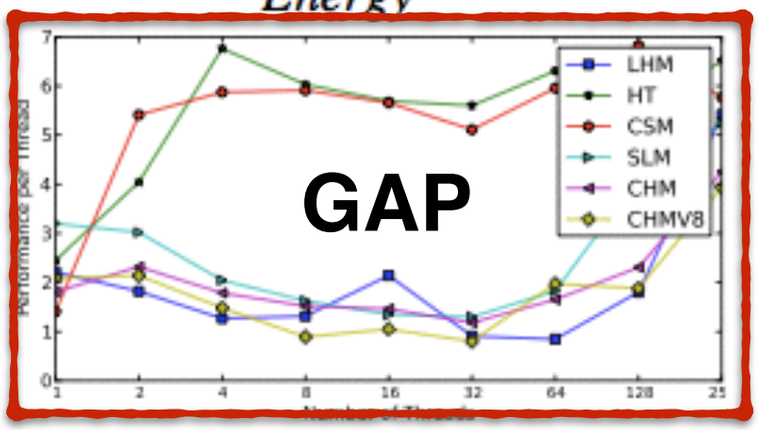
Removal

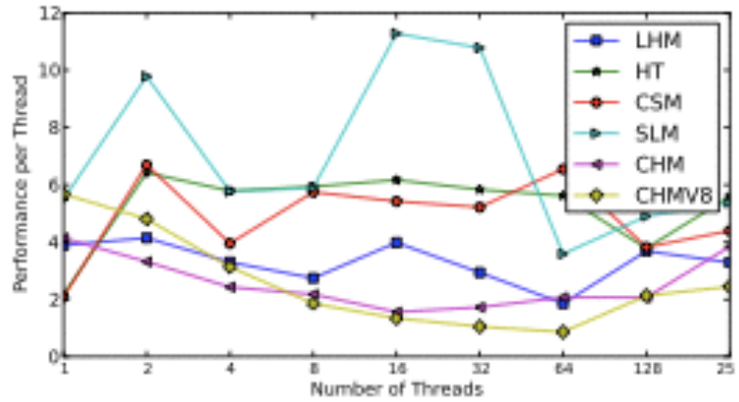
Do Maps Scale?

System#2

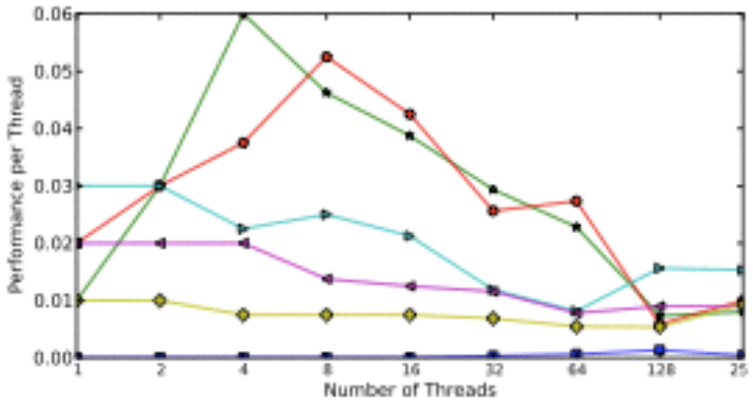
Energy



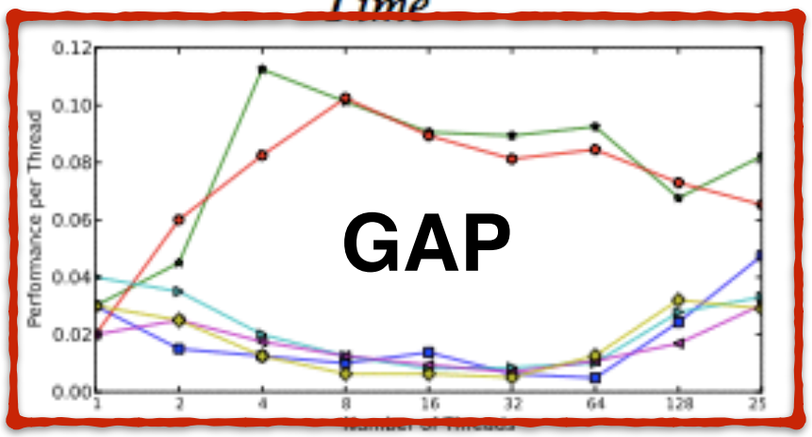




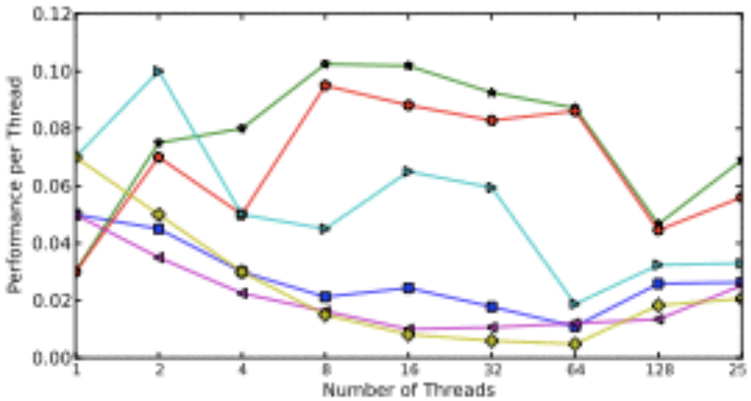
Time



Traversal

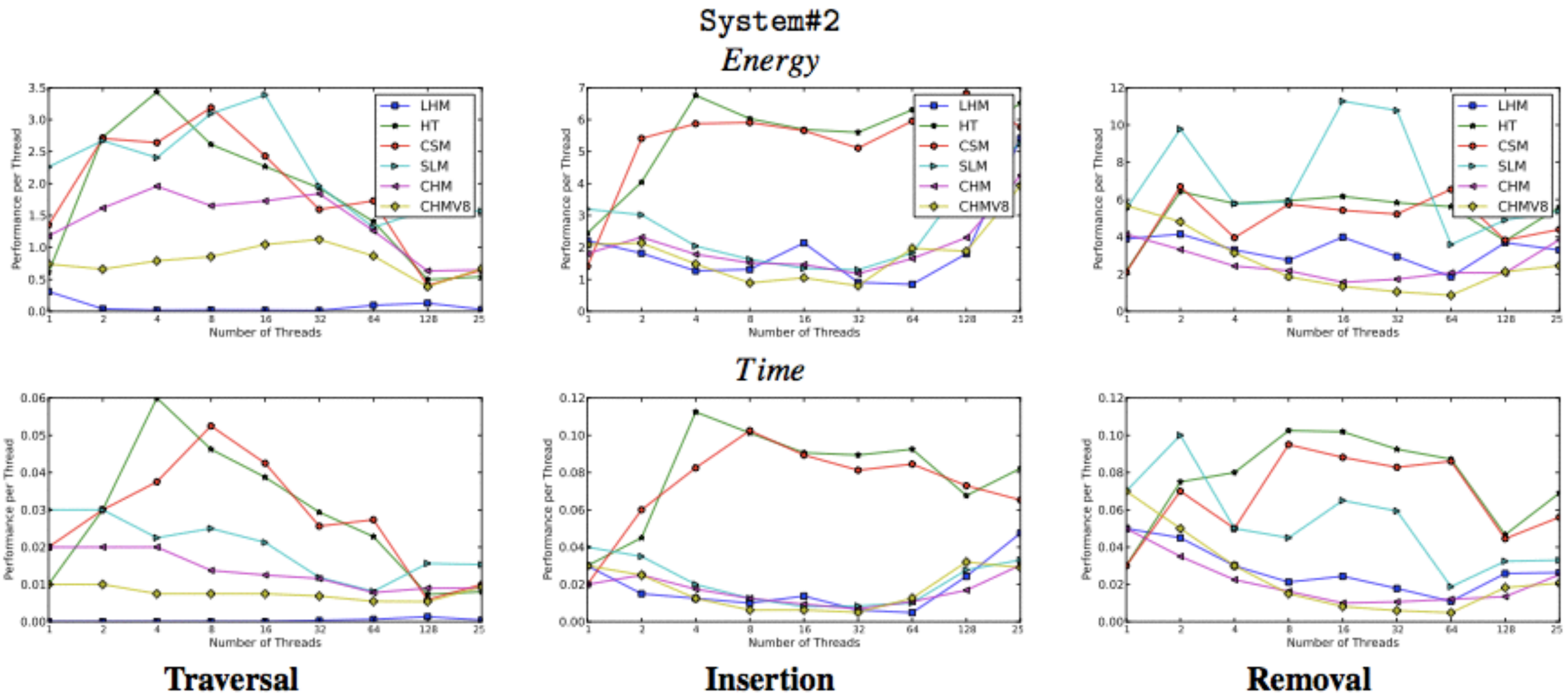


Insertion



Removal

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. <u>To characterize the energy-consumption</u> behavior of

