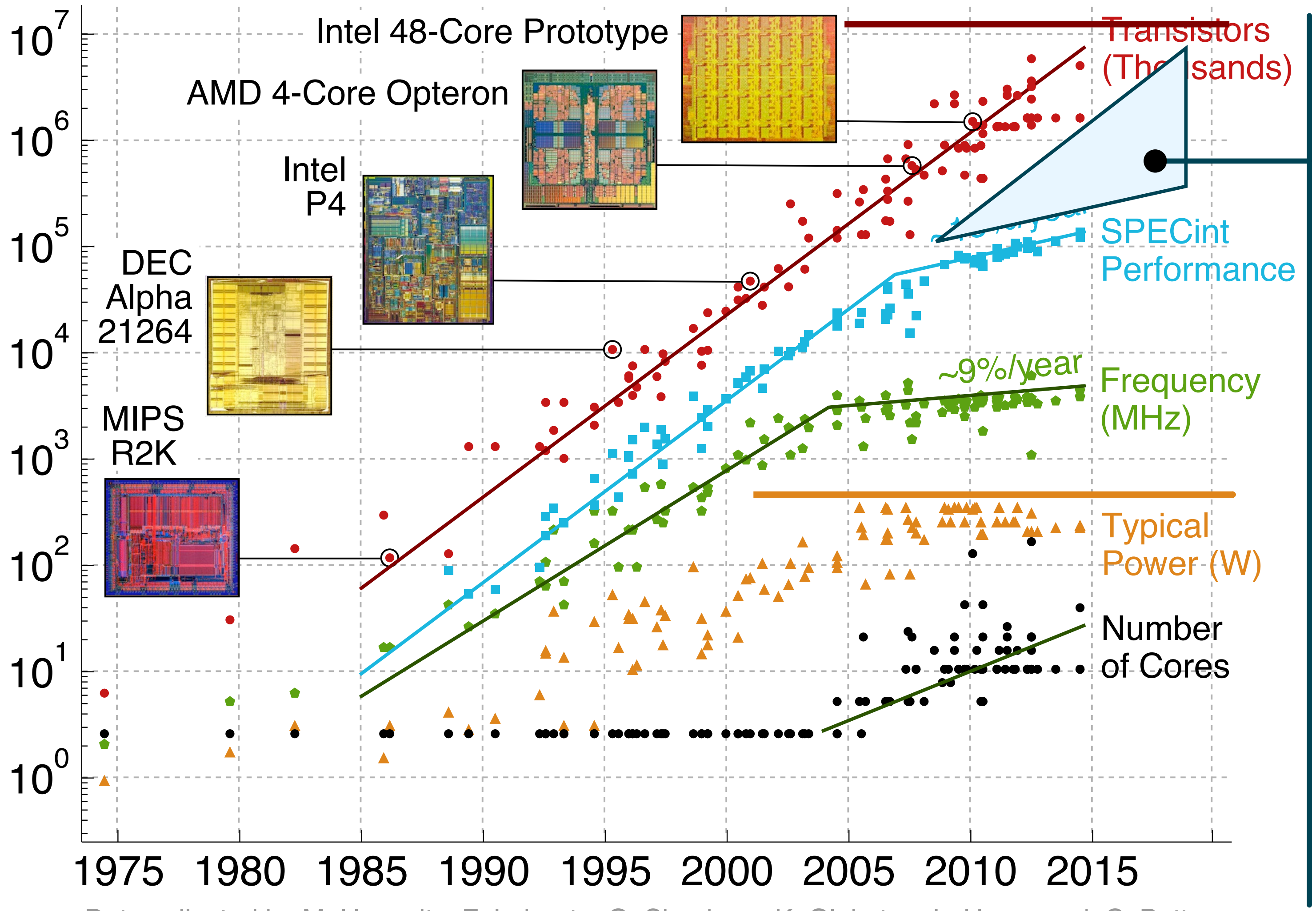
Intra-Core Loop-Task Accelerators for Task-Based Parallel Programs

Christopher Batten

Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University

Spring 2018

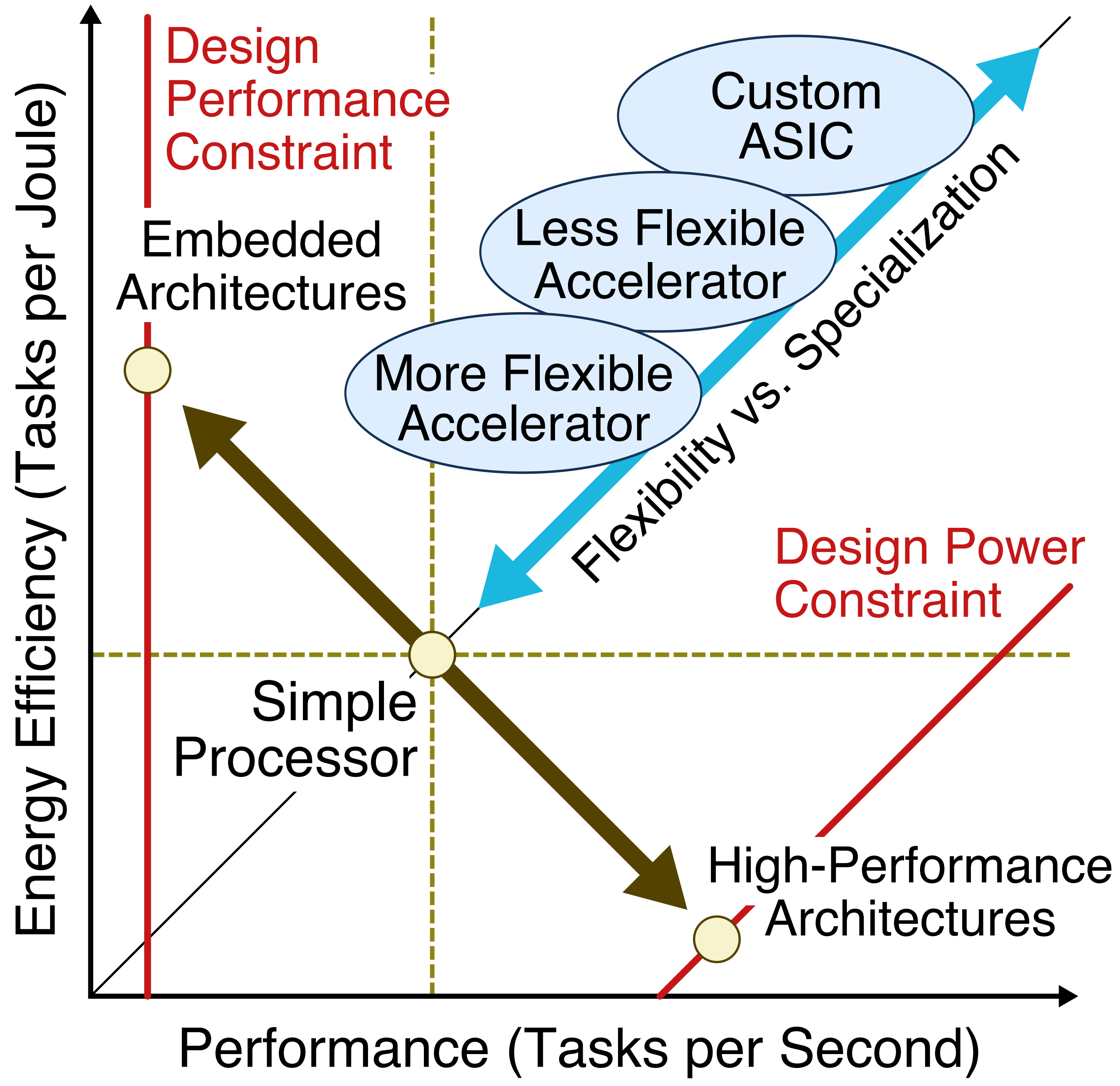
Motivating Trends in Computer Architecture



Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten

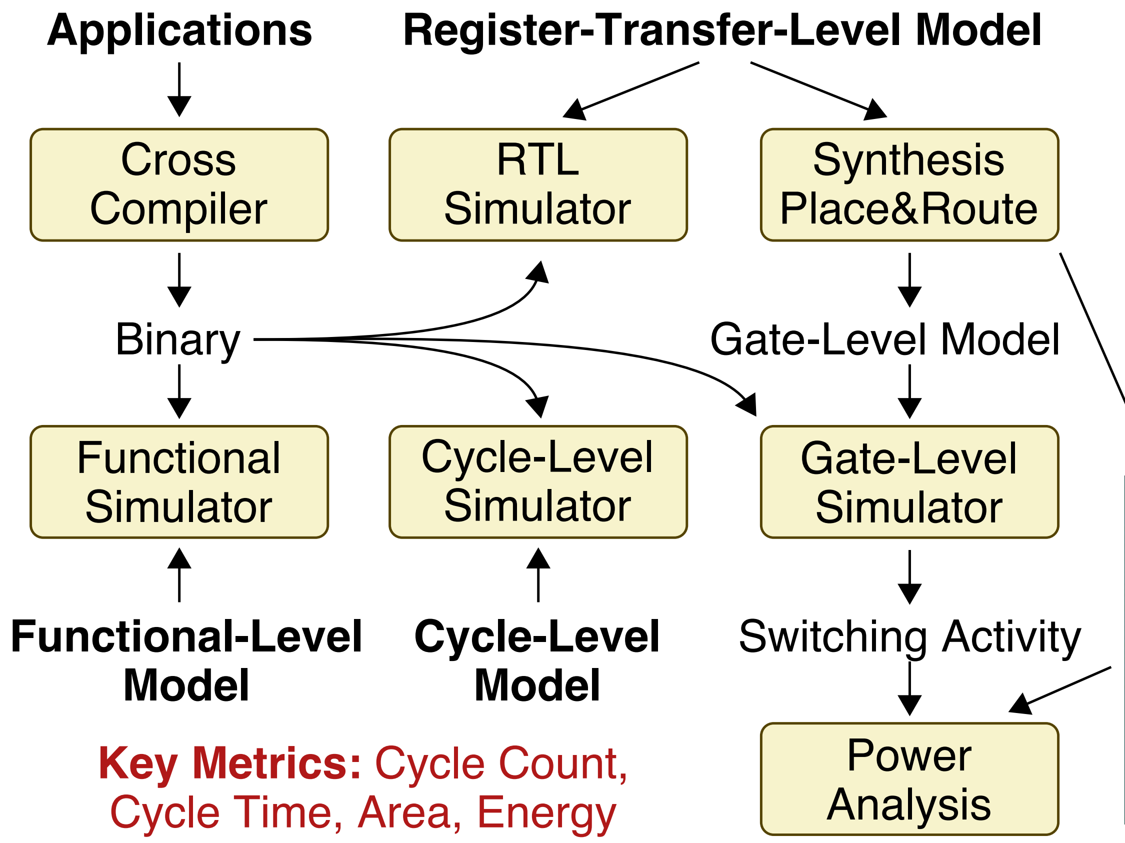
Hardware Specialization

- Data-Parallelism via GPGPUs & Vector
- Fine-Grain Task-Level Parallelism
- Instruction Set Specialization
- Subgraph Specialization
- Application-Specific Accelerators
- Domain-Specific Accelerators
- Coarse-Grain Reconfig Arrays
- Field-Programmable Gate Arrays


Design
Performance
Constraint
Embedded
Architectures
Custom
ASIC
Less Flexible
Accelerator
More Flexible
Accelerator
Flexibility vs. Specialization
Design Power
Constraint
Simple
Processor
High-Performance
Architectures
Energy Efficiency (Tasks per Joule)
Performance (Tasks per Second)

Vertically Integrated Research Methodology

Our research involves reconsidering all aspects of the computing stack including applications, programming frameworks, compiler optimizations, runtime systems, instruction set design, microarchitecture design, VLSI implementation, and hardware design methodologies



Key Metrics: Cycle Count, Cycle Time, Area, Energy

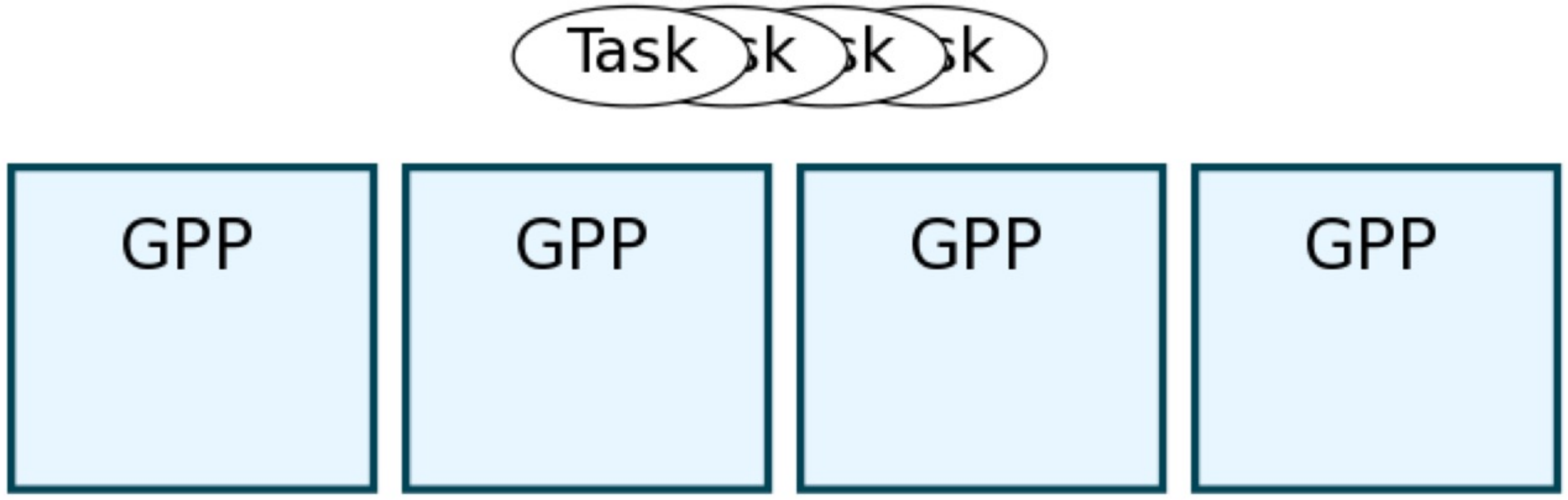
Experimenting with full-chip layout, FPGA prototypes, and test chips is a key part of our research methodology

