µ-op Fission: Hyper-threading without the Hyper-headache

Anthony Cartolano

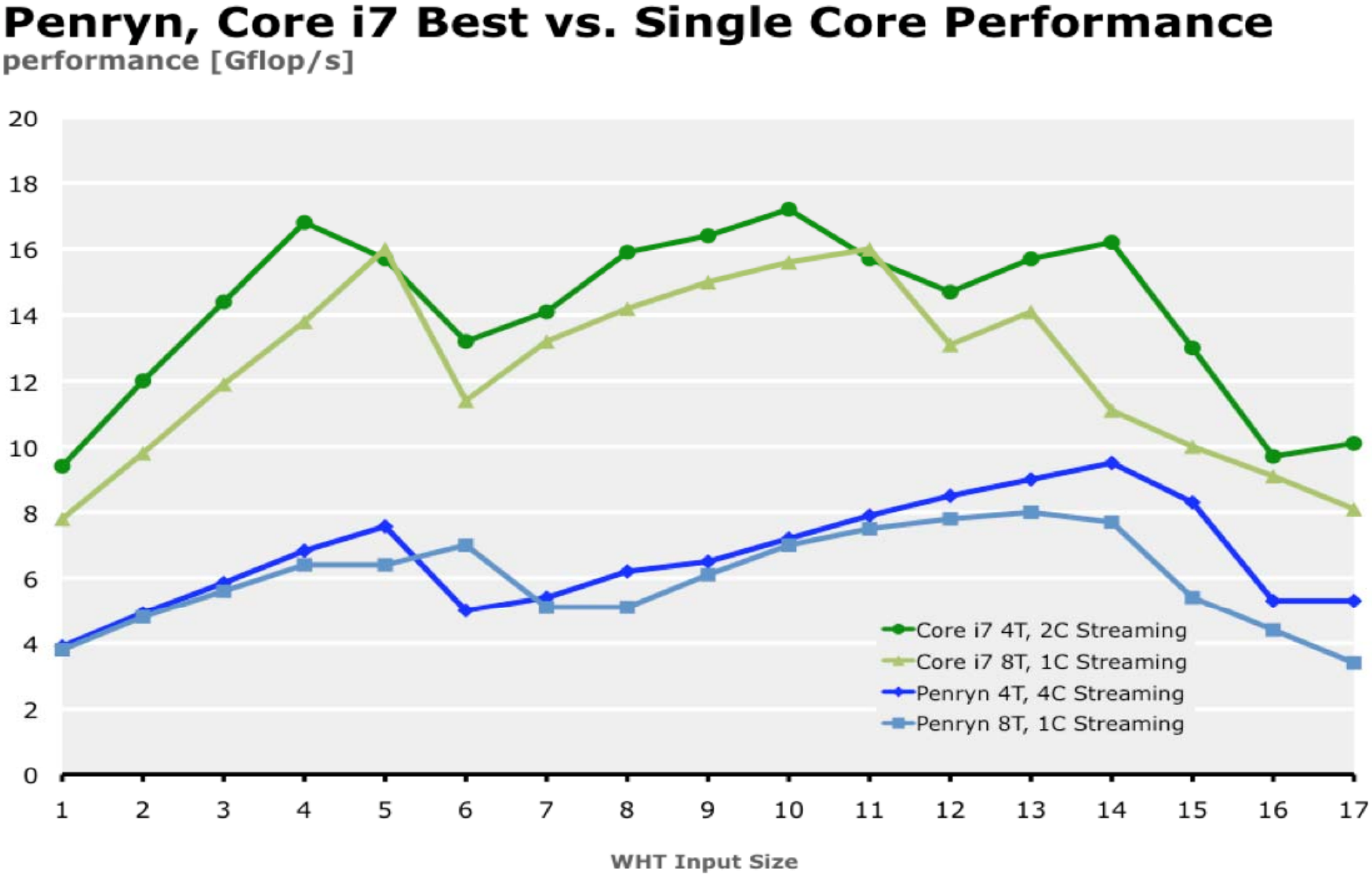
Robert Koutsoyannis

Daniel S. McFarlin

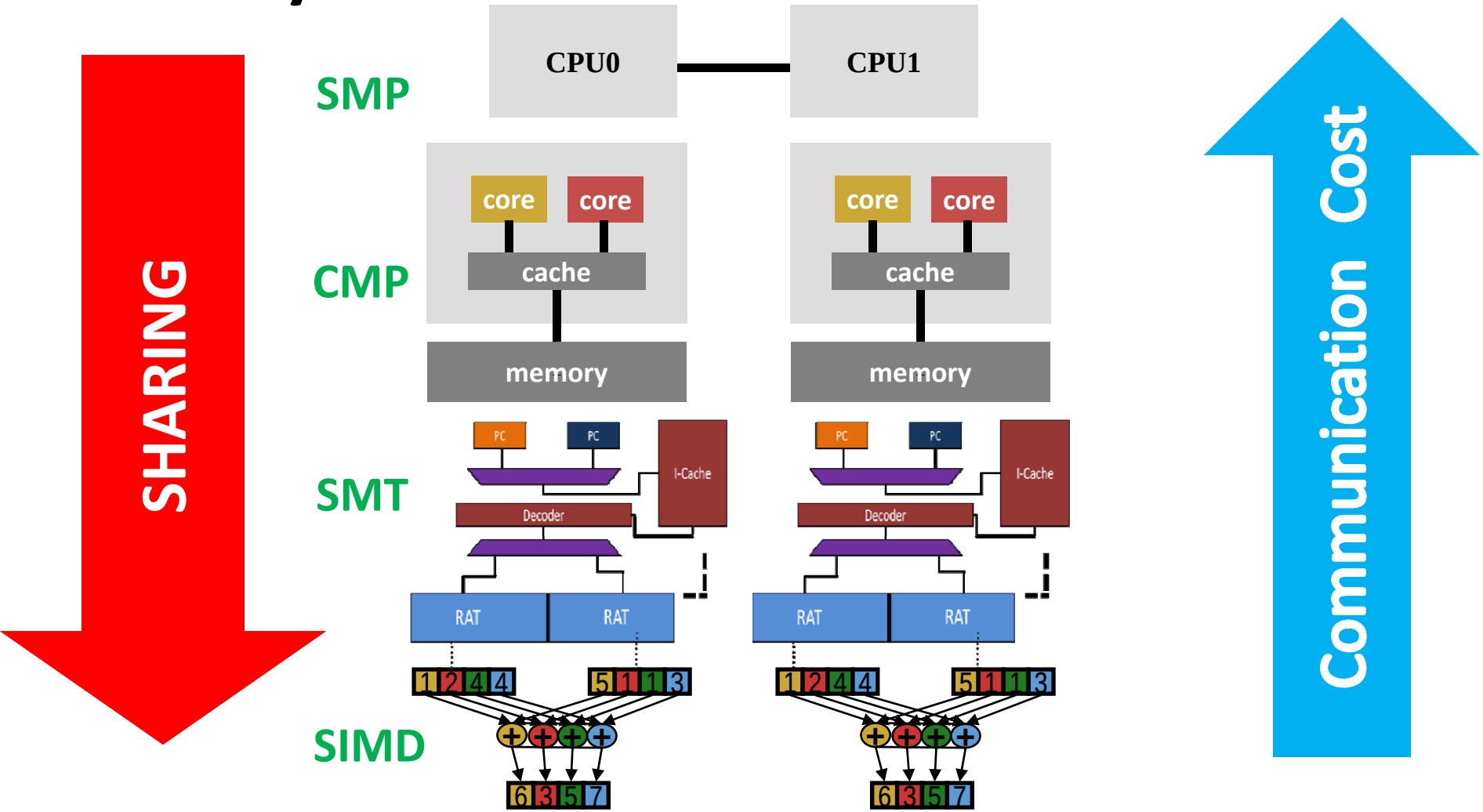
Carnegie Mellon University

Dept. of Electrical and Computer Engineering

Streaming WHT Results on Intel Multicore: Best Versus Single Core

