Energy-Effectiveness of Pre-execution and Energy-Aware P-Thread Selection

Vlad Petric, Amir Roth
University of Pennsylvania
Pre-Execution

What it is: a performance technique
What it does: hides microarch latencies
- Cache misses (branch mispredictions too)

How: p-threads (pre-execution “helper” threads)
- Statically isolate slices leading to cache misses
- Dynamically spawn copies in parallel with main thread

Performance-redundancy trade-off
Pre-Execution

What it is: a performance technique
What it does: hides microarch latencies
  • Cache misses (branch mispredictions too)
How: p-threads (pre-execution “helper” threads)
  • Statically isolate slices leading to cache misses
  • Dynamically spawn copies in parallel with main thread

Performance-redundancy trade-off

Previously: performance considerations only
  • **PTHSEL**: automated P-THread SELection framework
Pre-Execution

What it is: a performance technique
What it does: hides microarch latencies
- Cache misses (branch mispredictions too)
How: p-threads (pre-execution “helper” threads)
- Statically isolate slices leading to cache misses
- Dynamically spawn copies in parallel with main thread

Performance-redundancy trade-off
Previously: performance considerations only
- PTHSEL: automated P-THread SELection framework

This work: redundancy = energy
- PTHSEL\(_E\): manipulate performance/energy trade-off
Outline

Pre-Execution / DDMT primer

Performance and energy evaluation

PTHSEL: performance-only p-thread selection (review)

$\text{PTHSEL}^+_E$: energy-aware p-thread selection

- An explicit energy model
- A better latency reduction model

Performance and energy re-evaluation
DDMT (Data-Driven Multi-Threaded) Technology

- One implementation of pre-execution

1. P-threads derived from actual program
2. Control-less: all p-thread instances identical
3. Chain-less: number of spawns under tight control
DDMT (Data-Driven Multi-Threading)

- One implementation of pre-execution

1. P-threads derived from actual program
2. Control-less: all p-thread instances identical
3. Chain-less: number of spawns under tight control

2+3. Aggregate p-thread behavior easy to analyze
DDMT (Data-Driven Multi-Threading)

- One implementation of pre-execution

1. P-threads derived from actual program
2. Control-less: all p-thread instances identical
3. Chain-less: number of spawns under tight control

2+3. Aggregate p-thread behavior easy to analyze

1+2+3. PTHSEL: automated p-thread selection framework
Example I: P-Thread Generation

Static code:

```c
for (i = 0; i < 100; i++) {
    if (xn[i].cover == PART)
        id = xn[i].id;  \(70 \text{ times}\)
    else
        id = xn[i].g_id;  \(30 \text{ times}\)
    receipts += rx[id].price;  \(50 \text{ misses}\)
    ...
}
```

**Problem load**: 100 executions, 50 misses

Address-predicting this load is hard
Example I: P-Thread Generation

**Static code:**
```
for (i = 0; i < 100; i++) {
    if (xn[i].cover == PART)
        id = xn[i].id;
    else
        id = xn[i].g_id;
    receipts += rx[id].price;
...
}
```

**Execution:**
```
id = xn[i].id;
receipts += rx[id].price;
...
i++;

id = xn[i].id;
receipts += rx[id].price;
...
i++;

id = xn[i].g_id;
receipts += rx[id].price;
...
i++;

id = xn[i].id;
receipts += rx[id].price;
```
Example I: P-Thread Generation

Static code:

```c
for (i = 0; i < 100; i++) {
    if (xn[i].cover == PART)
        id = xn[i].id;
    else
        id = xn[i].g_id;
    receipts += rx[id].price;
    ... 
}
```

Execution:

```c
id = xn[i].id;
receipts += rx[id].price;
...
```

```c
i++;
```
Example I: P-Thread Generation

Static code:

```c
for (i = 0; i < 100; i++) {
    if (xn[i].cover == PART)
        id = xn[i].id;
    else
        id = xn[i].g_id;

    receipts += rx[id].price;
    ...
}
```

Execution:

```c
id = xn[i].id;
receipts += rx[id].price;
...
```

```c
id = xn[i].id;
receipts += rx[id].price;
...
i++;
```

```c
id = xn[i].g_id;
receipts += rx[id].price;
...
i++;
```

```c
id = xn[i].id;
receipts += rx[id].price;
...
i++;
```
Example I: P-Thread Generation