Projects Within the Batten Research Group

	GPGPU Architecture [ISCA'13] [MICRO'14a] (AFOSR)	Integrated Voltage Regulation [MICRO'14b] [TCAS'18]	Accelerator Centric SoC Design [HOTCHIPS'17] [DAC'16] [FPGA'17] (DARPA,NSF)	Deep Processing in Memory (NSF,SRC)	Accel for Loop-Level Parallelism [MICRO'14c] (DARPA,AFOSR, NSF)	Accel for Task-Level Parallelism [ISCA'16] [MICRO'17] (AFOSR,NSF)	PyMTL/Pydgin Frameworks [MICRO'14d] [ISPASS'15,'16] [DAC'18] (NSF,DARPA)
Apps	●	●	●	●	●	●	
Algos	•		●	●		●	
PL			•			•	•
Compiler	•		•		●	•	
ISA	●	•	•	●	●	●	●
uArch	●	●	●	●	●	●	●
RTL	●	●	●	●	●	●	●
VLSI	●	●	●	●	●	●	●
Circuits		●		●			
Tech				●			

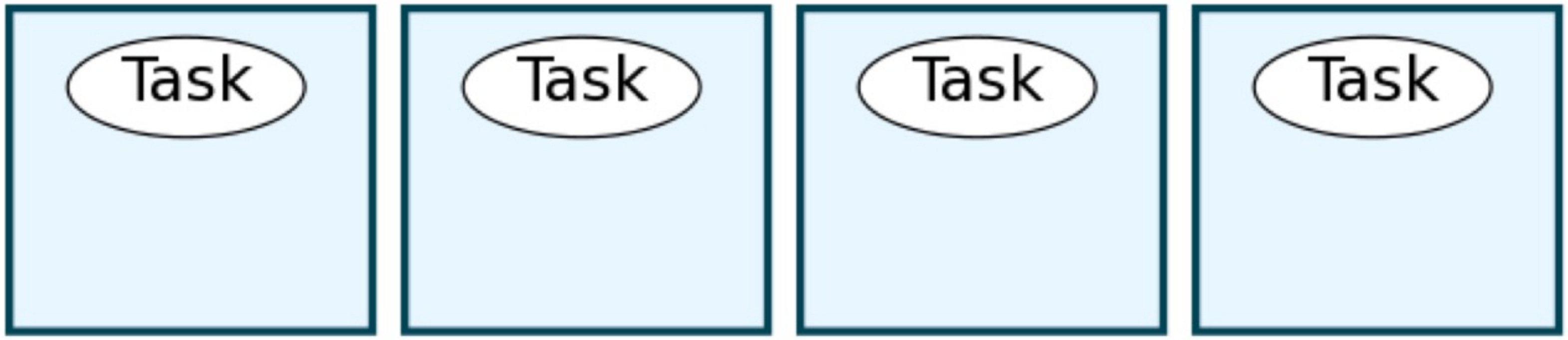
Inter-Core

- Task-Based Parallel Programming Frameworks
  - Intel TBB, Cilk



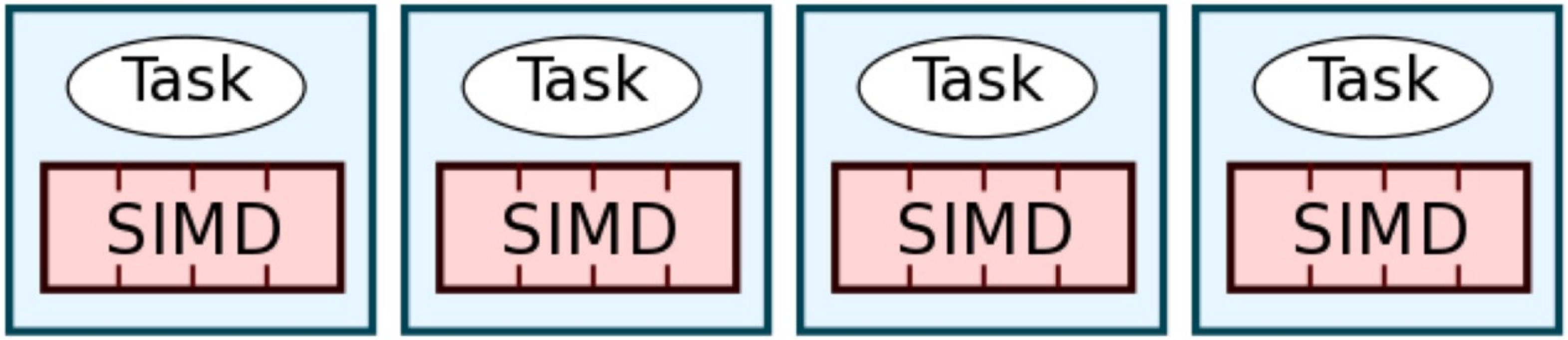
Inter-Core

- Task-Based Parallel Programming Frameworks
    - Intel TBB, Cilk



Inter-Core

- Task-Based Parallel Programming Frameworks
  - Intel TBB, Cilk



Intra-Core

- Packed-SIMD Vectorization
  - Intel AVX, Arm NEON

Challenges of Combining Tasks and Vectors

```
void app_kernel_tbb_avx(int N, float* src, float* dst) {
  // Pack data into padded aligned chunks
  //   src -> src_chunks[NUM_CHUNKS * SIMD_WIDTH]
  //   dst -> dst_chunks[NUM_CHUNKS * SIMD_WIDTH]
  ...

  // Use TBB across cores
  parallel_for (range(0, NUM_CHUNKS, TASK_SIZE), [&] (range r) {
    for (int i = r.begin(); i < r.end(); i++) {
      // Use packed-SIMD within a core
      #pragma simd vlen(SIMD_WIDTH)
      for (int j = 0; j < SIMD_WIDTH; j++) {
        if (src_chunks[i][j] > THRESHOLD)
          aligned_dst[i] = DoLightCompute aligned_src[i]);
        else
          aligned_dst[i] = DoHeavyCompute aligned_src[i]);
  ...
  ...
```

Challenge #1: Intra-Core Parallel Abstraction Gap

Challenges of Combining Tasks and Vectors

```
void app_kernel_tbb_avx(int N, float* src, float* dst) {
  // Pack data into padded aligned chunks
  //   src -> src_chunks[NUM_CHUNKS * SIMD_WIDTH]
  //   dst -> dst_chunks[NUM_CHUNKS * SIMD_WIDTH]
  ...

  // Use TBB across cores
  parallel_for (range(0, NUM_CHUNKS, TASK_SIZE), [&] (range r) {
    for (int i = r.begin(); i < r.end(); i++) {
      // Use packed-SIMD within a core
      #pragma simd vlen(SIMD_WIDTH)
      for (int j = 0; j < SIMD_WIDTH; j++) {
        if (src_chunks[i][j] > THRESHOLD)
          aligned_dst[i] = DoLightCompute(aligned_src[i]);
        else
          aligned_dst[i] = DoHeavyCompute(aligned_src[i]);
  ...
  ...
```

Challenge #1: Intra-Core Parallel Abstraction Gap

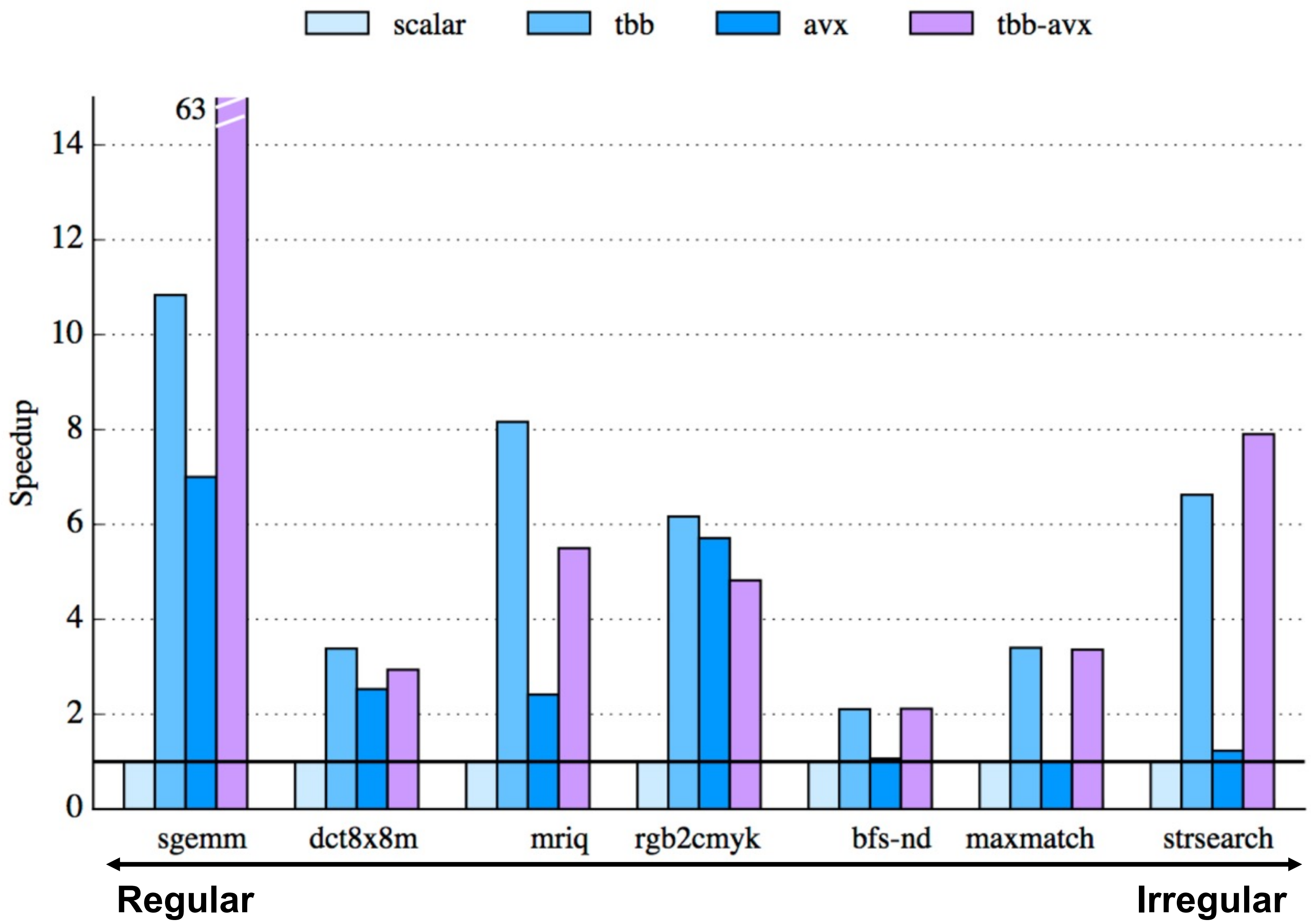
Challenges of Combining Tasks and Vectors

```
void app_kernel_tbb_avx(int N, float* src, float* dst) {
  // Pack data into padded aligned chunks
  //   src -> src_chunks[NUM_CHUNKS * SIMD_WIDTH]
  //   dst -> dst_chunks[NUM_CHUNKS * SIMD_WIDTH]
  ...

  // Use TBB across cores
  parallel_for (range(0, NUM_CHUNKS, TASK_SIZE), [&] (range r) {
    for (int i = r.begin(); i < r.end(); i++) {
      // Use packed-SIMD within a core
      #pragma simd vlen(SIMD_WIDTH)
      for (int j = 0; j < SIMD_WIDTH; j++) {
        if (src_chunks[i][j] > THRESHOLD)
          aligned_dst[i] = DoLightCompute(aligned_src[i]);
        else
          aligned_dst[i] = DoHeavyCompute(aligned_src[i]);
  ...
  ...
```

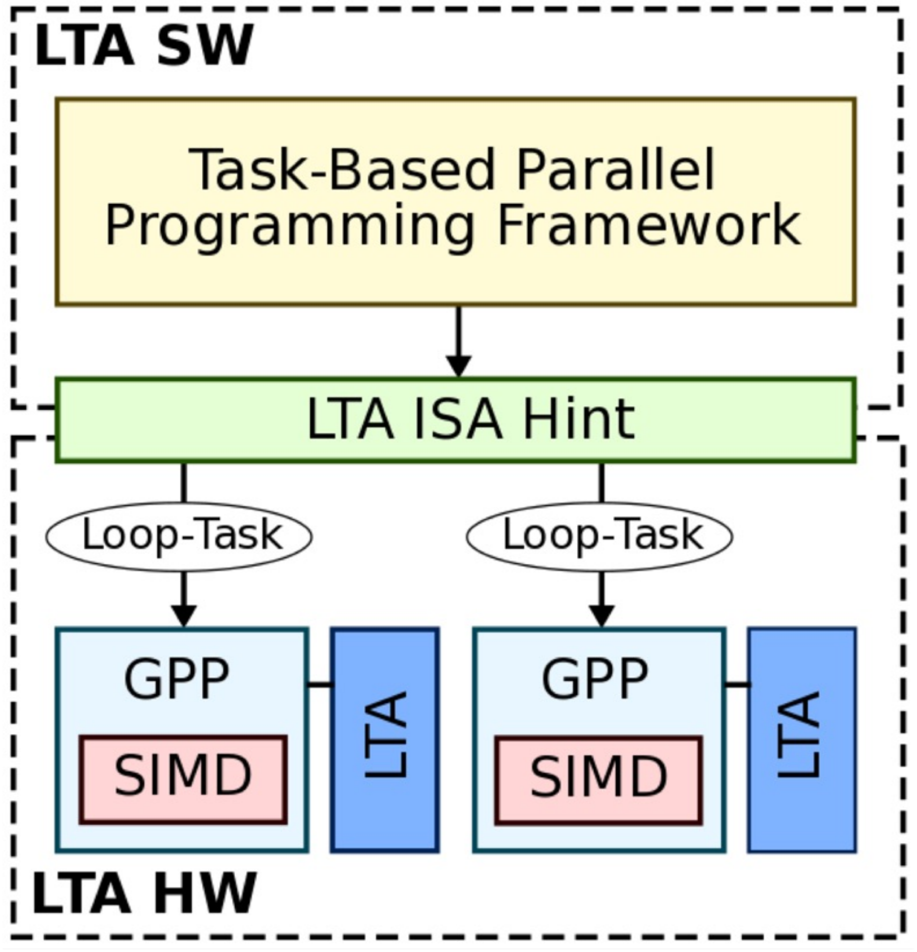
Challenge #1: Intra-Core Parallel Abstraction Gap