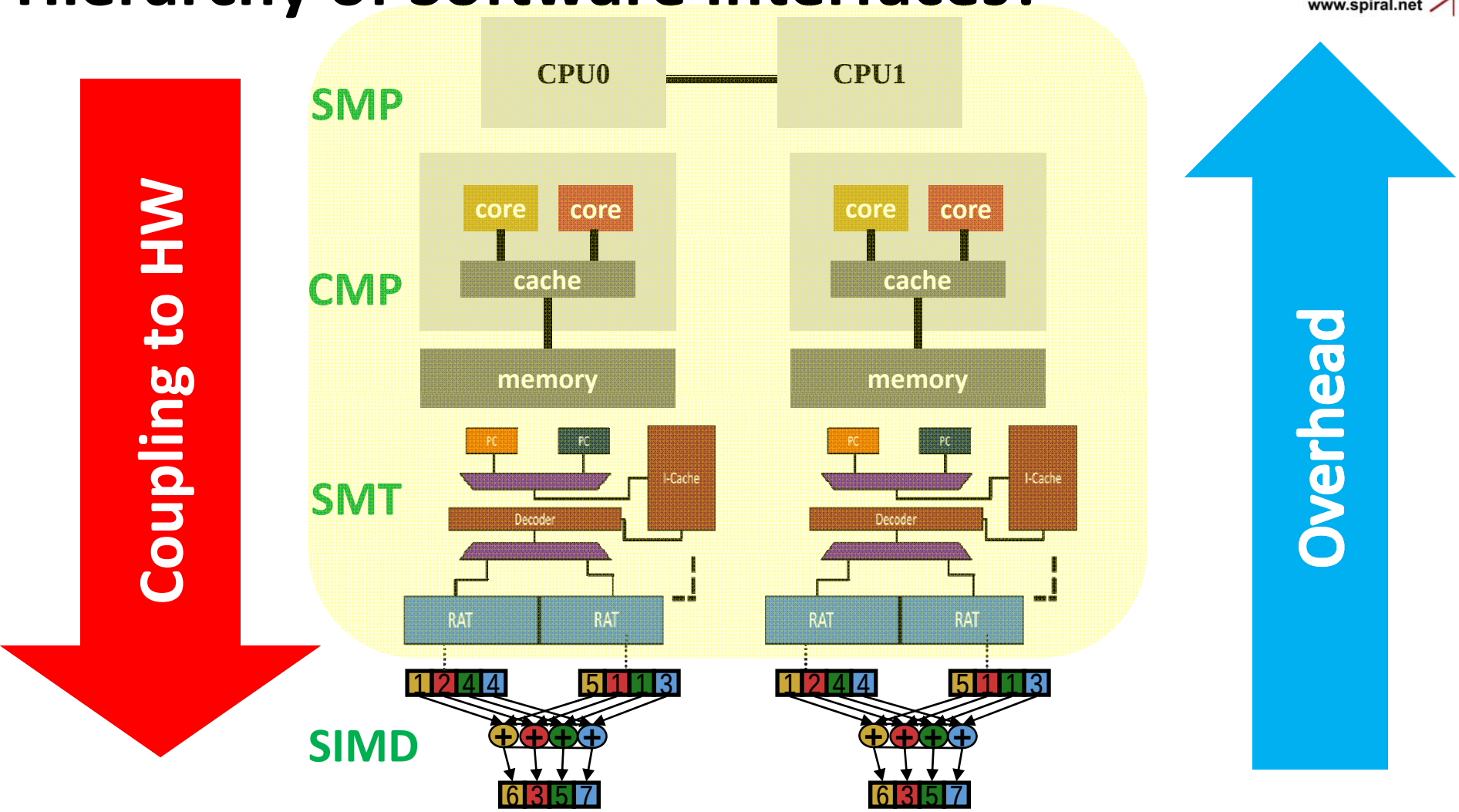


Hierarchy of Parallel Hardware Features



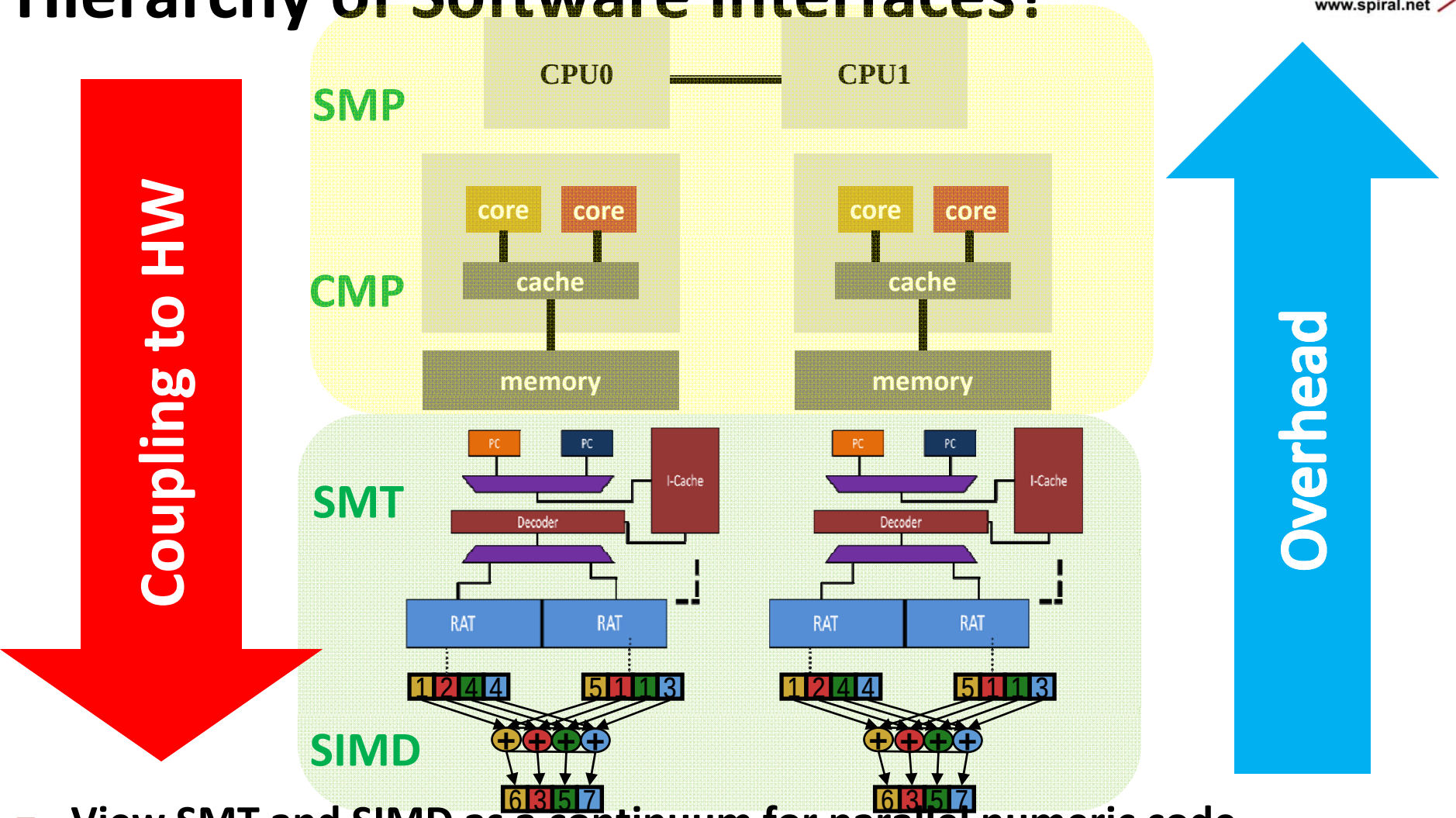
- Characterized by degree of sharing α (Cost of Communication)$^{-1}$
- SIMD: share register file, functional units, cache hierarchy
- SMP: share main memory and QPI link

Hierarchy of Software Interfaces?