Static code:

```c
for (i = 0; i < 100; i++) {
    if (xn[i].cover == PART)
        id = xn[i].id;
    else
        id = xn[i].g_id;
    receipts += rx[id].price;
    ...
}
```

Execution:

```c
id = xn[i].id;
receipts += rx[id].price;
...
i++;
```

```c
id = xn[i].id;
receipts += rx[id].price;
...
i++;
```

```c
id = xn[i].g_id;
receipts += rx[id].price;
...
i++;
```

```c
id = xn[i].id;
receipts += rx[id].price;
```
Example I: P-Thread Generation

Static code:
```
for (i = 0; i < 100; i++) {
    if (xn[i].cover == PART)
        id = xn[i].id;
    else
        id = xn[i].g_id;
    receipts += rx[id].price;
    ...
}
```

Execution:
```
id = xn[i].id;
receipts += rx[id].price;
...
i++;

id = xn[i].id;
receipts += rx[id].price;
...
i++;

id = xn[i].g_id;
receipts += rx[id].price;
...
i++;

id = xn[i].id;
receipts += rx[id].price;
```
Example I: P-Thread Generation

Static code:
for (i = 0; i < 100; i++) {
    if (xn[i].cover == PART)
        id = xn[i].id;
    else
        id = xn[i].g.id;
    receipts += rx[id].price;
    ...
}

Static p-thread:
i++;
i++;
i++;
id = xn[i].id;
prefetch &rx[id].price;
i++;

Execution:

id = xn[i].id;
receipts += rx[id].price;
...
i++;
id = xn[i].id;
receipts += rx[id].price;
...
i++;
id = xn[i].g.id;
receipts += rx[id].price;
i++;
id = xn[i].id;
receipts += rx[id].price;
Example 1: P-Thread Generation

Static code:

```c
for (i = 0; i < 100; i++) {
    if (xn[i].cover == PART)
        id = xn[i].id;
    else
        id = xn[i].g_id;
    receipts += rx[id].price;
    ...  
}
```

Static p-thread:

```c
i++;  
i++;  
i++;  
id = xn[i].id;  
prefetch &rx[id].price;
```

Execution:

```c
id = xn[i].id;  
receipts += rx[id].price;  
...
```

```c
i++;  
...  
```

```c
id = xn[i].id;  
receipts += rx[id].price;  
...
```

```c
i++;  
...  
```

```c
id = xn[i].g_id;  
prefetch &rx[id].price;  
receipts += rx[id].price;  
```
Example II: Runtime

Main-thread execution:

```java
id = xn[i].id;
receipts += rx[id].price;
...
i++;
```

P-thread execution:
Example II: Runtime

Main-thread execution:

\[
\begin{align*}
  \text{id} &= \text{xn}[i].\text{id}; \\
  \text{receipts} &= \text{rx}[\text{id}].\text{price}; \\
  \ldots &\quad \text{spawn} \\
  i++;
\end{align*}
\]
Example II: Runtime

Main-thread execution:

```java
id = xn[i].id;
receipts += rx[id].price;
...
```

P-thread execution:

```java
i++; spawn
id = xn[i].id;
receipts += rx[id].price;
...
```

```java
i++;
```

```java
i++;
```

```java
id = xn[i].id;
prefetch &rx[id].price;
```
Example II: Runtime

Main-thread execution:

```c
id = xn[i].id;
receipts += rx[id].price;
...
```

 spawn

```c
i++;
```

```c
id = xn[i].id;
receipts += rx[id].price;
...
```

```c
i++;
```

```c
id = xn[i].id;
receipts += rx[id].price;
...
```

```c
i++;
```
Example II: Runtime

Main-thread execution:

```c
id = xn[i].id;
receipts += rx[id].price;
...
```  

```
  spawn
  
i++;
```  

```
i++;
id = xn[i].id;
receipts += rx[id].price;
...`

```
i++;
id = xn[i].g_id;
receipts += rx[id].price;
...`

```
i++;
id = xn[i].id;
receipts += rx[id].price;
...`
```

P-thread execution:

```
i++;
prefetch &rx[id].price;
```  

```
  miss latency
  ```
Example II: Runtime

Main-thread execution:

id = xn[i].id;
receipts += rx[id].price;
...

i++;

id = xn[i].id;
receipts += rx[id].price;
...

i++;

id = xn[i].id;
receipts += rx[id].price;
...

i++;

P-thread execution:

i++;

id = xn[i].id;
prefetch &rx[id].price;

i++;

id = xn[i].id;
prefetch &rx[id].price;

i++;

id = xn[i].id;
prefetch &rx[id].price;
Performance/Energy Evaluation

Performance: SimpleScalar Alpha++
- 6-way superscalar out-of-order, 8-threads
- 32KB I/D$, 512KB L2, 200-cycle memory latency
- Critical path post-mortem based on Fields et al.