    1. Thread-safe collections

    2. <u>Thread management techniques</u>

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

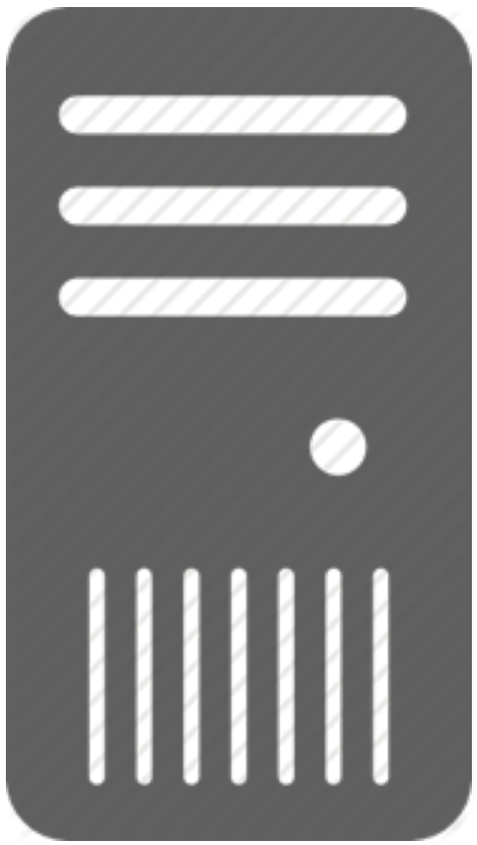
- **Embarrassingly parallel:** spectralnorm, sunflow, n-queens
- **Leaning parallel:** xalan, knucleotide, tomcat
- **Leaning serial:** mandelbrot, largestImage
- **Embarrassingly serial:** h2

