

Exploring Tradeoffs between Programmability and Efficiency in Data-Parallel Accelerators

Yunsup Lee¹, Rimas Avizienis¹, Alex Bishara¹,
Richard Xia¹, Derek Lockhart²,
Christopher Batten², Krste Asanovic¹

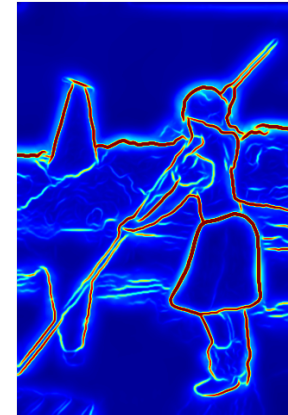
¹The Parallel Computing Lab, UC Berkeley

²Computer Systems Lab, Cornell University

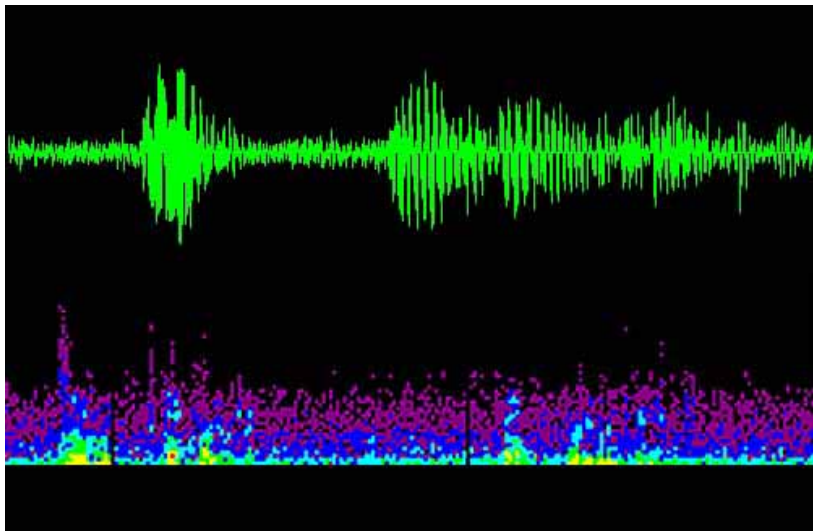
DLP Kernels Dominate Many Computational Workloads