- General Purpose SW threading model to rule them all: affinity knob
- Overhead: intrinsics vs. OpenMP vs. APIs + Runtimes
- Amortizing runtime overhead dictates partitioning: precludes SMT

Hierarchy of Software Interfaces?

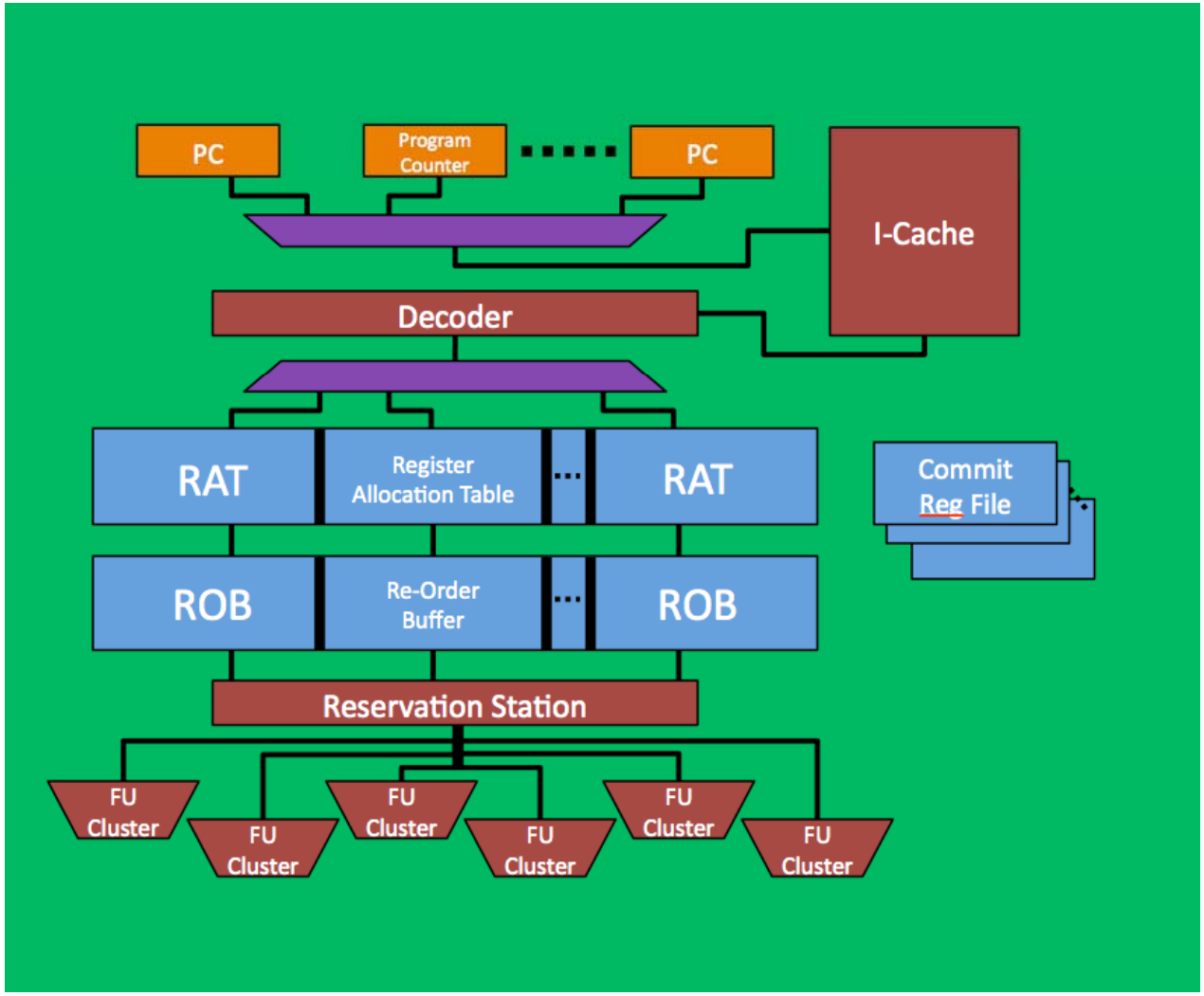


- View SMT and SIMD as a continuum for parallel numeric code
- Expose SMT to SW and Encode SMT loop parallelization with intrinsics to minimize overhead
- Requires HW/SW support and co-optimization to achieve

Outline

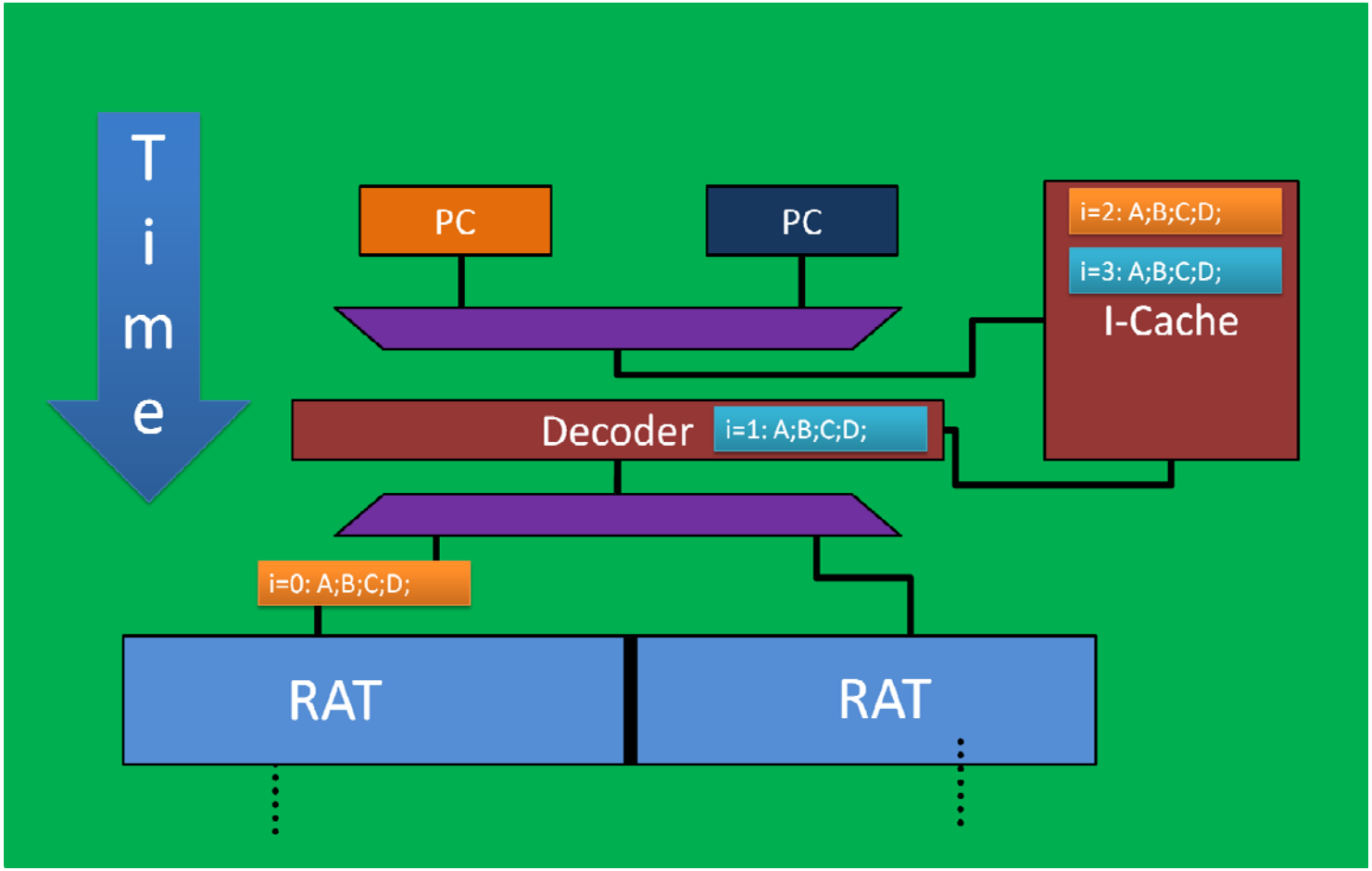
- Motivation
- SMT Hardware Bottlenecks
- μ-op Fission
- Required software support
- Experimental Results
- Concluding Remarks

