Understanding Energy Behaviors of Thread Management Constructs

Gustavo Pinto[1] Fernando Castor[1] David Liu[2]

{ghlp, castor}@cin.ufpe.br[1]

davidL@binghamton.edu[2]

VIRTUS IMPAVIDA

Centro de Informática UFPE

BINGHAMTON UNIVERSITY

STATE UNIVERSITY OF NEW YORK

Motivation

- **First,** the proliferation of multicore CPUs
- **Second**, the prevalence of multi-threaded programs

The Problem

- **First,** more cores → more power consumed
- **Second,** little is known about **energy behaviors** of multi-threaded program on the **application and programming language** level

This Talk

1. **programming abstractions** of thread management on energy efficiency
2. **programmer choices** of thread management on energy efficiency

for Java multi-threaded programs

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

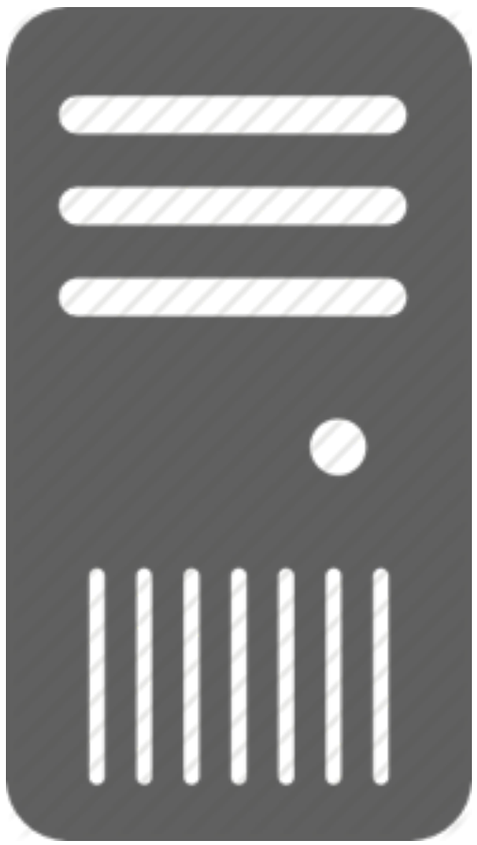
Benchmarks

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

Micro-benchmarks

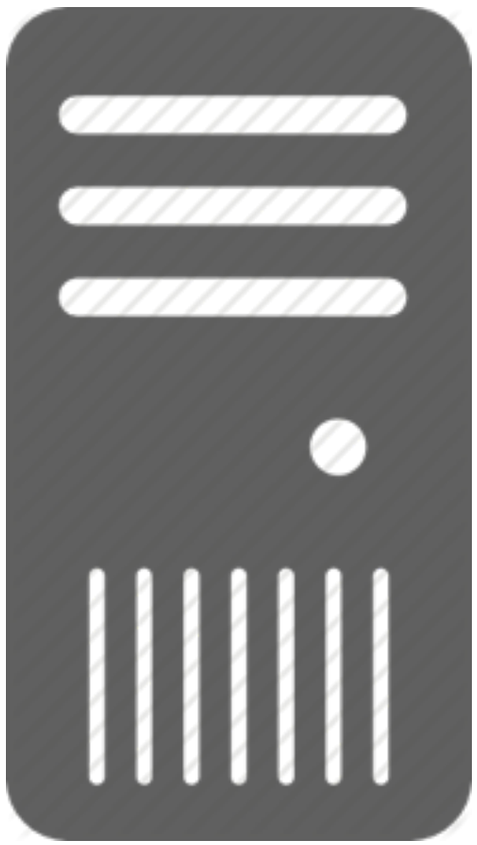
DaCapo benchmarks

Experimental Environment

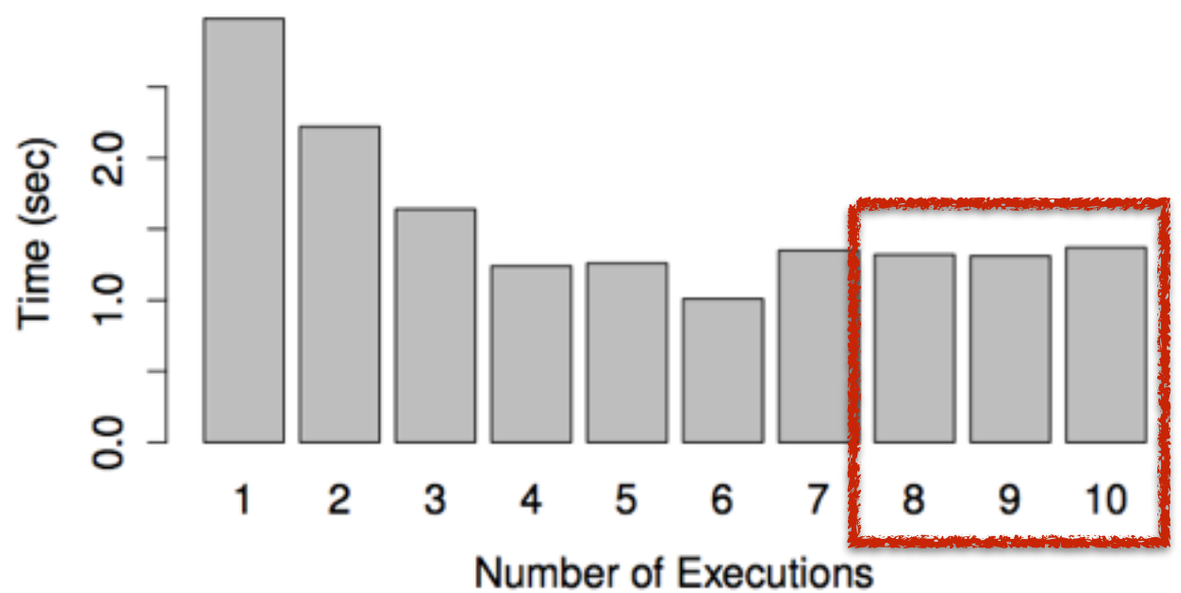


A 2×16-core AMD CPUs, running Debian Linux, 64GB of memory, JDK version 1.7.0 11, build 21, “ondemand” governor

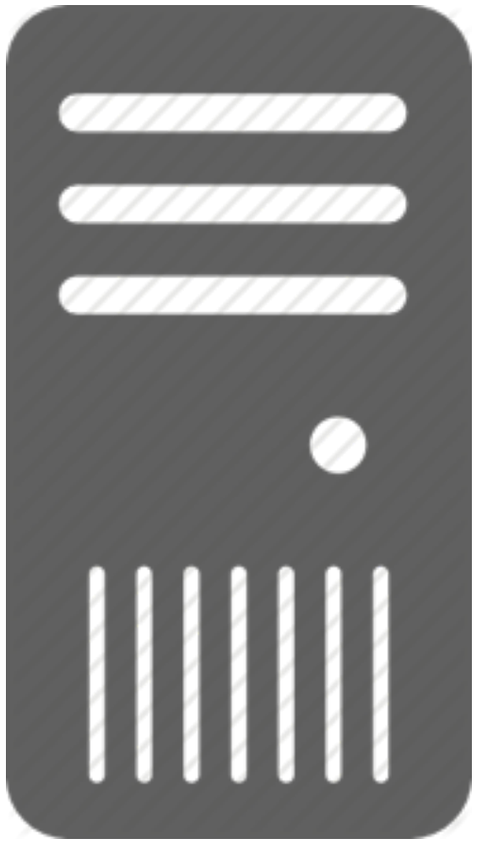
Experimental Environment



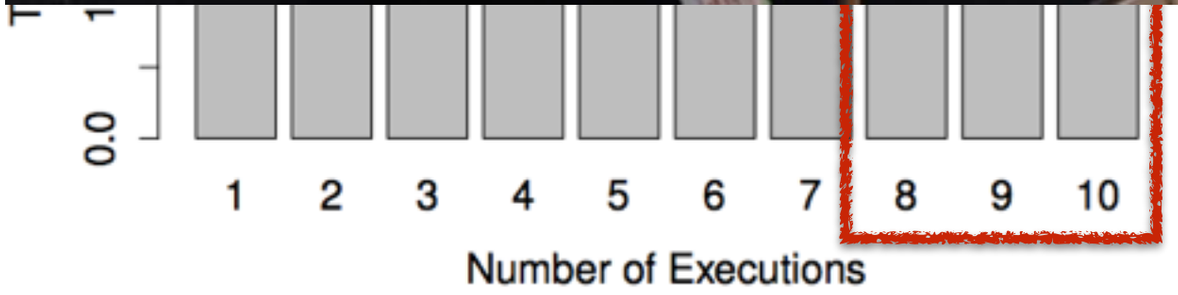
A 2×16-core AMD CPUs, running Debian Linux, 64GB of memory, JDK version 1.7.0 11, build 21, “ondemand” governor