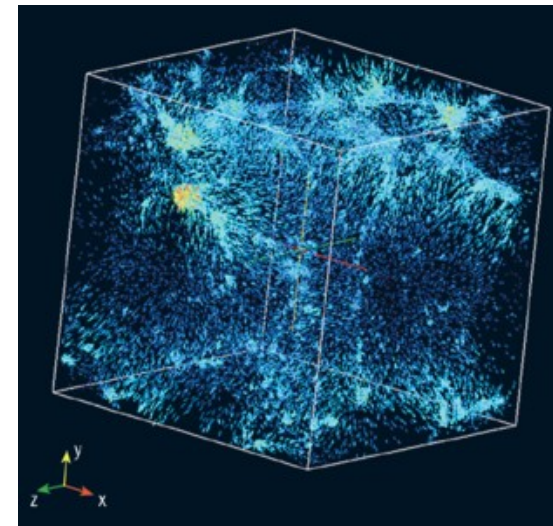
Graphics Rendering



Computer Vision



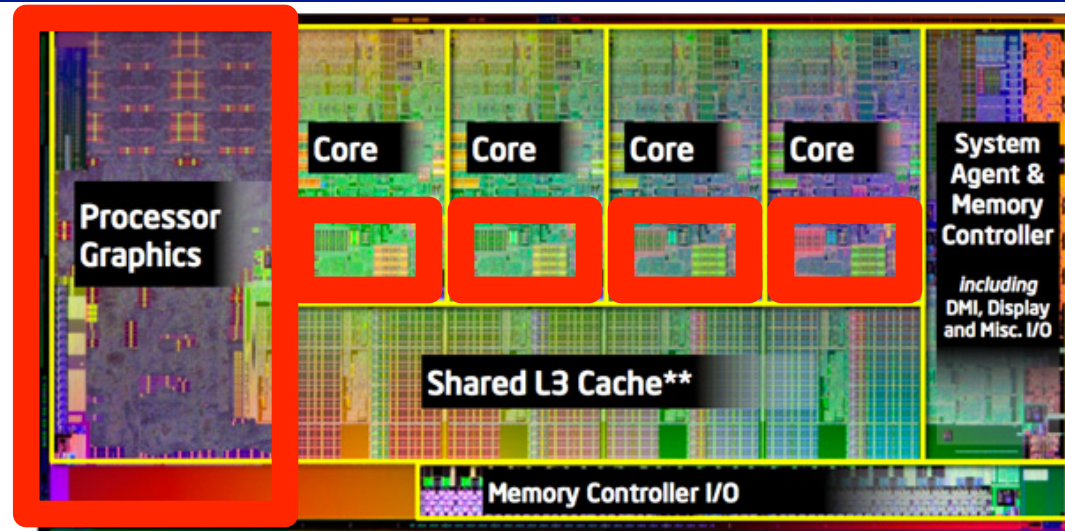
Audio Processing



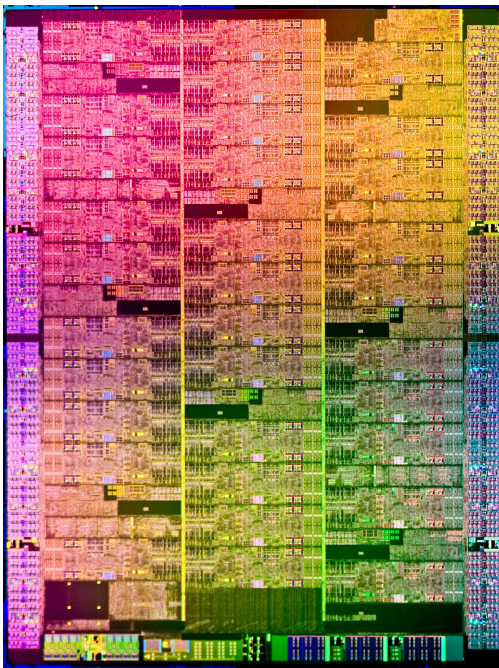
Physical Simulation

DLP Accelerators are Getting Popular

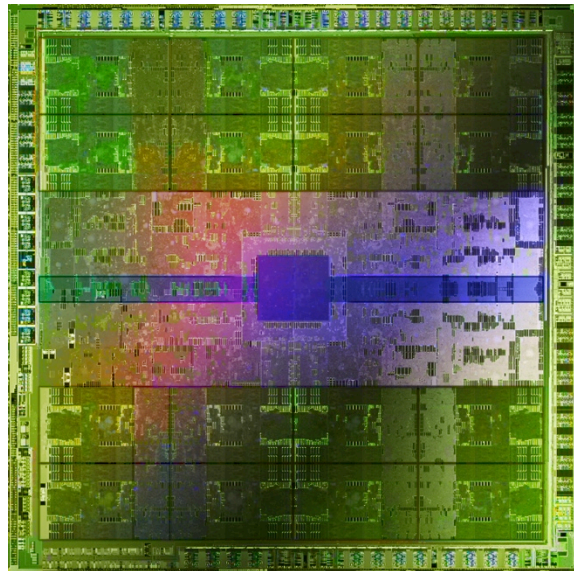
Sandy Bridge



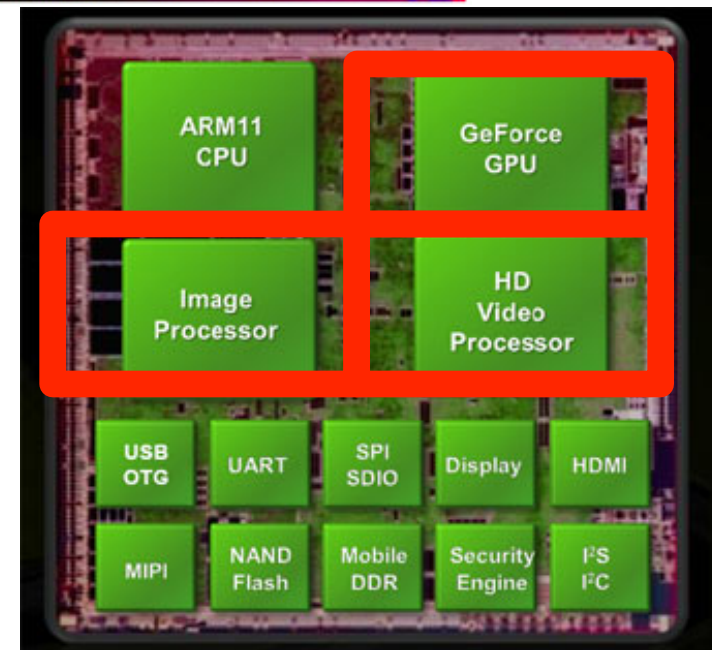
Knights Ferry



Fermi



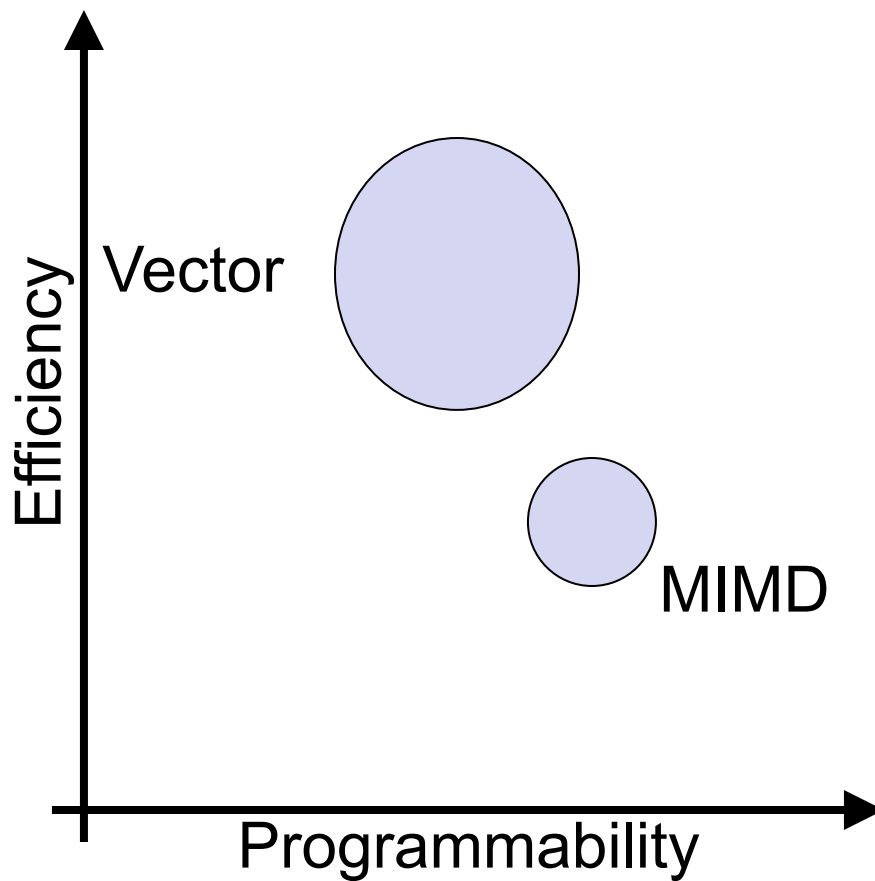
Tegra



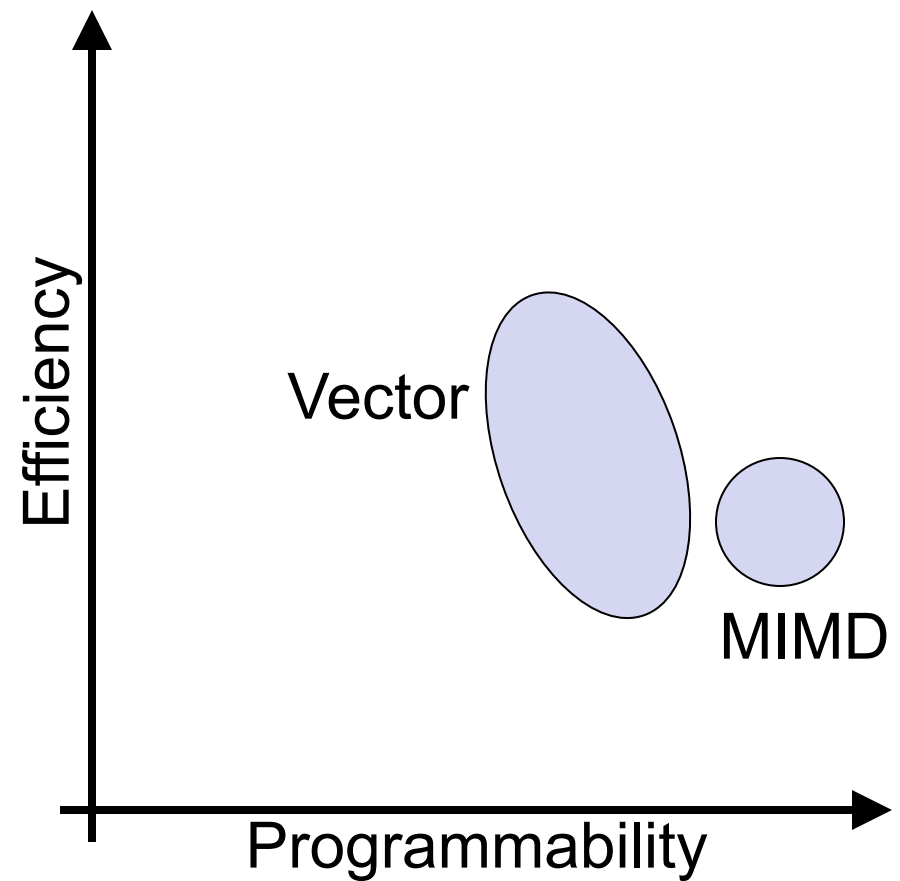
Important Metrics when Comparing DLP Accelerator Architectures

- **Performance per Unit Area**
- **Energy per Task**
- **Flexibility** (What can it run well?)
- **Programmability** (How hard is it to write code?)

Efficiency vs. Programmability: It's a tradeoff

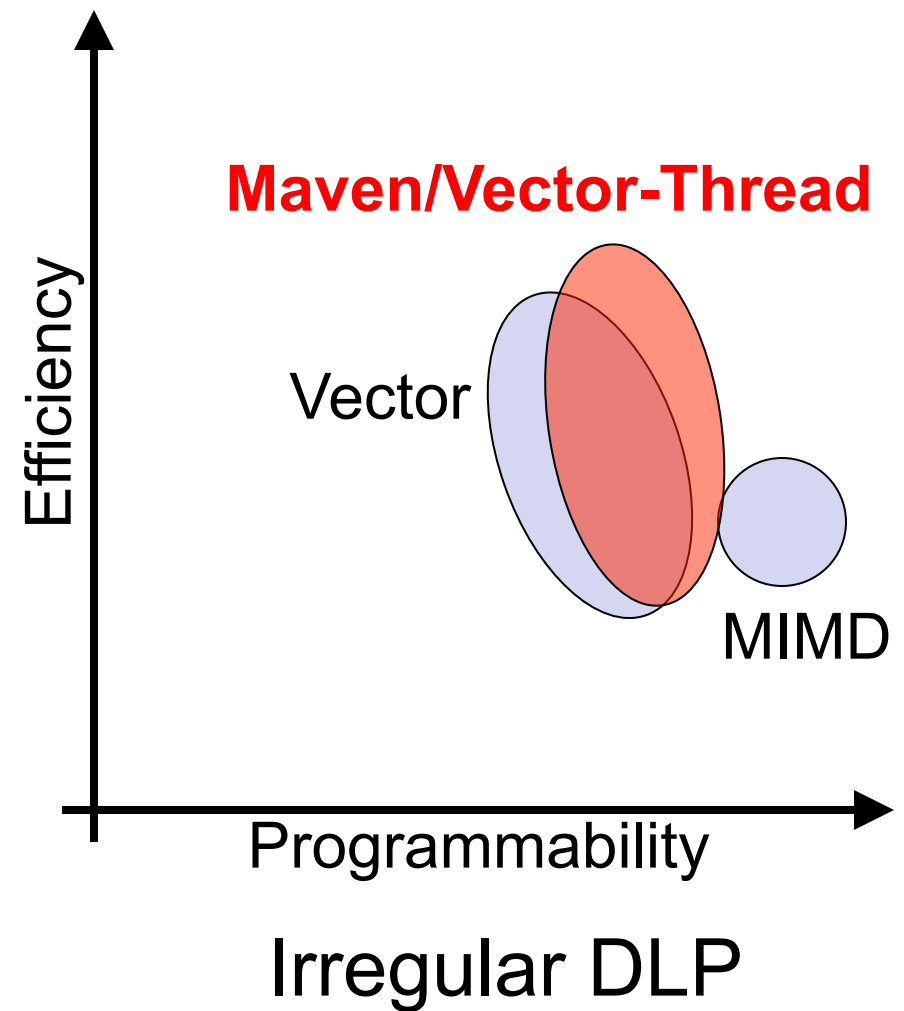
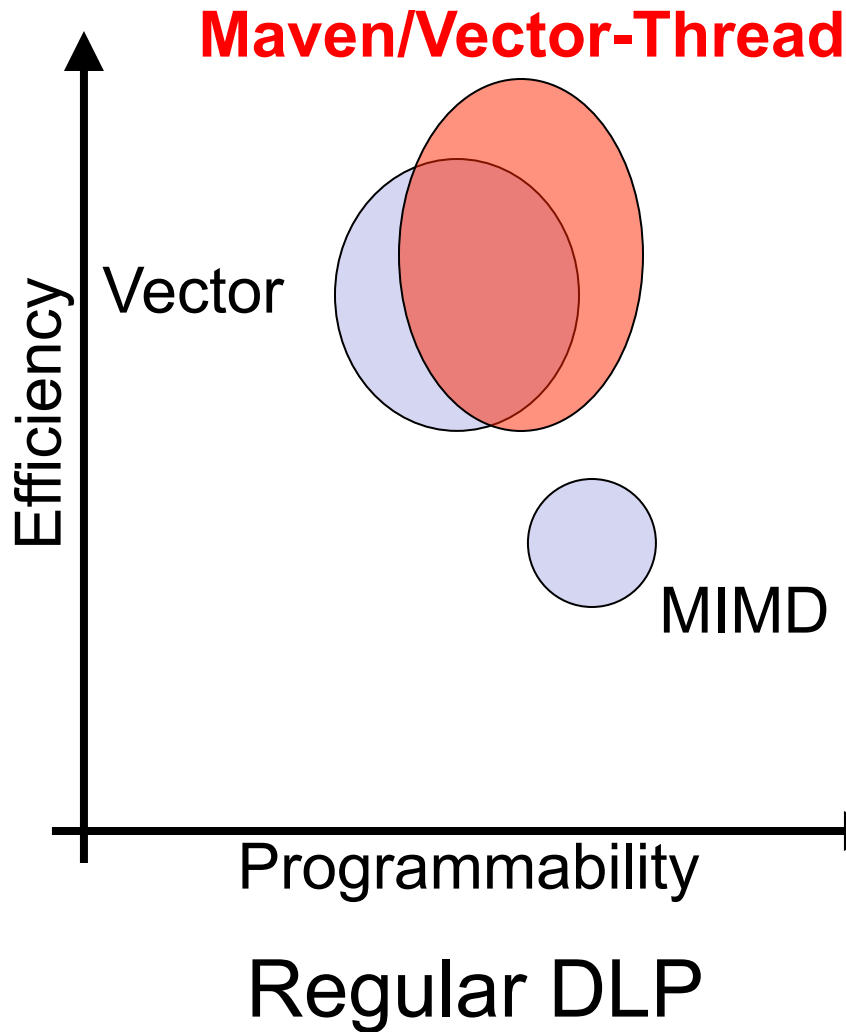