Micro-benchmarks

DaCapo benchmarks

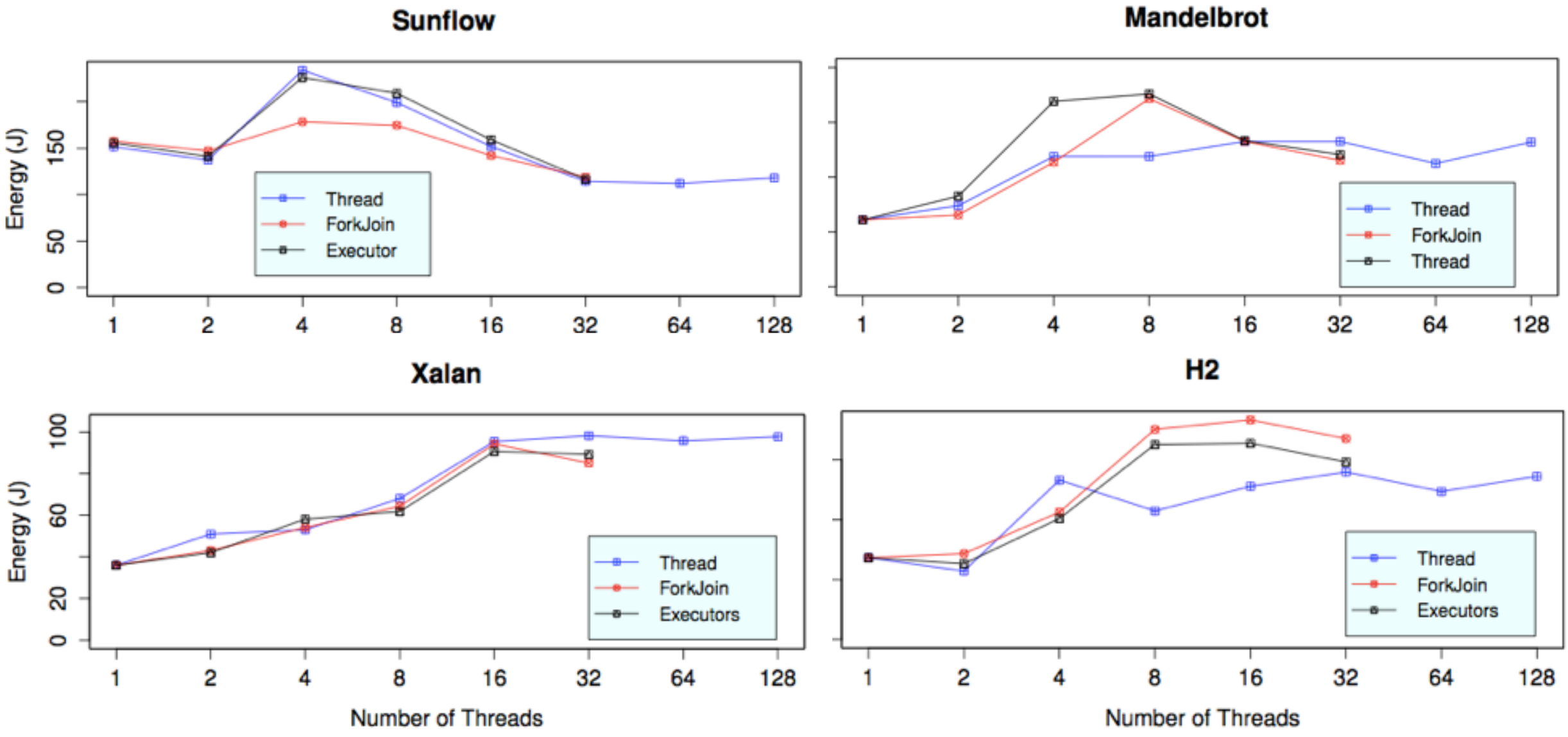
the DaCapo benchmark suite

Experimental Environment

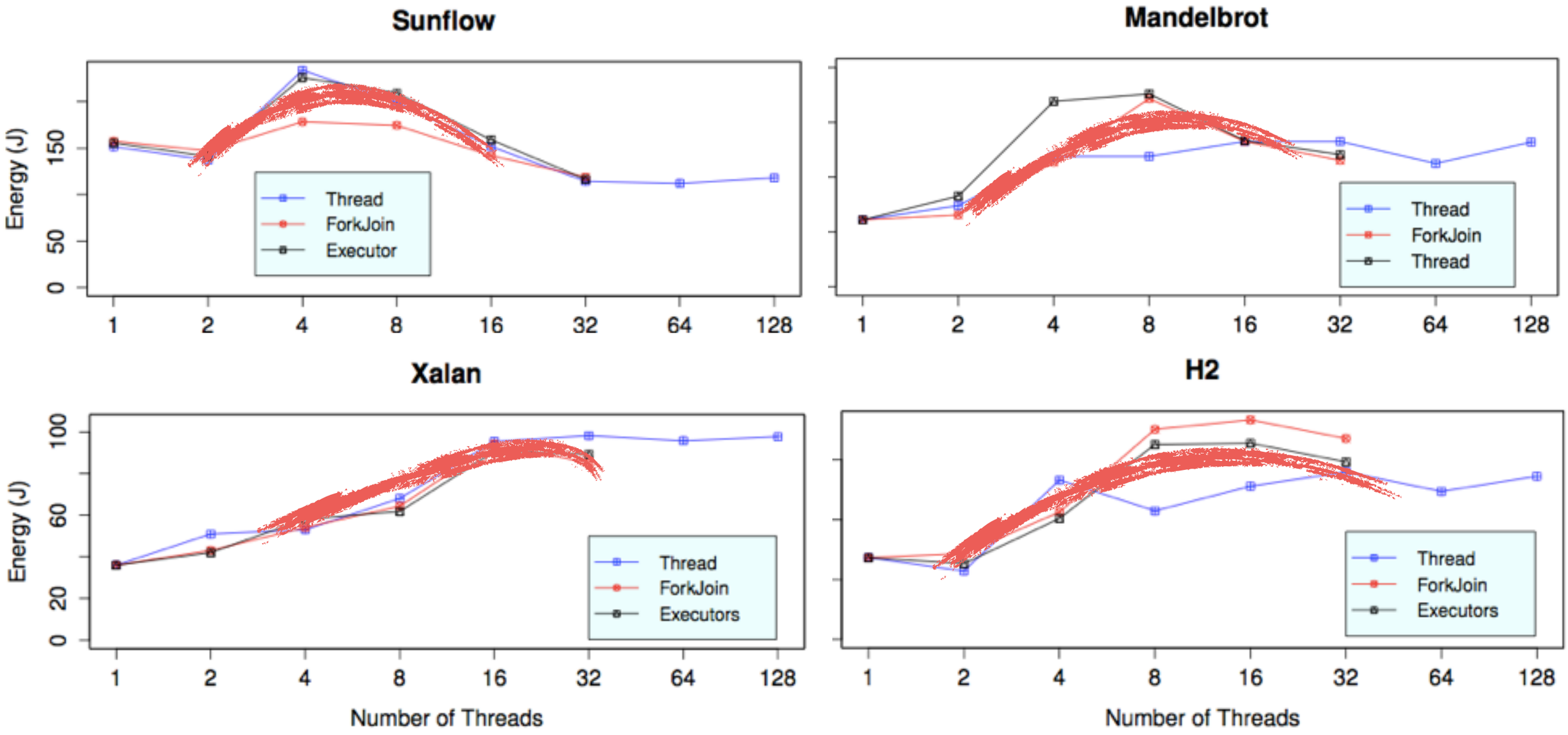


A 2×16-core **AMD CPUs**, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

Energy Consumption When Varying the Number of Threads



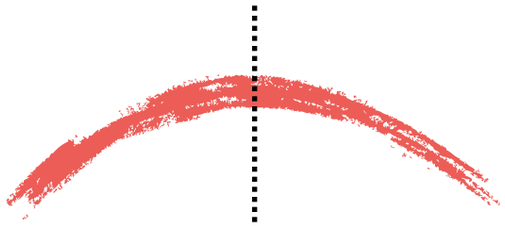
The Λ Curve


Sunflow
Energy (J)
0
50
150
Thread
ForkJoin
Executor
1
2
4
8
16
32
64
128
Mandelbrot
Thread
ForkJoin
Thread
Xalan
Energy (J)
0
20
60
100
Thread
ForkJoin
Executors
Number of Threads
H2
Thread
ForkJoin
Executors
Number of Threads

The Λ Curve

The Λ Curve

The Λ Curve



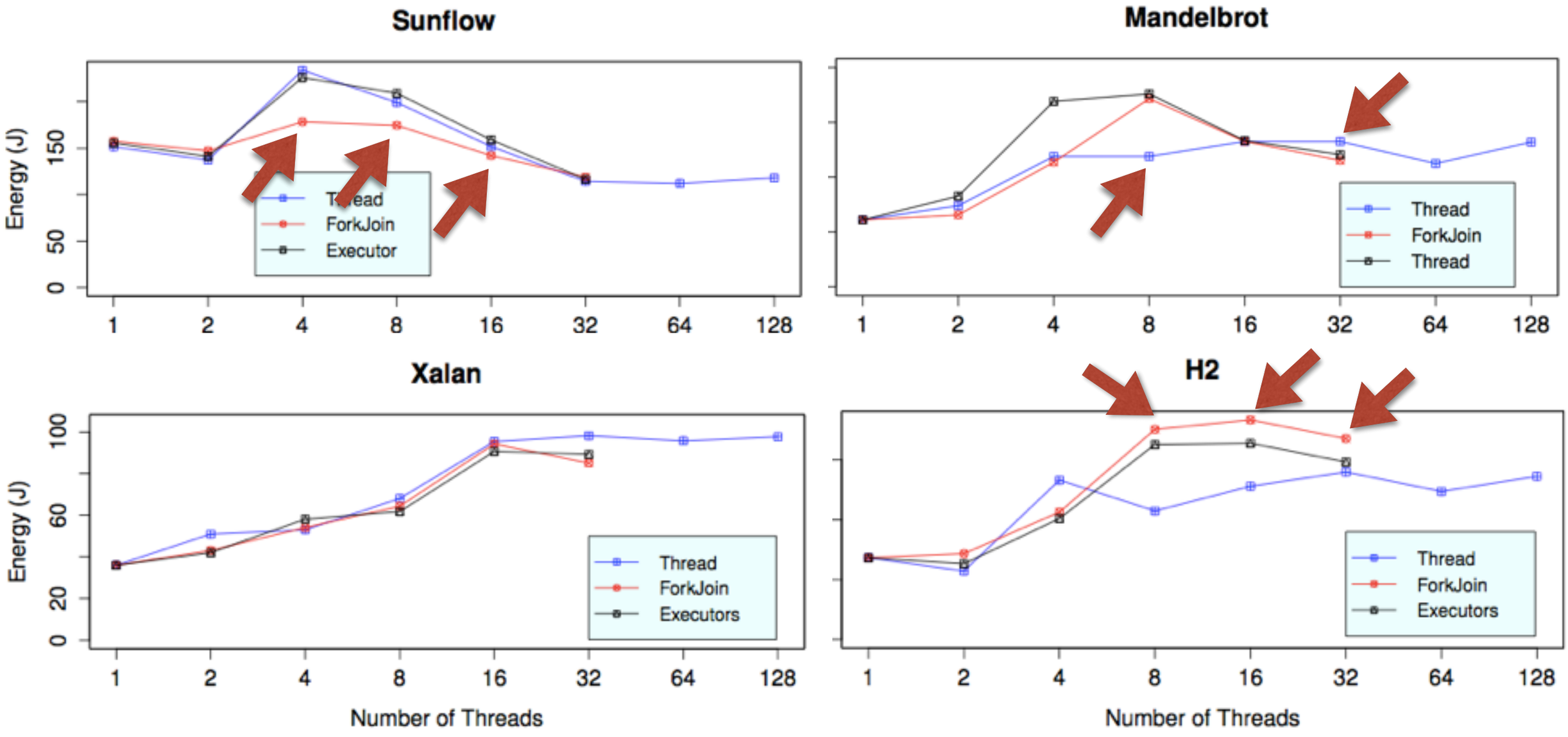
- More cores idle
- Frequency at a lower level