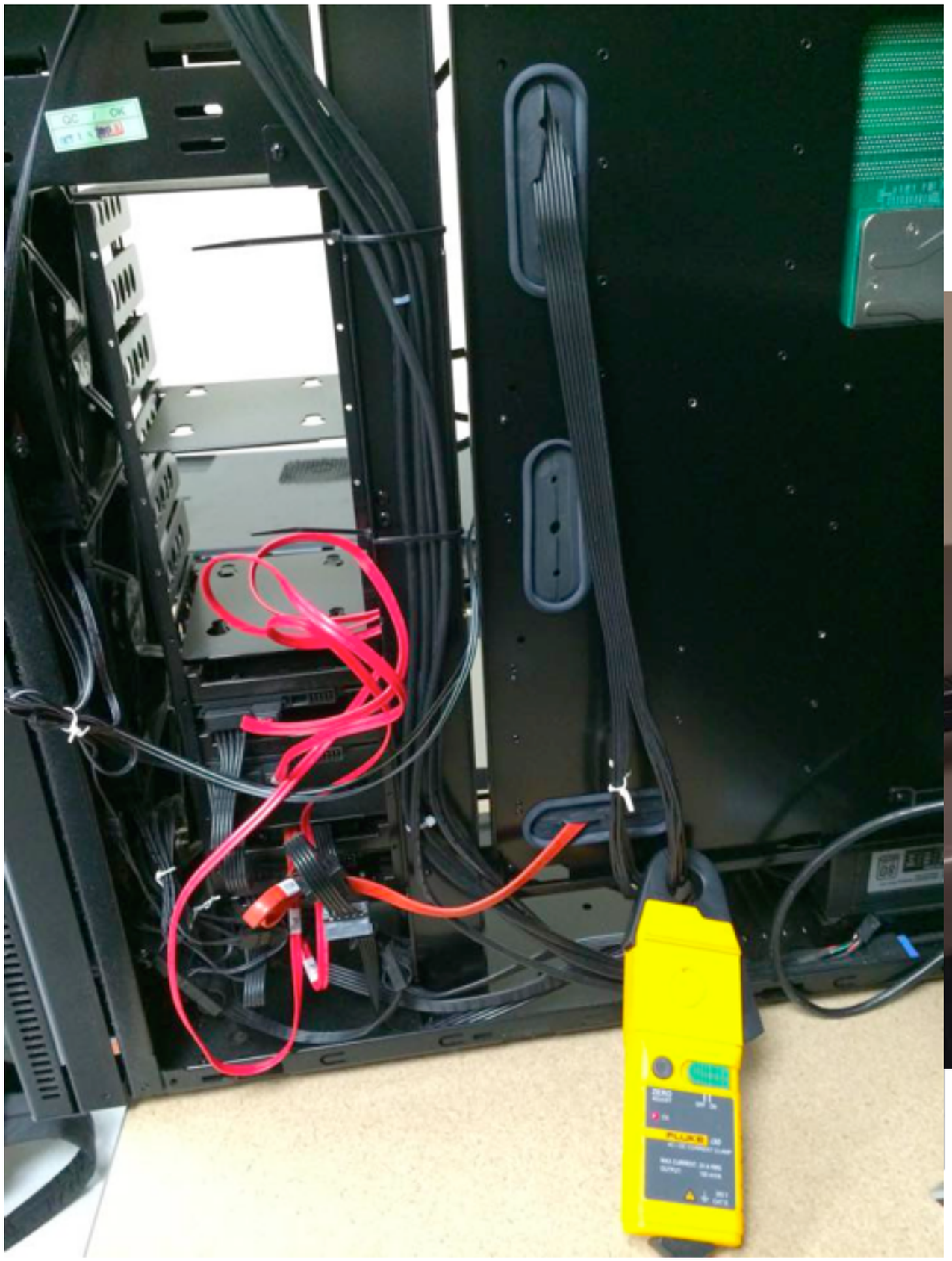
Experimental Environment

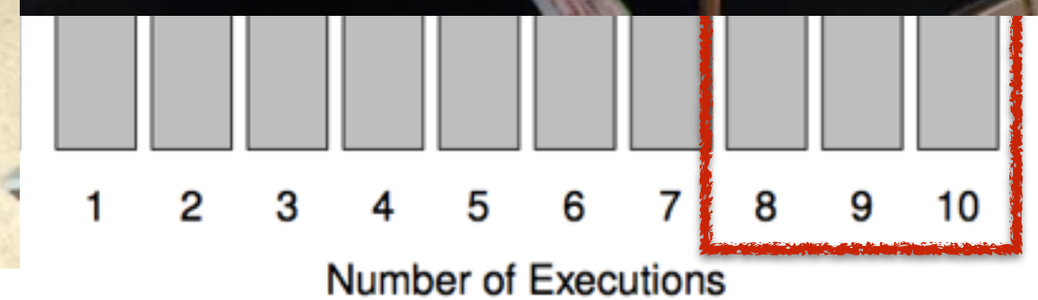


A 2×16-core
Linux, 64GB
11, build 21

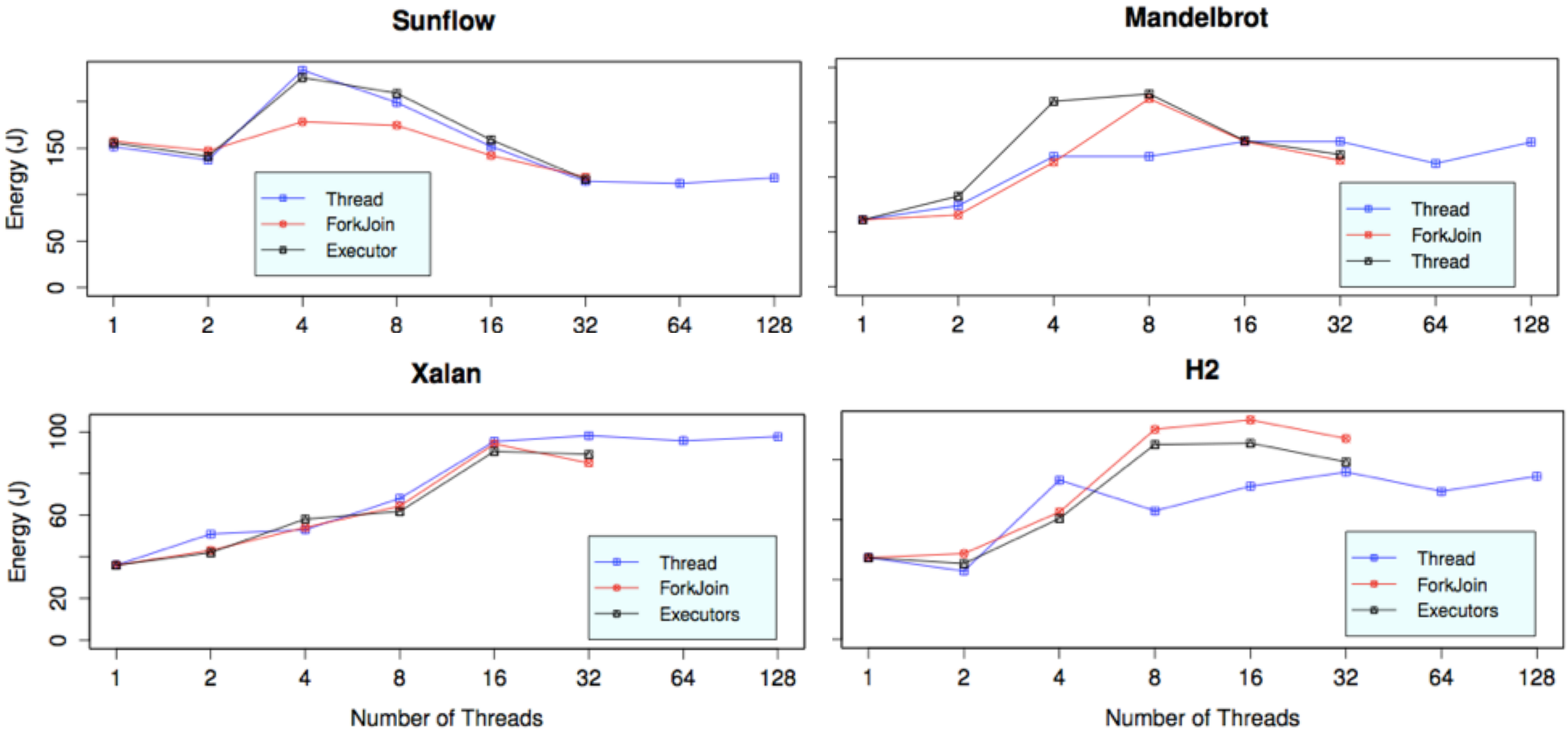


l Environment

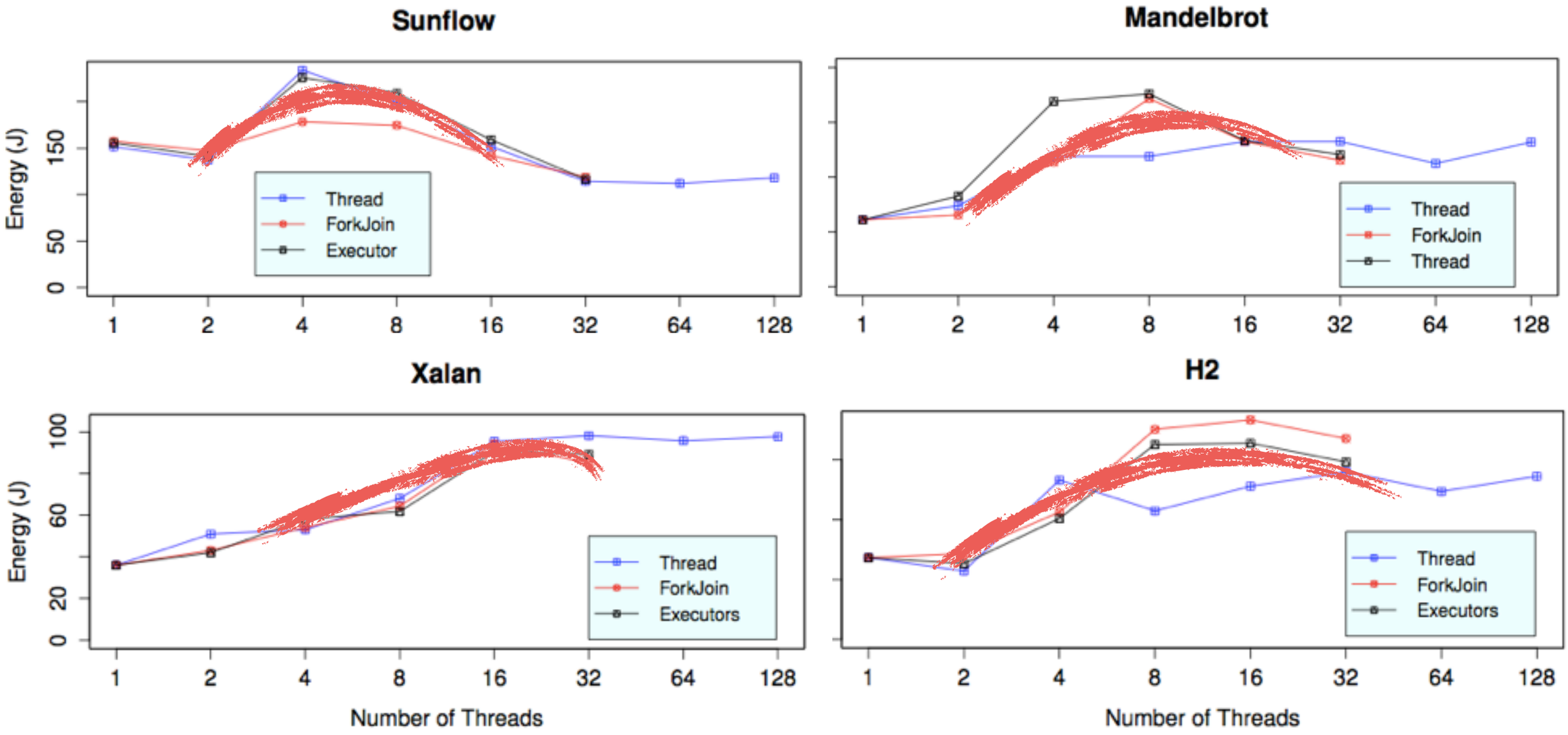