Regular DLP

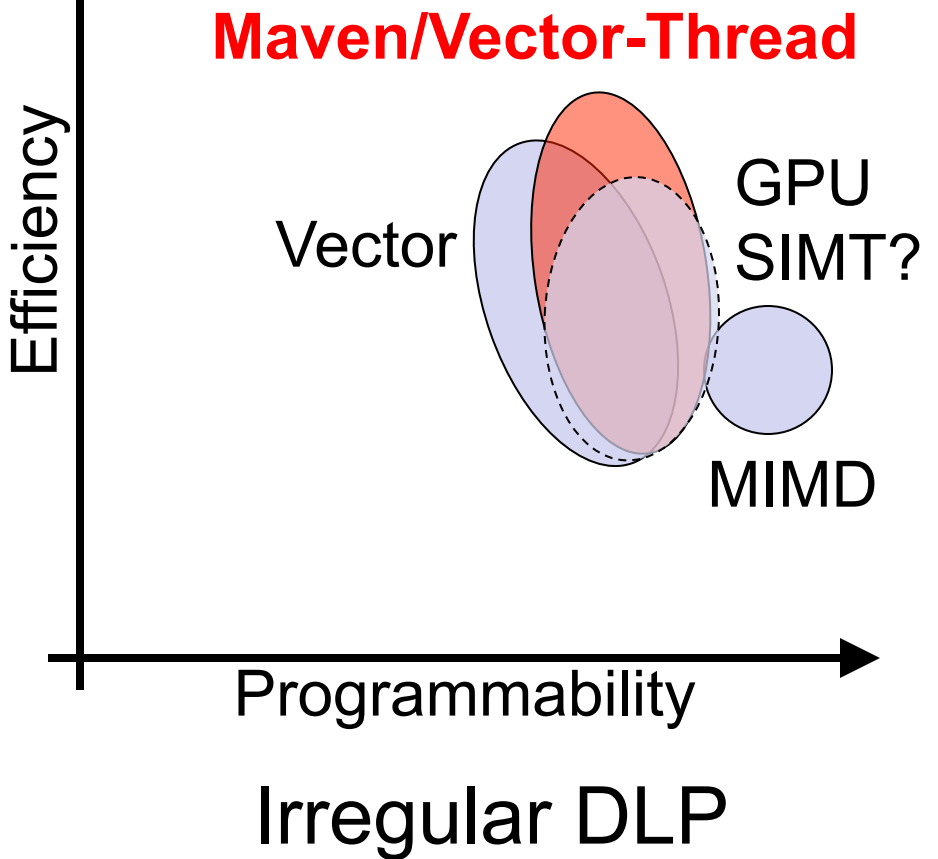
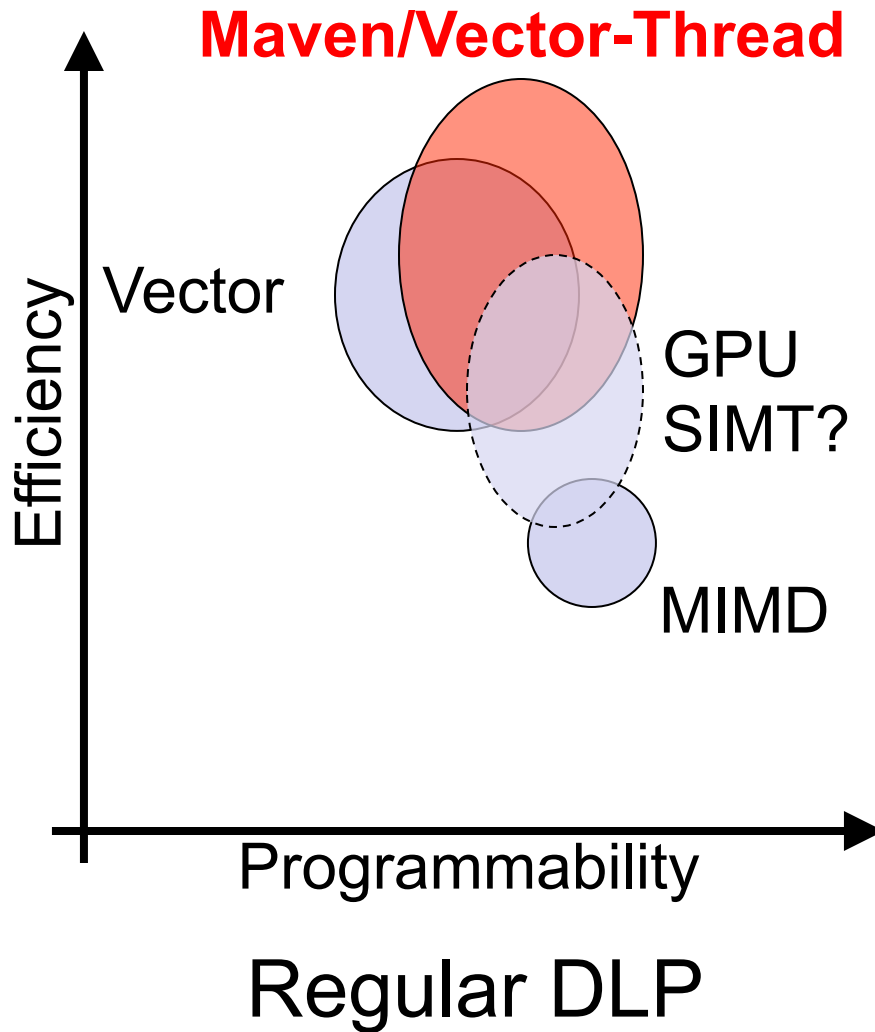


Irregular DLP

Maven Provides Both Greater Efficiency and Easier Programmability



Where does the GPU/SIMT fit in this picture?

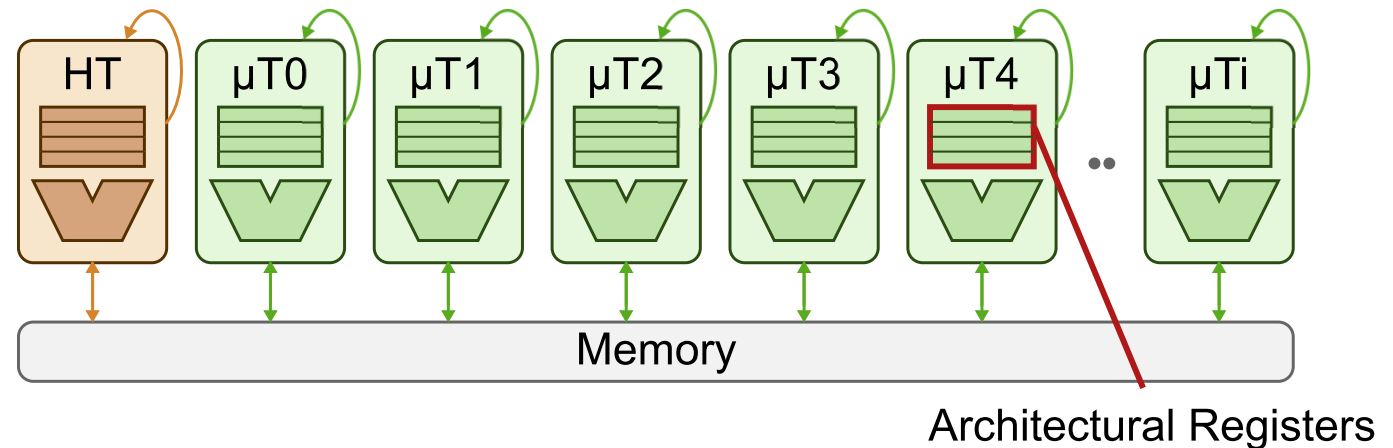


Outline

- **Data-Parallel Architecture Design Patterns**
 - MIMD, Vector-SIMD, Subword-SIMD, SIMT, Maven/Vector-Thread
- Microarchitectural Components
- Evaluation Framework
- Evaluation Results

DLP Pattern #1: MIMD

Programmer's
Logical
View



Vector-Vector Add

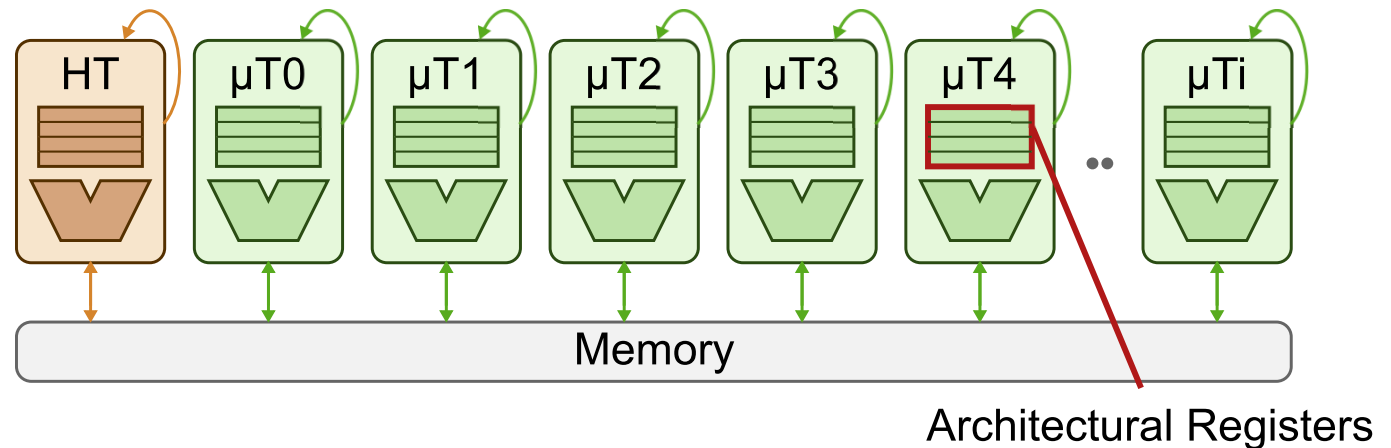
```
loop:
  load  a, a_ptr
  load  b, b_ptr
  add   c, a, b
  store c, c_ptr
  a_ptr++
  b_ptr++
  c_ptr++
  branch
```

Masked Filter

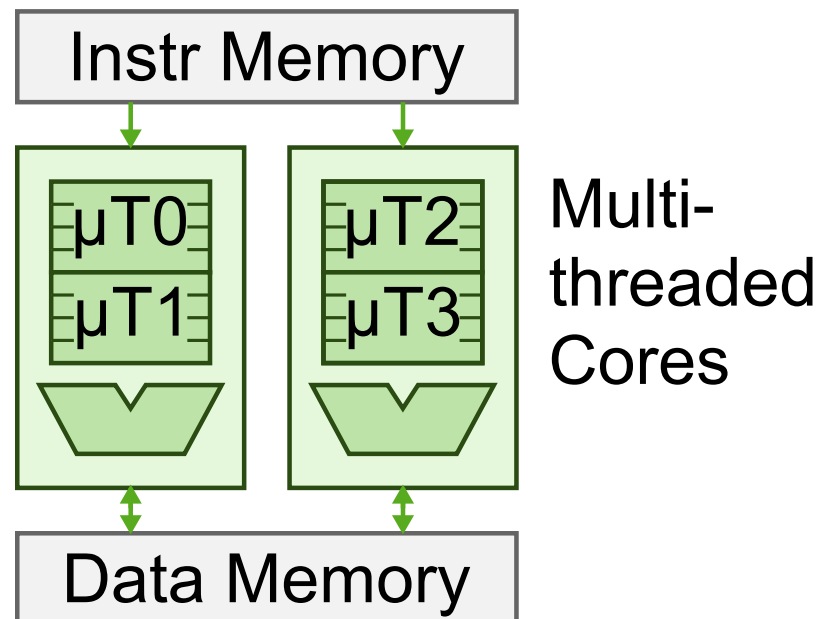
```
loop:
  load  a, a_ptr
  a_ptr++
  branch a = 0
  op0   } FILTER OP
  op1   }
  ...   }
  branch
```