Challenge #2: Inefficient Execution of Irregular Tasks

Native Performance Results



Loop-Task Accelerator (LTA) Vision

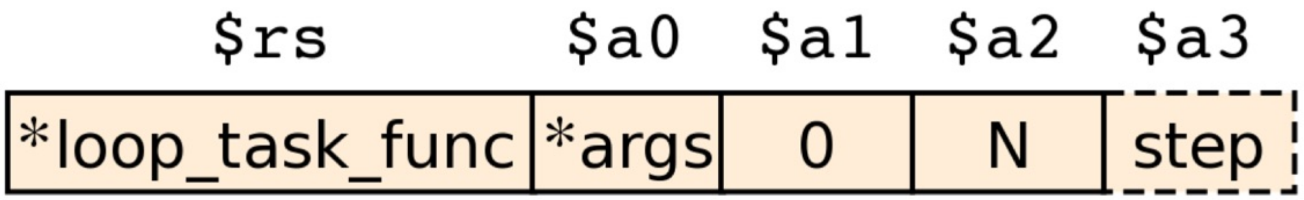


- Motivation
- **Challenge #1: LTA SW**
- Challenge #2: LTA HW
- Evaluation

LTA SW: API and ISA Hint

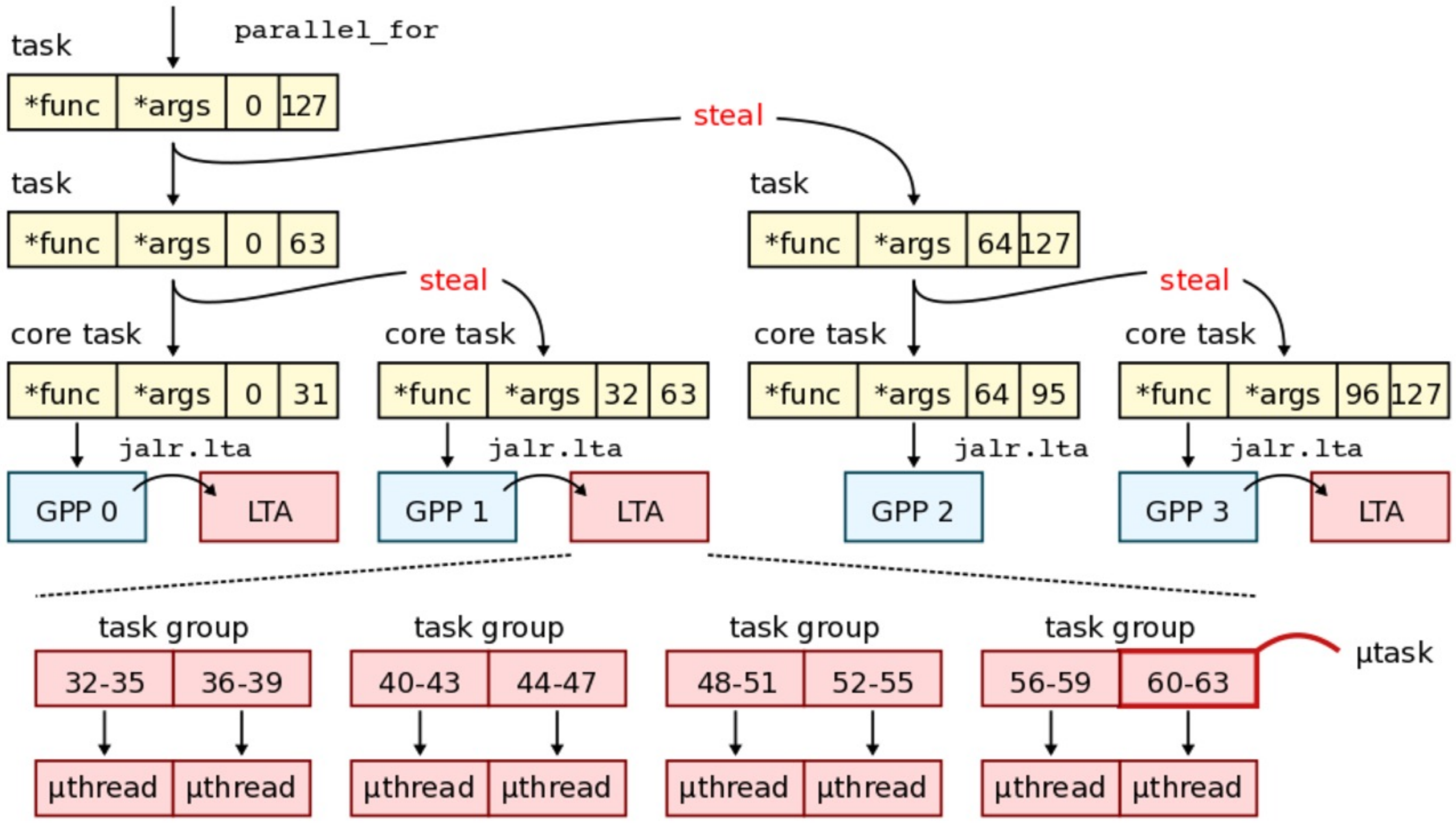
```
void app_kernel_lta(int N, float* src, float* dst) {
  LTA_PARALLEL_FOR(0, N, (dst, src), ({
    if (src[i] > THRESHOLD)
      dst[i] = DoComputeLight(src[i]);
    else
      dst[i] = DoComputeHeavy(src[i]);
  }));
}

void loop_task_func(void* a, int start, int end, int step=1);
```

```
jalr.lta $rd, $rs
```

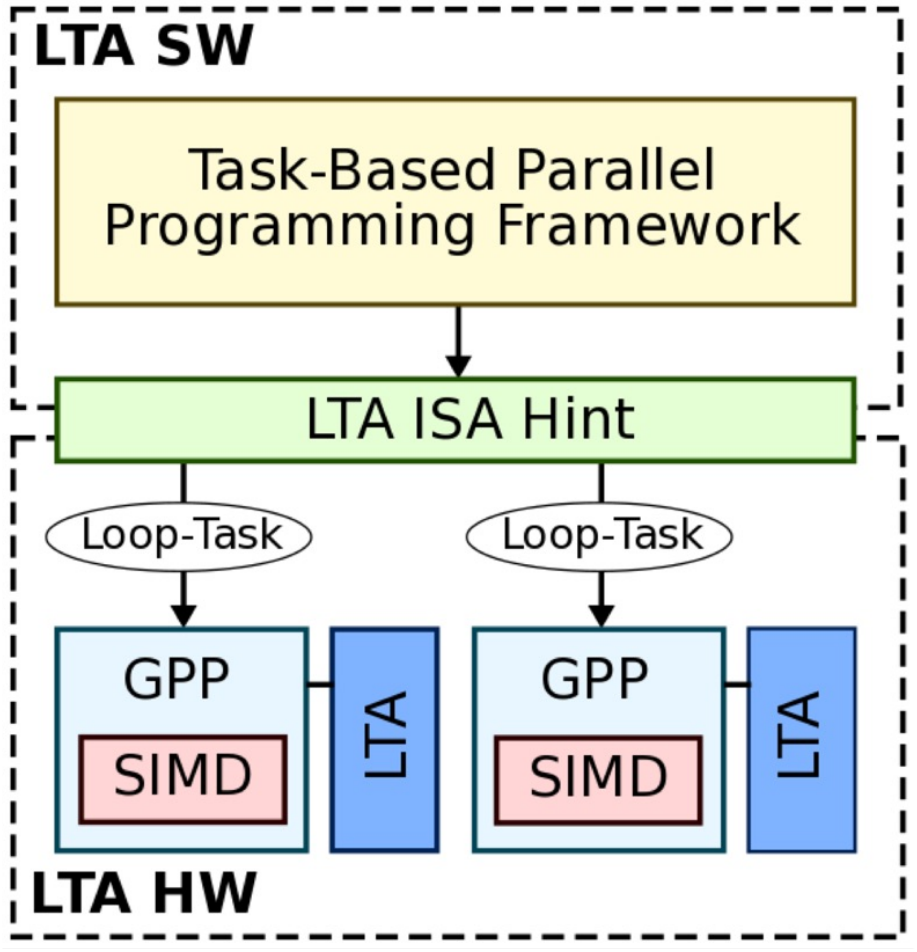


Hint that hardware can potentially accelerate task execution

LTA SW: Task-Based Runtime

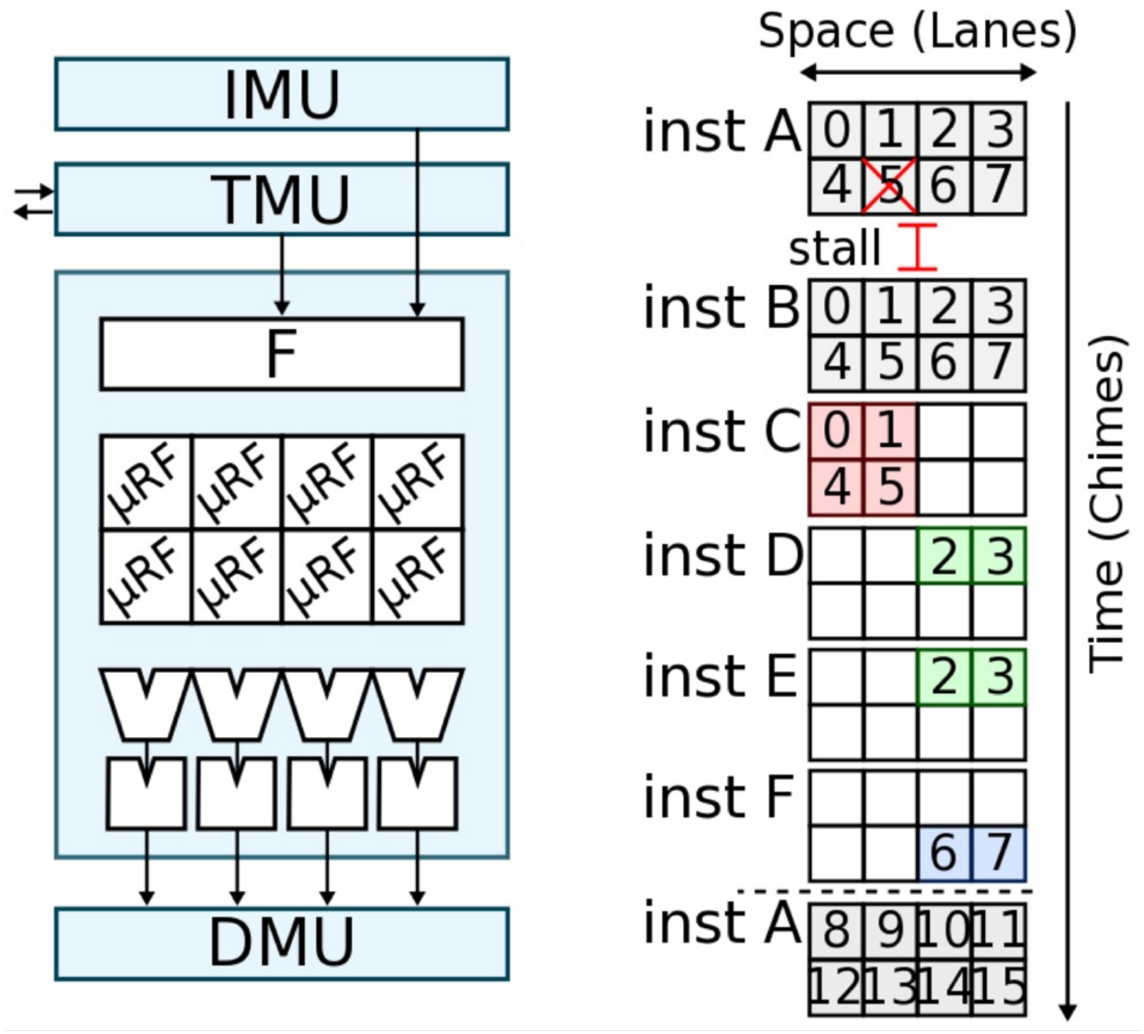


Loop-Task Accelerator (LTA) Vision



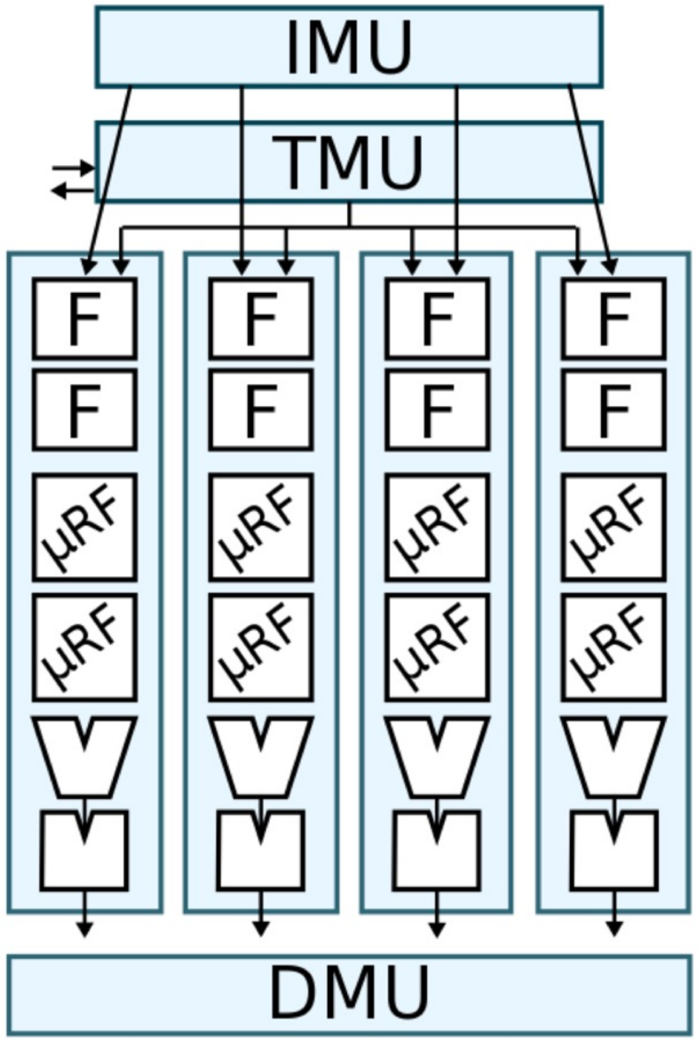
- Motivation
- Challenge #1: LTA SW
- **Challenge #2: LTA HW**
- Evaluation