Energy Consumption When Varying the Number of Threads



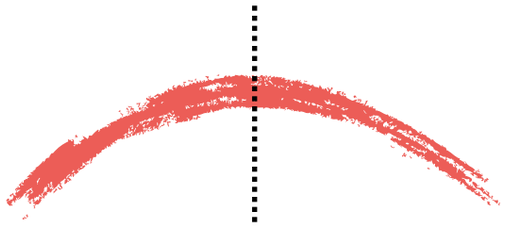
The Λ Curve


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



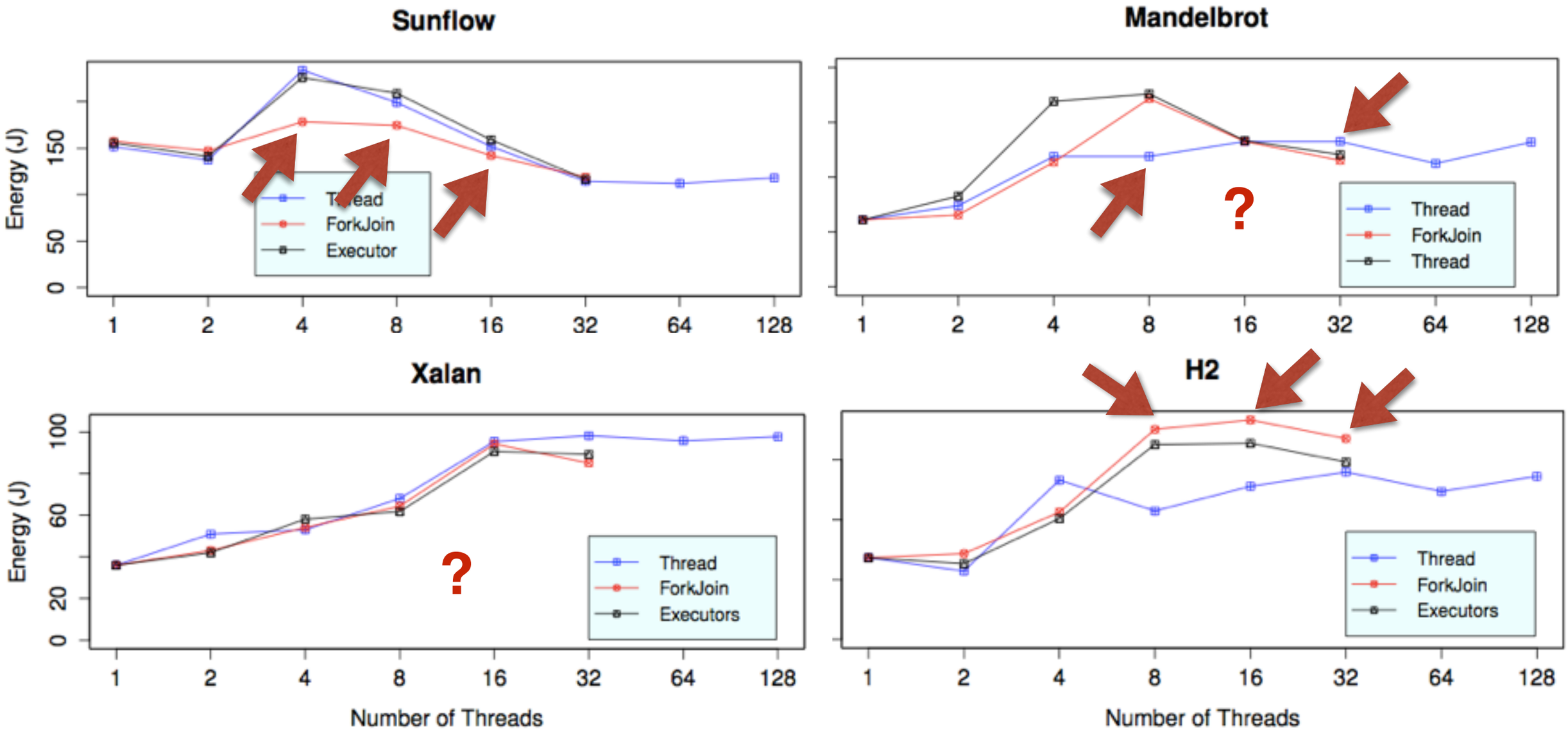
- More cores idle
- CPU frequency at a lower level