DLP Pattern #1: MIMD

Programmer's
Logical
View



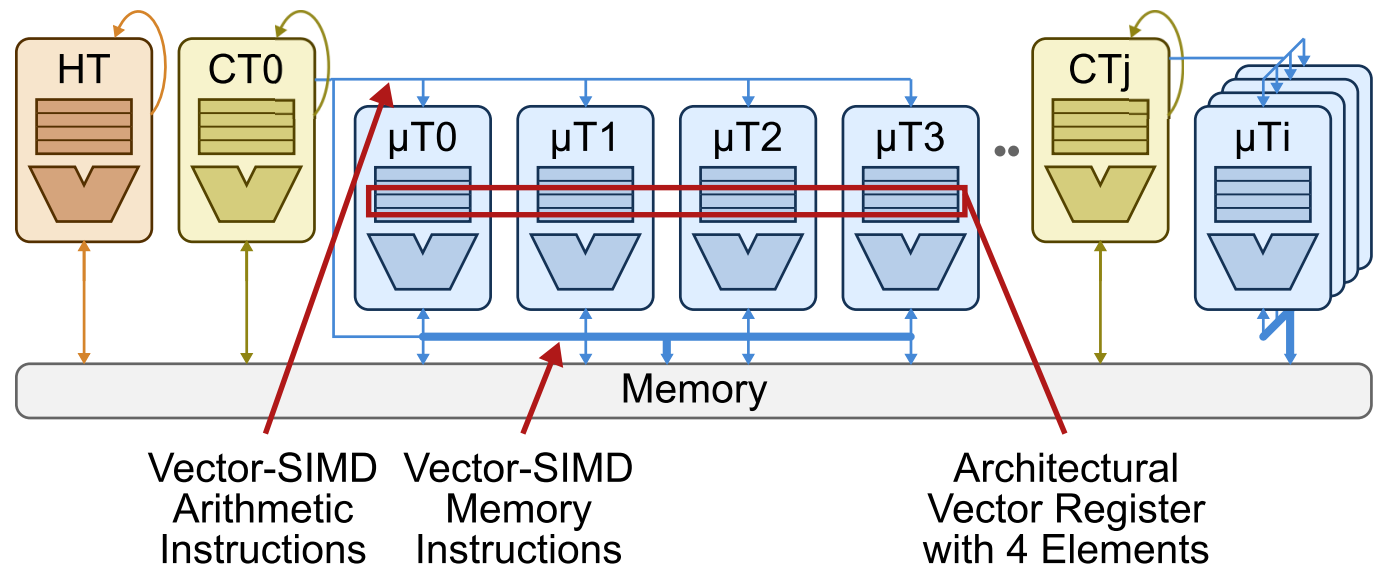
Typical
Micro-
architecture



Examples:
Tilera
Rigel

DLP Pattern #2: Vector-SIMD

Programmer's
Logical
View



Vector-Vector Add

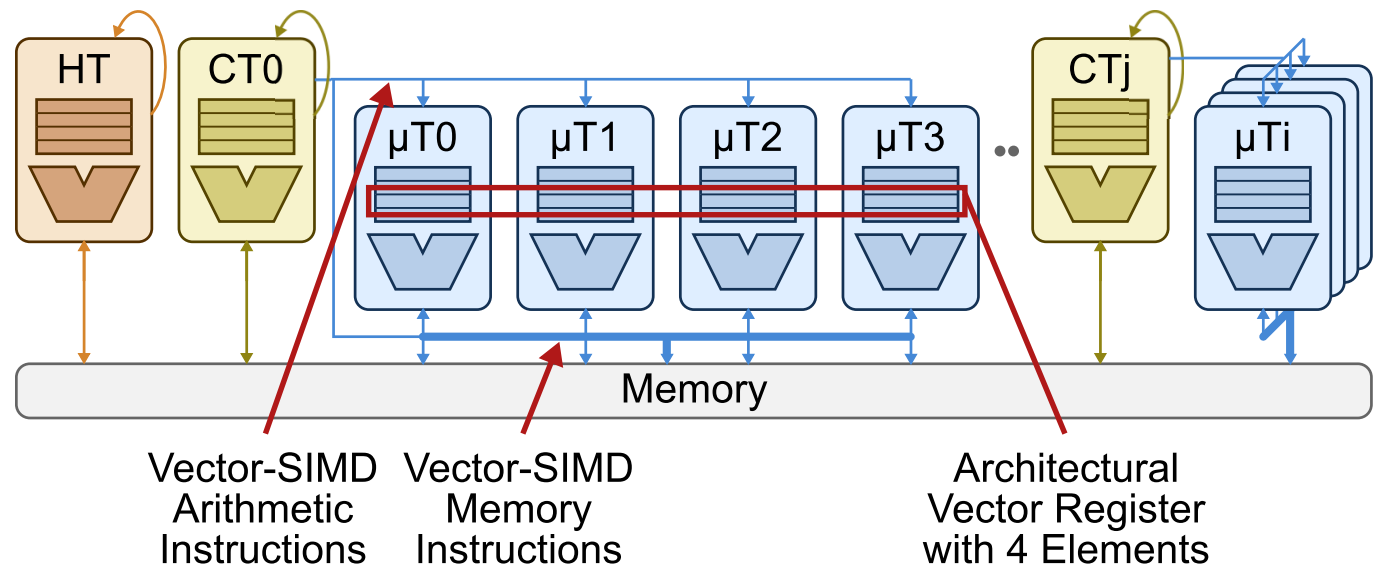
```
loop:
  vload  A, a_ptr
  vload  B, b_ptr
  vadd   C, A, B
  vstore C, c_ptr
  a_ptr++
  b_ptr++
  c_ptr++
  branch
```

Masked Filter

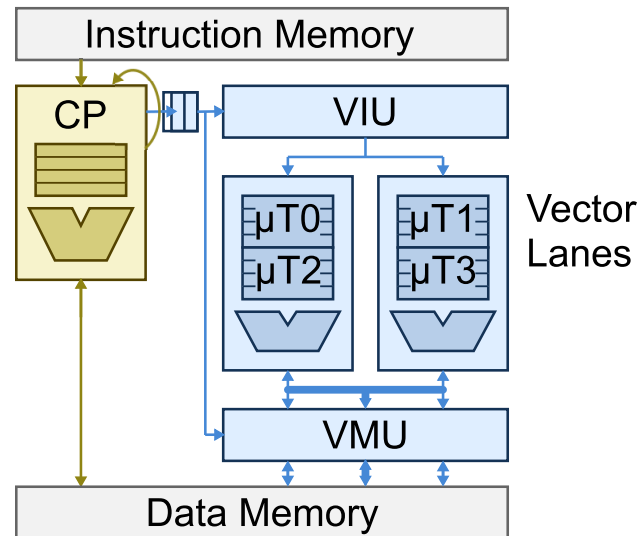
```
loop:
  vload  A, a_ptr
  vset   F, A = 0
  vop0   under flag F
  vop1   under flag F
  ...
  a_ptr++
  branch
```