The Λ Curve

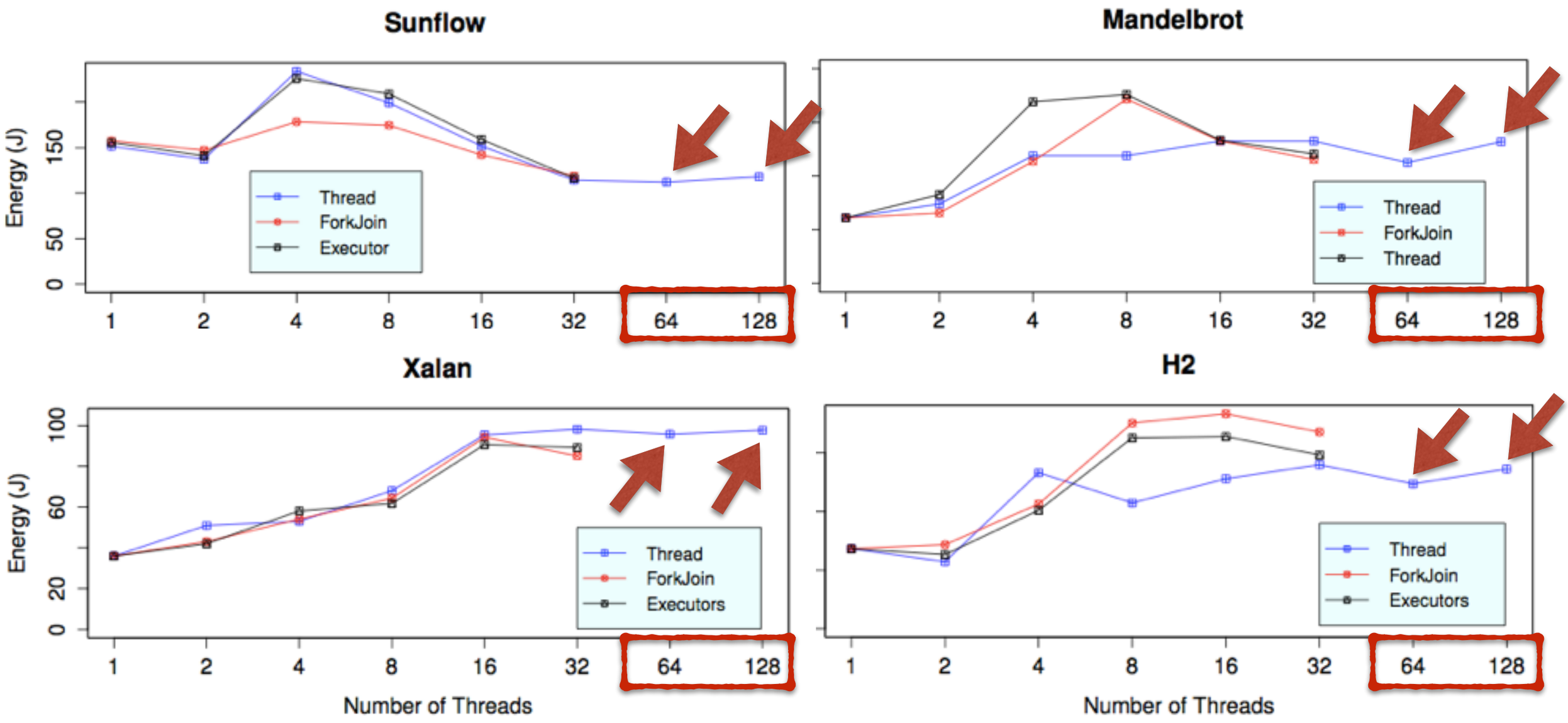
- More cores idle
- Frequency at a lower level

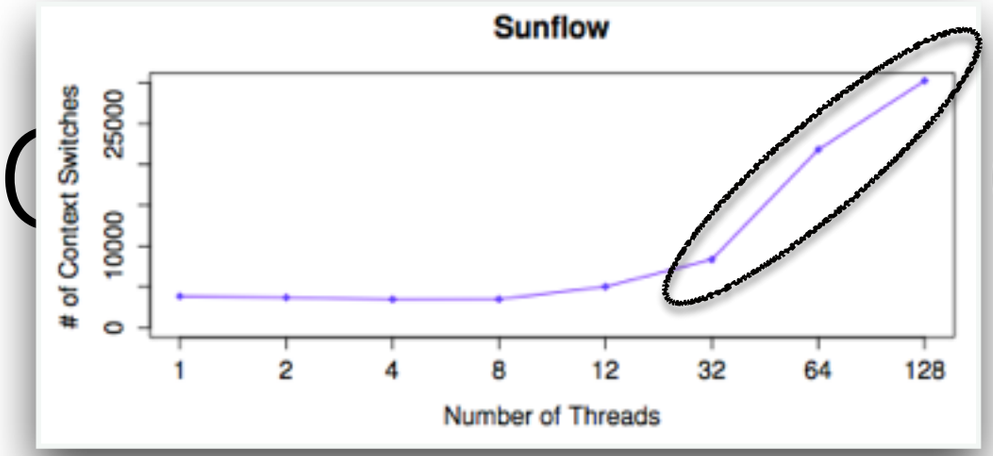
- More threads used, performance increase
- The greater the ratio between speedup and power, the steeper the \

Which programming style should I use?

