Adding PEP to Multicore

Eric Lau

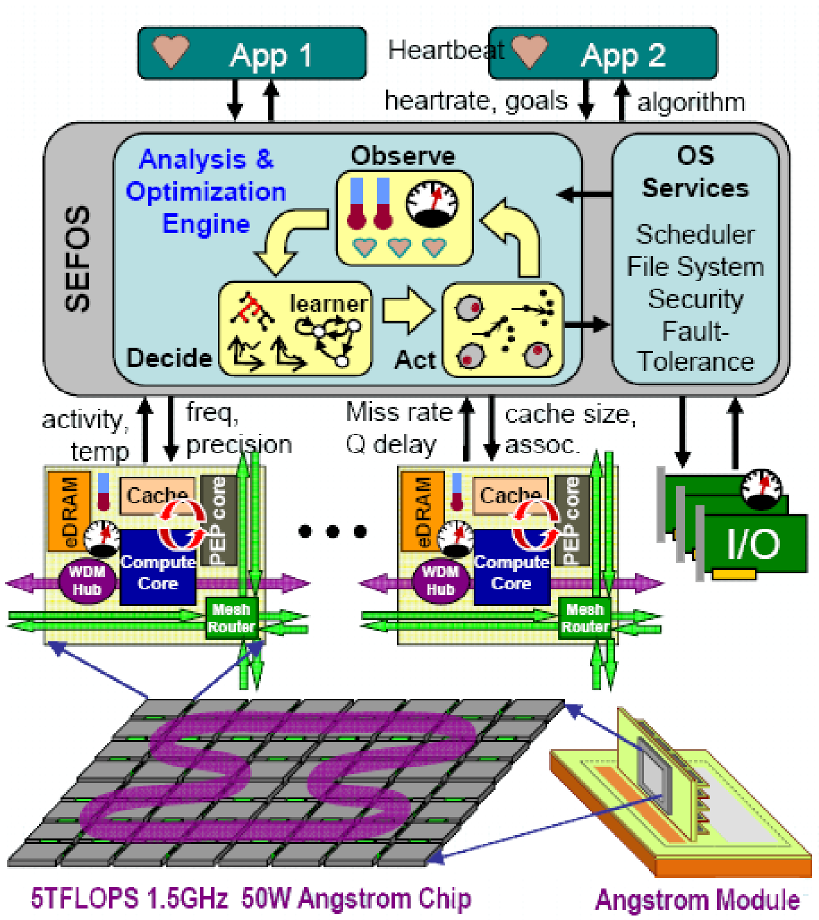
Jason Miller, Inseok Choi, Omer Khan, Jim Holt, Anantha Chandrakasan, Donald Yeung, Saman Amarasinghe, Anant Agarwal

Feb 16, 2011

Outline

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

Angstrom



Decrease programming effort

Increase performance

Increase resiliency

Increase energy efficiency

Positive Energy Partnerships

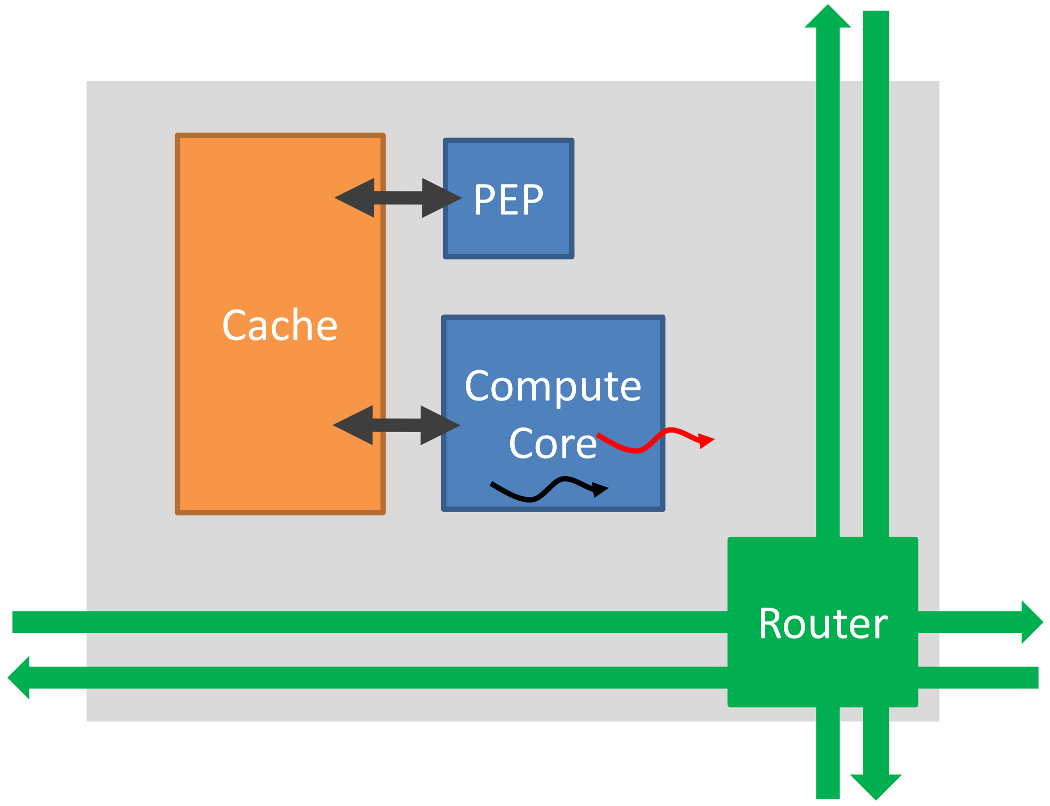
- *Non-application resources that perform tasks for a net gain in energy.*