DLP Pattern #2: Vector-SIMD

Programmer's
Logical
View



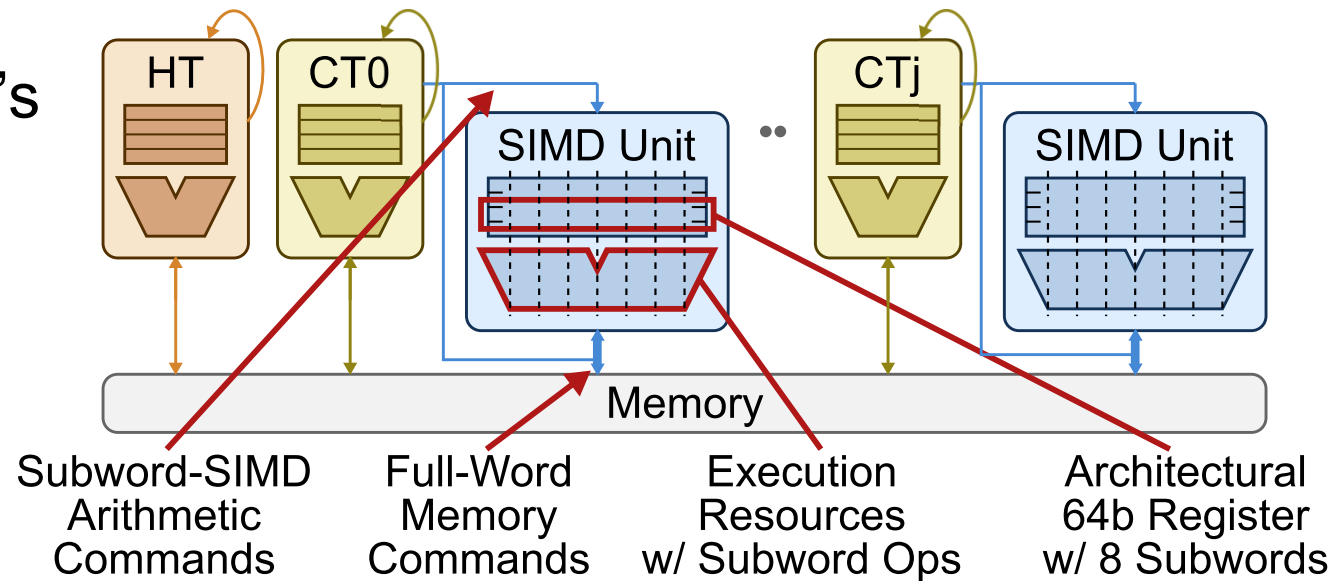
Typical
Micro-
architecture



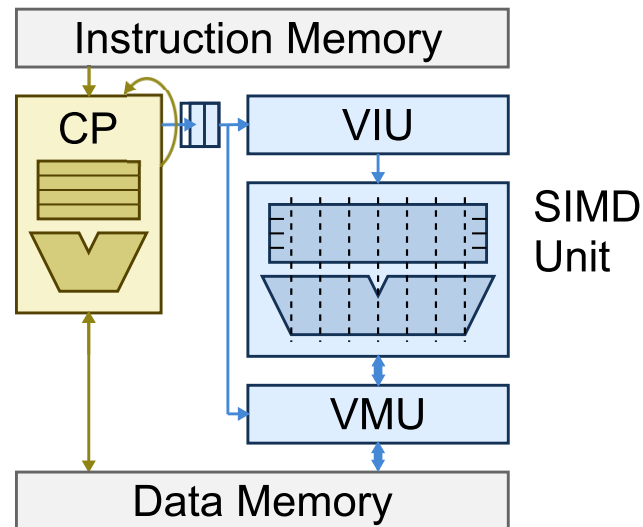
Examples:
T0
Cray-1

DLP Pattern #3: Subword-SIMD

Programmer's
Logical
View



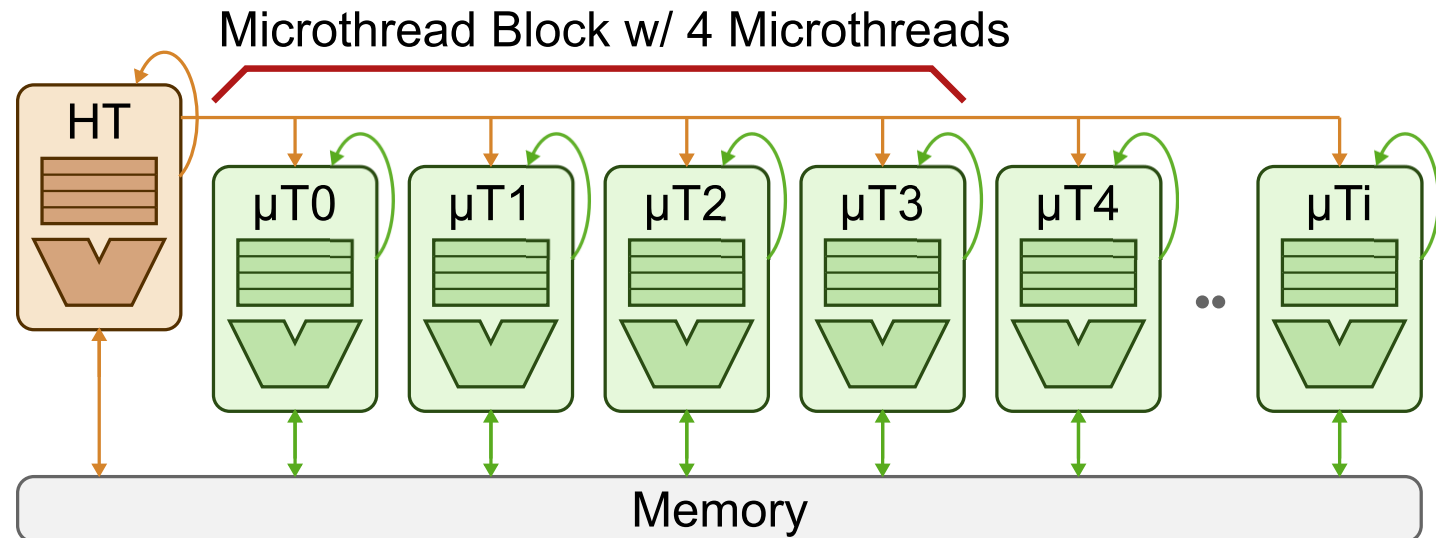
Typical
Micro-
architecture



Examples:
AVX/SSE

DLP Pattern #4: GPU/SIMT

Programmer's
Logical
View



Vector-Vector Add

```
a_ptr = a_base * thread_id
b_ptr = b_base * thread_id
c_ptr = c_base * thread_id
```

```
load  a, a_ptr
load  b, b_ptr
add   c, a, b
store c, c_ptr
```

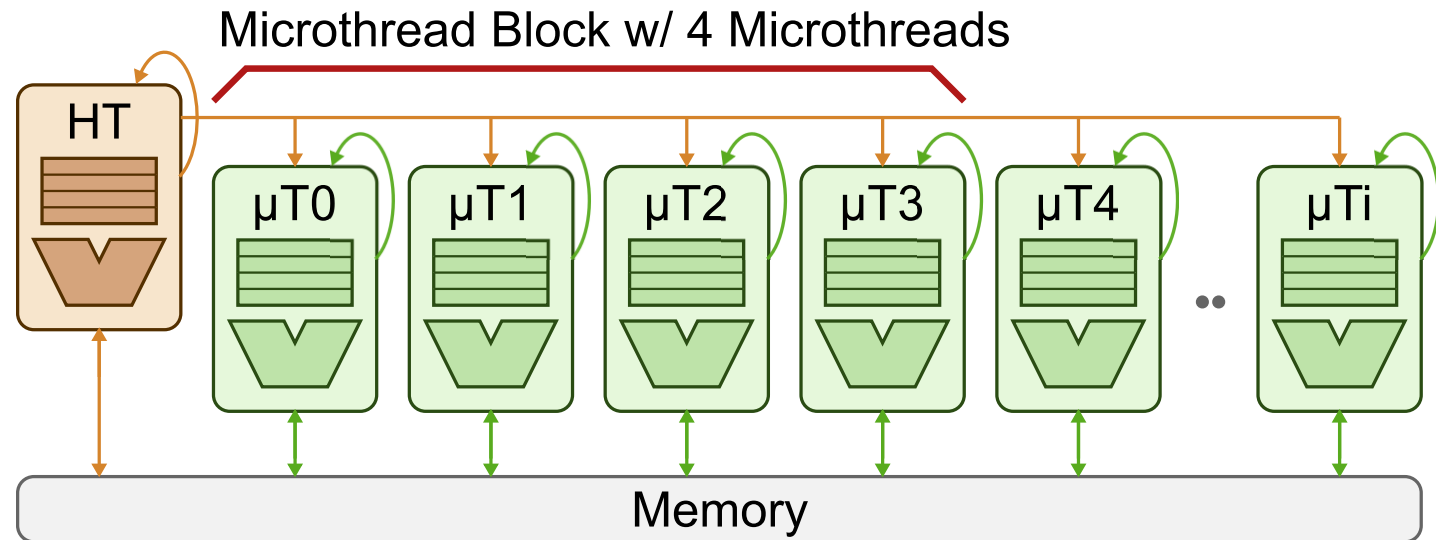
Masked Filter

```
a_ptr = a_base * thread_id
```

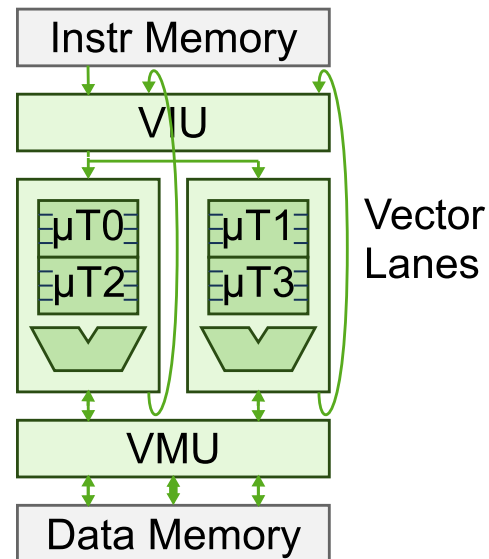
```
load  a, a_ptr
branch a = 0
op0
op1
...
```

DLP Pattern #4: GPU/SIMT

Programmer's
Logical
View



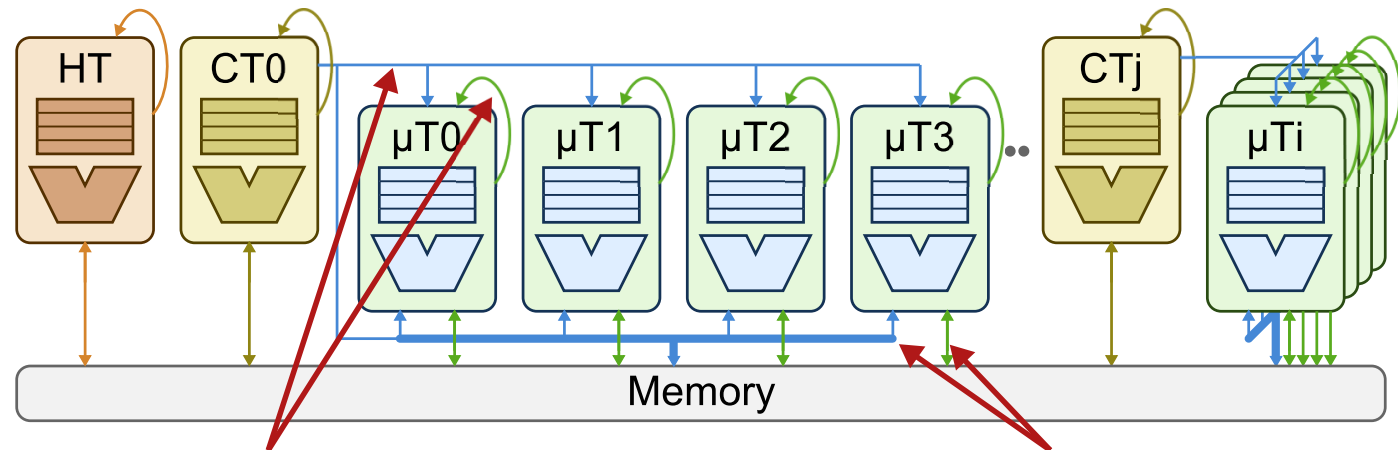
Typical
Micro-
architecture



Example:
Fermi

DLP Pattern #5: Vector-Thread (VT)

Programmer's
Logical
View



Vector and Scalar
Control Flow Instructions

Vector and Scalar
Memory Instructions

Vector-Vector Add

```
loop:
  vload  A, a_ptr
  vload  B, b_ptr
  vfetch ut_code
  vstore C, c_ptr
  a_ptr++
  b_ptr++
  c_ptr++
  branch
```

ut_code:
 add c, a, b
 stop

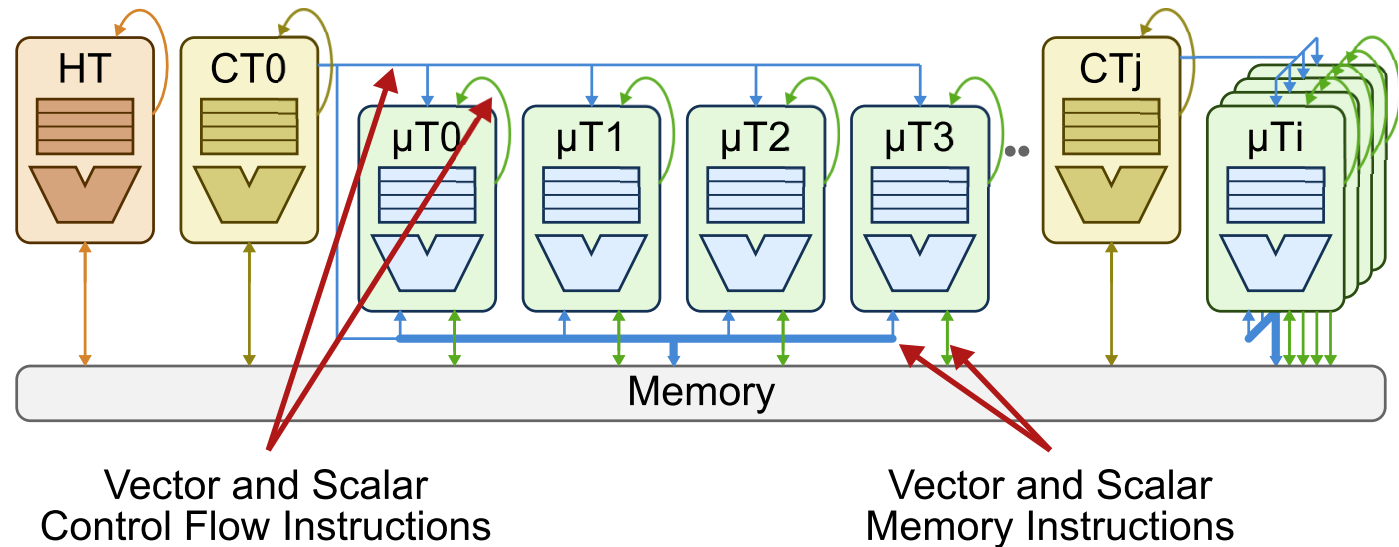
Masked Filter

```
loop:
  vload  A, a_ptr
  vfetch ut_code
  a_ptr++
  branch
```