The Λ Curve

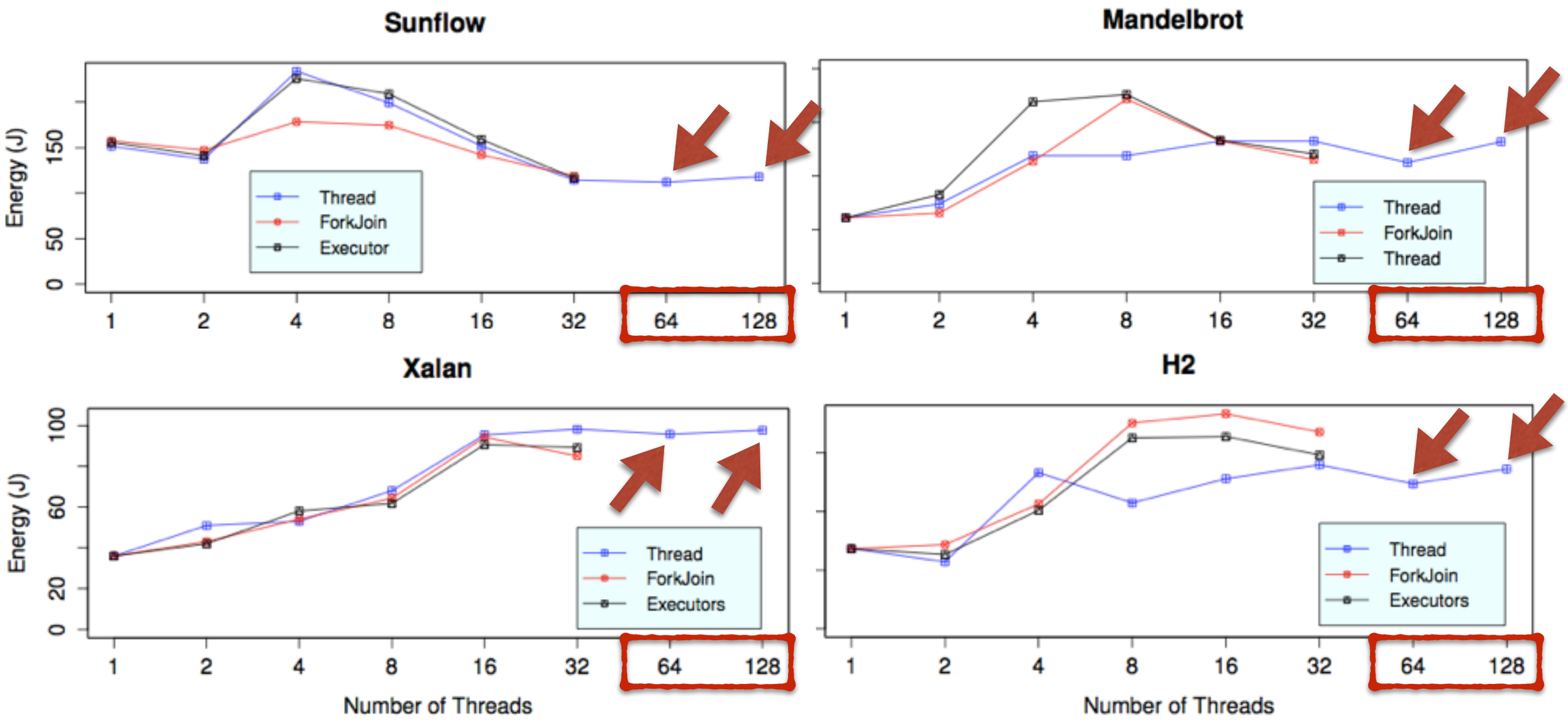
- More cores idle
- CPU frequency at a lower level

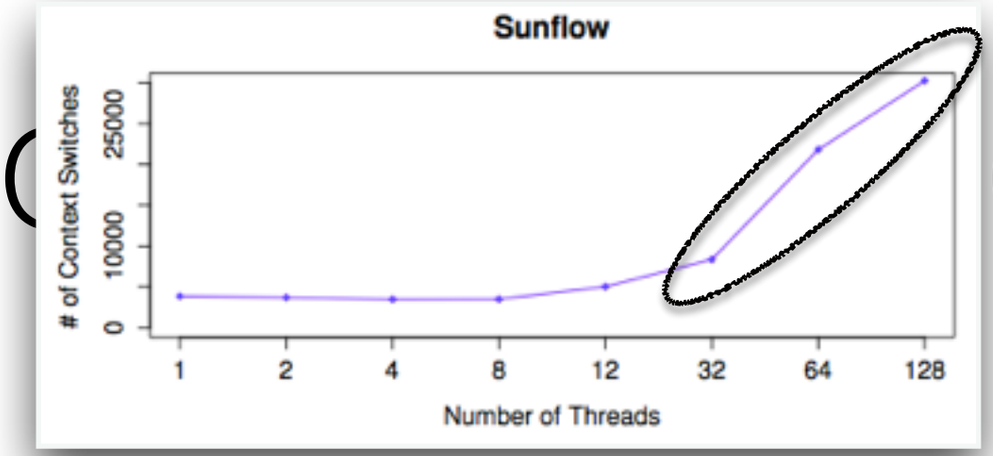
- More threads used, program completes sooner
- The greater the ratio between speedup and power, the steeper the \

Which programming style should I use?