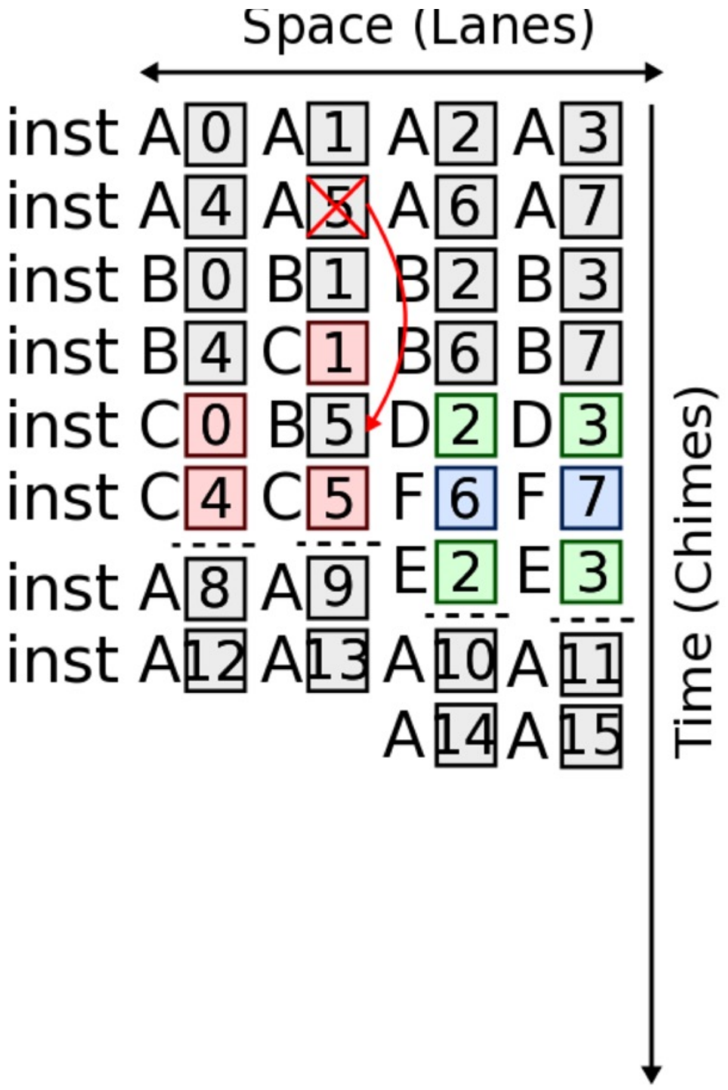
LTA HW: Fully Coupled LTA



Coupling better for regular workloads (amortize frontend/memory)

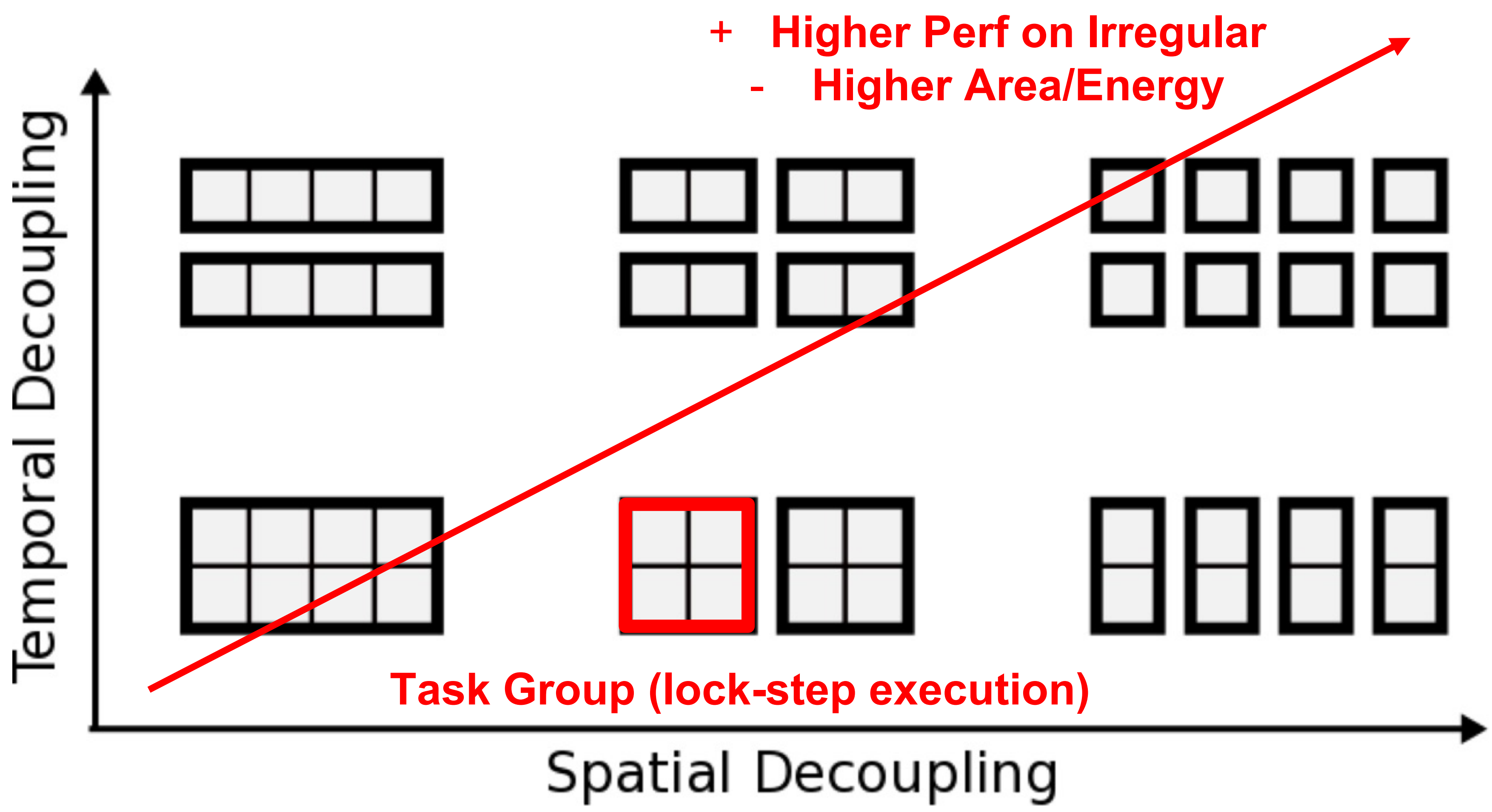
LTA HW: Fully Decoupled LTA





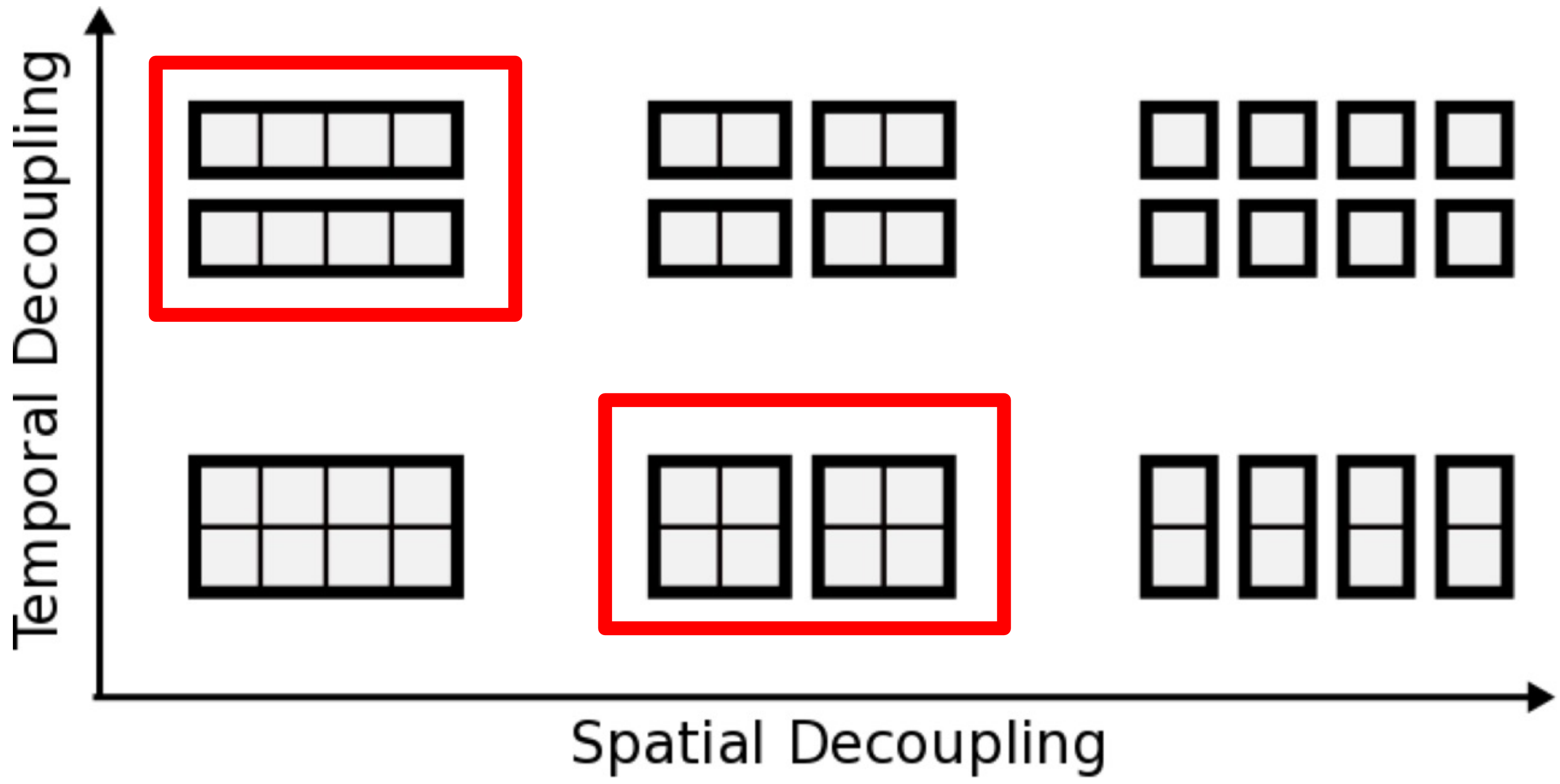
Decoupling better for irregular workloads (hide latencies)

LTA HW: Task-Coupling Taxonomy

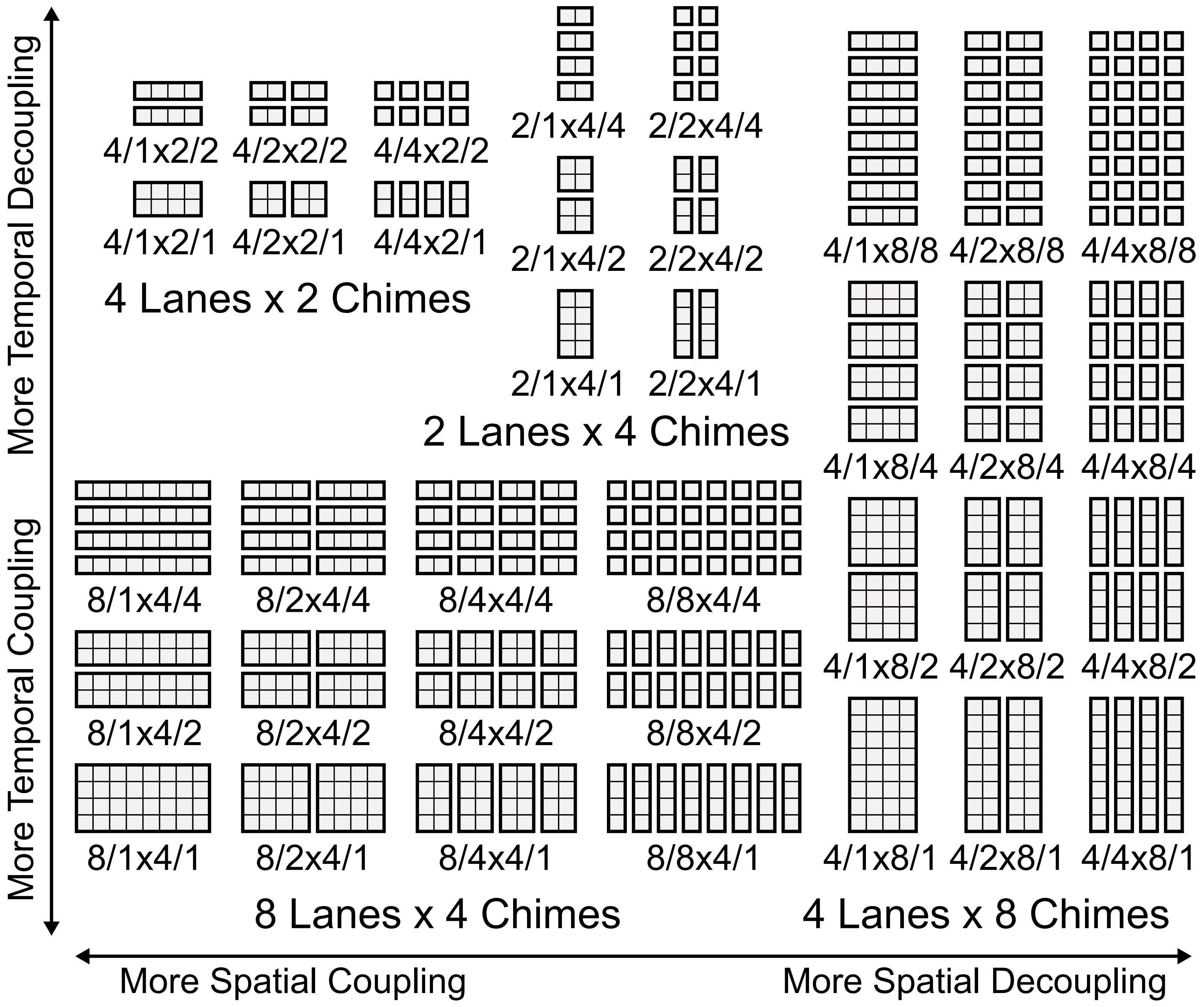


More decoupling (more task groups) in either space or time improves performance on irregular workloads at the cost of area/energy