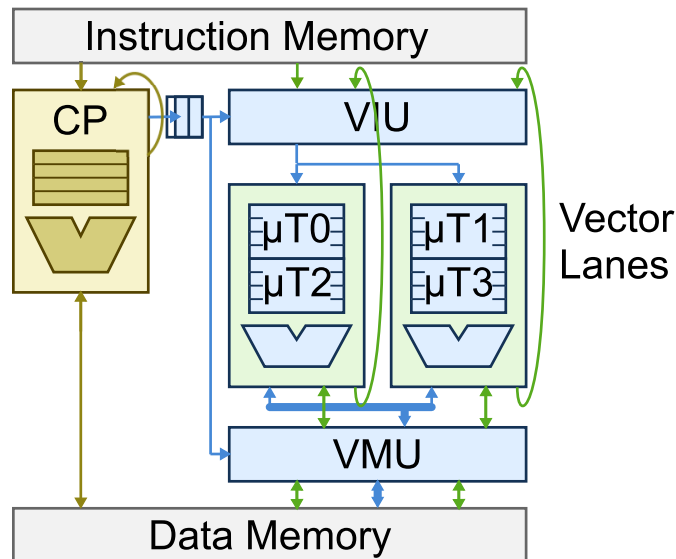
ut_code:
 branch a = 0
 op0
 op1
 ...
 stop

DLP Pattern #5: Vector-Thread (VT)

Programmer's
Logical
View



Typical
Micro-
architecture

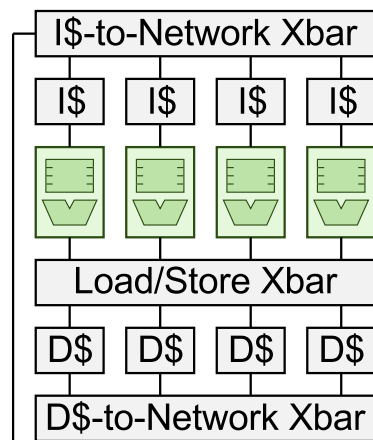
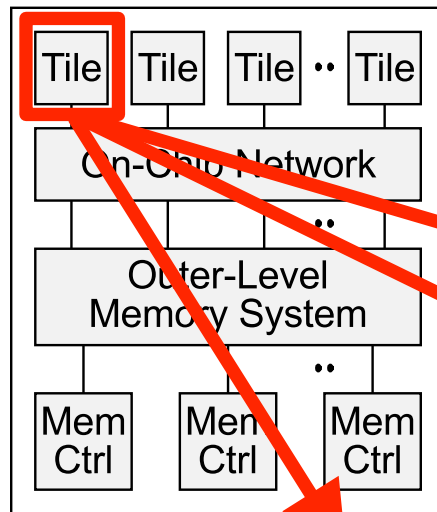


Examples:
Scale
Maven

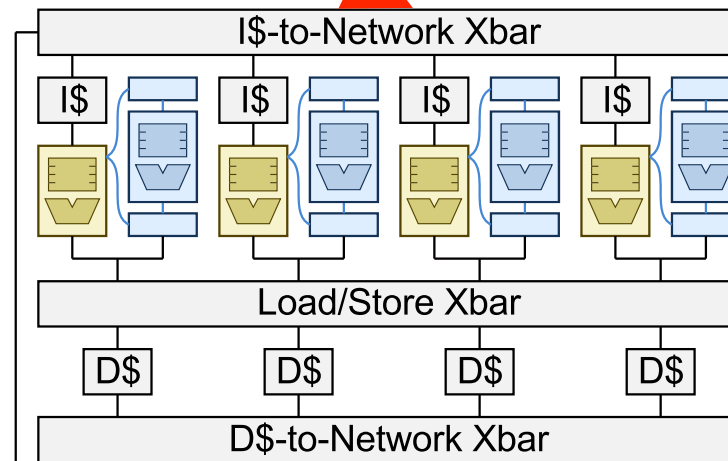
Outline

- Data Parallel Architectural Design Patterns
- **Microarchitectural Components**
- Evaluation Framework
- Evaluation Results