Energy: Wattch/CACTI++
- 180nm, 2GHz, 1.5V, aggressive clock-gating
- 5% of max energy saved in “sleep mode”
  - e.g. Pentium 4 Mobile

Benchmarks: SPECint2000
- Only subset that has L2 misses
Performance

lower is better

Execution latency reduced 14%

+ Memory latency: reduced 20%

– Fetch bandwidth: increased 8% (much more for bzip2, mcf)
Energy

again, lower is better

Energy increased 12% → Energy-delay reduced 2%

- Dynamic p-thread energy
Energy

Energy increased 12% → Energy-delay reduced 2%
- Dynamic p-thread energy

“Energy-negative”, “ED-neutral” ...
Energy

again, lower is better

Energy increased 12% → Energy-delay reduced 2%

- Dynamic p-thread energy

“Energy-negative”, “ED-neutral” ... we can do better
Outline

Pre-Execution / DDMT primer
Performance and energy evaluation

PTHSEL: performance-only p-thread selection (review)

PTHSEL$_{+E}$: energy-aware p-thread selection
- An explicit energy model
- A better latency reduction model

Performance and energy re-evaluation
Slice tree

- All possible static p-threads
- Node $\rightarrow$ spawn-point
- Path to root $\rightarrow$ p-thread body
Slice tree

- All possible static p-threads
- Node → spawn-point
- Path to root → p-thread body

```
i++
i++
i++
i++
```
```
i++
i++
i++
```
```
```
```
```
iprefixh &rx[id]
```
```
```
i++
i++
i++
id=xn[i].id
```
```
i++
id=xn[i].g_id
```
PTHSEL Overview

P-thread generation: easy
P-thread selection: hard

Short p-threads
- Lower overhead
- Lower latency tolerance

Long p-threads
**PTHSEL Overview**

- **P-thread generation**: easy
- **P-thread selection**: hard

**Short p-threads**
- + Lower overhead
- - Lower latency tolerance

**Long p-threads**
- - Higher overhead
- + Higher latency tolerance

- `i++`  
- `id=xn[i].id`  
- `i++`  
- `id=xn[i].g_id`  
- `i++`  
- `i++`  
- `i++`  
- `prefetch &rx[id]`
P-Thread generation: easy
P-Thread selection: hard

Short p-threads
+ Lower overhead
- Lower latency tolerance

Long p-threads
- Higher overhead
+ Higher latency tolerance

PTHSEL finds the sweetspot... quantitatively
Benefit: miss latency reduction

- \( LBENEFIT(p) = MISSES(p) \times LRED-MISS(p) \)
- \( LRED-MISS(p) \): dataflow height calculation
**Benefit: miss latency reduction**

- $LBENEFIT(p) = MISSES(p) \times LRED\text{-}MISS(p)$
- $LRED\text{-}MISS(p)$: dataflow height calculation

**Cost: fetch bandwidth contention**

- $LCOST(p) = SPAWNS(p) \times SIZE(p) \times DISCOUNT$
- $SPAWNS(p) \geq MISSES(p)$: both from profile
- $DISCOUNT$: unused bandwidth is “free”
Benefit: miss latency reduction
- $LBENEFIT(p) = MISSES(p) \times LRED-MISS(p)$
- $LRED-MISS(p)$: dataflow height calculation

Cost: fetch bandwidth contention
- $LCOST(p) = SPAWNS(p) \times SIZE(p) \times DISCOUNT$
- $SPAWNS(p) \geq MISSES(p)$: both from profile
- $DISCOUNT$: unused bandwidth is “free”

Advantage: benefit – cost
- $LADV(p) = LBENEFIT(p) - LCOST(p)$
PTHSEL Latency Model

Benefit: miss latency reduction
- \( LBENEFIT(p) = MISSES(p) \times LRED-MISS(p) \)
- \( LRED-MISS(p) \): dataflow height calculation

Cost: fetch bandwidth contention
- \( LCOST(p) = SPAWNS(p) \times SIZE(p) \times DISCOUNT \)
- \( SPAWNS(p) \geq MISSES(p) \): both from profile
- \( DISCOUNT \): unused bandwidth is “free”