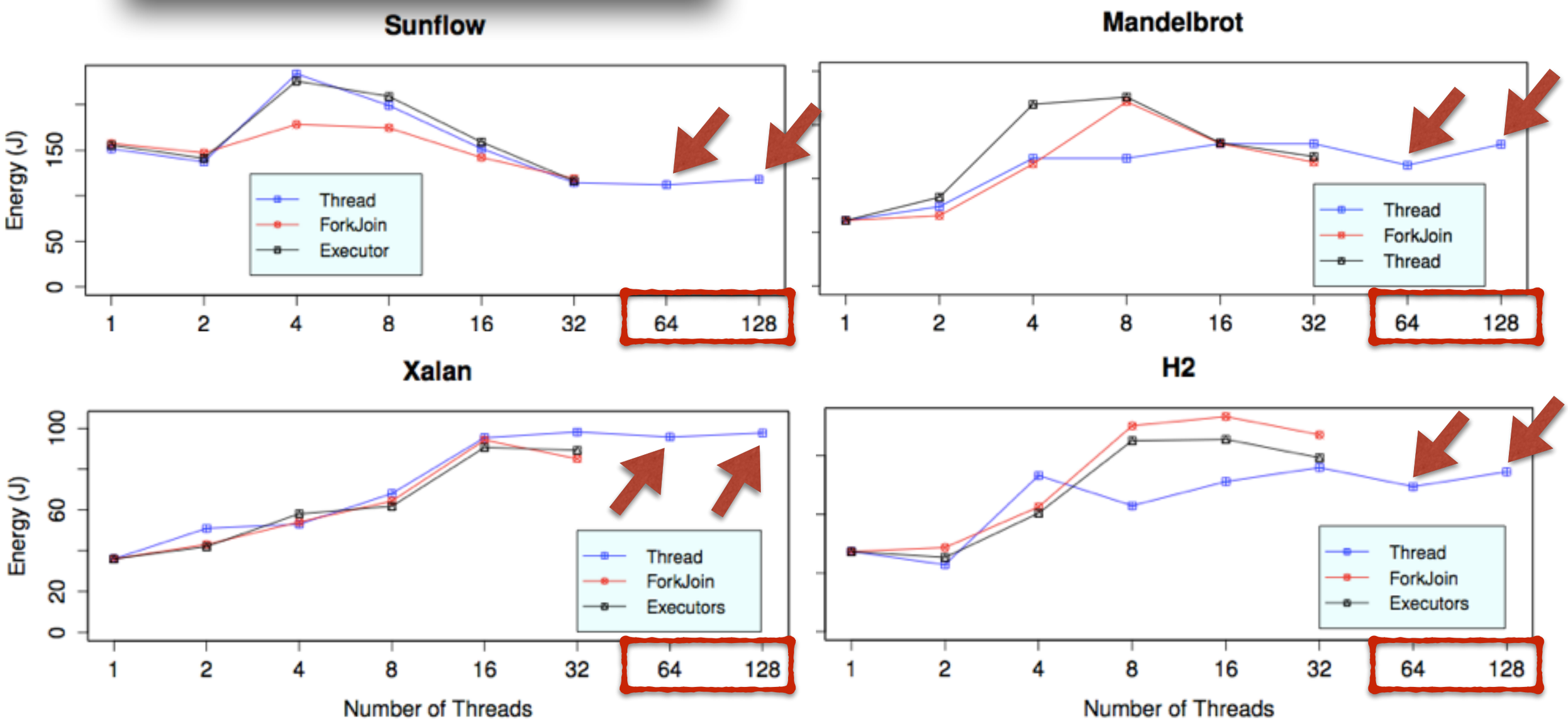


Overpopulating Cores with Threads

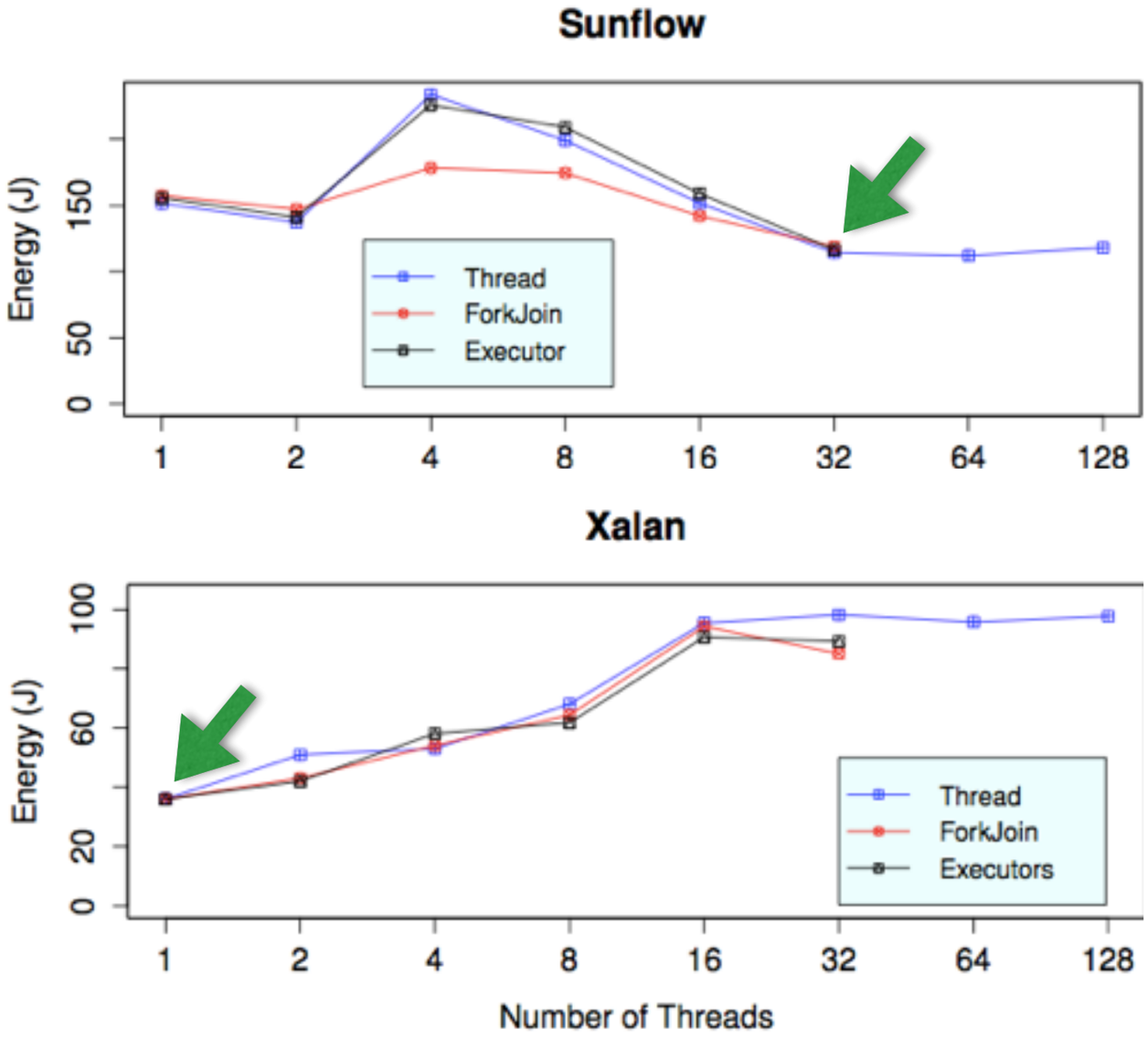


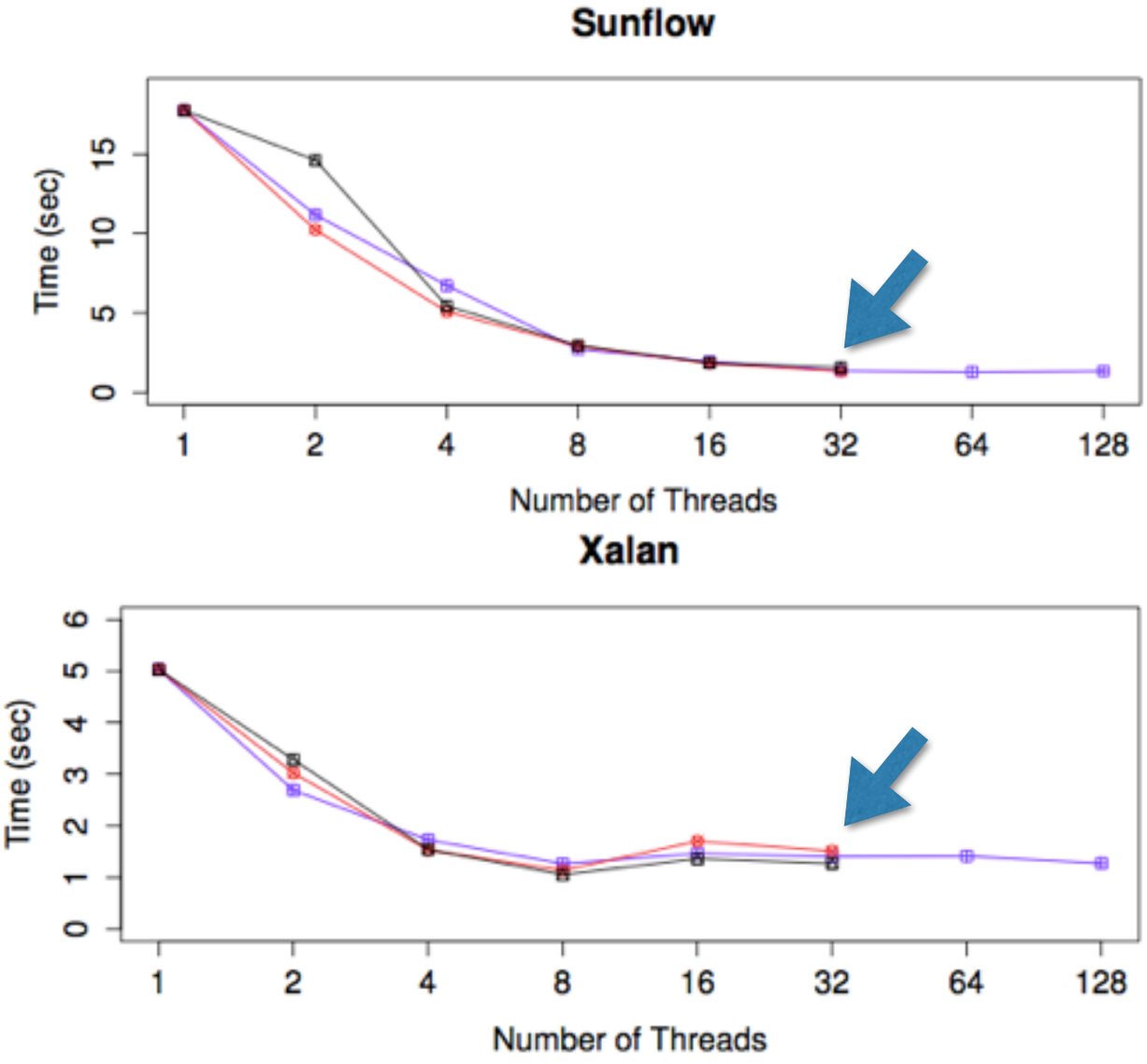


Cores with Threads



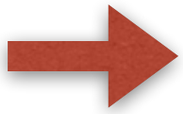
Faster ≠ Greener





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();
   }
}
```



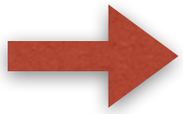
```
public void compute() {
   ...
   List<NQueensSolver> tasks = new ArrayList<>(
        size);
   for (int i = 0; i < tasks.size(); i++) {
      int[] newElements = new int[depth + 1];
      System.arraycopy(currentElements, 0,
                       newElements, 0, depth);
      tasks.add(new NQueensSolver(newElements));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

Copy

Fork/Join

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);
}
```

Copy

Fork/Join

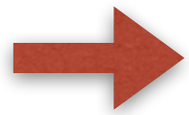
±10% of energy savings!

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)
      );
    }
  }
}
```