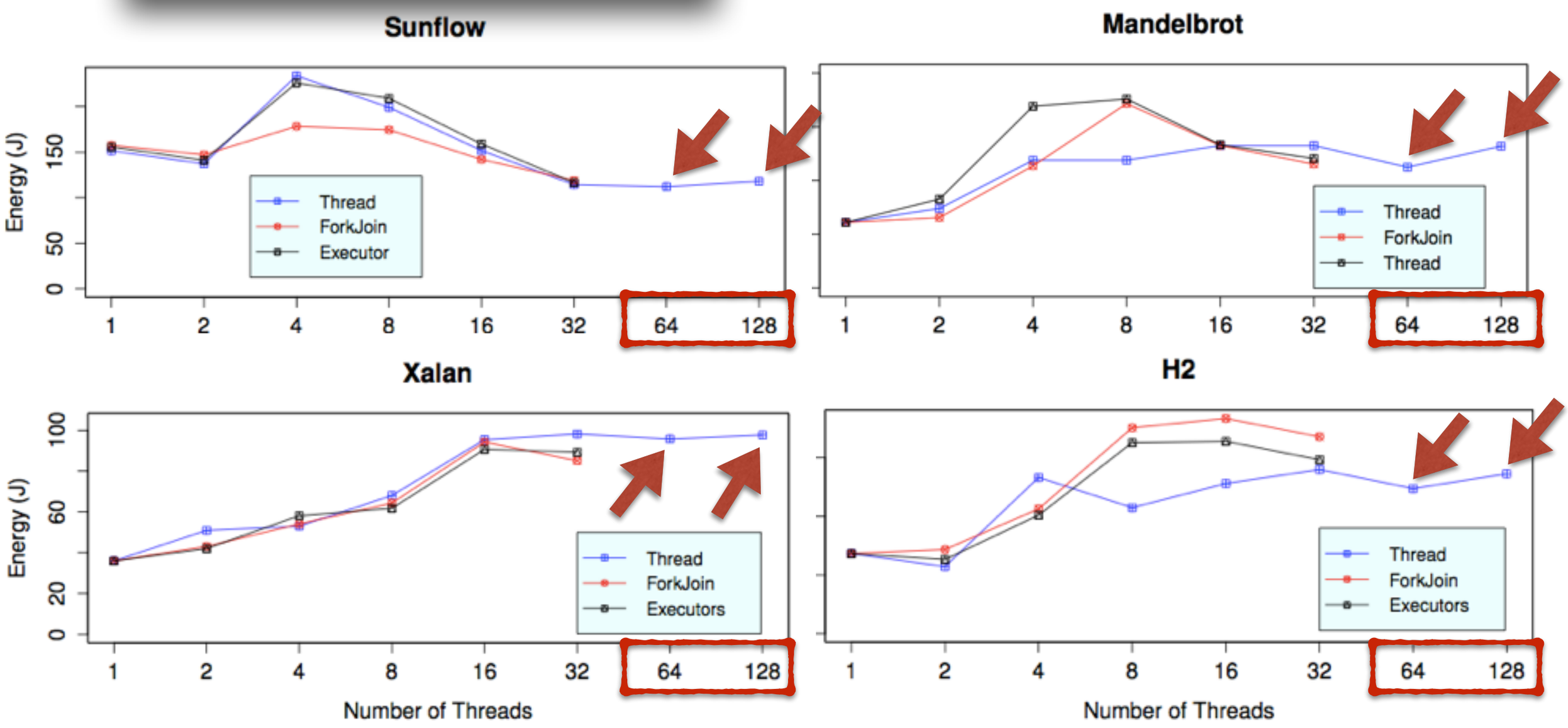


Overpopulating Cores with Threads

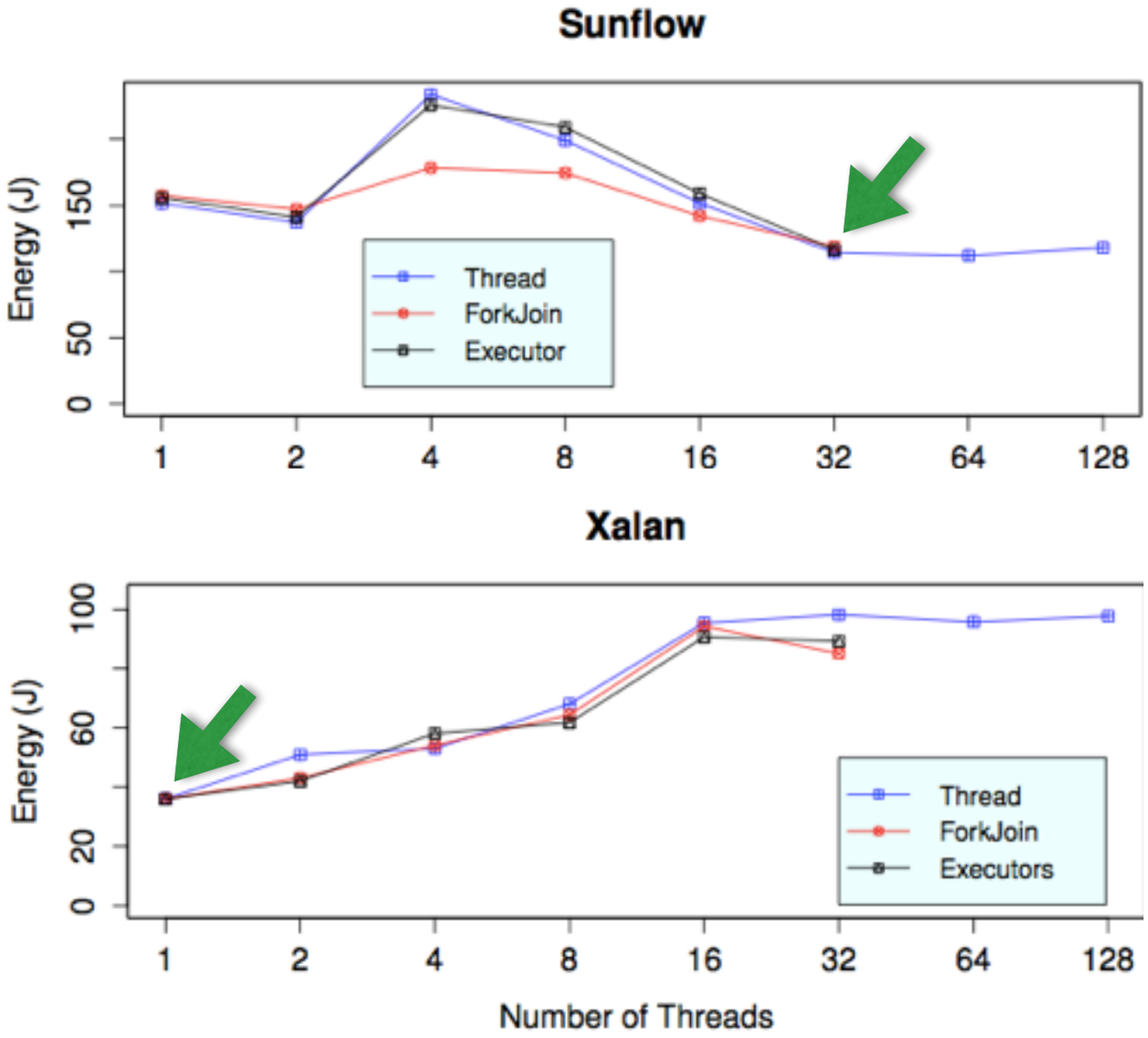


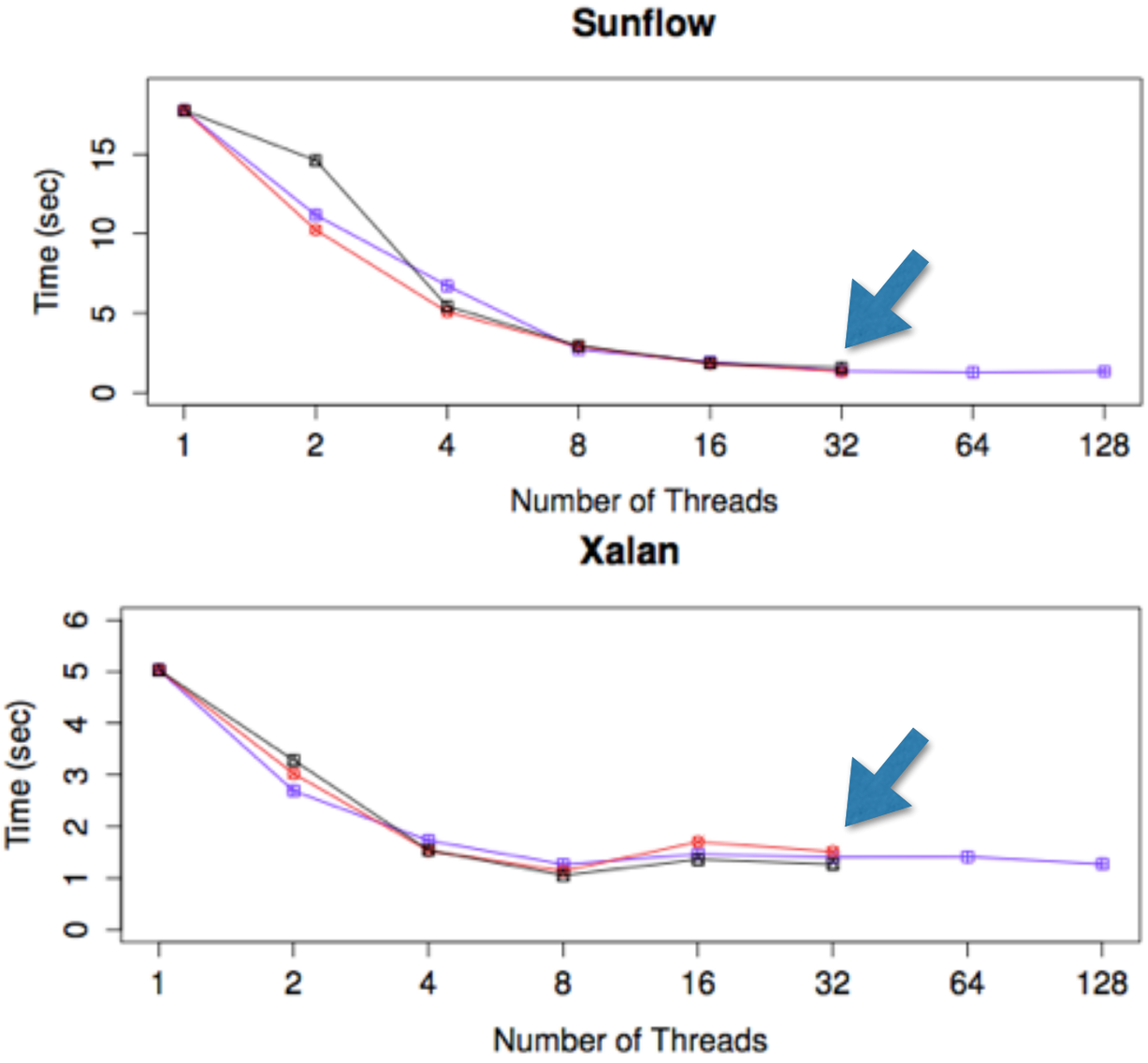


Cores with Threads



Faster ≠ Greener





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 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

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)
      );
    }
  }
}
```

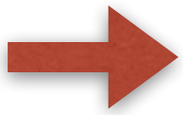
Copying

Sharing

±15% of energy savings!

Copy-Fork

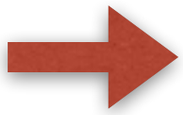
```
public void compute() {
   ...
   NQueensSolver[] tasks = new NQueensSolver[size
       ];
   for (int i = 0; i < tasks.length; i++) {
      int[] newElements = new int[depth + 1];
      System.arraycopy(currentElements, 0,
                       newElements, 0, depth);
      tasks[i] = new NQueensSolver(newElements);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

```
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();
   }
}
```



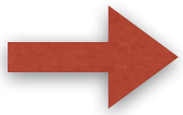
```
public void compute() {
   ...
   List<NQueensSolver> tasks = new ArrayList<>(
       size);
   for (int i = 0; i < tasks.size(); i++) {
      int[] newElements = new int[depth + 1];
      System.arraycopy(currentElements, 0,
                       newElements, 0, depth);
      tasks.add(new NQueensSolver(newElements));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

Before

After

Copy-Fork

```
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);
}
```

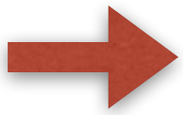
Before

After

Copy/Fork/Copy/Fork/…

Copy-Fork

```
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);
}
```

Before

After

Copy/Fork/Copy/Fork/…

Copy-Fork

```
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);
}
```

Before

After

Copy/Fork/Copy/Fork/…

Copy/../Copy/Fork/…/Fork

Copy-Fork