- Hardware Partnerships
  - shared resources
  - hard-wired events

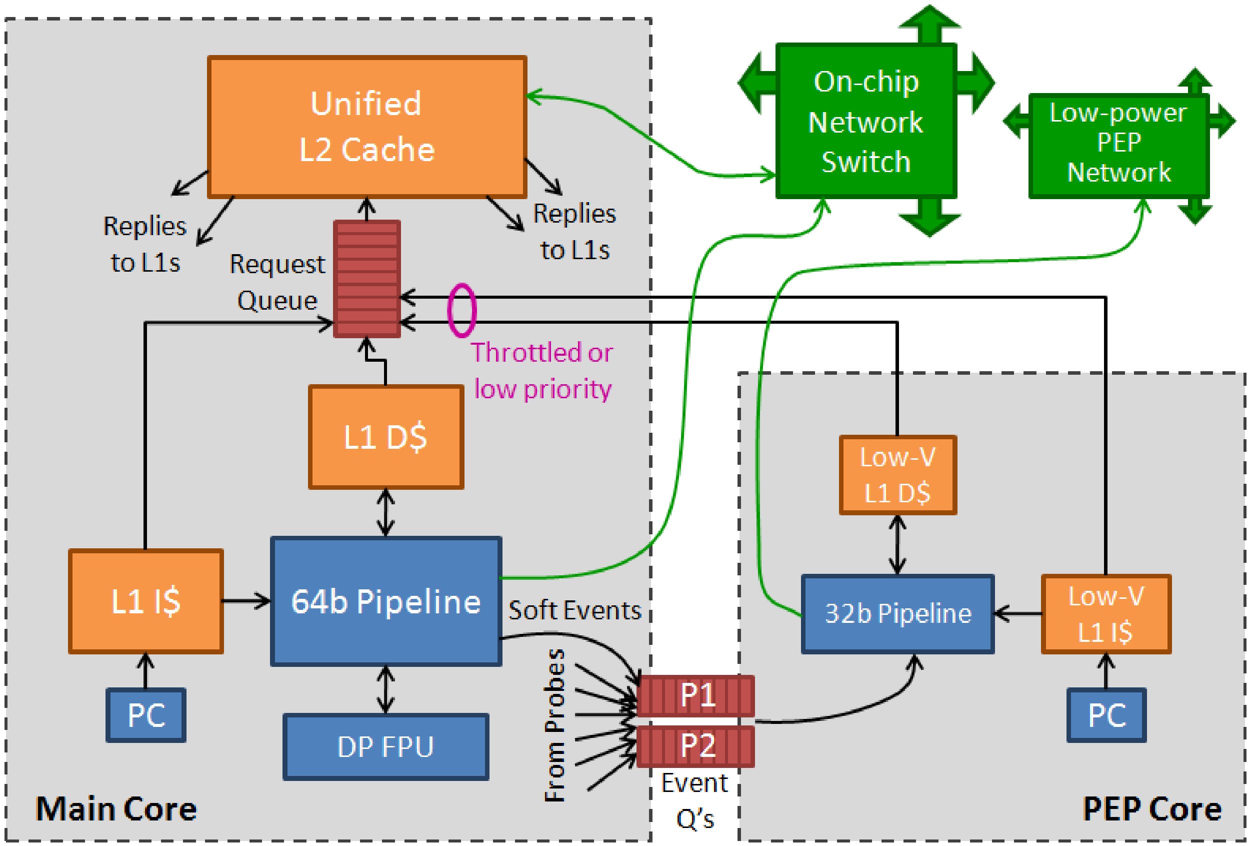
- Software Partnerships
  - expose to h/w ISA
  - algorithms
  - event servicing

Positive Energy Partnerships

- PEP Cores



PEP Core Architecture



PEP Core Architecture

- Area Considerations

Core	Node (nm)	SRAM	Size (mm^2)	Scaled Size (mm^2)
µController	65	128K	1.5	0.03
RAW	180	128K	16	0.25
Intel Core 2	65	2M/4M	80	9.0