Copying

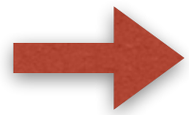
Sharing

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)
      );
    }
  }
}
```

Copying

Sharing

±15% of energy savings!

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 Re
  public Task (User[]
      int to) {}
  protected void comp
    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

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

```
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)
      );
    }
  }
}
```

Copying

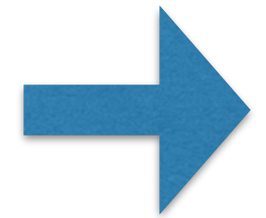
Sharing

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)
      );
    }
  }
}
```

Copying

Sharing

Solution

1. Add field variable

```
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

```
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 (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

```
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

```
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;

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

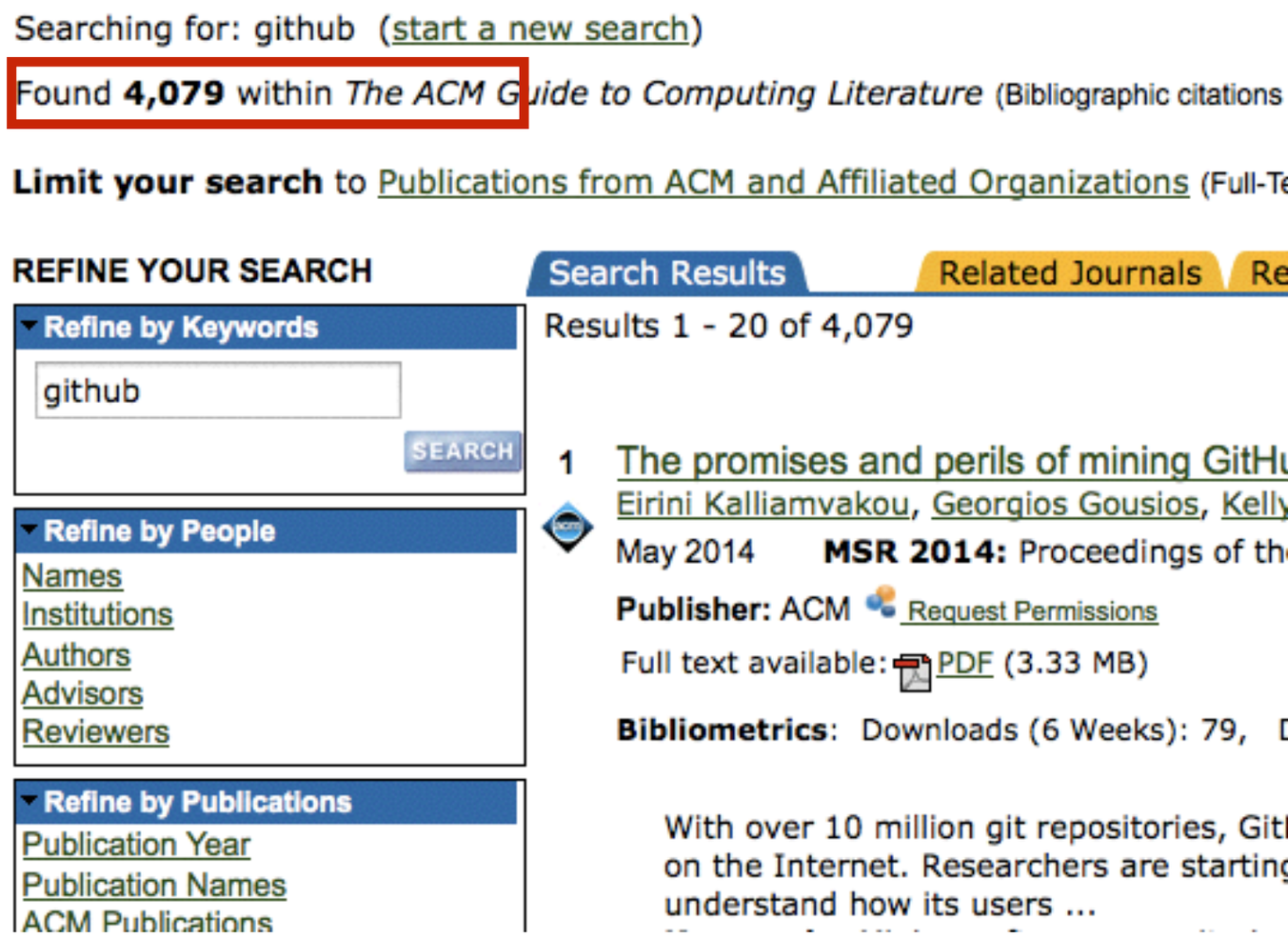
4Mi+ Users

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

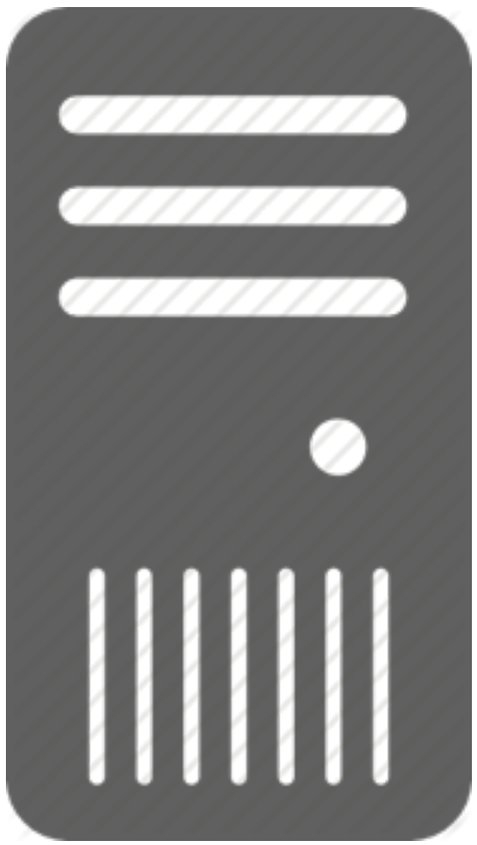
4Mi+ Users

19Mi+ Repositories

Subjects	# LoC	# Dev	Last Commit
itemupdown	4,925	1	08-10-2013
jAcer	4,476	2	12-06-2014
educational	1,323	1	05-05-2014
scalatuts	253	2	11-17-2013
knn	3,099	1	11-25-2014
netflixoss	231,361	1	09-14-2013
doms-transformers	3,714	4	06-19-2014
ForkAndJoinUtility	127	1	03-31-2013
exhibitor	15,314	14	11-08-2014
Solitaire	1,527	1	11-18-2011
javaOneBR-2012	518	1	12-02-2012
mywiki	1,920	2	10-04-2012
ejisto	12,330	1	08-06-2014
cq4j	5,815	1	10-15-2013
MagicSquares	664	1	10-25-2013