```
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);
}
```

Before

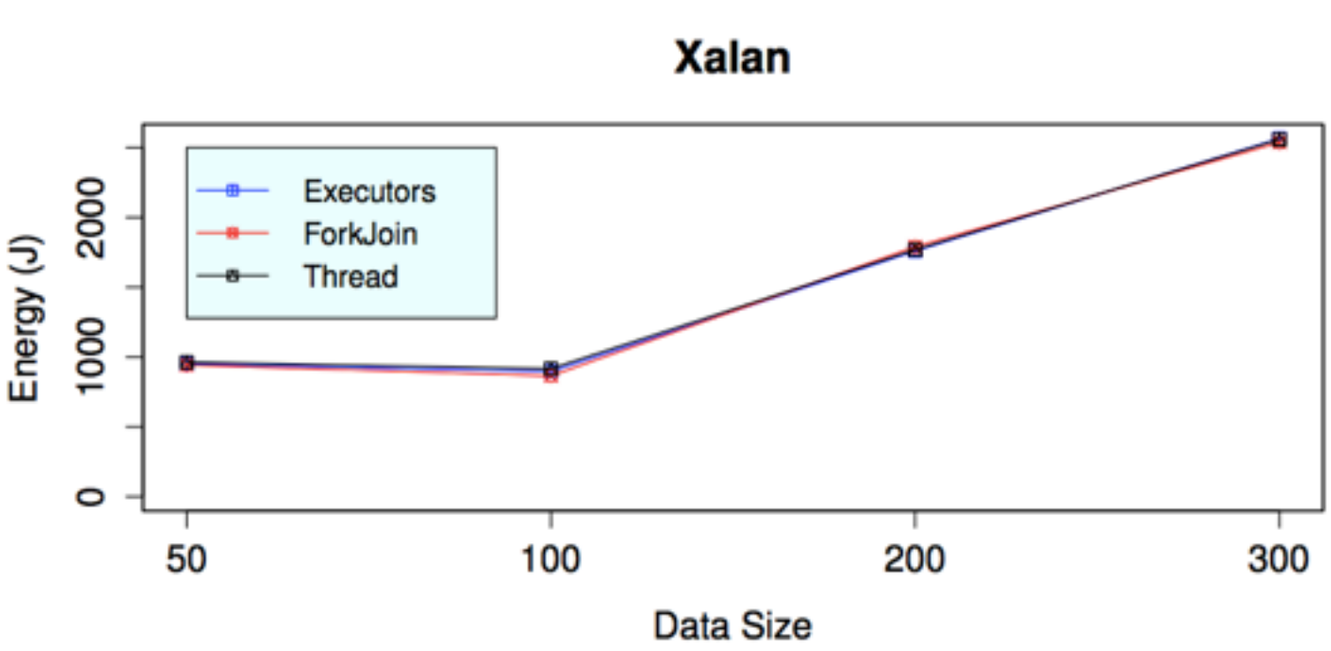
After

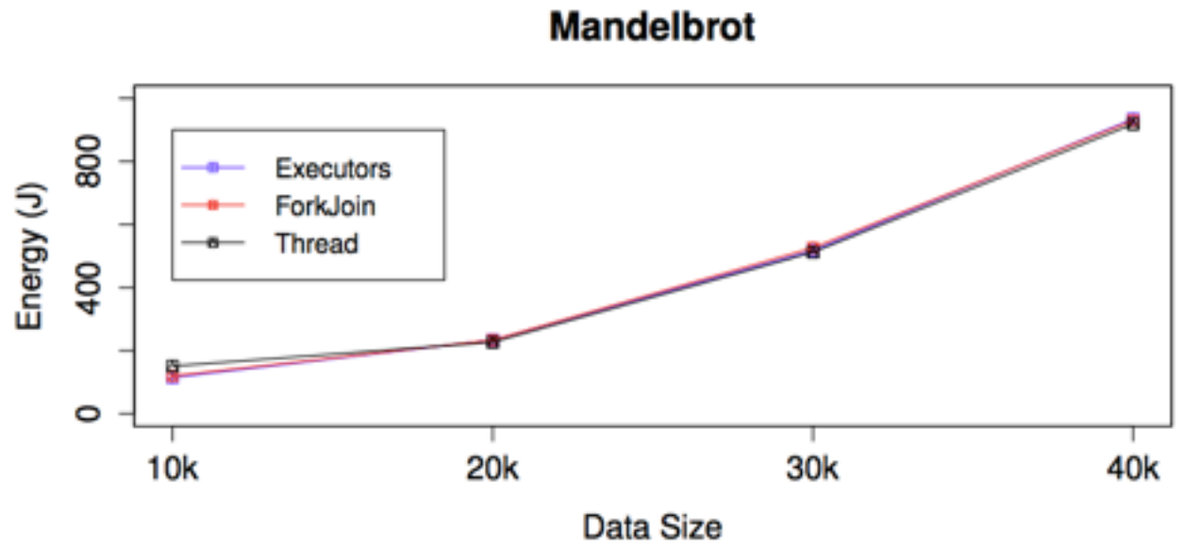
Copy/Fork/Copy/Fork/…

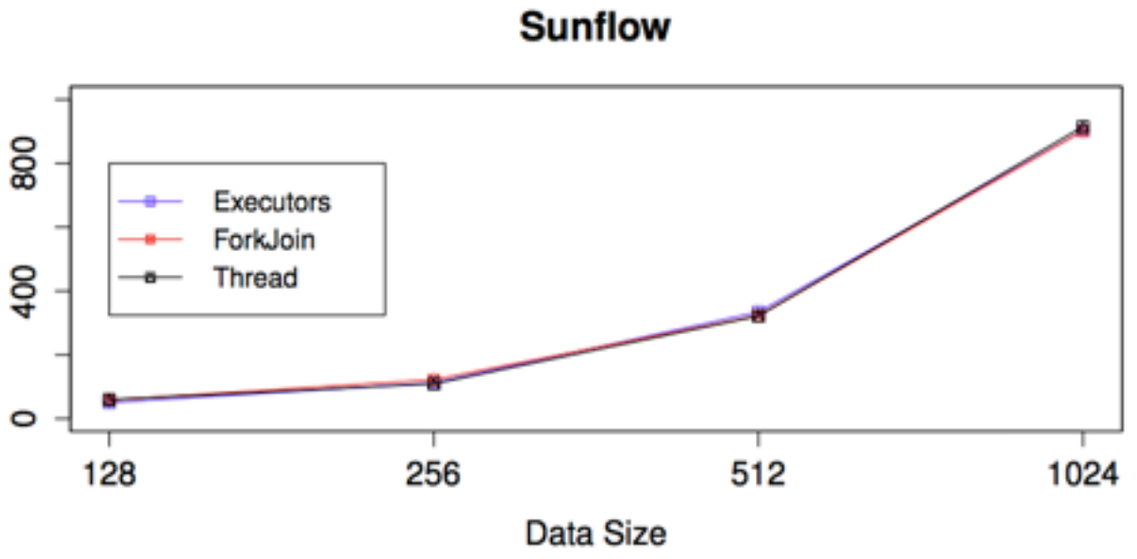
Copy/../Copy/Fork/…/Fork

±10% of energy savings!

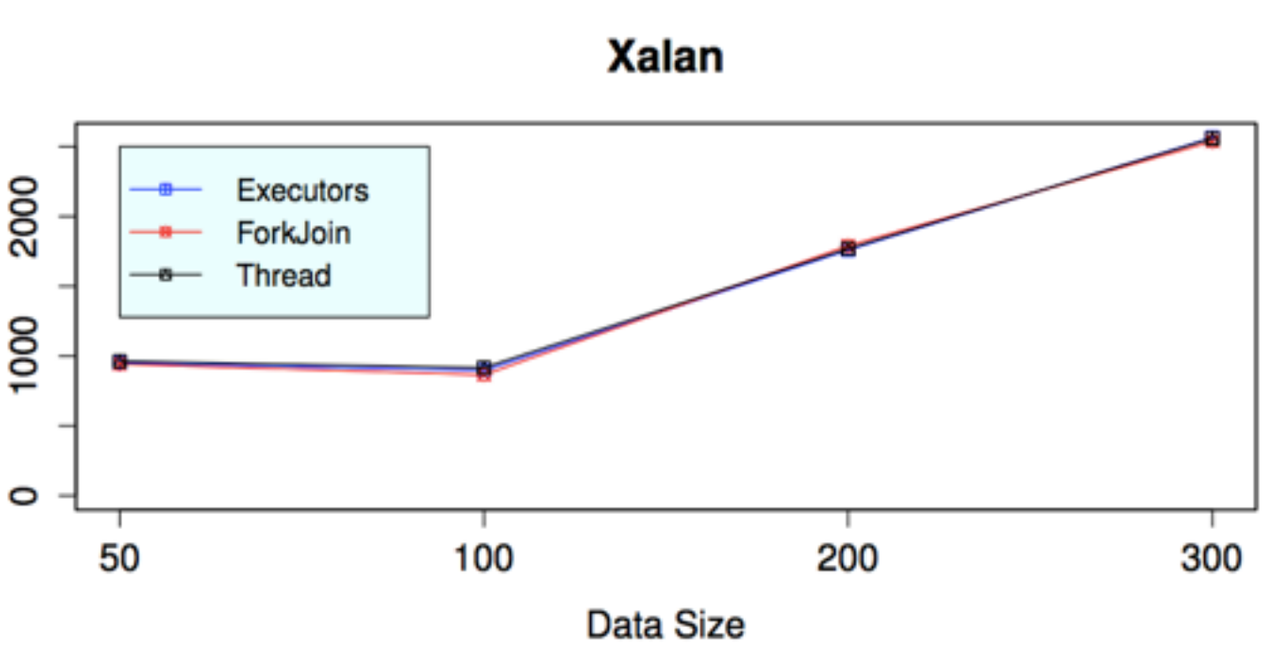
Data Size

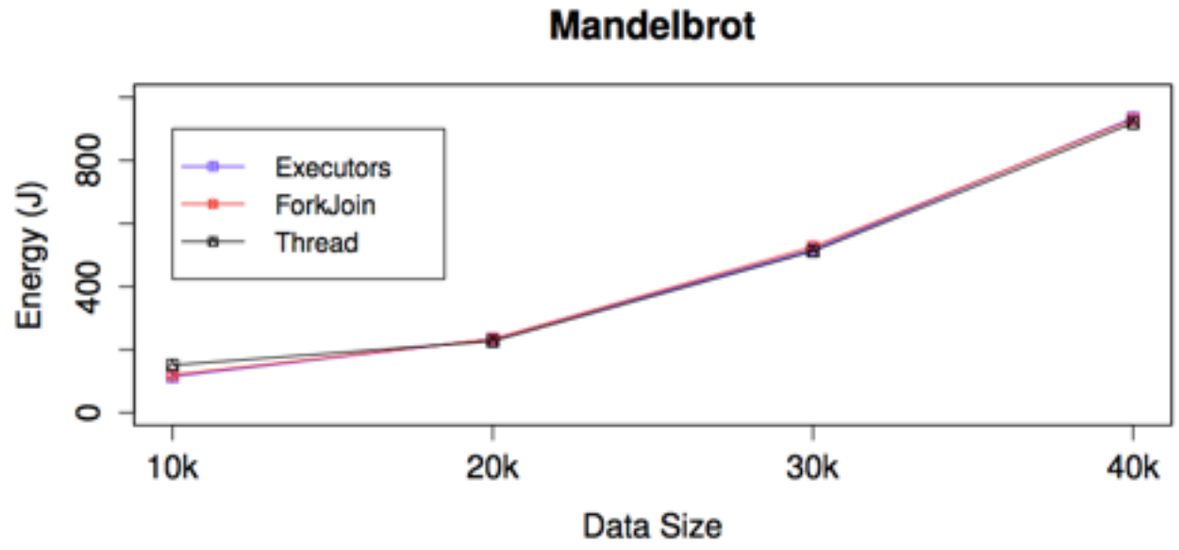

Xalan
Executors
ForkJoin
Thread
Energy (J)
0
1000
2000
50
100
200
300
Data Size

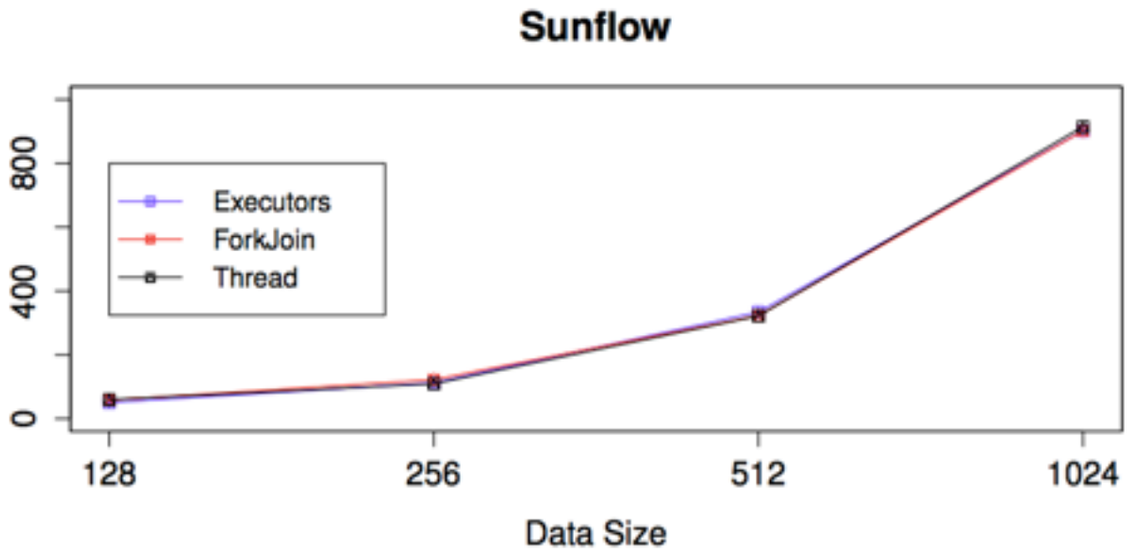

Mandelbrot
Executors
ForkJoin
Thread
Energy (J)
0
400
800
10k
20k
30k
40k
Data Size


Sunflow
Executors
ForkJoin
Thread
0
400
800
128
256
512
1024
Data Size

Data Size





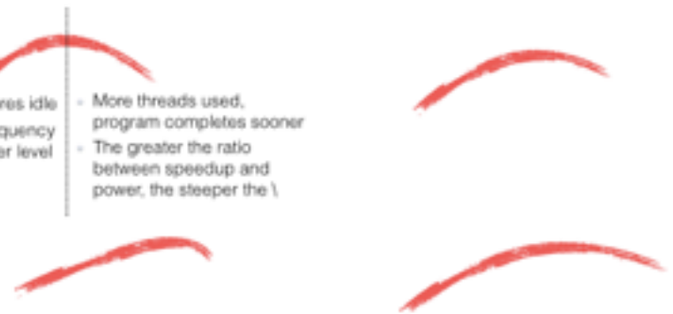


For most of the cases, energy consumption is linear to data size!