Generic N-way SMT



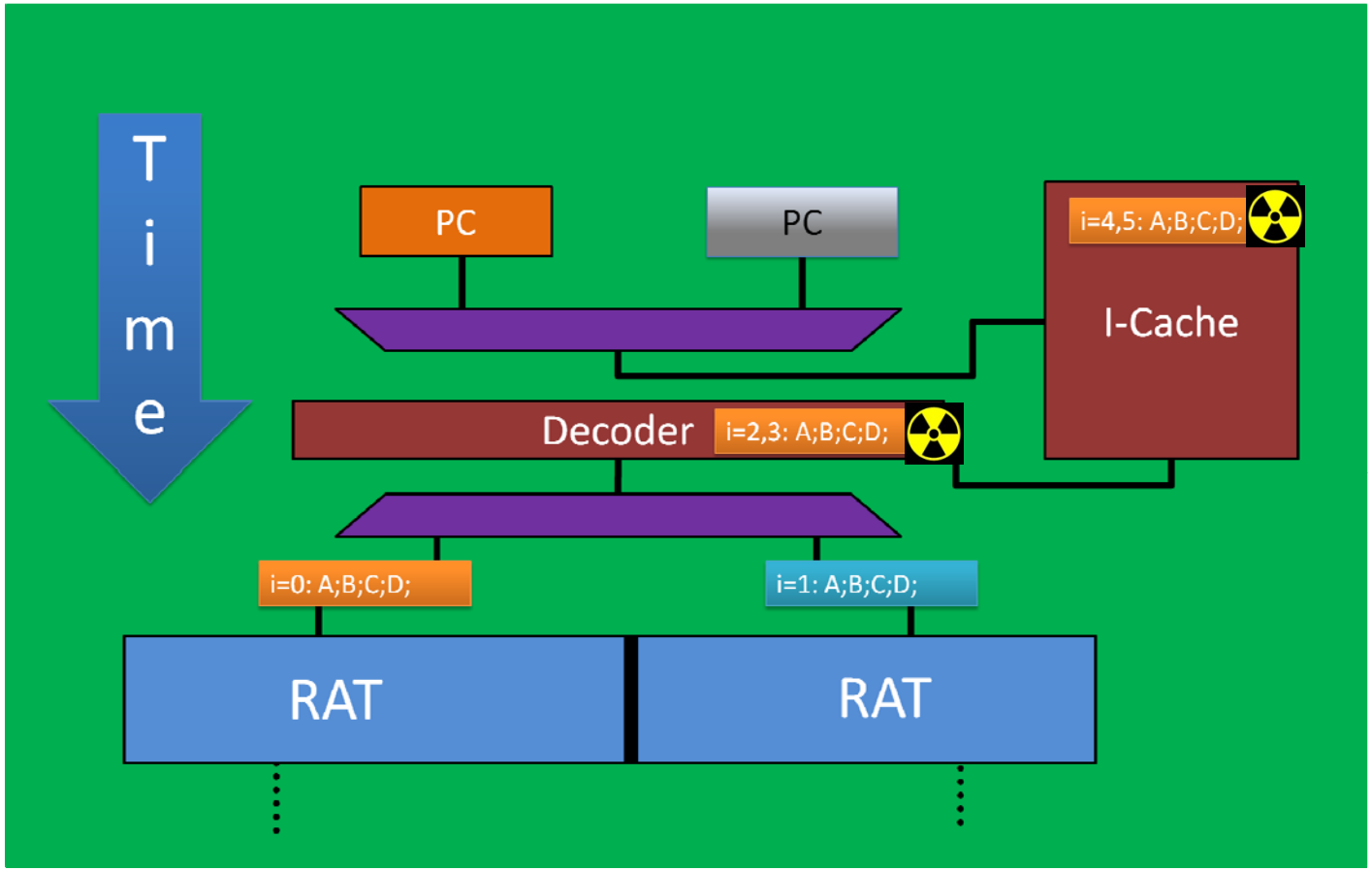
- **Narrow Front-End vs. Wide Backend**

2-Way SMT Case Study



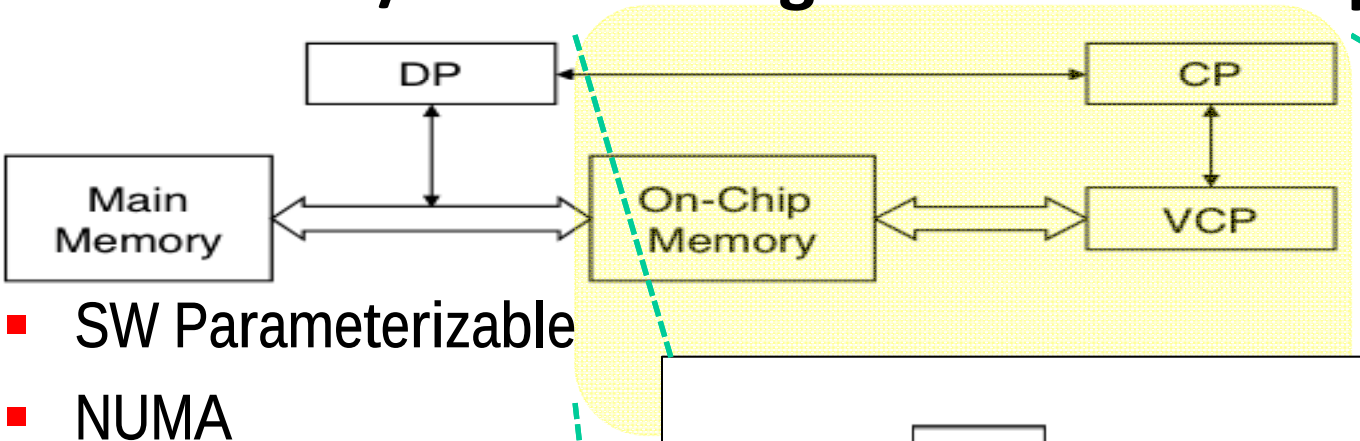
- Front-End (Fetch & Decode) is implemented as Fine-Grained Multi-threading
  - Done to reduce i-cache, decode latency
  - Can lead to back-end starvation