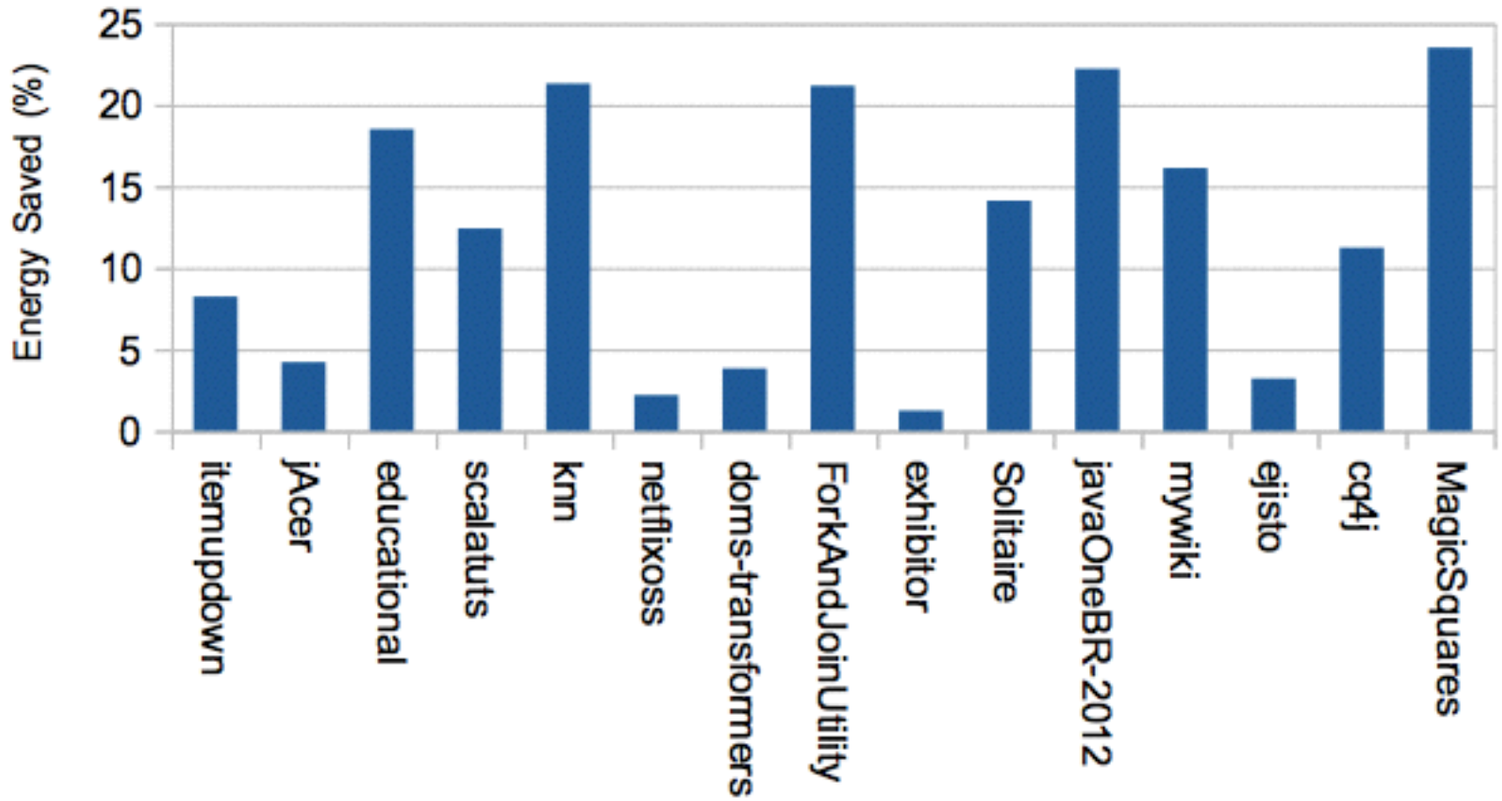
“GitHub is the largest code host on the planet with over 19.9 mi repositories.”
https://github.com/features

Experimental Environment

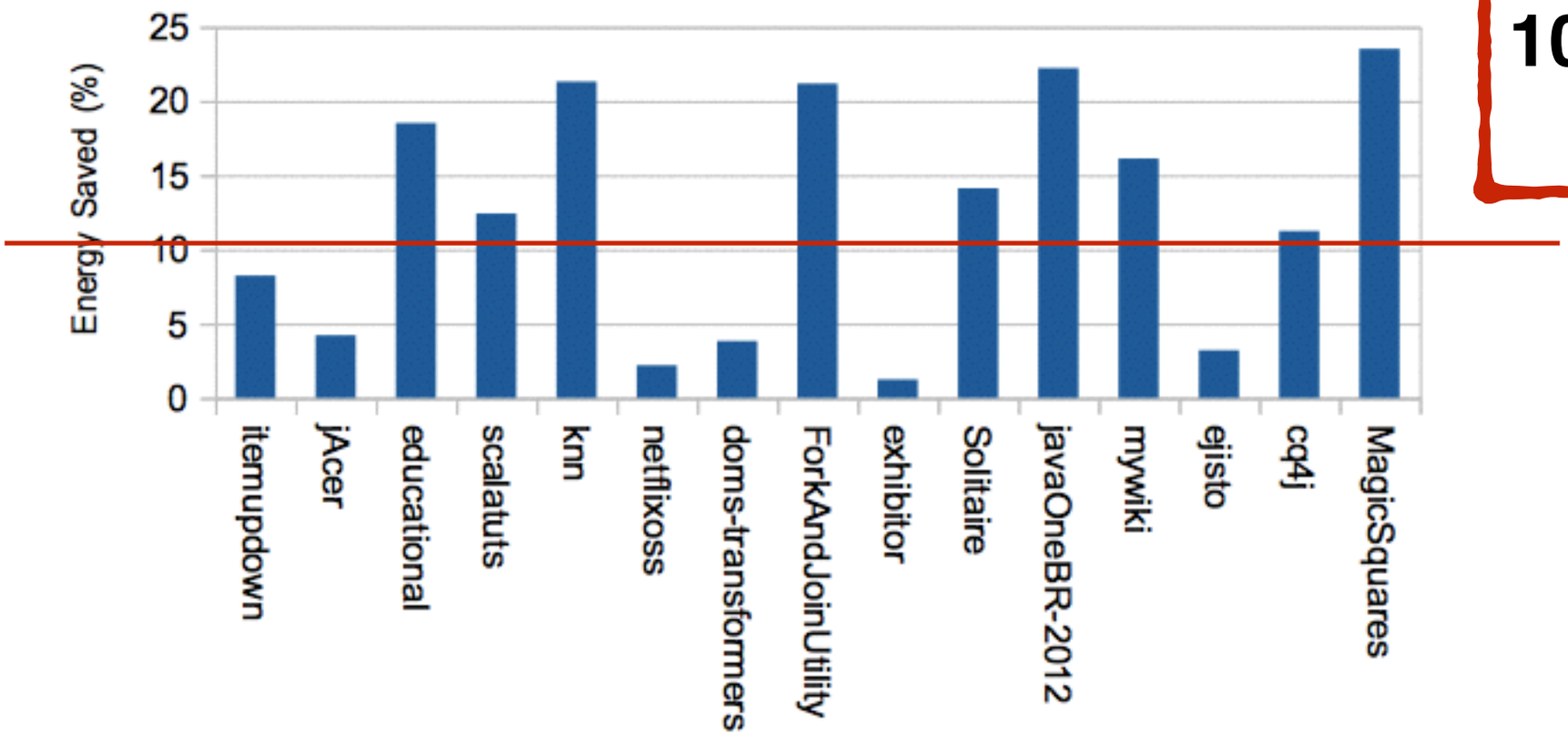


A 2×8-core (32-cores w/ hyper-threading) **Intel CPU**,running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.

Results


Energy Saved (%)
25
20
15
10
5
0
itemupdown
jAcer
educational
scalatuts
knn
netflixoss
doms-transformers
ForkAndJoinUtility
exhibitor
Solitaire
javaOneBR-2012
mywiki
ejisto
cq4j
MagicSquares

Results



9/15 crossed the **10%** energy saving!

Patches

15 Pull Requests

- Proposed — #1 **Improving ForkJoin usage for better performance** — czbabl/itemupdown
- Proposed — #206 **Improving ForkJoin usage for better performance** on Dec 5, — Netflix/exhibitor
- Merged — #1 **Improving ForkJoin usage for better performance** — toby1984/jAcer
- Proposed — #1 **Improving ForkJoin usage for better performance** on Dec 5, 20 — robitobi/Solitaire
- Proposed — #1 **Improving ForkJoin usage for better performance** — nikkrichko/Educational
- Merged — #1 **Improving ForkJoin usage for better performance** on Dec 5, 20 — mariofts/javaOneBR-2012
- Merged — #1 **Improving ForkJoin usage for better performance** — mayukh42/scalatuts
- Proposed — #1 **Improving ForkJoin usage for better performance** on Dec 5, 20 — cacling/mywiki
- Merged — #1 **Improving ForkJoin usage for better performance** — tfreese/knn
- Merged — #14 **Improving ForkJoin usage for better performance** on Dec 5, 2 — ejisto/ejisto
- Closed — #1 **Improving ForkJoin usage for better performance** — TarantulaTechnology/netflixoss
- Proposed — #1 **Improving ForkJoin usage for better performance** on Dec 5, 20 — ctpahhik/cq4j
- Proposed — #1 **Improving ForkJoin usage for better performance** — statsbiblioteket/doms-transformers
- Merged — #1 **Updating ForkJoin usage for better performance and energy e** — cjlarose/MagicSquares
- Merged — #1 **Improving ForkJoin usage for better performance** on Dec 5, 2014 — 2sbsbsb/ForkAndJoinUtility