- 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

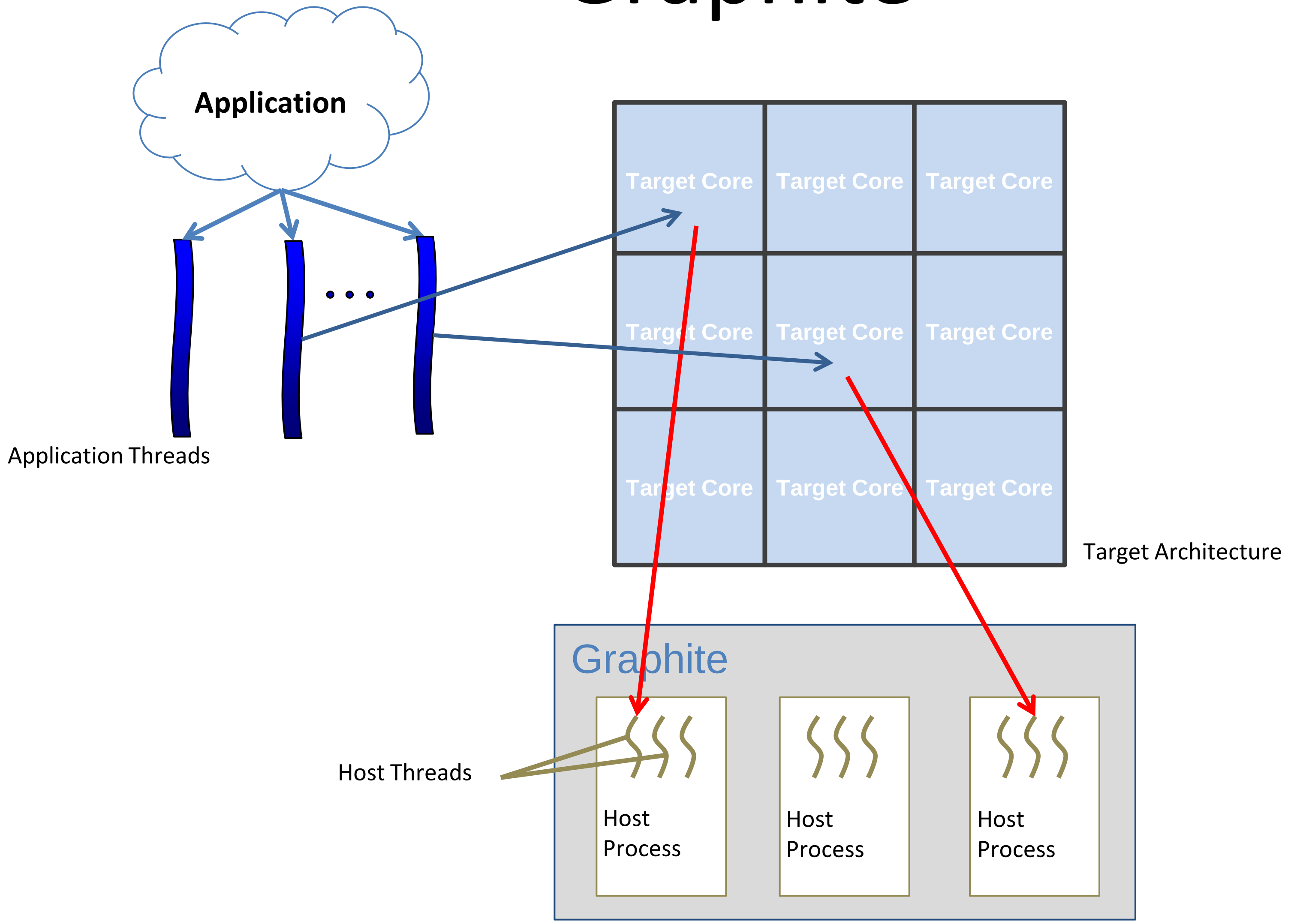
PEP Core Architecture

- Power Considerations

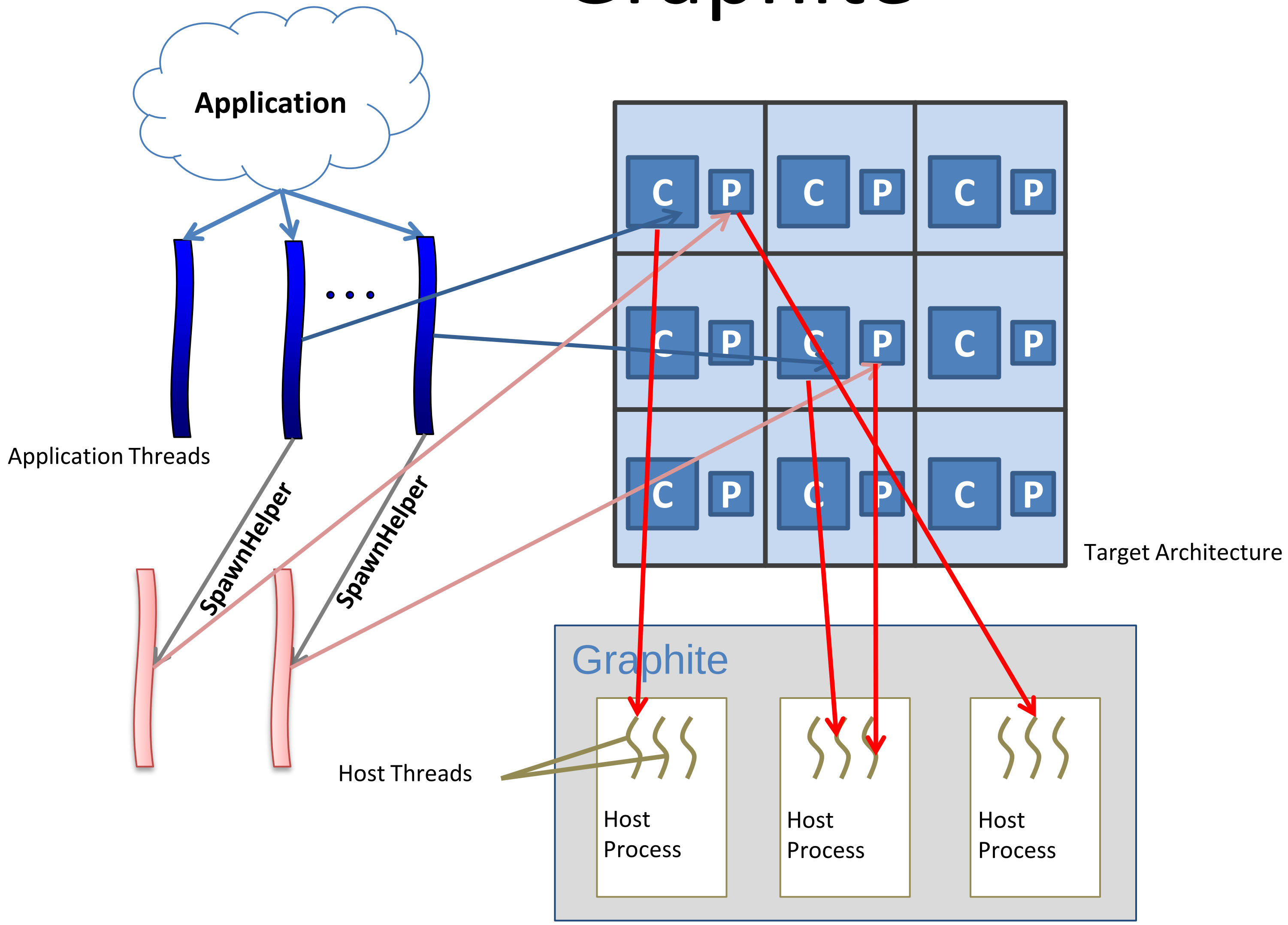
Core	Energy/Cycle
µController	27.2pJ
TilePro64	286pJ
Intel Core 2	101 000pJ