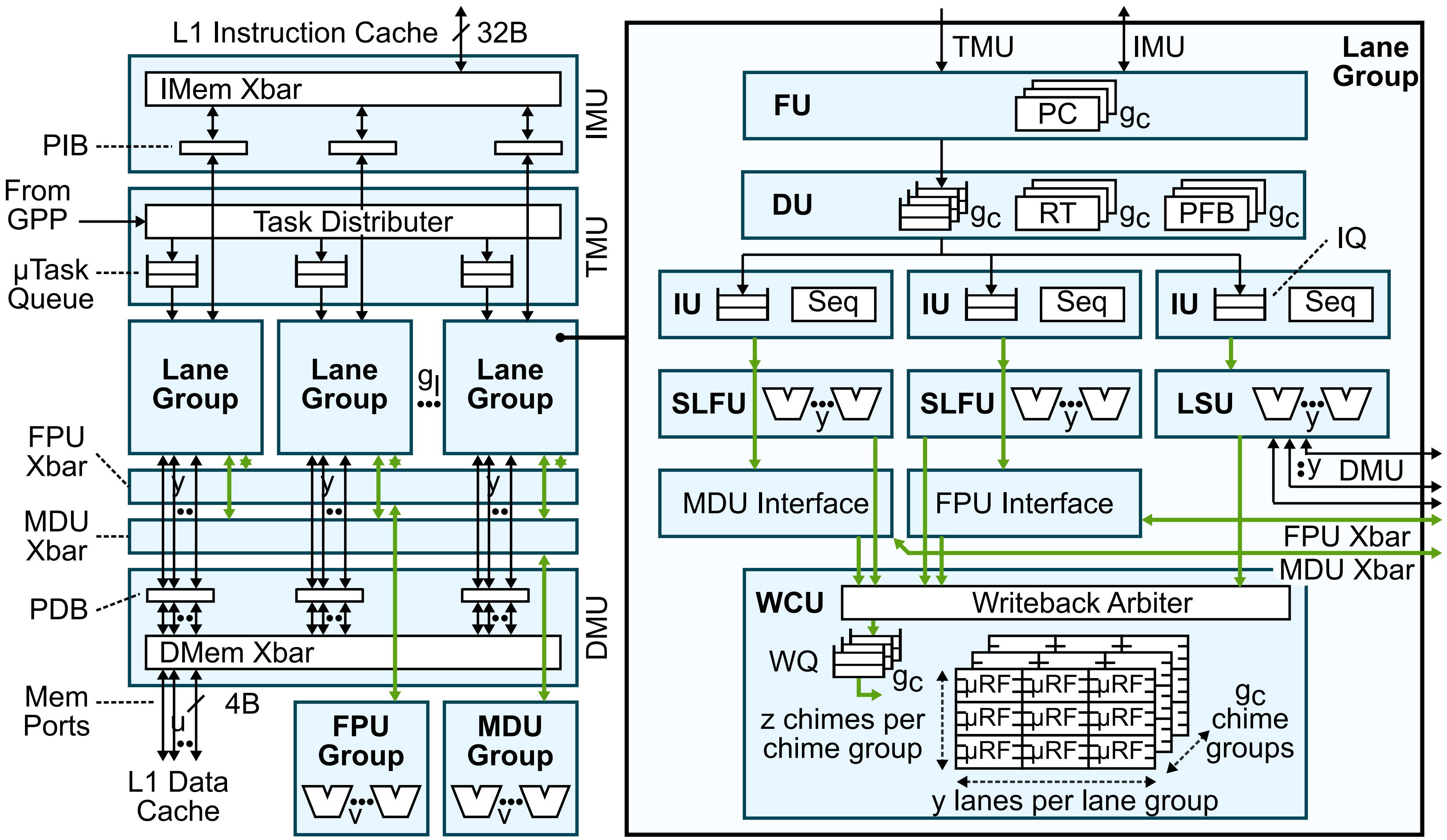
LTA HW: Task-Coupling Taxonomy



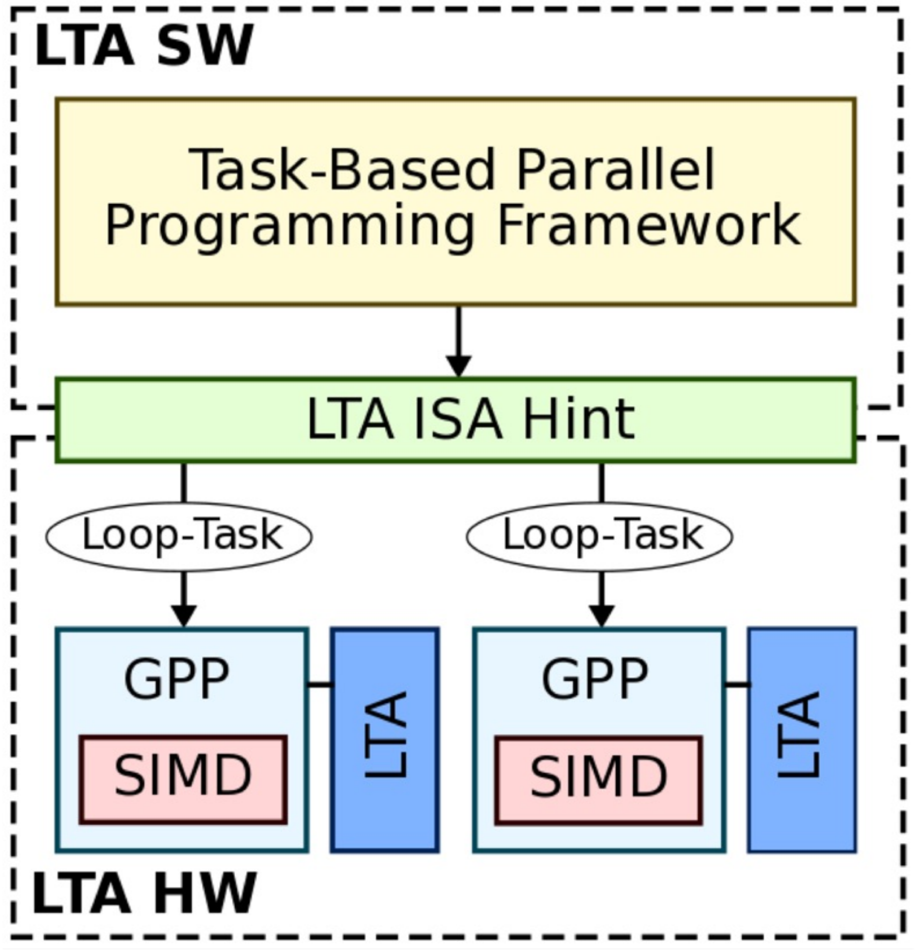
Does it matter whether we decouple in space or in time?


4/1x2/2
4/2x2/2
4/4x2/2
4/1x2/1
4/2x2/1
4/4x2/1
4 Lanes x 2 Chimes
2/1x4/4
2/2x4/4
2/1x4/2
2/2x4/2
2/1x4/1
2/2x4/1
2 Lanes x 4 Chimes
4/1x8/8
4/2x8/8
4/4x8/8
4/1x8/4
4/2x8/4
4/4x8/4
4/1x8/2
4/2x8/2
4/4x8/2
4/1x8/1
4/2x8/1
4/4x8/1
4 Lanes x 8 Chimes
8/1x4/4
8/2x4/4
8/4x4/4
8/8x4/4
8/1x4/2
8/2x4/2
8/4x4/2
8/8x4/2
8/1x4/1
8/2x4/1
8/4x4/1
8/8x4/1
8 Lanes x 4 Chimes
More Temporal Decoupling
More Temporal Coupling
More Spatial Coupling
More Spatial Decoupling

LTA HW: Microarchitectural Template



Loop-Task Accelerator (LTA) Vision

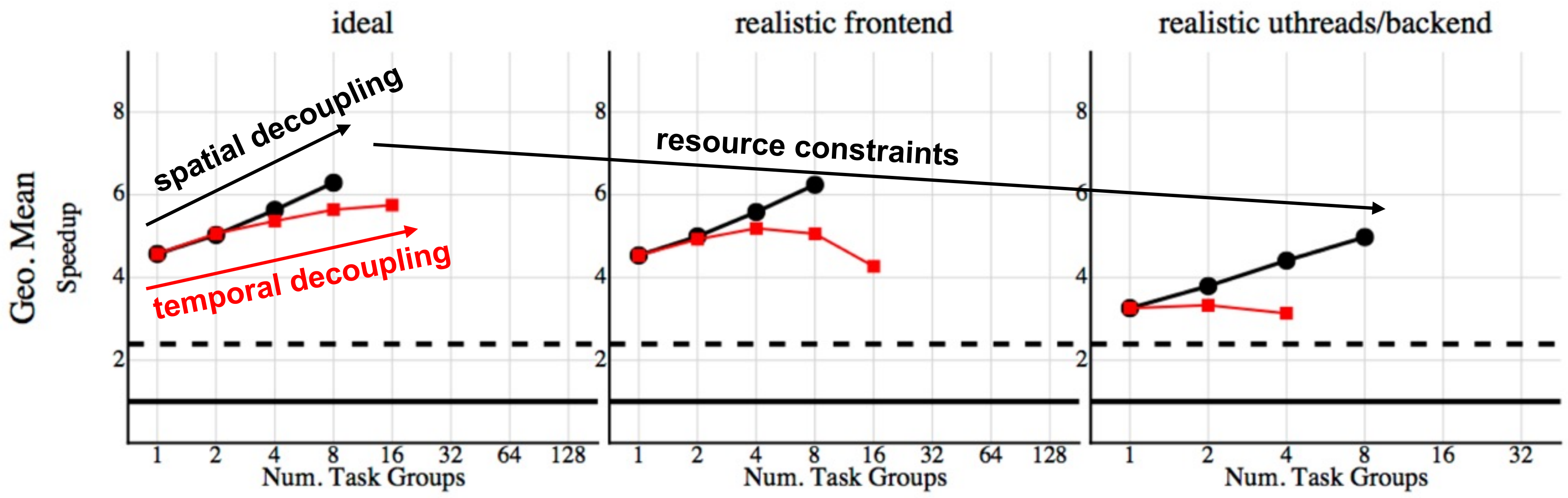


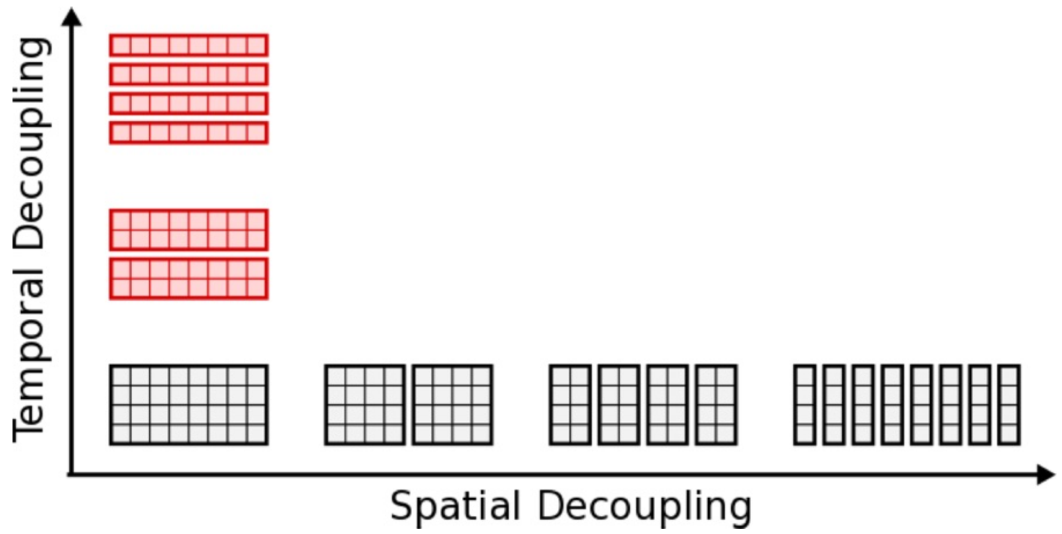
- Motivation
- Challenge #1: LTA SW
- Challenge #2: LTA HW
- **Evaluation**