The Λ Curve

- More cores idle
- CPU frequency at a lower level

- More threads used, program completes sooner
- The greater the ratio between speedup and power, the steeper the \



13

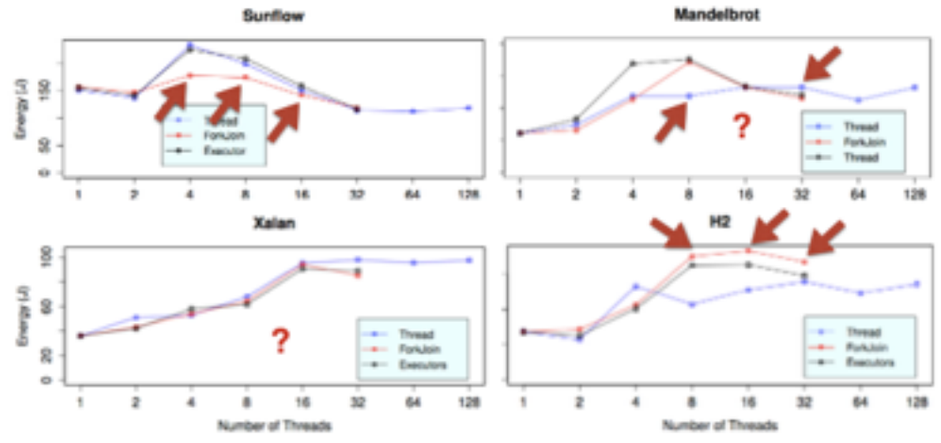
The Λ Curve

- More cores idle
- CPU frequency at a lower level

- More threads used, program completes sooner
- The greater the ratio between speedup and power, the steeper the \

13

Which programming style should I use?

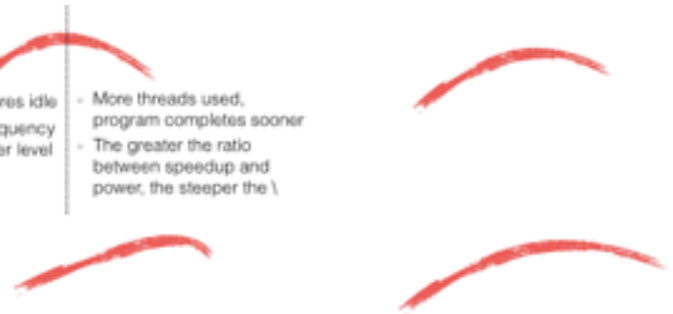


14

The Λ Curve

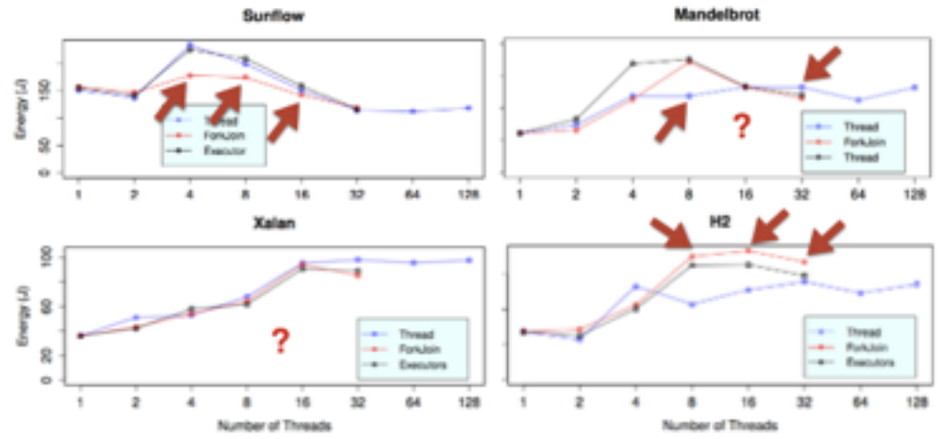
- More cores idle
- CPU frequency at a lower level

- More threads used, program completes sooner
- The greater the ratio between speedup and power, the steeper the \



13

Which programming style should I use?



14

Copy-Fork Patterns