µ-op Fission

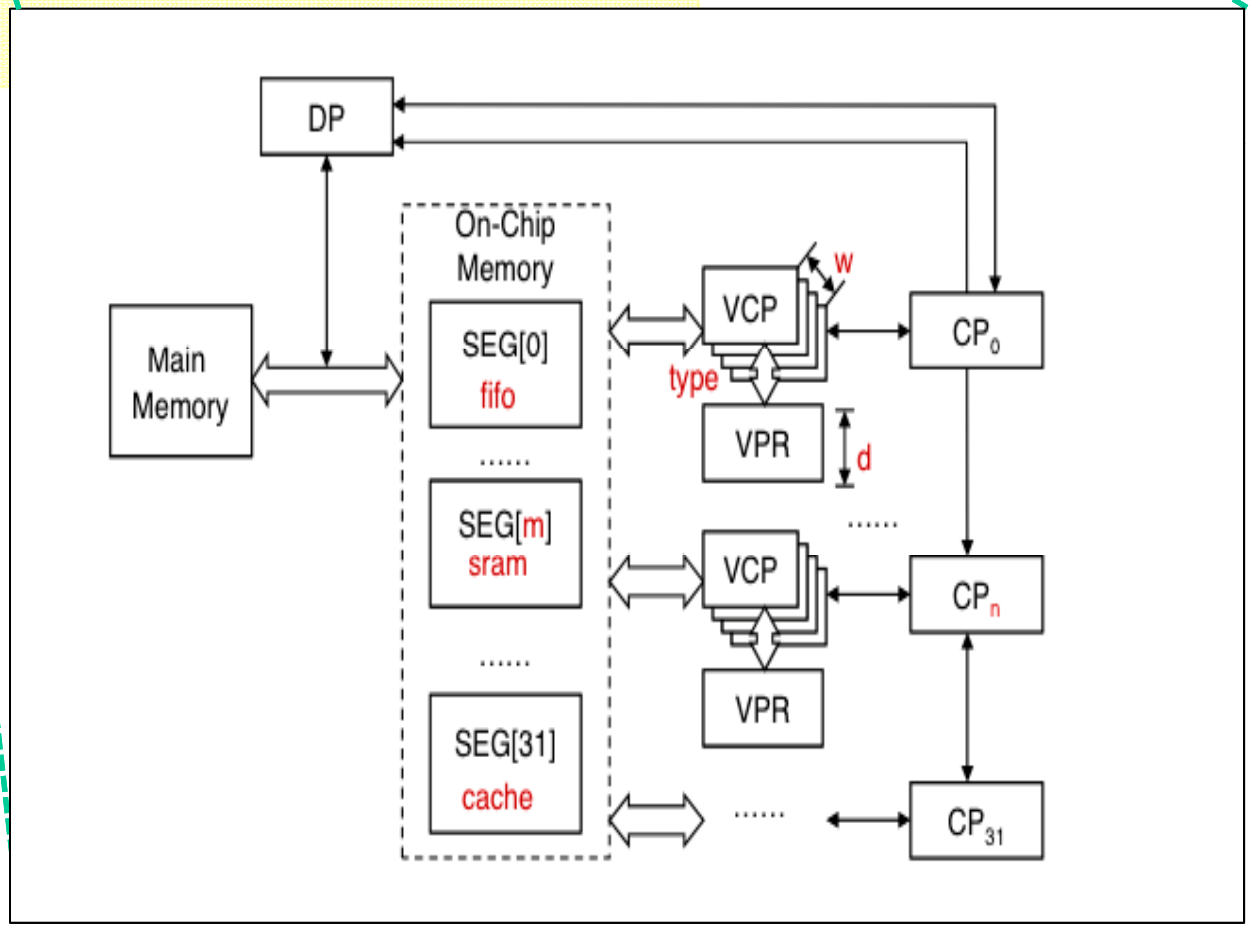


- 2 iterations compiler-fused with clone bit
  - Equivalent to unroll-and-jam optimization
- Decoder fissions iterations into available RATs
  - Reduces fetch & decode bandwidth and power
  - Allows narrow front-end to keep wide back-end pipeline full

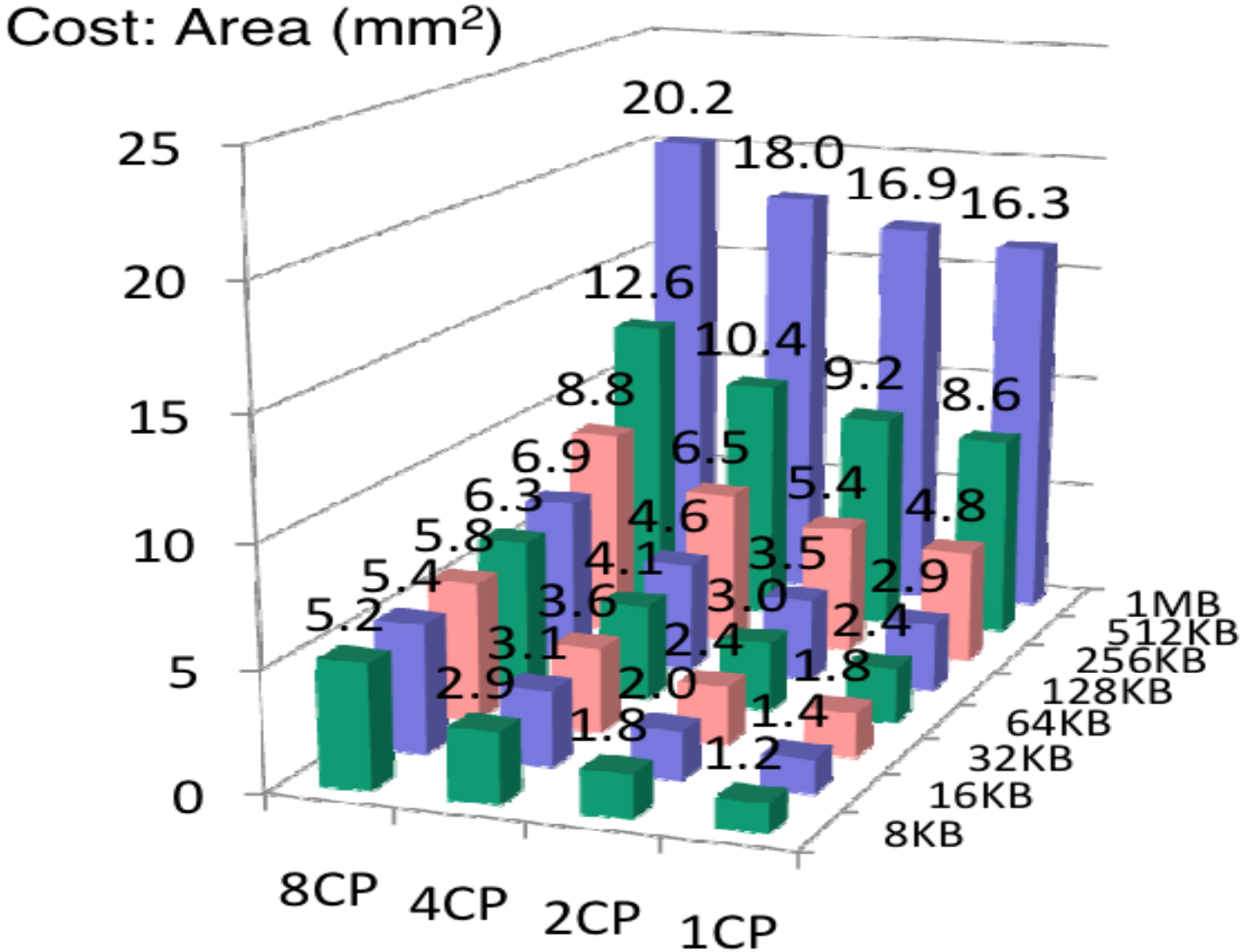
Formal HW/SW Co-Design: The Data Pump Architecture



- SW Parameterizable
- NUMA
- SW Programmable Memory Controller
- Decoupled Compute & Communication Instruction Streams
- Asynch Gather/Scatter
  - MEM <-> LM
  - VRF <-> LM
- Simple Manycore
- SIMD



High Dimensional Non-Uniform HW Parameter Space



Software Architecture: Spiral

SPIRAL
www.spiral.net

- **Library generator for linear transforms (DFT, DCT, DWT, filters, ….) *and recently more …***