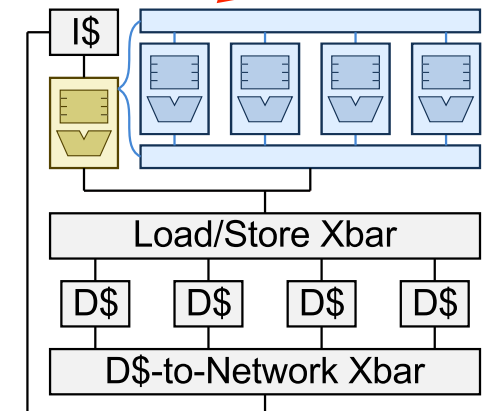
Focus on the Tile



MIMD Tile



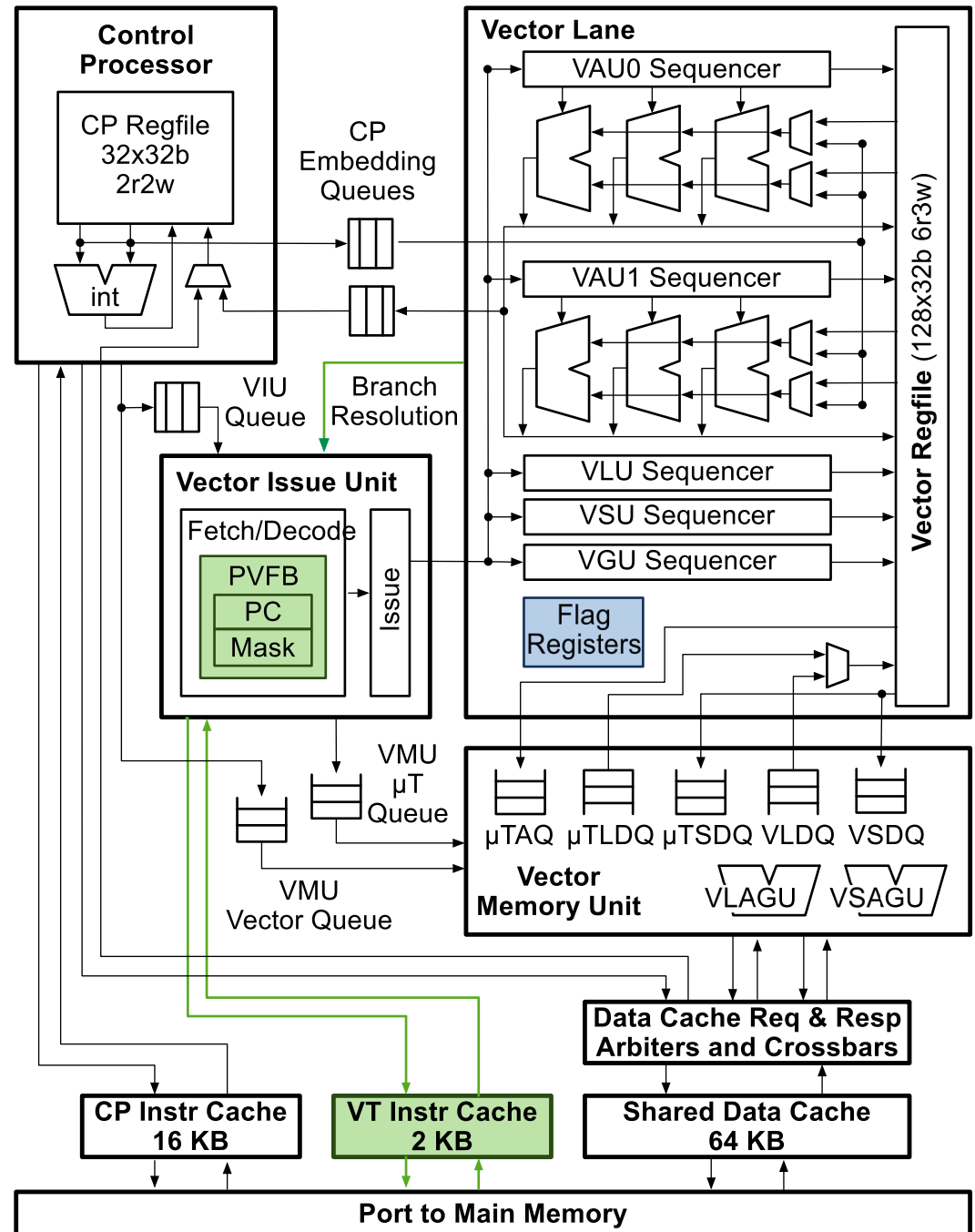
Vector Tile with
Four Single-Lane Cores



Vector Tile with
One Four-Lane Core

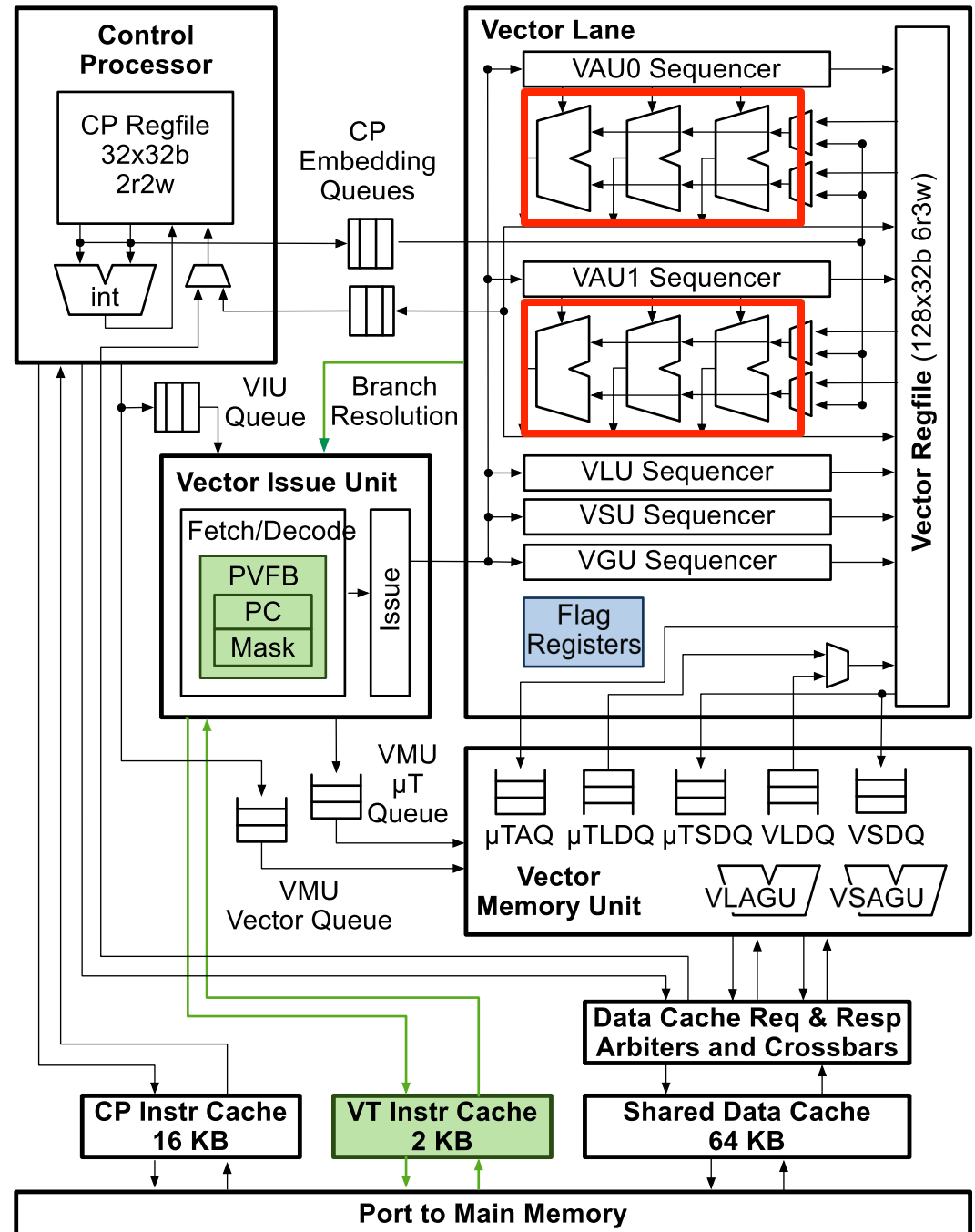
uArchitecture

- Developed a library of parameterized synthesizable RTL components



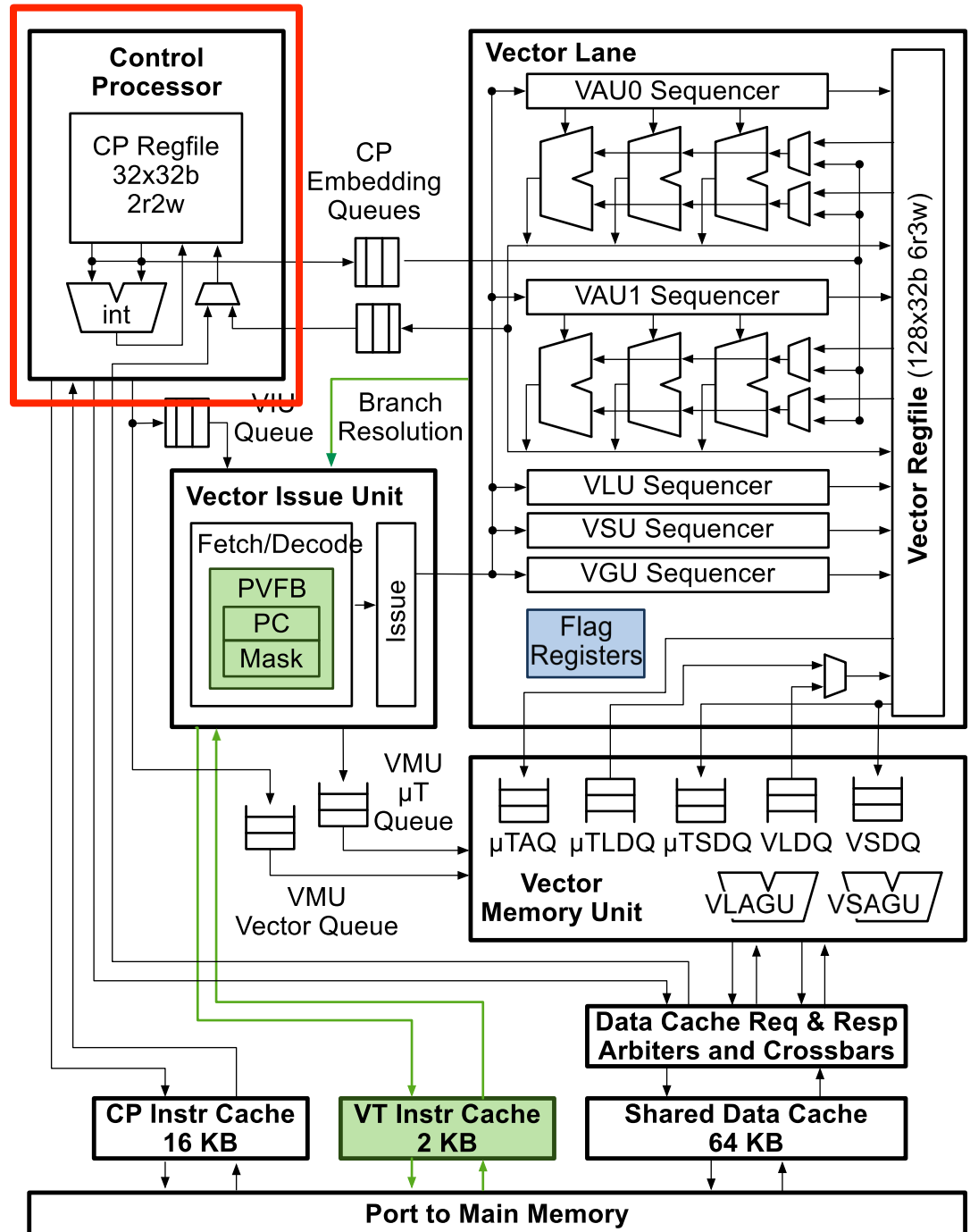
Retimable Long-latency Functional Units

- 32-bit integer multiplier, divider
- Single-precision floating-point add, multiply, divide, square root



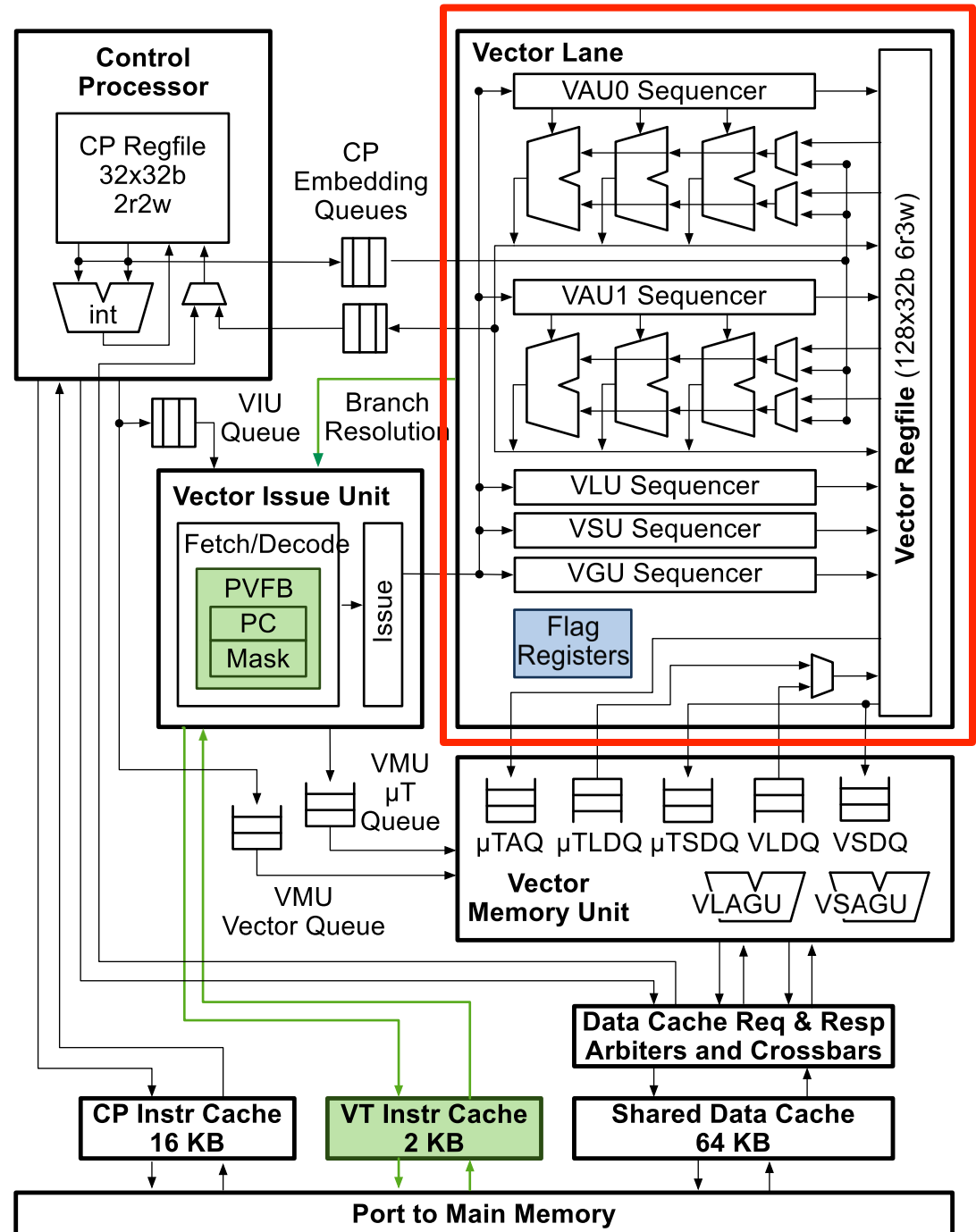
5-stage Multi-threaded Scalar Core

- Change number of entries in register file (32,64,128,256) to vary degree of multi-threading (1,2,4,8 threads)