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

Graphite


Application
Application Threads
Target Core
Target Core
Target Core
Target Core
Target Core
Target Core
Target Core
Target Core
Target Core
Target Architecture
Graphite
Host Threads
Host Process
Host Process
Host Process

Graphite

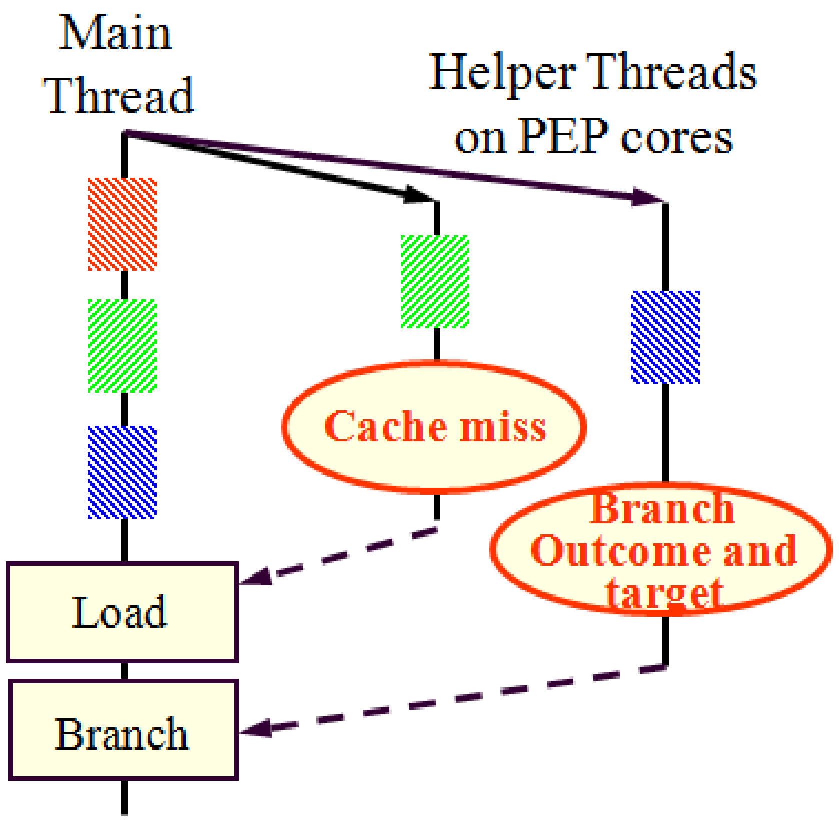

Application
. . .
Application Threads
SpawnHelper
SpawnHelper
Host Threads
C
P
Target Architecture
Graphite
Host Process
Host Process
Host Process

Potential Applications

- Case Study
  - Memory Prefetching
- Other Applications
  - Security
  - Reliability
  - Event Probing

