Outline

• Motivation
• PEP Concept
• PEP Core Architecture
• Graphite Simulation
• Applications
• Conclusion
Motivation

PEP Concept

Architecture

Graphite

Applications

Conclusion

Angstrom

Decrease programming effort

Increase performance

Increase resiliency

Increase energy efficiency
Positive Energy Partnerships

• *Non-application resources that perform tasks for a net gain in energy.*

• Hardware Partnerships
  • shared resources
  • hard-wired events

• Software Partnerships
  • expose to h/w ISA
  • algorithms
  • event servicing
Positive Energy Partnerships

• PEP Cores
PEP Core Architecture
PEP Core Architecture

• Area Considerations

<table>
<thead>
<tr>
<th>Core</th>
<th>Node (nm)</th>
<th>SRAM</th>
<th>Size (mm²)</th>
<th>Scaled Size (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>μController</td>
<td>65</td>
<td>128K</td>
<td>1.5</td>
<td>0.03</td>
</tr>
<tr>
<td>RAW</td>
<td>180</td>
<td>128K</td>
<td>16</td>
<td>0.25</td>
</tr>
<tr>
<td>Intel Core 2</td>
<td>65</td>
<td>2M/4M</td>
<td>80</td>
<td>9.0</td>
</tr>
</tbody>
</table>

• μController: 16-bit, RISC, in-order datapath, unified cache
• RAW: 32-bit, RISC, 8-stage, in-order datapath, split cache
• Core 2: 64-bit, x86, 14-stage, out-of-order datapath, split cache
PEP Core Architecture

- Power Considerations

<table>
<thead>
<tr>
<th>Core</th>
<th>Energy/Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>μController</td>
<td>27.2pJ</td>
</tr>
<tr>
<td>TilePro64</td>
<td>286pJ</td>
</tr>
<tr>
<td>Intel Core 2</td>
<td>101 000pJ</td>
</tr>
</tbody>
</table>

- μController: 16-bit, RISC, 1 MHz
- TilePro64: 64-bit, VLIW, 700-866 MHz
- Core 2: 64-bit, x86, 1-2.4GHz
Graphite

Application

Target Architecture

Graphite

Host Threads

Host Process

Host Process

Host Process
Graphite

Target Architecture

Application

Host Threads

SpawnHelper

Application Threads

Graphite

Host Process

Host Process

Host Process
Potential Applications

• Case Study
  – Memory Prefetching

• Other Applications
  – Security
  – Reliability
  – Event Probing
Case Study: Memory Prefetching

• Pre-Execution
  • PEP cores are free to run with minimal energy:
    – Low clock frequency
    – Low power hardware
    – Tight coupling
  • Helper thread extracted from main thread.
    – Static compiler
    – Dynamic extraction
Case Study: Helper Threads

- EM3D benchmark (Execution Time)
Case Study: Helper Threads

- EM3D benchmark (Energy Efficiency)
Potential Applications

• Security
  – PEP core does not run application code
    • Immunity to application level attacks!
  – Dynamic Information Flow Tracking (DIFT)
    • Taint pointers using PEP core
    • Run DIFT instructions in PEP
Potential Applications

• Security
  – Helper thread runs ahead only on DIFT instructions
  – Tight coupling allows access to register values
Potential Applications

• Reliability
  – With 1000 cores on a single chip, and ultra low voltage logic, transient faults are a terrible issue.

  – Redundant Multithreading (RMT)
    • Create a leading thread and a trailing thread.
    • Perform exact same execution on both threads.
    • Commit results only if both threads yield same results.
Potential Applications

- **Reliability**

  - **Implementation**
    - SMT: Low hardware overhead; incurs switching overhead.
    - CMP: Simple to implement; repeats mis-speculations.
    - PEP cores gain advantages of both!
Potential Applications

• Self-Aware Computing
  – Requires a way to monitor itself:
    • SMT
    • Extra thread
    • Extra core

  – PEP core possesses detached execution context.
    • Main core keeps running!
Potential Applications

• Event Probing
Potential Applications

• Event Probing
Conclusion

• PEP cores allow for several optimizations with a low energy/area cost (<10%).

• Next steps:
  – Evaluate more applications with more benchmarks.
  – Implement a comprehensive scheme for all these applications.
  – Determine proper communication protocol between PEP cores and main cores.
Case Study: Memory Prefetching

```c
CarbonSpawnHelperThread(helper);
...
CarbonCondBroadcast(&resume_cond);
cur_node = node_vec;
for (n = 0; n < N; ++n, ++cur_node) {
    CarbonMutexLock(&counter_lock);
    shared_counter++;
    CarbonMutexunlock(&counter_lock);
    cur_value = val(cur_node);
    values = cur_node->values;
    coeffs = cur_node->coeffs;
    for (i = 0; i < stop; i++)
        cur_value -= values[i]*coeffs[i];
    val(cur_node) = cur_value;
}
```

```c
void * helper()
{
    while(1)
    {
        CarbonCondWait(&resume_cond);
        // Synchronization Code...
        local_counter = shared_counter++;
        ...
        values = cur_node->values;
        coeffs = cur_node->coeffs;
        for (i = 0; i < stop; i++)
            coeff_dummy = coeffs[i];
            value_dummy = values[i];
    }
}
```
Case Study: Memory Prefetching

**MAIN**

\[
\text{while (l\_counter < n\_nodes)} \{
    \text{while (1)} \{
        \text{while (1)} \{
            \text{CarbonMutexLock (&counter\_lock);}
            \text{c\_counter = shared\_counter;}
            \text{CarbonMutexUnlock (&counter\_lock);}
            \text{offset = l\_counter - c\_counter;}
            \text{if (offset >= MIN && offset < MAX)} \text{break;}
            \text{else if (offset <= MIN\_OFFSET)} \text{l\_counter = c\_counter + OFFSET; break;}
        \}
        \text{while (prev\_l\_counter < l\_counter)} \{
            \text{cur\_node++;}
            \text{prev\_l\_counter++;}
        \}
    \}
\]

**HELPER**

\[
\text{CarbonSpawnHelperThread (helper); ...}
\text{CarbonCondBroadcast (&resume\_cond);}
\text{cur\_node = node\_vec;}
\text{for (n = 0; n < N; ++n, ++cur\_node)} \{
    \text{CarbonMutexLock (&counter\_lock);}
    \text{shared\_counter++;}
    \text{CarbonMutexUnlock (&counter\_lock);}
    \text{cur\_value = val (cur\_node);}
    \text{values = cur\_node->values;}
    \text{coeffs = cur\_node->coeffs;}
    \text{for (i = 0; i < stop; i++)}
        \text{cur\_value -= values[i]*coeffs[i];}
    \text{val(cur\_node) = cur\_value;}
\}
Backup Slides

![Diagram of a system with clock gating, 128kb SRAM, cache, and power gating components.](image-url)
Case Study: Memory Prefetching

• Simulation Results on Graphite
  – Graphite instantiates two cores per tile.
  – Each core runs a separate context (stack, registers, etc.)
  – PEP and main cores contain private L1 and shared L2.
  – Synchronization through finite barrier across all threads.

• EM3D benchmark (Olden Suite)