Patches

15 Pull Requests

Proposed #1 **Improving ForkJoin usage for better performance** czbabl/itemupdown

Merged #1 **Improving ForkJoin usage for better performance** toby1984/jAcer

Proposed #1 **Improving ForkJoin usage for better performance** nikkrichko/Educational

Merged #1 **Improving ForkJoin usage for better performance** mayukh42/scalatuts

Merged #1 **Improving ForkJoin usage for better performance** tfreese/knn

Closed #1 **Improving ForkJoin usage for better performance** TarantulaTechnology/netflixoss

Proposed #1 **Improving ForkJoin usage for better performance** statsbiblioteket/doms-transformers

Merged #1 **Improving ForkJoin usage for better performance** on Dec 5, 2014 2sbsbsb/ForkAndJoinUtility

Proposed #206 **Improving ForkJoin** Netflix/exhibitor **e** on Dec 5,

Proposed #1 **Improving ForkJoin u** robitobi/Solitaire n Dec 5, 20

Merged #1 **Improving ForkJoin u** mariofts/javaOneBR-2012 n Dec 5, 20

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5, 20 cacling/mywiki

Merged #14 **Improving ForkJoin usage for better performance** on Dec 5, 2 ejisto/ejisto

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5, 20 ctpahhik/cq4j

Merged #1 **Updating ForkJoin usage for better performance and energy e** cjlarose/MagicSquares

7/9 of projects that **replied** have **accepted**

Patches

15 Pull Requests

Proposed #1 **Improving ForkJoin usage for better performance** czbabl/itemupdown

Merged #1 **Improving ForkJoin usage for better performance** toby1984/jAcer

Proposed #1 **Improving ForkJoin usage for better performance** nikkrichko/Educational

Merged #1 **Improving ForkJoin usage for better performance** mayukh42/scalatuts

Merged #1 **Improving ForkJoin usage for better performance** on Dec 5, 2014 2sbsbsb/ForkAndJoinUtility

Proposed #206 **Improving ForkJoin** Netflix/exhibitor on Dec 5,

Proposed #1 **Improving ForkJoin u** robitobi/Solitaire n Dec 5, 20

Merged #1 **Improving ForkJoin u** mariofts/javaOneBR-2012 n Dec 5, 20

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5, 20 cacling/mywiki

Merged #14 **Improving ForkJoin usage for better performance** on Dec 5, 2

Close e on Dec 5, 20

Propos and energy e

mariofts commented on Dec 8, 2014 — Owner

Hello **@gustavopinto**

Thanks for the improvement, I didn't notice that. Can you share your work after you finish it?

7/9 of projects that **replied** have **accepted**

Patches

15 Pull Requests

Proposed #1 **Improving ForkJoin usage for better performance** czbabl/itemupdown

Merged #1 **Improving ForkJoin usage for better performance** toby1984/jAcer

Proposed #1 **Improving ForkJoin usage for better performance** nikkrichko/Educational

Merged #1 **Improving ForkJoin usage for better performance** mayukh42/scalatuts

Merged #1 **Improving ForkJoin usage for better performance** 2sbsbsb/ForkAndJoinUtility

Proposed #206 **Improving ForkJoin** Netflix/exhibitor on Dec 5,

Proposed #1 **Improving ForkJoin u** robitobi/Solitaire n Dec 5, 20

Merged #1 **Improving ForkJoin u** mariofts/javaOneBR-2012 n Dec 5, 20

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5, 20 cacling/mywiki

Merged #14 **Improving ForkJoin usage for better performance** on Dec 5, 2

Close e on Dec 5, 20

Propos and energy e

mariofts commented on Dec 8, 2014 — Owner

Hello **@gustavopinto**

Thanks for the improvement, I didn't notice that. Can you share your work after you finish it?

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 researc articles related to energy and power. They select 16 papers. Six categories are created from those papers, and f 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 questions is: When to refactor?

Threads

ForkJoin

Conclusions

However, the questions is: When to refactor?

Threads

ForkJoin

Conclusions

However, the questions is: When to refactor?

Non Thread-Safe
Data Structures

Thread-Safe
Data Structures

Conclusions

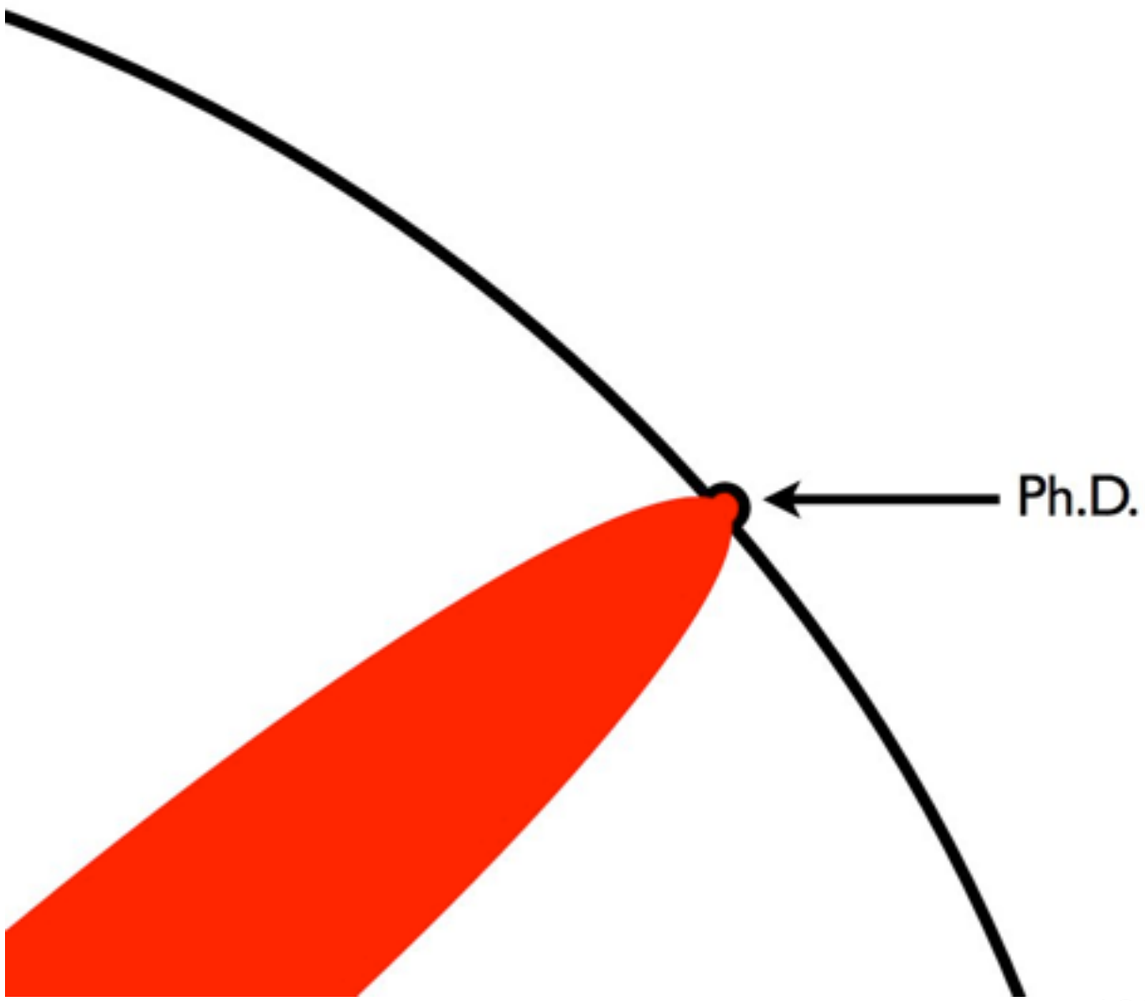
However, the questions is: When to refactor?

Non Thread-Safe
Data Structures

Thread-Safe
Data Structures

Conclusions

- This thesis just scratched the surface



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