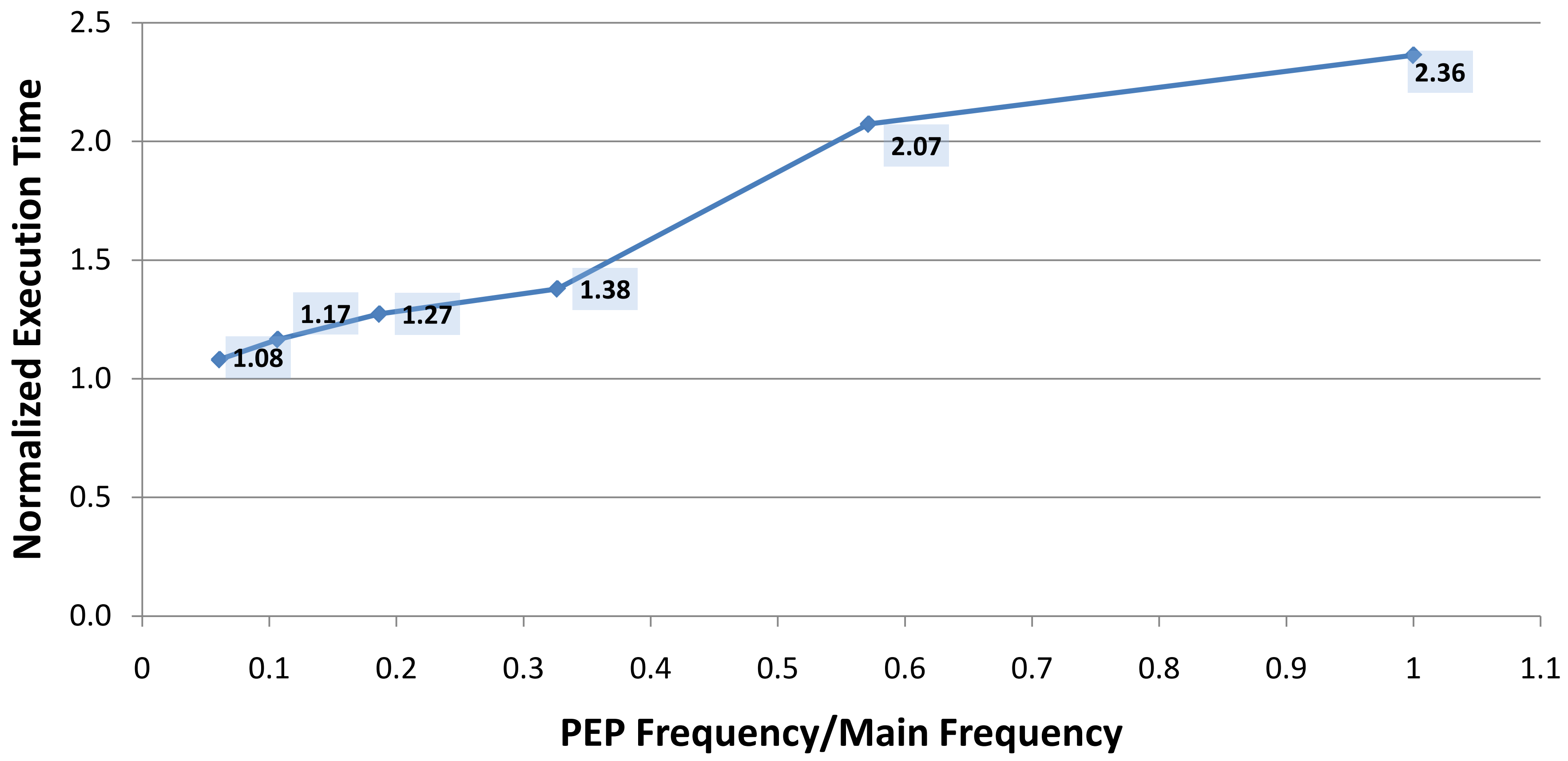
Case Study: Memory Prefetching

- 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



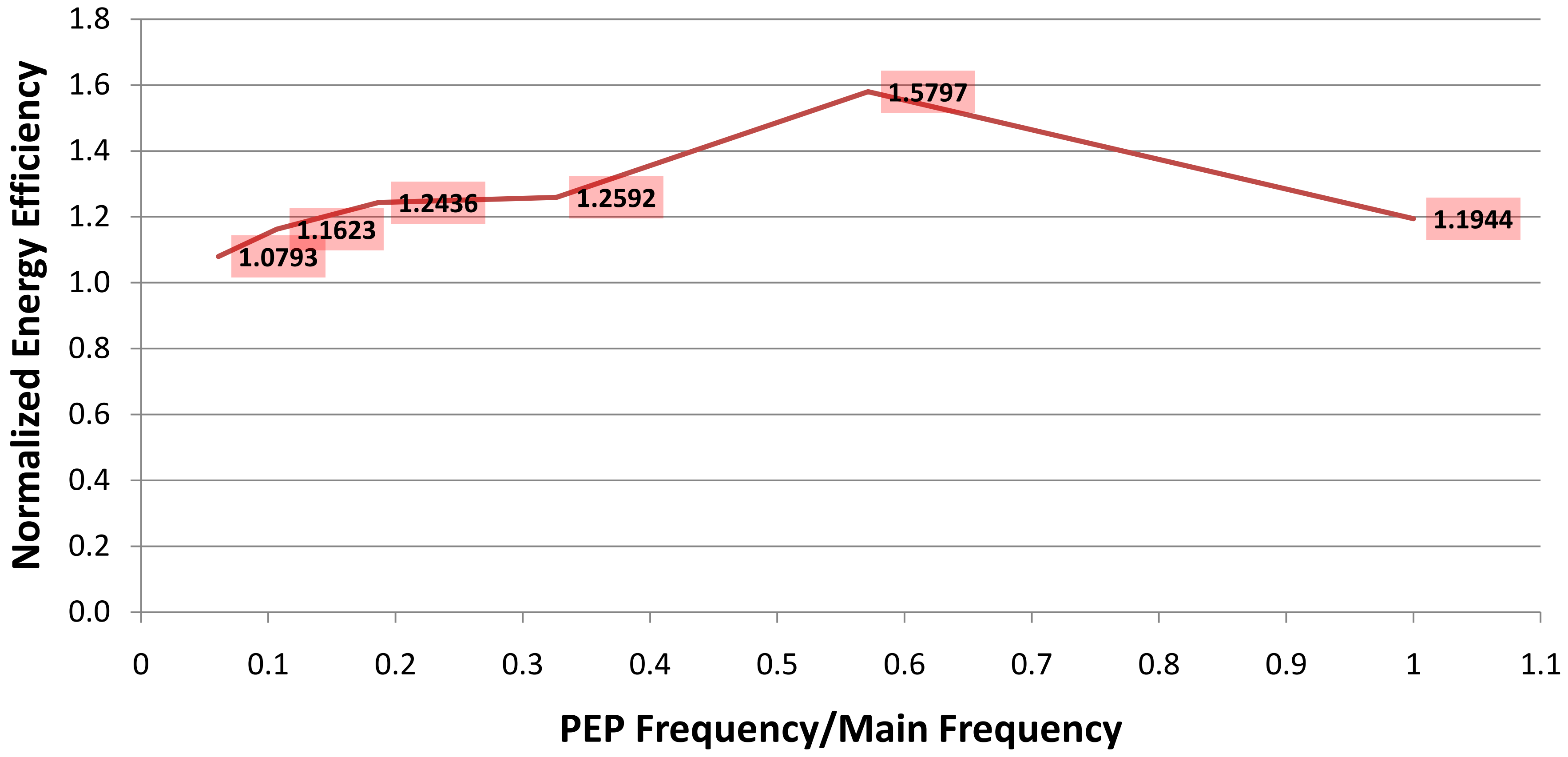
Case Study: Helper Threads

- EM3D benchmark (Execution Time)