Evaluation: Methodology

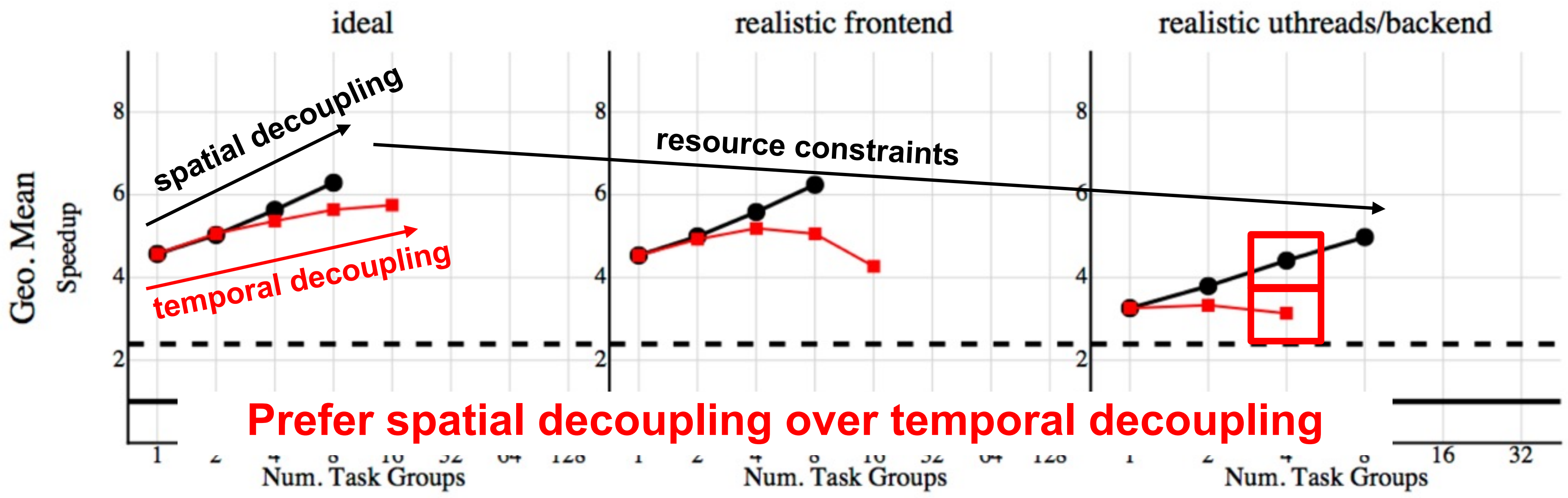
- Ported 16 application kernels from PBBS and in-house benchmark suites with diverse loop-task parallelism
  - Scientific computing: N-body simulation, MRI-Q, SGEMM
  - Image processing: bilateral filter, RGB-to-CMYK, DCT
  - Graph algorithms: breadth-first search, maximal matching
  - Search/Sort algorithms: radix sort, substring matching
- gem5 + PyMTL co-simulation for cycle-level performance
- Component/event-based area/energy modeling
  - Uses area/energy dictionary backed by VLSI results and McPAT