Advantage: benefit – cost
- \( LADV(p) = LBENEFIT(p) - LCOST(p) \)

Sweetspot? as p-threads get longer...
- Some things get better, others get worse
From PTHSEL to PTHSEL$^E$

PTHSEL: p-threads target latency reduction
From PTHSEL to $\text{PTHSEL} + E$

$\text{PTHSEL}$: p-threads target latency reduction

$\text{PTHSEL} + E$: p-threads target energy reduction

- Or any latency/energy combination (e.g., ED, $ED^2$)
- New benefit/cost functions, e.g., $EADV(p)$
  - Explicit energy model
  - Better latency model
PTHSEL$_E$ Energy Model

Energy cost: dynamic p-thread energy consumption

- $ECOST(p) = SPAWNS(p) \times SIZE(p) \times E_{insn}$
- $E_{insn} = E_{I\$} + E_{rename} + \ldots$ (see paper)
- No DISCOUNT: energy is never “free”
**PTHSEL + \( E \) Energy Model**

Energy cost: dynamic p-thread energy consumption

- \( ECOST(p) = SPAWNS(p) \times SIZE(p) \times E_{insn} \)
- \( E_{insn} = E_{I\$} + E_{rename} + \ldots \) (see paper)
- No **DISCOUNT**: energy is never “free”

Energy benefit: truly idle → “sleep mode”

- \( EBENEFIT(p) = LADV(p) \times E_{idle} \)
- \( E_{idle} \): per-cycle energy saved by “sleeping”
**PTHSEL + Energy Model**

Energy cost: dynamic p-thread energy consumption

- $ECOST(p) = SPAWNS(p) \times SIZE(p) \times E_{\text{insn}}$
- $E_{\text{insn}} = E_{I\$} + E_{\text{rename}} + \ldots$ (see paper)
- No **DISCOUNT**: energy is never “free”

Energy benefit: truly idle $\rightarrow$ “sleep mode”

- $EBENEFIT(p) = LADV(p) \times E_{\text{idle}}$
- $E_{\text{idle}}$: per-cycle energy saved by “sleeping”

Energy advantage

- $EADV(p) = EBENEFIT(p) - ECOST(p)$
PTHSEL + \( E \) Energy Model

Energy cost: dynamic p-thread energy consumption

- \( ECOST(p) = SPAWNS(p) \times SIZE(p) \times E_{insn} \)
- \( E_{insn} = E_{I\$} + E_{rename} + \ldots \) (see paper)
- No \textbf{DISCOUNT}: energy is never “free”

Energy benefit: truly idle → “sleep mode”

- \( EBENEFIT(p) = LADV(p) \times E_{idle} \)
- \( E_{idle} \): per-cycle energy saved by “sleeping”

Energy advantage

- \( EADV(p) = EBENEFIT(p) - ECOST(p) \)

Energy constants: \( E_{insn}, E_{idle} \)

- Reverse engineered or OEM supplied
A Better Latency Model

$EADV(p)$ builds on $LADV(p)$

- But $LADV(p)$ not accurate enough to build on

  - Proof? slowdown in $mcf$

![Diagram showing latency and execution time comparison.](image-url)
A Better Latency Model

\( EADV(p) \) builds on \( LADV(p) \)
- But \( LADV(p) \) not accurate enough to build on
  - Proof? slowdown in \( mcf \)

Diagnosis: optimistic \( LRED\text{-}MISS(p) \)
- Miss latency 1-to-1 with execution time
  - Doesn’t account for MLP
  - P-threads with little/no actual advantage
**A Better Latency Model**

\(EADV(p)\) builds on \(LADV(p)\)

- But \(LADV(p)\) not accurate enough to build on
  - Proof? slowdown in \(mcf\)

**Diagnosis: optimistic \(LRED-MISS(p)\)**

- Miss latency 1-to-1 with execution time
- Doesn’t account for MLP
- P-threads with little/no actual advantage

**Fix: critical-path based \(LRED-MISS(p)\)**

- Miss latency 1-to-1 with execution time **while miss is critical**
- See paper for details
PTHSEL + $E$ “Targets”

Latency: $LADV(p)$

Energy: $EADV(p)$
PTHSEL + $E$ “Targets”