Case Study: Helper Threads

- EM3D benchmark (Energy Efficiency)

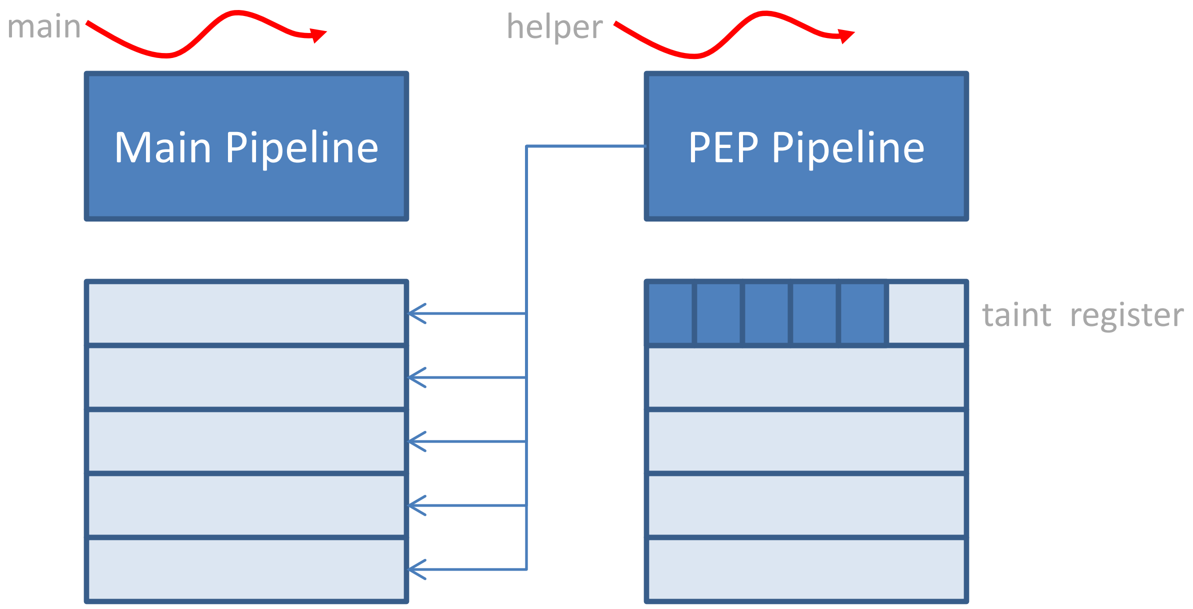


Potential Applications

- 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

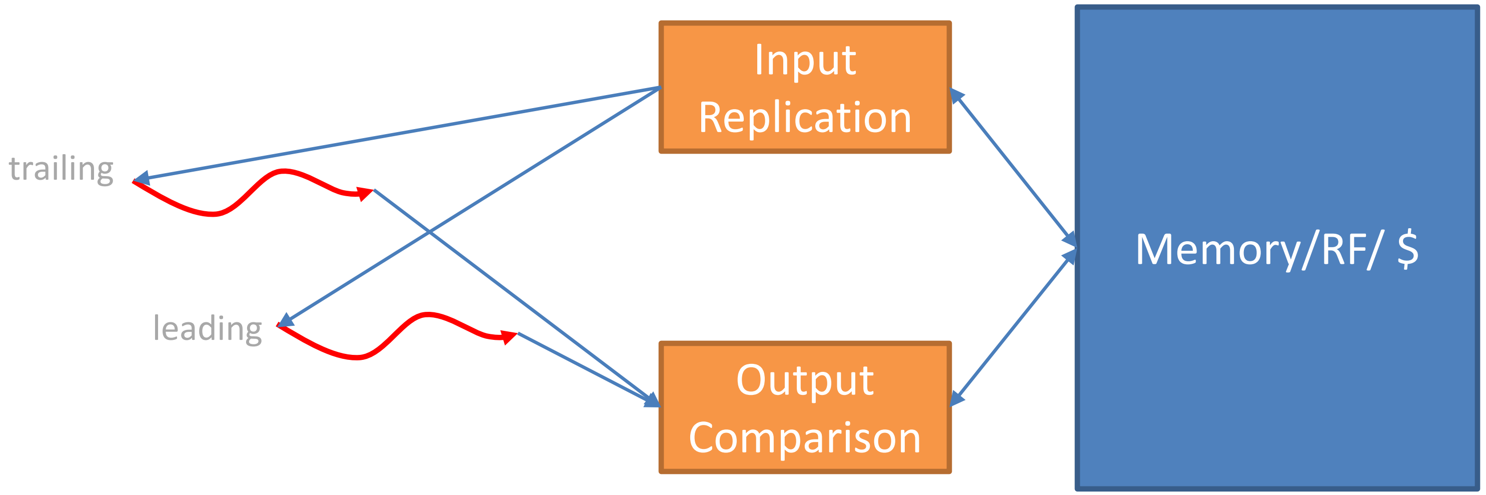


Potential Applications

- 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



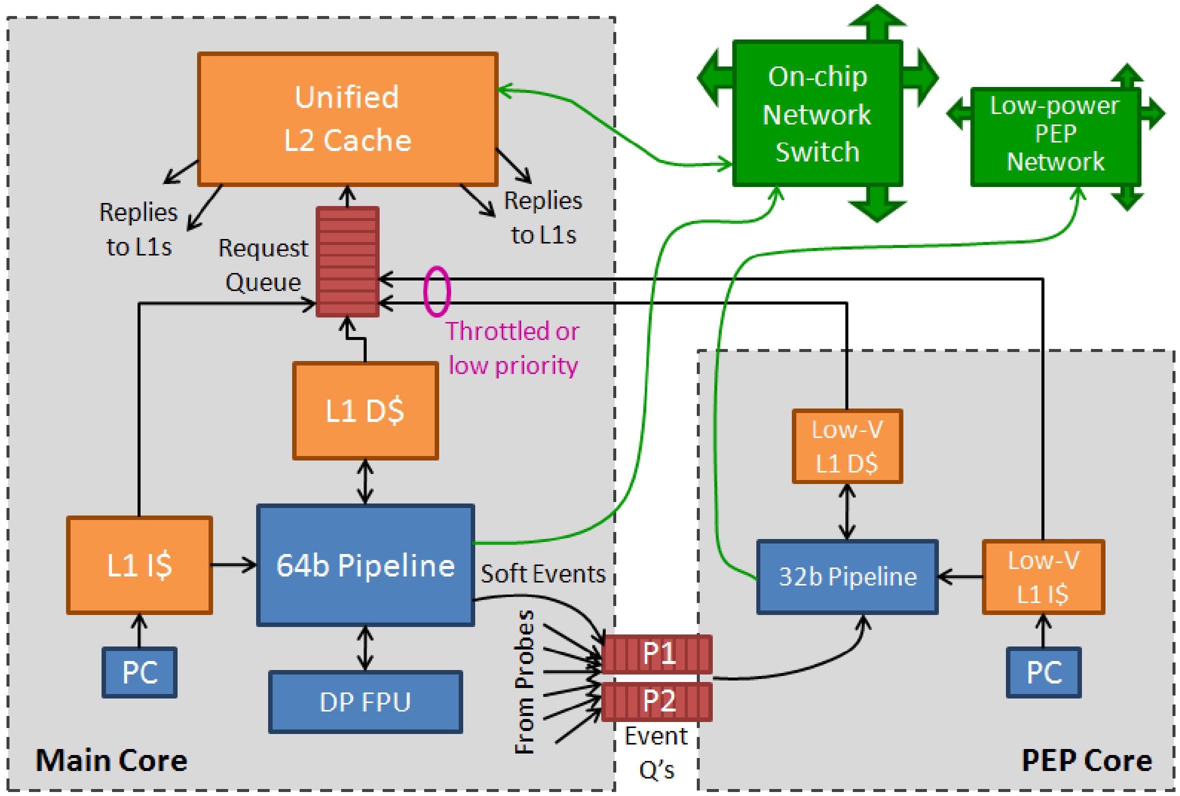
  - Implementation
    - SMT: Low hardware overhead; incurs switching overhead.
    - CMP: Simple to implement; repeats mis-speculations.
    - PEP cores gain advantages of both!

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!