```
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

Copy/Fork/Copy/Fork/…/Compute

Copy/../Copy/Compute/Fork/…/Fork

±10% of energy savings!

29

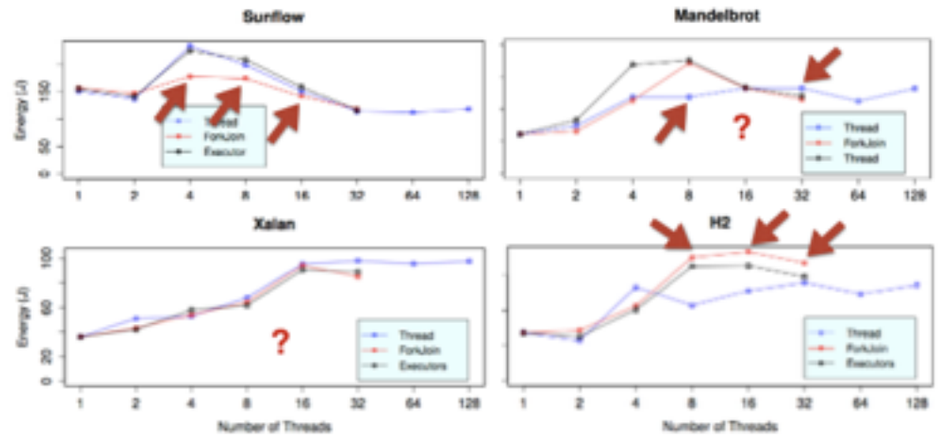
The Λ Curve

- More cores idle
- CPU frequency at a lower level

- More threads used, program completes sooner
- The greater the ratio between speedup and power, the steeper the \

13

Which programming style should I use?



14

Copy-Fork Patterns

```
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

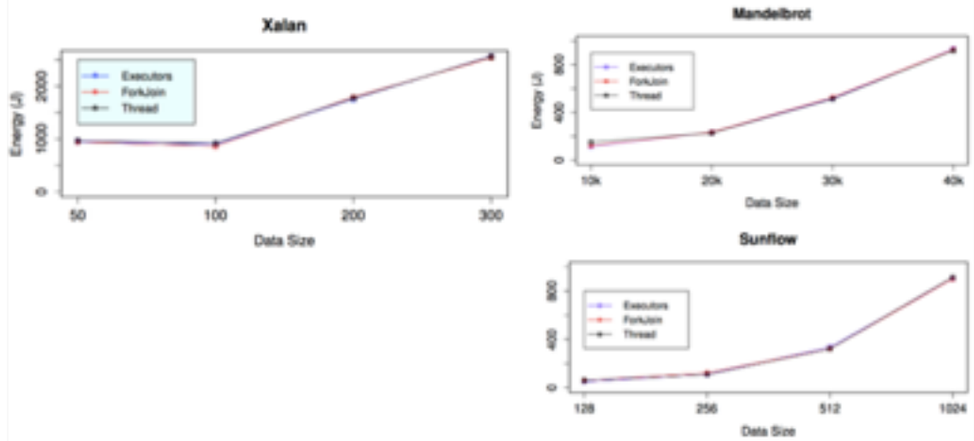
Copy/../Copy/Compute/Fork/.../Fork

Copy/Fork/Copy/Fork/.../Compute

±10% of energy savings!

29

Data Size



32

Understanding Energy Behaviors of Thread Management Constructs

Gustavo Pinto[1] Fernando Castor[1] David Liu[2]

{ghlp, castor}@cin.ufpe.br[1]

davidL@binghamton.edu[2]

VIRTUS IMPAVIDA

Centro de Informática UFPE

BINGHAMTON UNIVERSITY

STATE UNIVERSITY OF NEW YORK

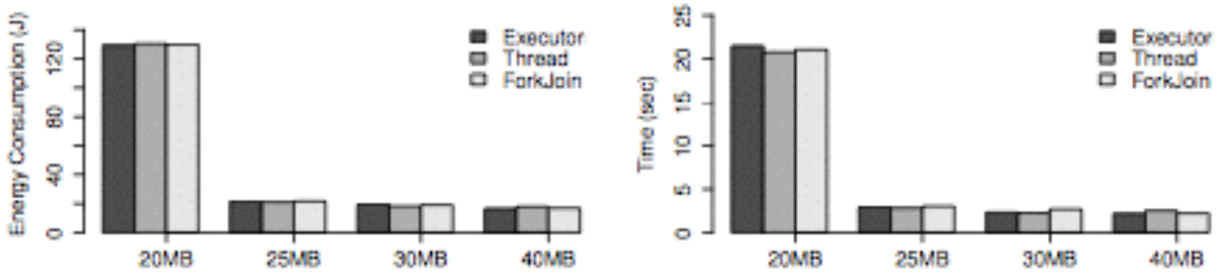


Figure 17. Heap Size Effect (`sunflow`, 32 threads, 256 tasks, 256 as image data size)

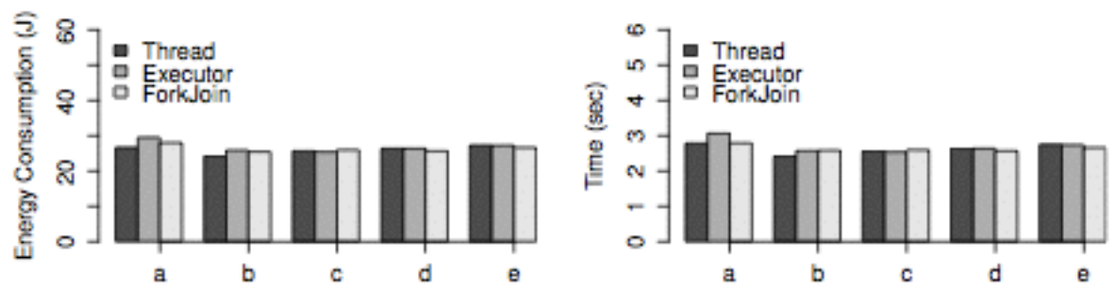
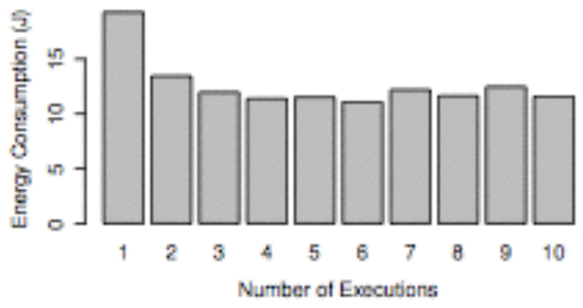
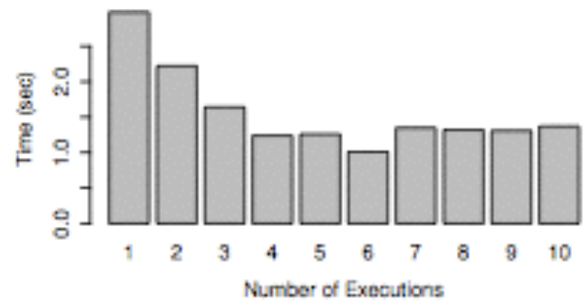


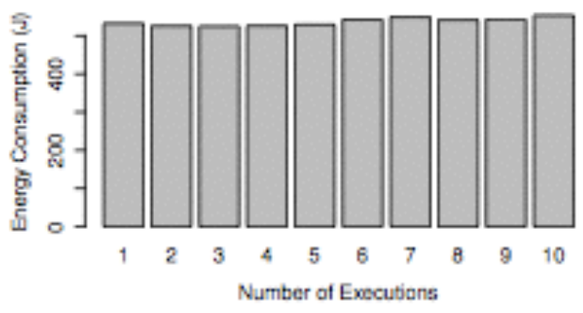
Figure 18. GC Effect (`xalan`, 32 threads, 64 tasks, 300 transformation files. GC strategies are: a: SerialGC, b: ParallelGC, c: `ParallelOldGC`, d: `ConcMarkSweepGC`, e: `G1GC`)



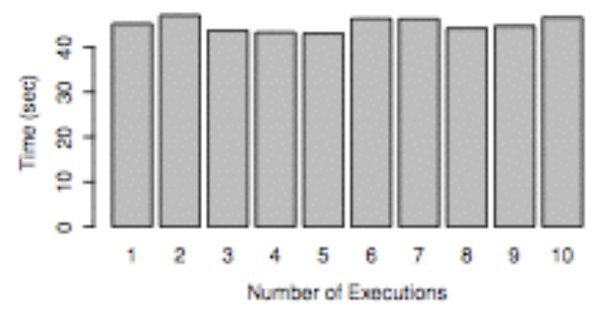
(a) Energy with JIT



(b) Time with JIT



(c) Energy without JIT



(d) Time without JIT

Figure 19. JIT Effect (`sunflow`, 32 threads, 256 tasks, 256 as image data size, 10 runs on X-axis)