- **Wide range of platforms supported: scalar, fixed point, vector, parallel, Verilog, GPU**

- **Research Goal: "Teach" computers to write fast libraries**
  - Complete automation of implementation and optimization
  - Conquer the "high" algorithm level for automation

- **When a new platform comes out: Regenerate a retuned library**

- **When a new platform paradigm comes out (e.g., CPU+GPU): Update the tool rather than rewriting the library**

Intel uses Spiral to generate parts of their MKL and IPP libraries

Spiral: A Domain Specific Program Generator

SPIRAL www.spiral.net

Transform
user specified

$$\mathrm{DFT}_8$$

Fast algorithm in SPL
many choices

$$(\mathrm{DFT}_2 \otimes I_4)\, T^8_4\, (I_2 \otimes ((\mathrm{DFT}_2 \otimes I_2) \cdot T^4_2\, (I_2 \otimes \mathrm{DFT}_2)\, L^4_2))\, L^8_2$$

∑-SPL:

$$\sum \left(S_j\, \mathrm{DFT}_2\, G_j\right) \sum \left(\sum \left(S_{k,l}\, \mathrm{diag}\left(\mathrm{t}_{k,l}\right) \mathrm{DFT}_2\, G_l\right) \sum \left(S_m\, \mathrm{diag}\left(\mathrm{t}_m\right) \mathrm{DFT}_2\, G_{k,m}\right)\right)$$

C Code:

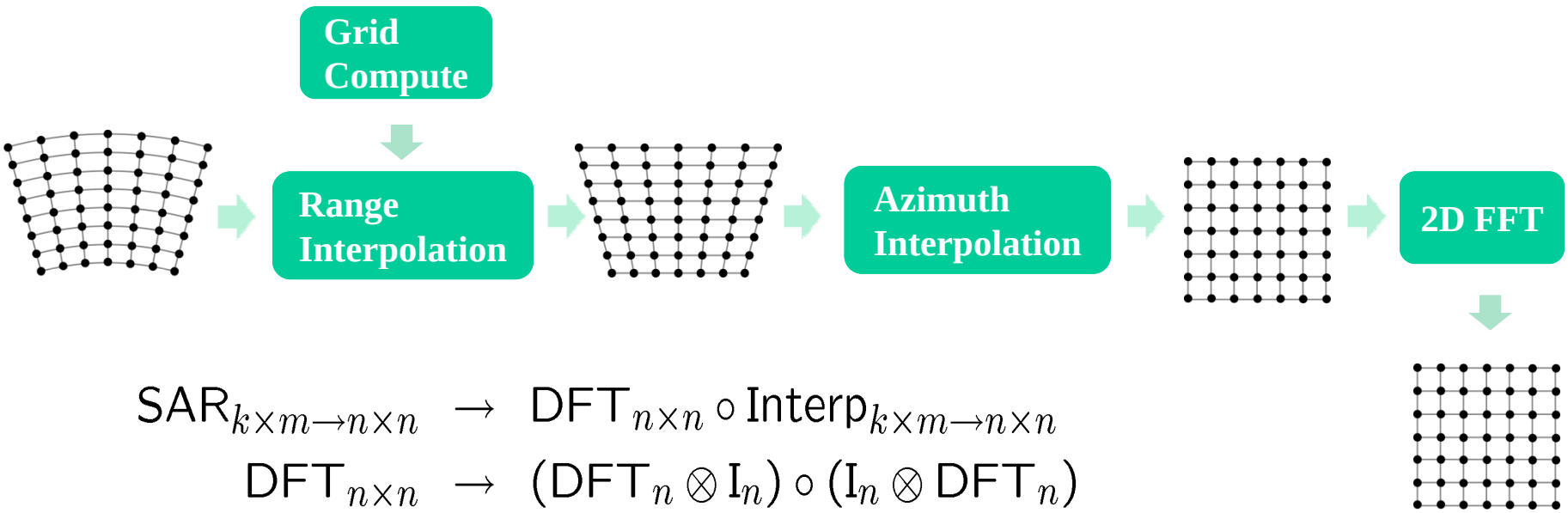
```
void sub(double *y, double *x) {
  double f0, f1, f2, f3, f4, f7, f8, f10, f11;
  f0 = x[0] - x[3];
  f1 = x[0] + x[3];
  f2 = x[1] - x[2];
  f3 = x[1] + x[2];
  f4 = f1 - f3;
  y[0] = f1 + f3;
  y[2] = 0.7071067811865476 * f4;
  f7 = 0.9238795325112867 * f0;
  f8 = 0.3826834323650898 * f2;
  y[1] = f7 + f8;
  f10 = 0.3826834323650898 * f0;
  f11 = (-0.9238795325112867) * f2;
  y[3] = f10 + f11;
}
```

Optimization at all abstraction levels

- **parallelization**
  vectorization
- **loop optimizations**
- **constant folding**
  scheduling
  ……

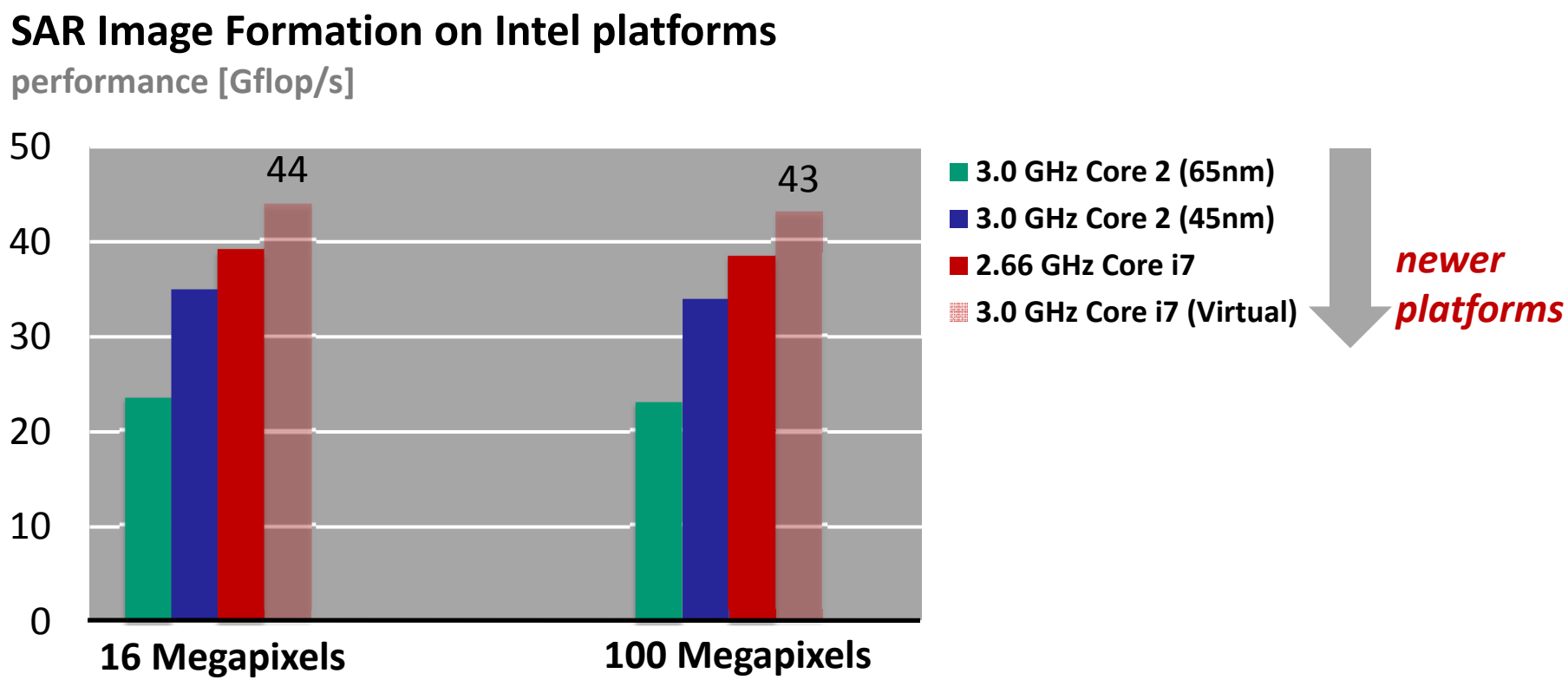
Iteration of this process to search for the fastest