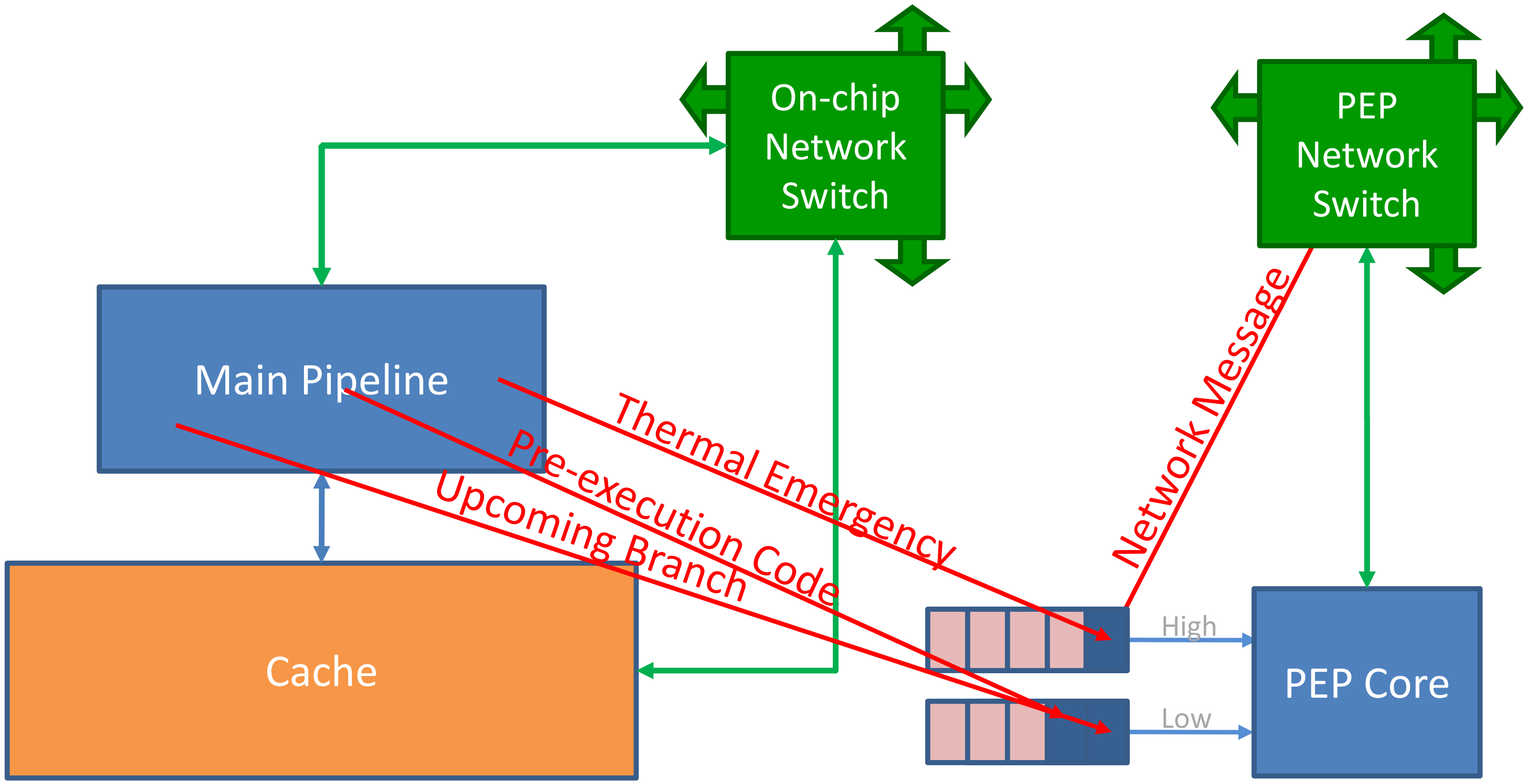
Potential Applications

- 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

```
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;
}
```

MAIN

```
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];
     }
  }
}
```

HELPER

Case Study: Memory Prefetching

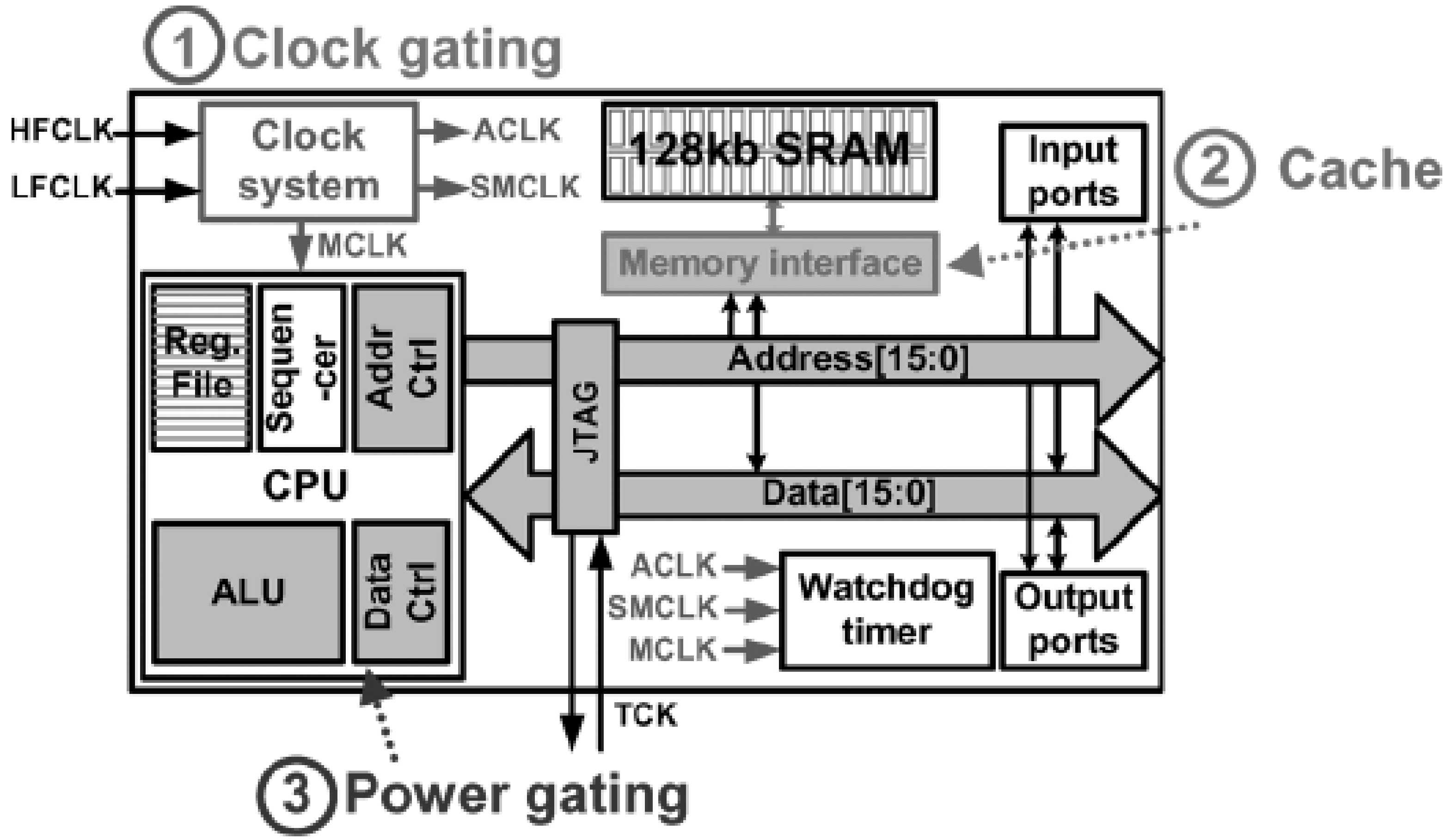
```
CarbonSpawnHelperThread(helper);
...

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;
}
```

MAIN

```
while (l_counter < n_nodes)
{
 while(1) {
    CarbonMutexLock(&counter_lock);
    c_counter = shared_counter;
    CarbonMutexUnlock(&counter_lock);

    offset = l_counter - c_counter;

    if (offset >= MIN && offset < MAX)
      break;
    else if ( offset <= MIN_OFFSET)
      l_counter = c_counter + OFFSET;
      break;
  }

  while(prev_l_counter < l_counter)
  {
    cur_node++;
    prev_l_counter++;
  }
}
```

HELPER

Backup Slides



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)