Evaluation: Design-Space Exploration

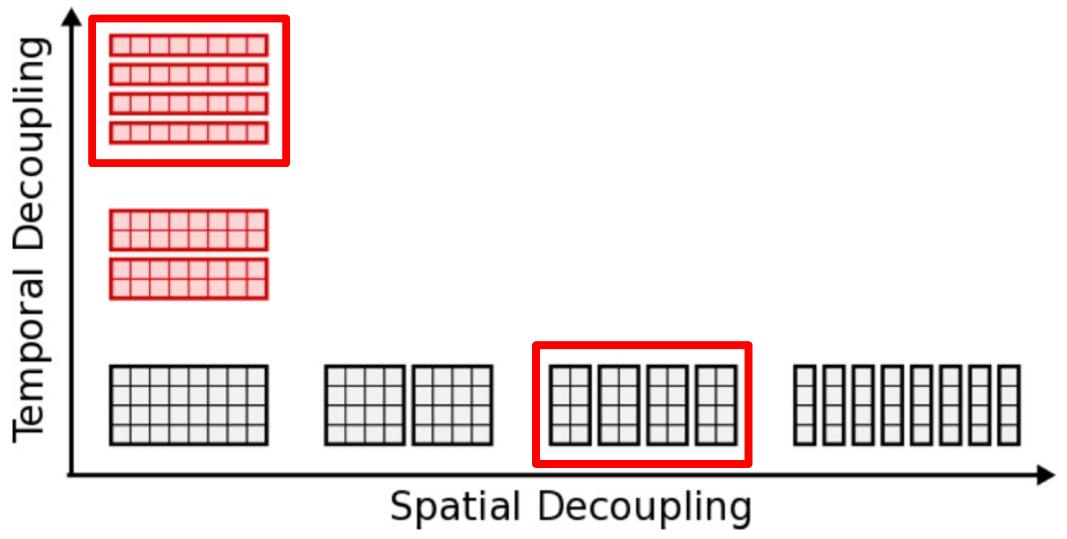




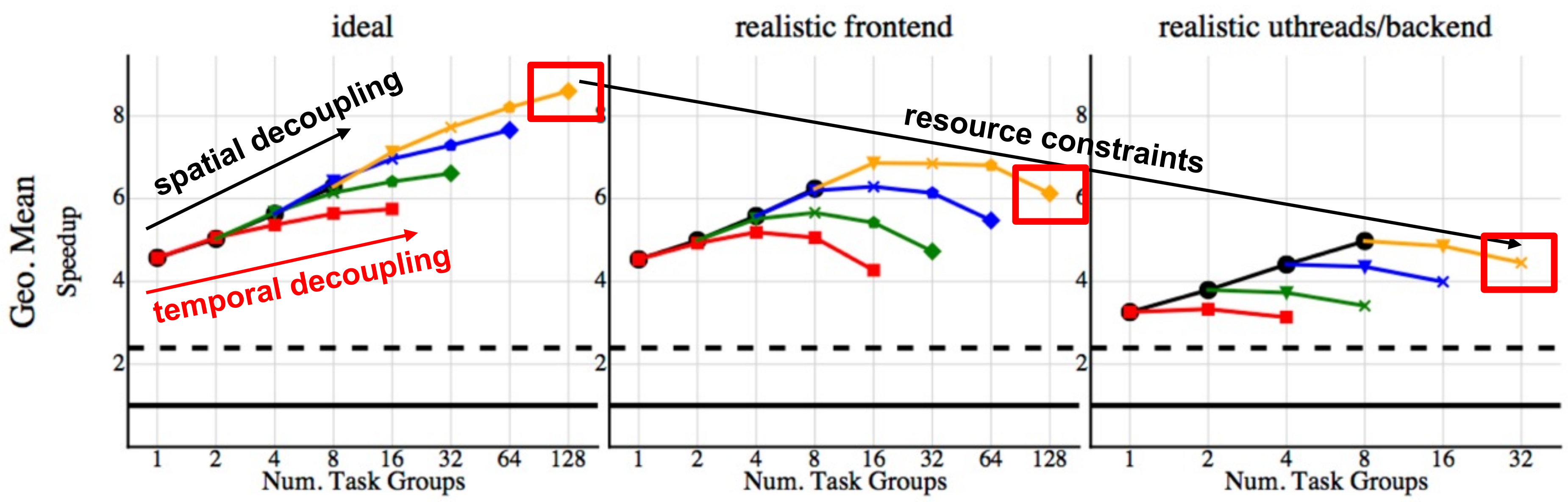
Evaluation: Design-Space Exploration

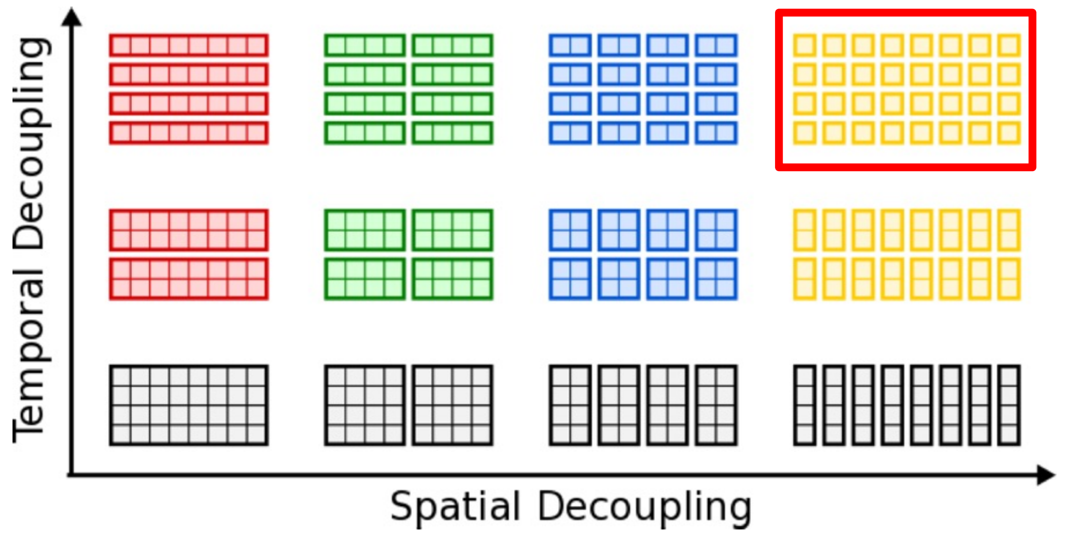


Prefer spatial decoupling over temporal decoupling

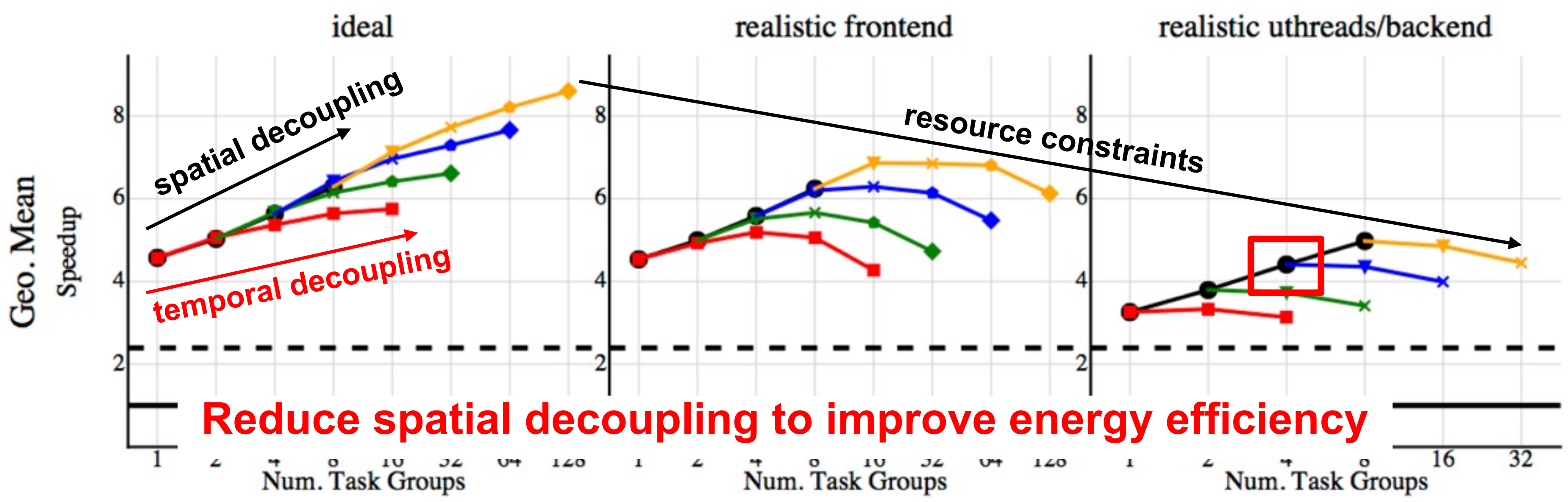


Evaluation: Design-Space Exploration

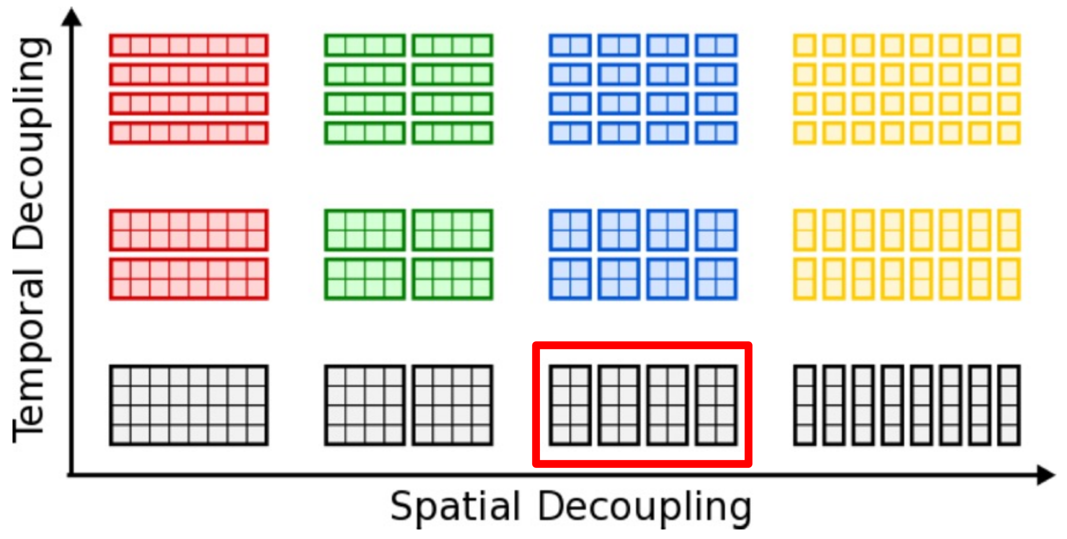




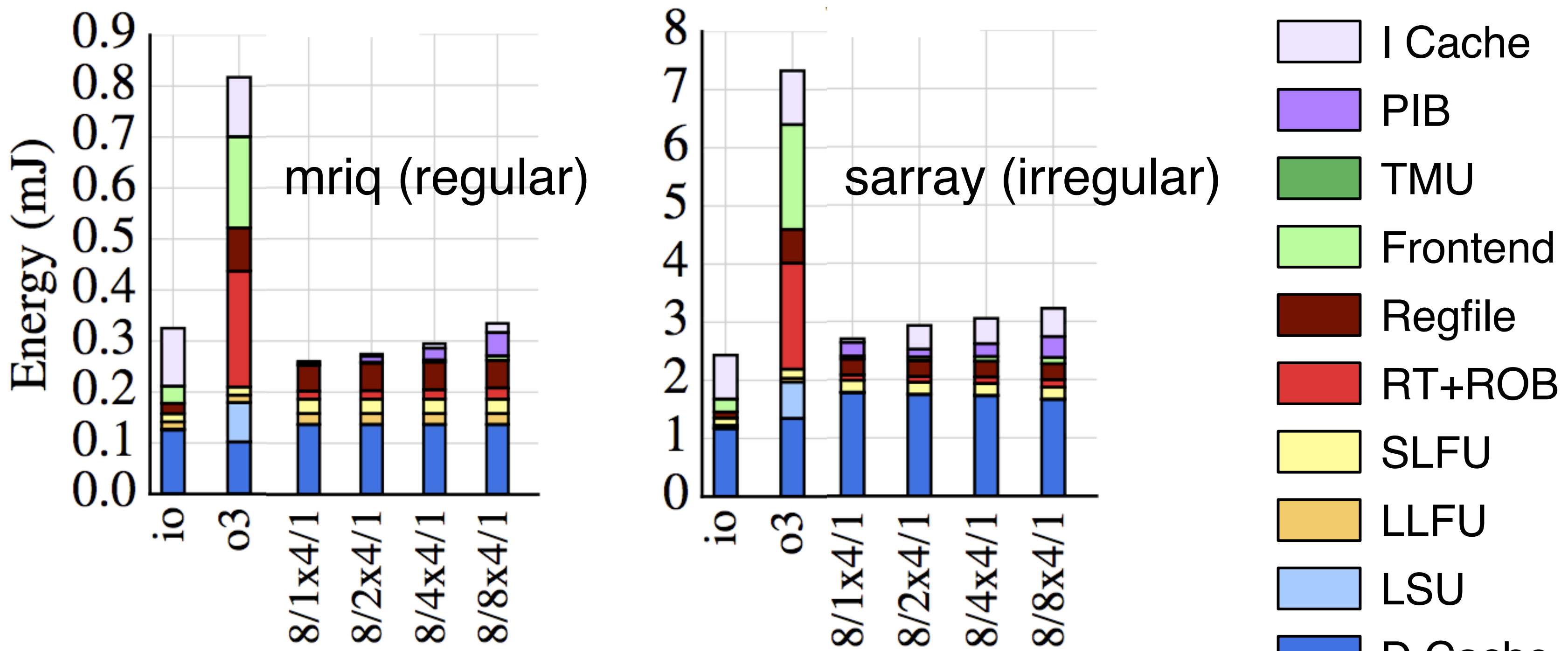
Evaluation: Design-Space Exploration

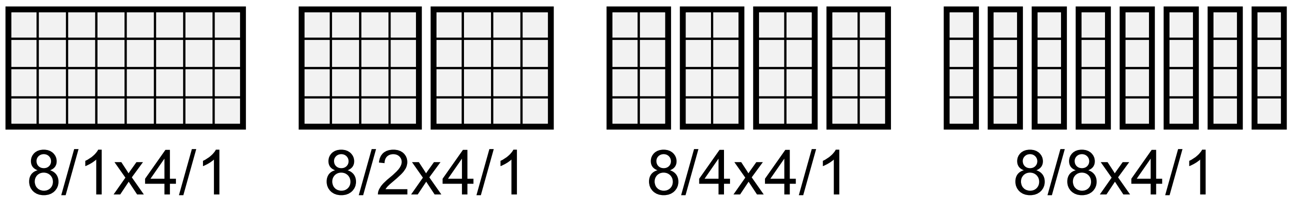


Reduce spatial decoupling to improve energy efficiency



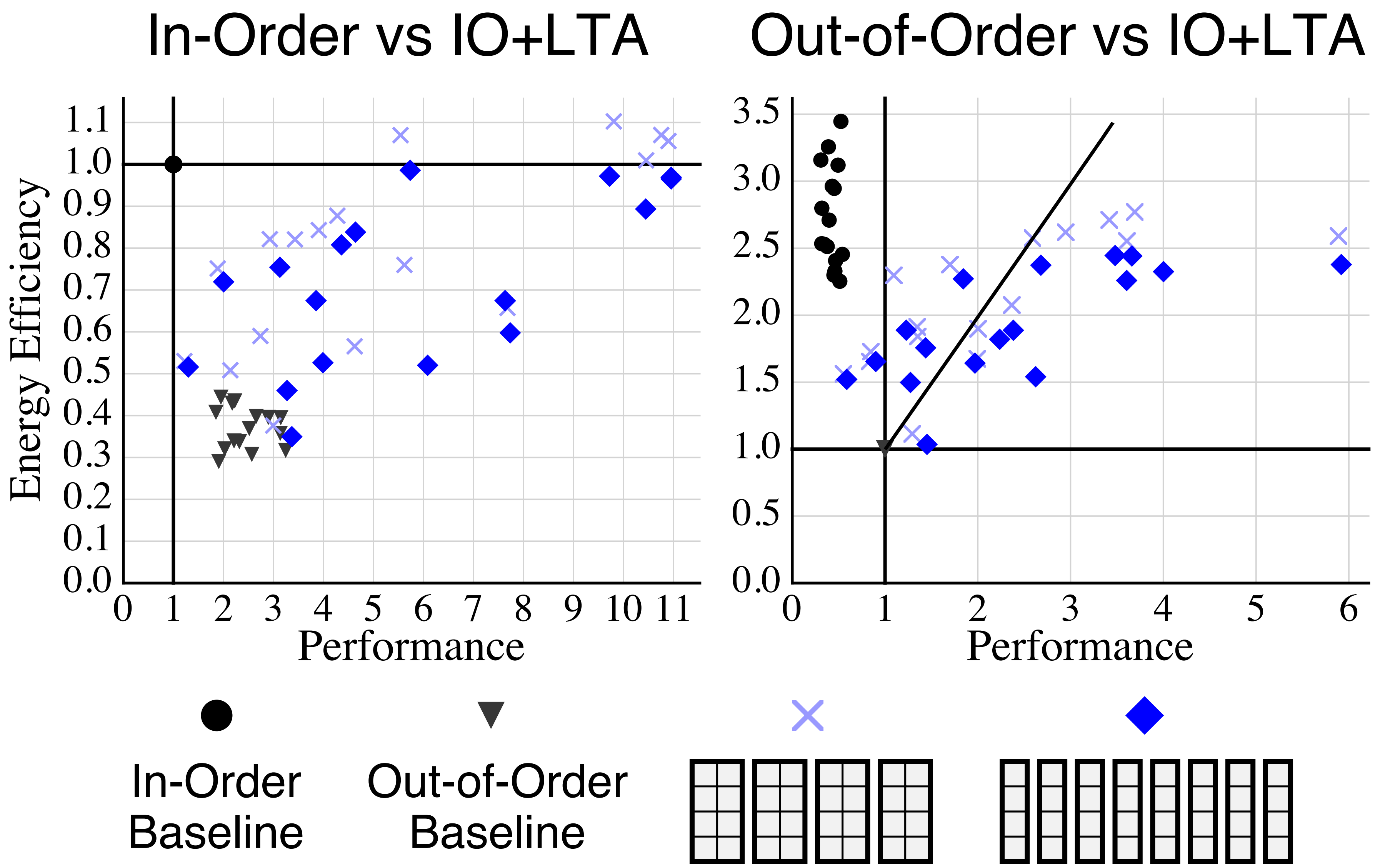
Evaluation: Energy Breakdown



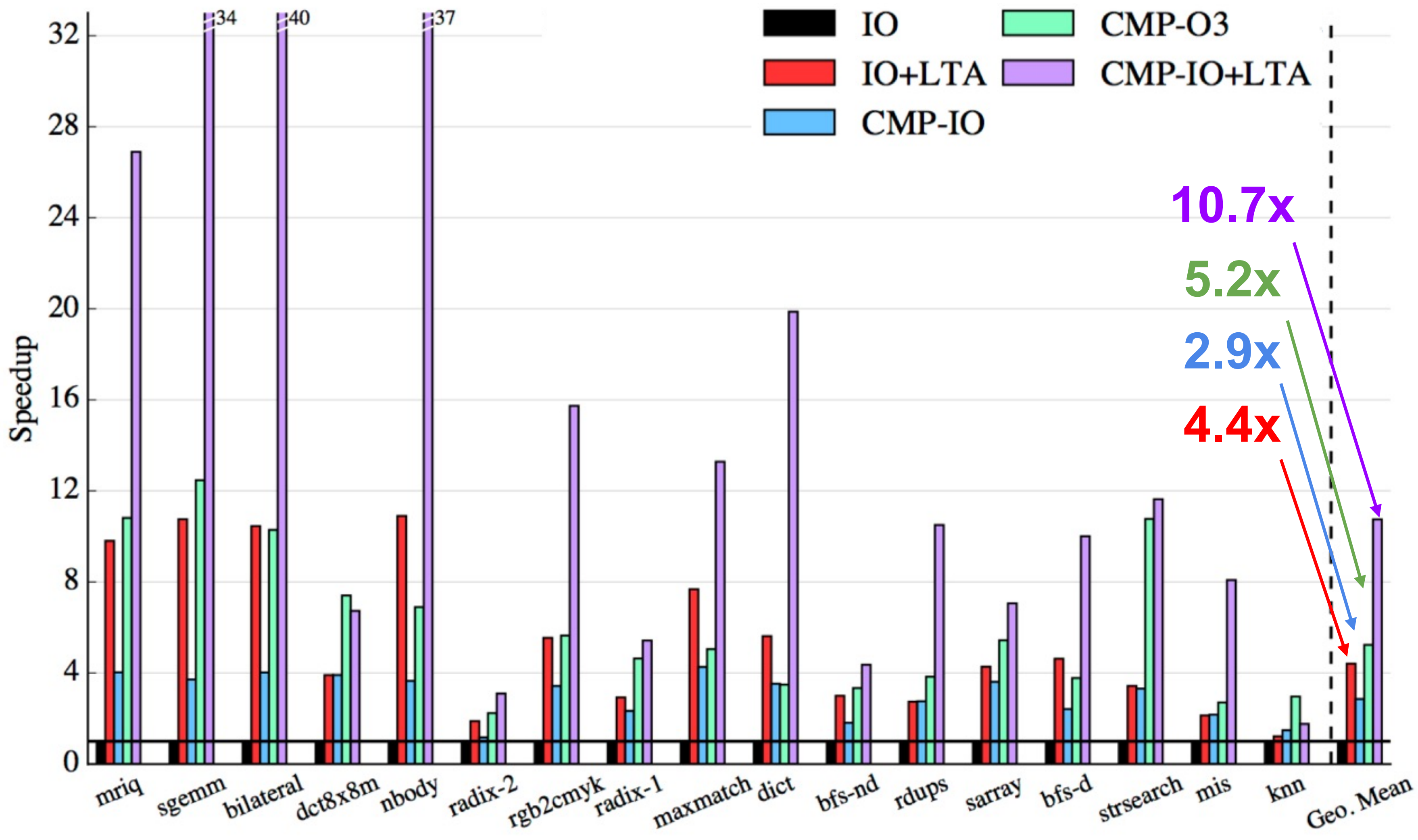


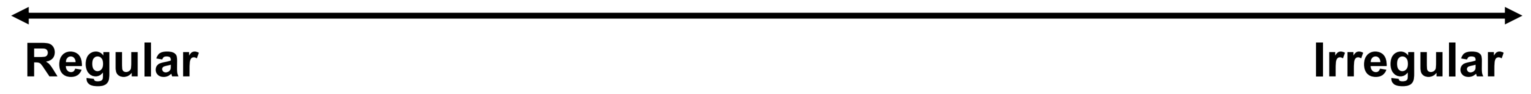
Conservative comparison since IO/O3 running *serial* baseline, while LTA is using *parallel runtime* even on a single core