Spiral Formula Representation of SAR



$$
\begin{aligned}
\mathsf{SAR}_{k\times m\to n\times n} &\to \mathsf{DFT}_{n\times n} \circ \mathsf{Interp}_{k\times m\to n\times n} \\
\mathsf{DFT}_{n\times n} &\to (\mathsf{DFT}_n \otimes \mathrm{I}_n) \circ (\mathrm{I}_n \otimes \mathsf{DFT}_n) \\
\mathsf{Interp}_{k\times m\to n\times n} &\to (\mathsf{Interp}_{k\to n} \otimes_i \mathrm{I}_n) \circ (\mathrm{I}_k \otimes_i \mathsf{Interp}_{m\to n}) \\
\mathsf{Interp}_{r\to s} &\to \left(\bigoplus_{i=0}^{n-2} \mathsf{InterpSeg}_k\right) \oplus \mathsf{InterpSegPruned}_{k,\ell} \\
\mathsf{InterpSeg}_k &\to \mathsf{G}_f^{u\cdot n\to k} \circ \mathsf{iPrunedDFT}_{n\to u\cdot n} \circ \left(\frac{1}{n}\right) \circ \mathsf{DFT}_n
\end{aligned}
$$

Spiral's Automatically Generated PFA SAR Image Formation Code

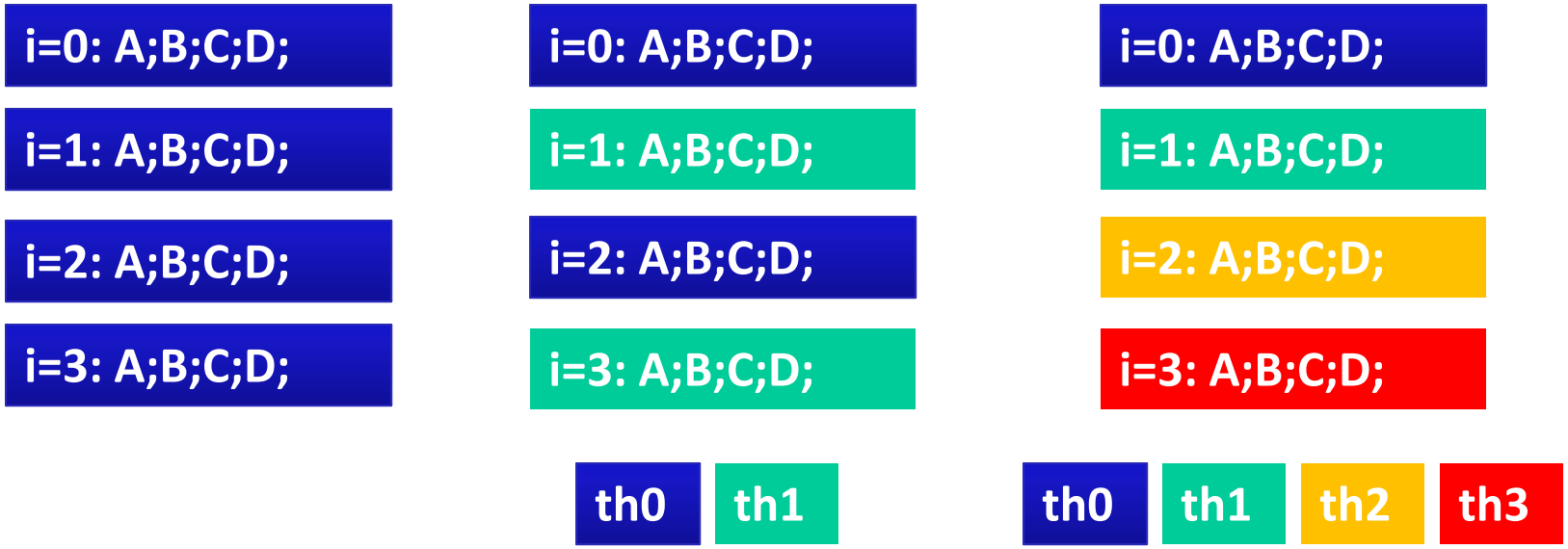


- Algorithm byJ. Rudin (best paper award, HPEC 2007): 30 Gflop/s on Cell
- Each implementation: vectorized, threaded, cache tuned, ~13 MB of code
- *Code was not written by a human*

Required Software Support

- **Executable has stub code which initializes pagetables and other CPU control registers at load time on all HW contexts**

- **Compiler performs virtual Loop-Unroll-and-Jam on tagged loops**
  - Maximizes sharing
  - SMT Thread Cyclic Partitioning



Fork Support

- **Need a lightweight fork mechanism**
  - Presently, Can only communicate between SMT register files via memory

Compile Time

```
store 0, (a0,0);   store 1, (a0,1);   store 2, (a0,2);   store 3, (a0,3);

load (a0, APIC_ID), i;

load (a0,0), i;   load (a0,1), i;   load (a0,2), i;   load (a0,3), i;
```

  - Load/Store Queue prevents materialization in most cases

- Prefer to have multi-assign statement for loop index with a vector input

```
i= {0,1,2,3};   →   i=0;   i=1;   i=2;   i=3;
```

- Need broadcast assignment for live-in set to the loop

```
load addr, a0;   →   load addr, a0;   load addr, a0;
                     load addr, a0;   load addr, a0;
```

Sample Loop Execution

Compile Time

```
i = {0,1};
load addr, a0;
L0:  cmp  i,  n;
jmpgte  END;
A;B;C;D;
add 2,  i;
jmp L0
END:  waitrobs
```

```
i = 0;
load addr, a0;
L0:  cmp  i,  n;
jmpgte  END;
A;B;C;D;
add 2,  i;
jmp L0
END:  waitrobs
```