Latency: $LADV(p)$
Energy: $EADV(p)$

ED: $EDADV(p) = L_0 \times E_0 - (L_0 - LADV(p)) \times (E_0 - EADV(p))$

- $L_0, E_0$: profiling ($E_0/L_0$ is enough)
PTHSEL $+E$ “Targets”

Latency: $LADV(p)$
Energy: $EADV(p)$

ED: $EDADV(p) = L_0 \times E_0 - (L_0 - LADV(p)) \times (E_0 - EADV(p))$

- $L_0, E_0$: profiling ($E_0/L_0$ is enough)

ED$^2$: similar

$E^W D^{(1-W)}$: choose your precise metric
Outline

Pre-Execution / DDMT primer
Performance and energy evaluation
PTHSEL: performance-only p-thread selection (review)

$\text{PTHSEL} + E$: energy-aware p-thread selection
  - An explicit energy model
  - A better latency reduction model

Performance and energy re-evaluation
Performance/Energy Re-evaluation

now, higher is better

O: PTHSEL (latency)
Performance/Energy Re-evaluation

now, higher is better

O: PTHSEL (latency)
\( L: \) PTHSEL_{+E} latency

+ PTHSEL_{+E} fixes PTHSEL latency model (mcf)
Performance/Energy Re-evaluation

now, higher is better

\[ \text{IPC} \quad \text{energy} \quad \text{ED} \]

\[ \begin{array}{c}
\text{O: PTHSEL (latency)} \\
\text{L: PTHSEL}_{+E} \text{ latency} \\
\text{E: PTHSEL}_{+E} \text{ energy} \\
\end{array} \]

\[ + \text{ PTHSEL}_{+E} \text{ fixes PTHSEL latency model (mcf)} \]
Performance/Energy Re-evaluation

now, higher is better

O: PTHSEL (latency)
L: PTHSEL\(_E\) latency
E: PTHSEL\(_E\) energy
P: PTHSEL\(_E\) ED

+ PTHSEL\(_E\) fixes PTHSEL latency model (mcf)
+ PTHSEL\(_E\) is “robust”
  ● Targeting X actually minimizes X (X = latency, energy, ED)
Performance/Energy Re-evaluation

again, higher is better

PTHSEL: +14% latency, -12% energy, +2% ED
Performance/Energy Re-evaluation

again, higher is better

PTHSEL: +14% latency, −12% energy, +2% ED
PTHSEL$_{E}$: +16% latency, +1% energy, +9% ED

• Not all at once: your choice
$E_{idle}$: The Energy Reduction Lever

- $E_{idle}=0$: worst-case
  - Energy reduction impossible
  + ED neutrality possible

As $E_{idle}$ increases ...
pre-execution's energy picture improves
$E_{\text{idle}}$: The Energy Reduction Lever

- $E_{\text{idle}}=0$: worst-case
  - Energy reduction impossible
  + ED neutrality possible

- $E_{\text{idle}}=5$: current
  + ED reduction
  + Energy neutrality

As $E_{\text{idle}}$ increases, the energy picture improves.
$E_{idle}$: The Energy Reduction Lever

higher is still better

$E_{idle}=0$: worst-case
- Energy reduction impossible
+ ED neutrality possible

$E_{idle}=5$: current
+ ED reduction
+ Energy neutrality

$E_{idle}=10$: future
+ ED reduction
+ Energy reduction
$E_{idle}$: The Energy Reduction Lever

- $E_{idle}=0$: worst-case
  - Energy reduction impossible
  + ED neutrality possible

- $E_{idle}=5$: current
  + ED reduction
  + Energy neutrality

- $E_{idle}=10$: future
  + ED reduction
  + Energy reduction

**As $E_{idle}$ increases ...**

pre-execution’s energy picture improves
Conclusion

Pre-Execution: a performance technique

PTHSEL: quantitative p-thread selection framework
+ Precise control over latency/redundancy tradeoff

To date: only performance considered
• Pre-execution is “energy-negative”, “ED-neutral”
+ Not bad for a performance technique, but...
Conclusion

Pre-Execution: a performance technique

PTHSEL: quantitative p-thread selection framework
  + Precise control over latency/redundancy tradeoff

To date: only performance considered
  ● Pre-execution is “energy-negative”, “ED-neutral”
  + Not bad for a performance technique, but...

PTHSEL \(+E\)
  ● Choose your metric: latency, energy, ED, ED^2, etc.
  ● Energy reduction lever: E_{idle} (“sleep mode”)
  + As E_{idle} grows ... pre-execution’s energy improves