Evaluation: Energy Efficiency vs. Performance



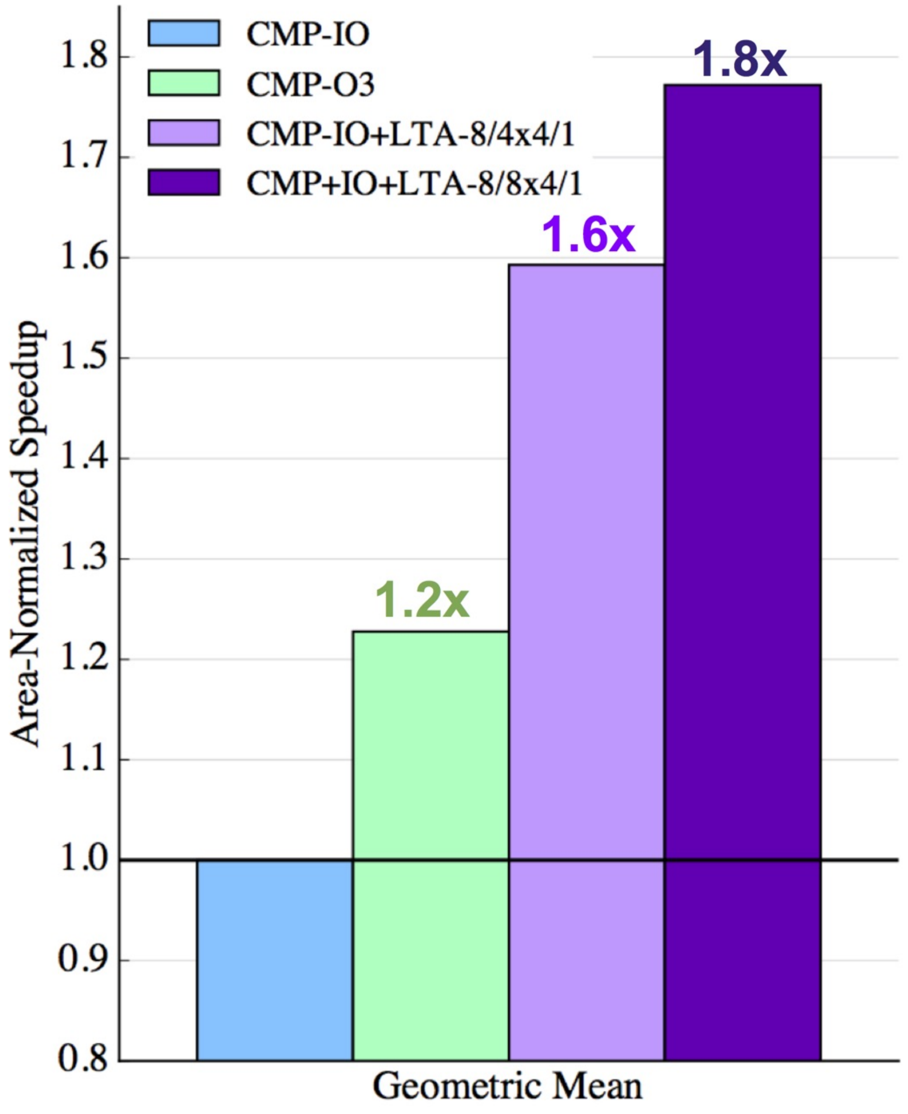
Evaluation: Multicore LTA Performance





Evaluation: Multicore Energy and Area

- Both the baseline CMP and the CMP+LTA designs use the same application code and almost the exact same parallel runtime
- CMP+LTA vs. CMP-IO: improves energy efficiency by 1.1× geo mean
- CMP+LTA vs. CMP-O3: improves energy efficiency by 3.2× geo mean

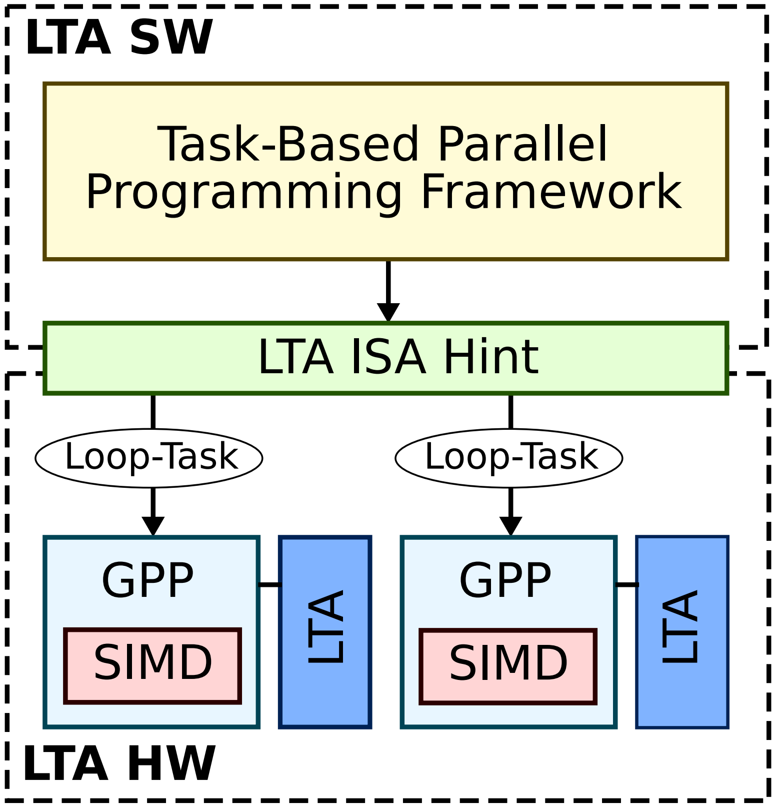


Related Work

- **Challenge #1: Intra-Core Parallel Abstraction Gap**
  - Persistent threads for GPGPUs (S. Tzeng et al.)
  - OpenCL, OpenMP, C++ AMP
  - Cilk for packed-SIMD (B. Ren et al.)
  - and more ...
- **Challenge #2: Inefficient Execution of Irregular Tasks**
  - Variable warp sizing (T. Rogers et al.)
  - Temporal SIMT (S. Keckler et al.)
  - Vector-lane threading (S. Rivoire et al.)
  - and more ...
- See MICRO'17 paper for detailed references ...

LTA Take-Away Points

- Intra-core parallel abstraction gap and inefficient execution of irregular tasks are fundamental challenges for CMPs
- LTAs address both challenges with a lightweight ISA hint and a flexible microarchitectural template



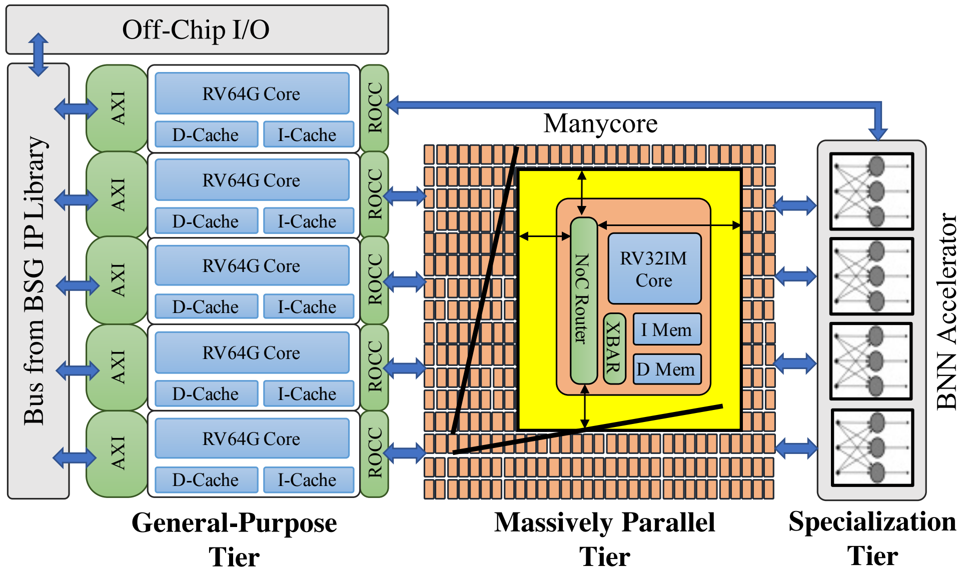
- Results suggest in a resource-constrained environment, architects should favor spatial decoupling over temporal decoupling

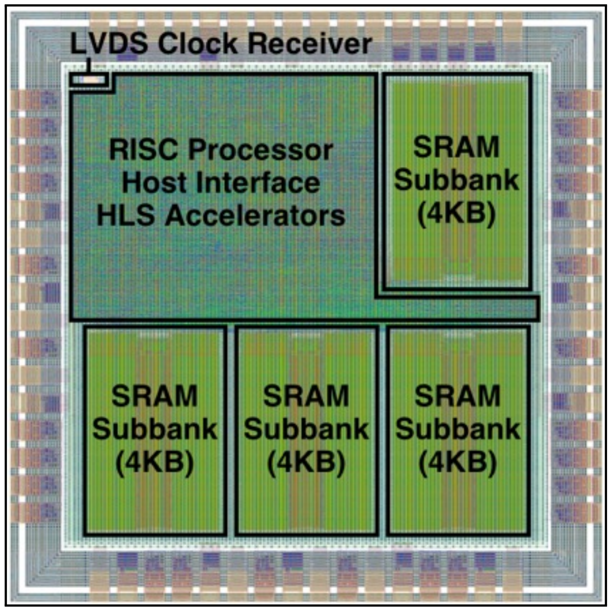
Upcoming Computer Laboratory Seminar

"A New Era of Open-Source SoC Design"
Wednesday, May 16th @ 4:15pm

Celerity: An Accelerator-Centric System-on-Chip

PyMTL: A Python-Based Hardware Modeling Framework



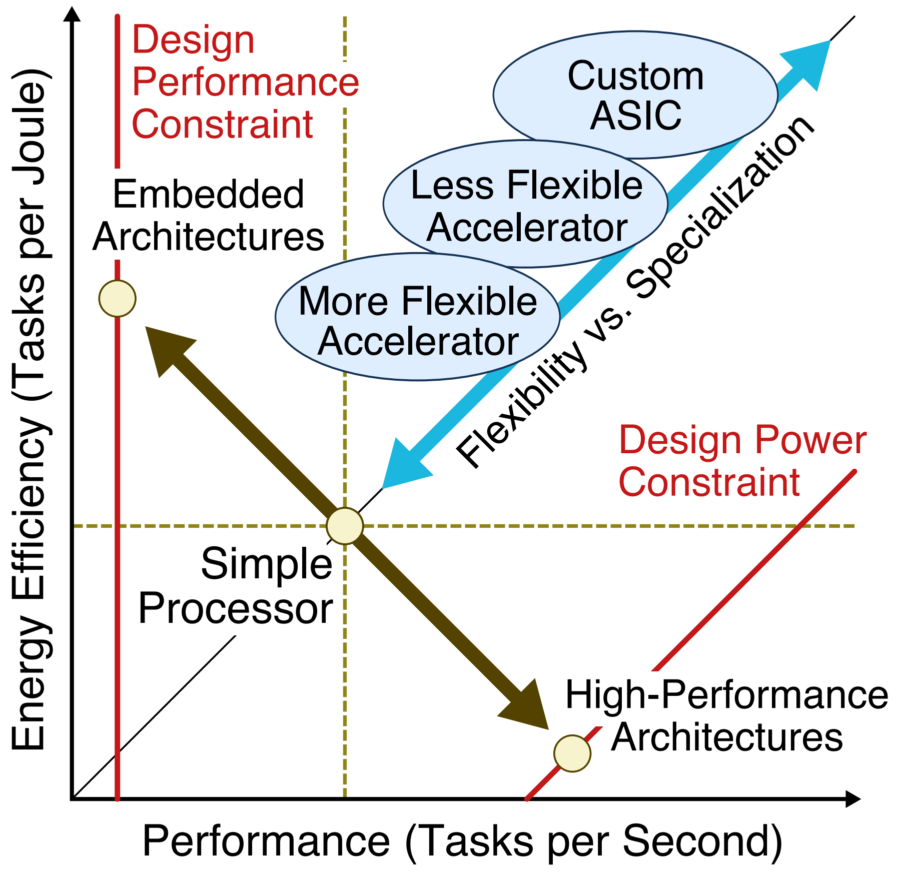


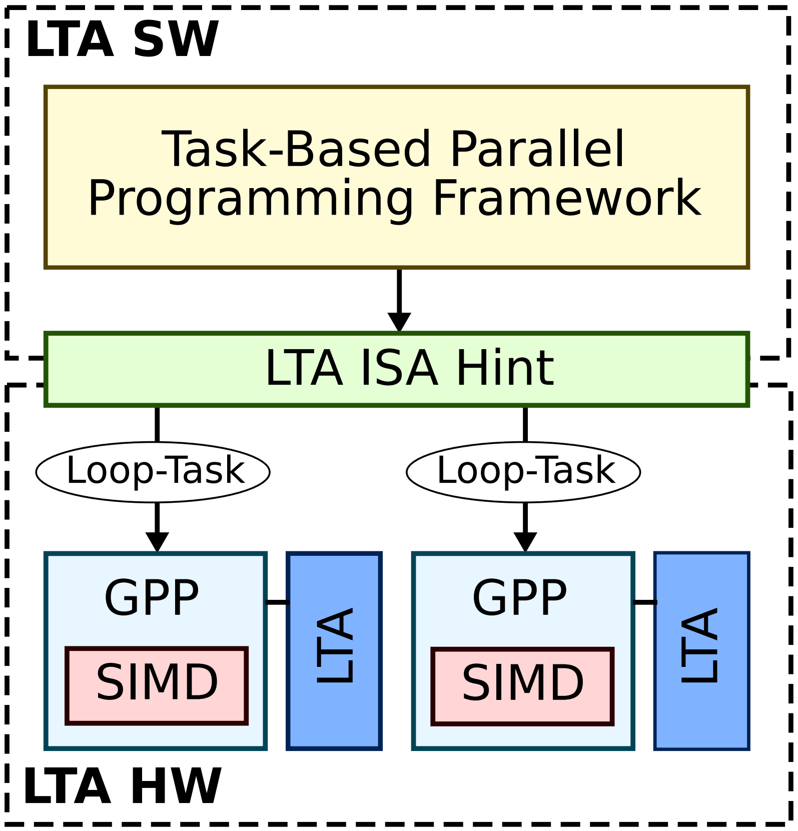
Ji Kim, Shreesha Srinath, Christopher Torng, Berkin Ilbeyi, Moyang Wang
Shunning Jiang, Khalid Al-Hawaj, Tuan Ta, Lin Cheng
and many M.S./B.S. students

Equipment, Tools, and IP
Intel, NVIDIA, Synopsys, Cadence, Xilinx, ARM

Batten Research Group

Exploring cross-layer hardware specialization
using a vertically integrated research methodology


Design Performance Constraint
Embedded Architectures
Custom ASIC
Less Flexible Accelerator
More Flexible Accelerator
Flexibility vs. Specialization
Design Power Constraint
Simple Processor
High-Performance Architectures
Energy Efficiency (Tasks per Joule)
Performance (Tasks per Second)


LTA SW
Task-Based Parallel Programming Framework
LTA ISA Hint
Loop-Task
Loop-Task
GPP
SIMD
LTA
GPP
SIMD
LTA
LTA HW