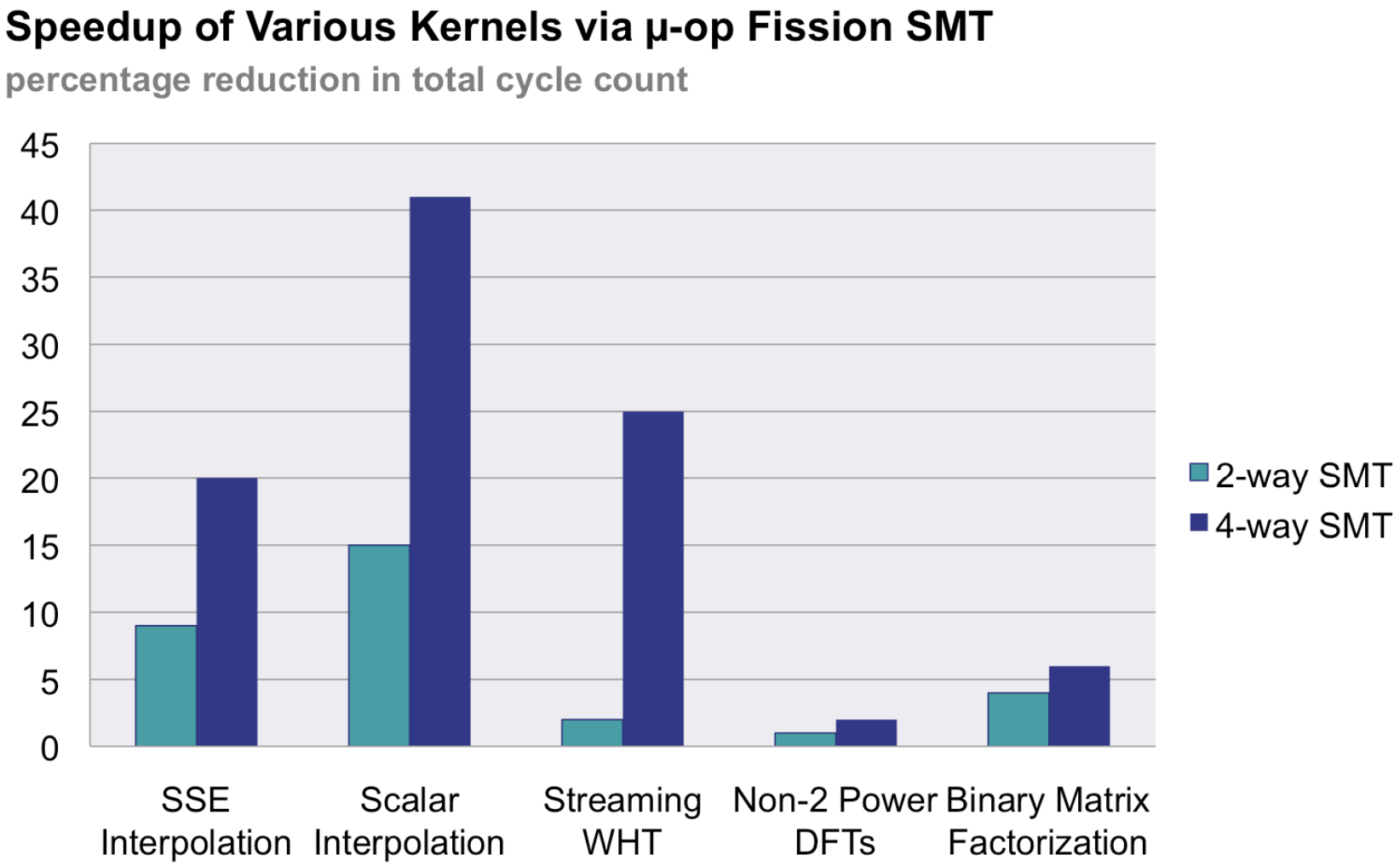
```
i = 1;
load addr, a0;
L0:  cmp  i,  n;
flushgte
A;B;C;D;
add 2,  i;
```

Experimental Setup

Architecture Parameter(s)	Value(s)
Fetch/Decode Width	8
Dispatch/Issue Width	4
Commit Width	4
Load/Store Queue	48/32
ROB Size	256
Physical Register File Size	256
Physical Register Files	3
Load/Store/Arithmetic/FP Units	2/2/2/2
L1 Cache Size/Latency	16 KB/1 cycle
L2 Cache Size/Latency	256 KB/5 cycles
L3 Cache Size/Latency	4 MB/8 cycles
Main Memory Latency	140 cycles

- **Used PTLSim with above configuration**
  - **Larger ROB size + physical register size than Nehalem**
  - **Smaller number of functional units**
  - **Simulate µ-op fission with explicit unroll-and-jam of source code coupled with penalized functional unit latencies**

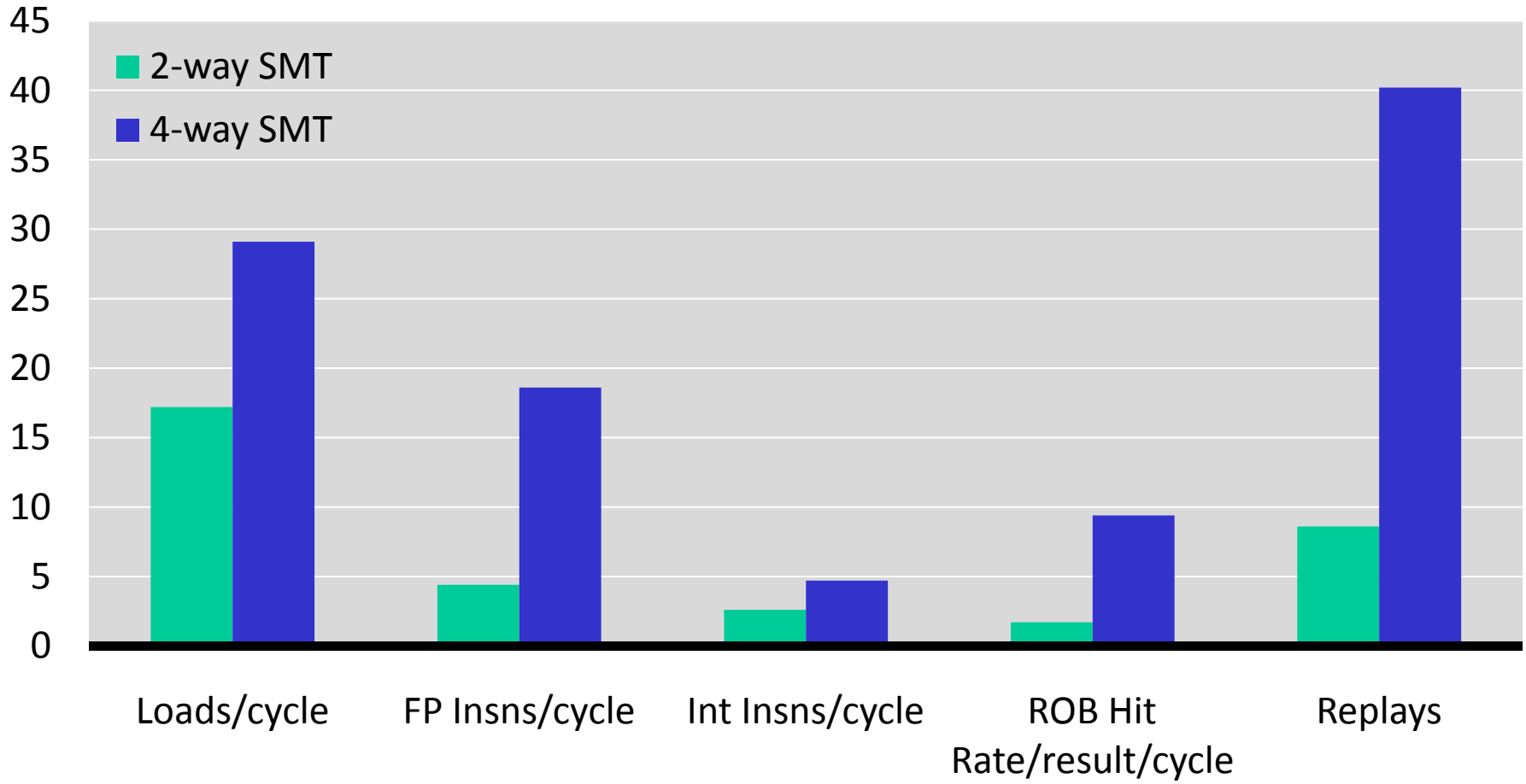
Experimental Results



Results Drilldown

Performance Improvement of Various µArch Metrics for µ-op Fissioned SSE interpolation Kernel

percentage improvement over baseline



Concluding Remarks

- Demonstrated a HW/SW Co-optimization approach to SMT parallelization
- Preliminary evaluation suggests performance benefit for a range of numerical kernels
- Scales with number of SMT contexts
- "Thread Fusion" research suggests a 10-15% power consumption reduction is possible due to reduced fetch/decode
- Future work: handling control-flow with predication and diverge-merge

THANK YOU!

SPIRAL
www.spiral.net

Questions?