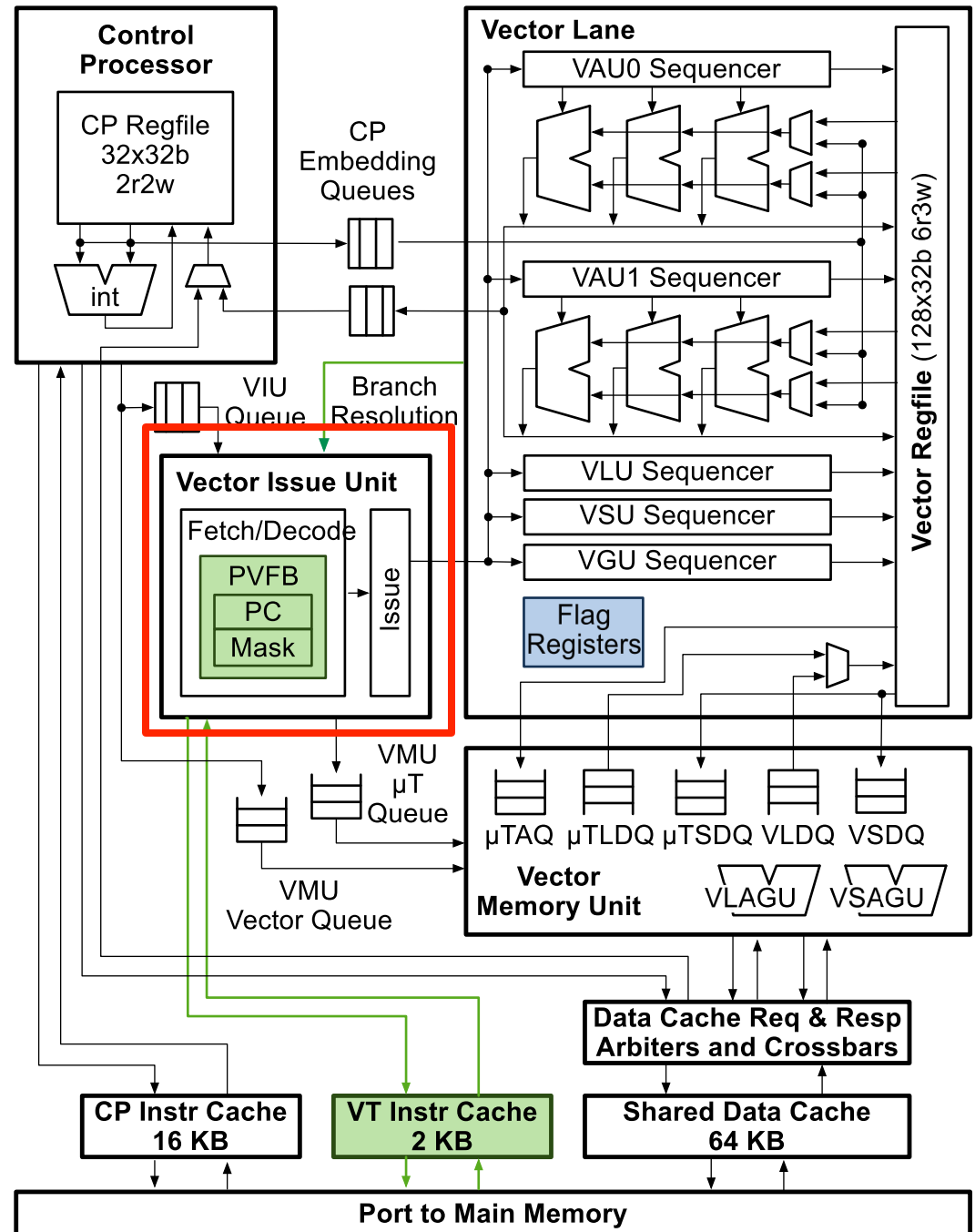
Vector Lanes

- Vector registers and ALUs
- Density-time Execution
- Replicate the lanes and execute in lock step for higher throughput
- Vector-SIMD: Flag Registers



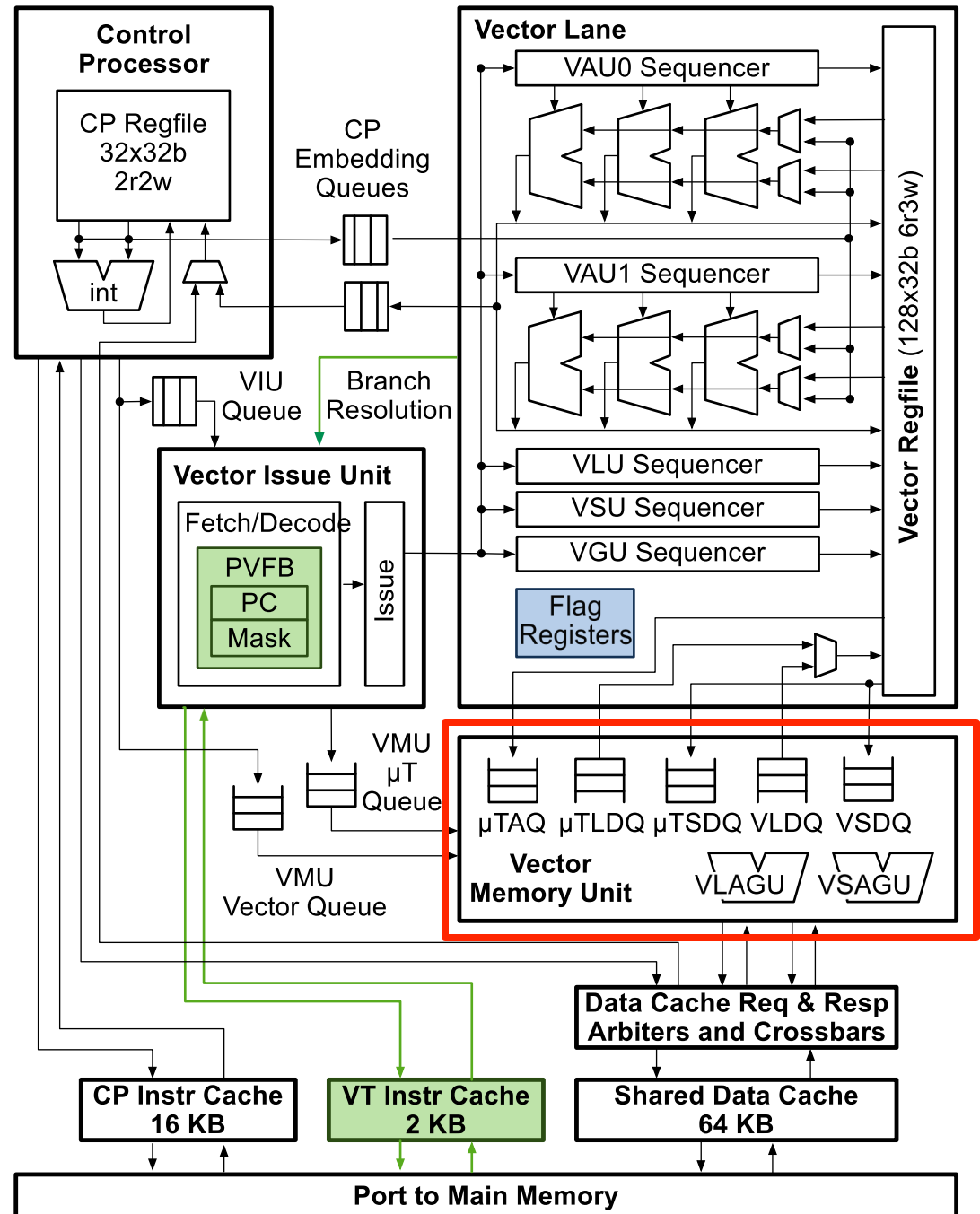
Vector Issue Unit

- Vector-SIMD: VIU only handles scheduling, data dependent control done by flag registers
- Maven: VIU fetches instructions, PVFB handles uT branches and does control flow convergence



Vector Memory Unit

- VMU Handles unit stride, constant stride vector memory operations
- Vector-SIMD: VMU handles scatter, gather
- Maven: VMU handles uT loads and stores

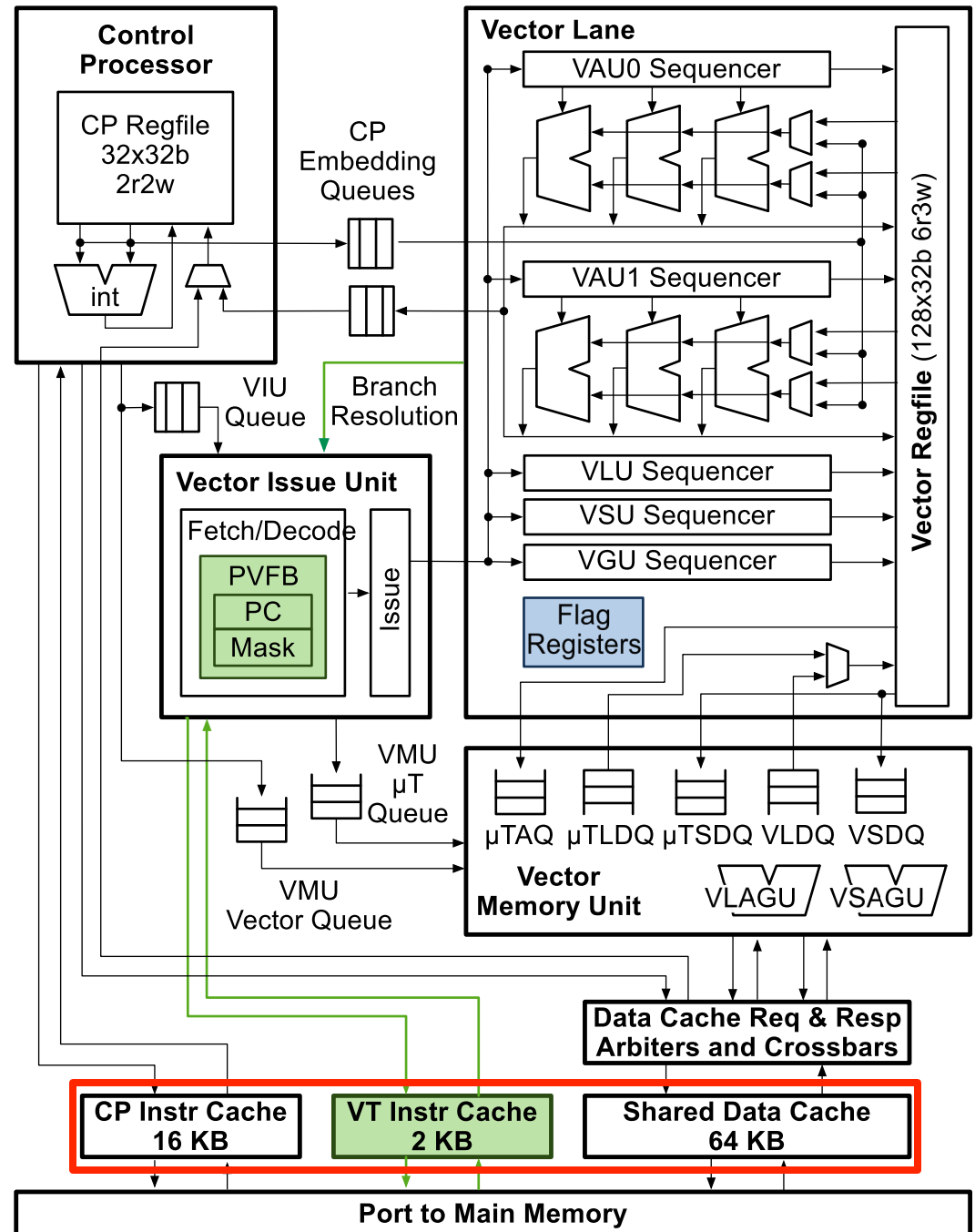


Blocking, Non-blocking Caches

- Access Port Width
- Refill Port Width
- Cache Line Size
- Total Capacity
- Associativity

Only for Non-blocking Caches:

- # MSHR
- # secondary misses per MSHR



A Big Design Space ...

- Number of entries in scalar register file
 - 32,64,128,256 (1,2,4,8 threads)
- Number of entries in vector register file
 - 32,64,128,256
- Architecture of vector register file
 - 6r3w unified register file, 4x 2r1w banked register file
- Per-bank integer ALU
- Density time execution
- Pending Vector Fragment Buffer (PVFB)
 - FIFO, 1-stack, 2-stack

Outline

- Data Parallel Architectural Design Patterns
- Microarchitectural Components
- **Evaluation Framework**
- Evaluation Results

Programming Methodology

- Use GCC C++ Cross Compiler (which we ported)
- MIMD
 - Custom application-scheduled lightweight threading lib
- Vector-SIMD
 - Leverage built-in GCC vectorizer for mapping very simple regular DLP code
 - Use GCC's inline assembly extensions for more complicated code
- Maven
 - Use C++ Macros with special library, which glues the control thread and microthreads
 - Automatic vector register allocation added to GCC

Microbenchmarks & Application Kernels

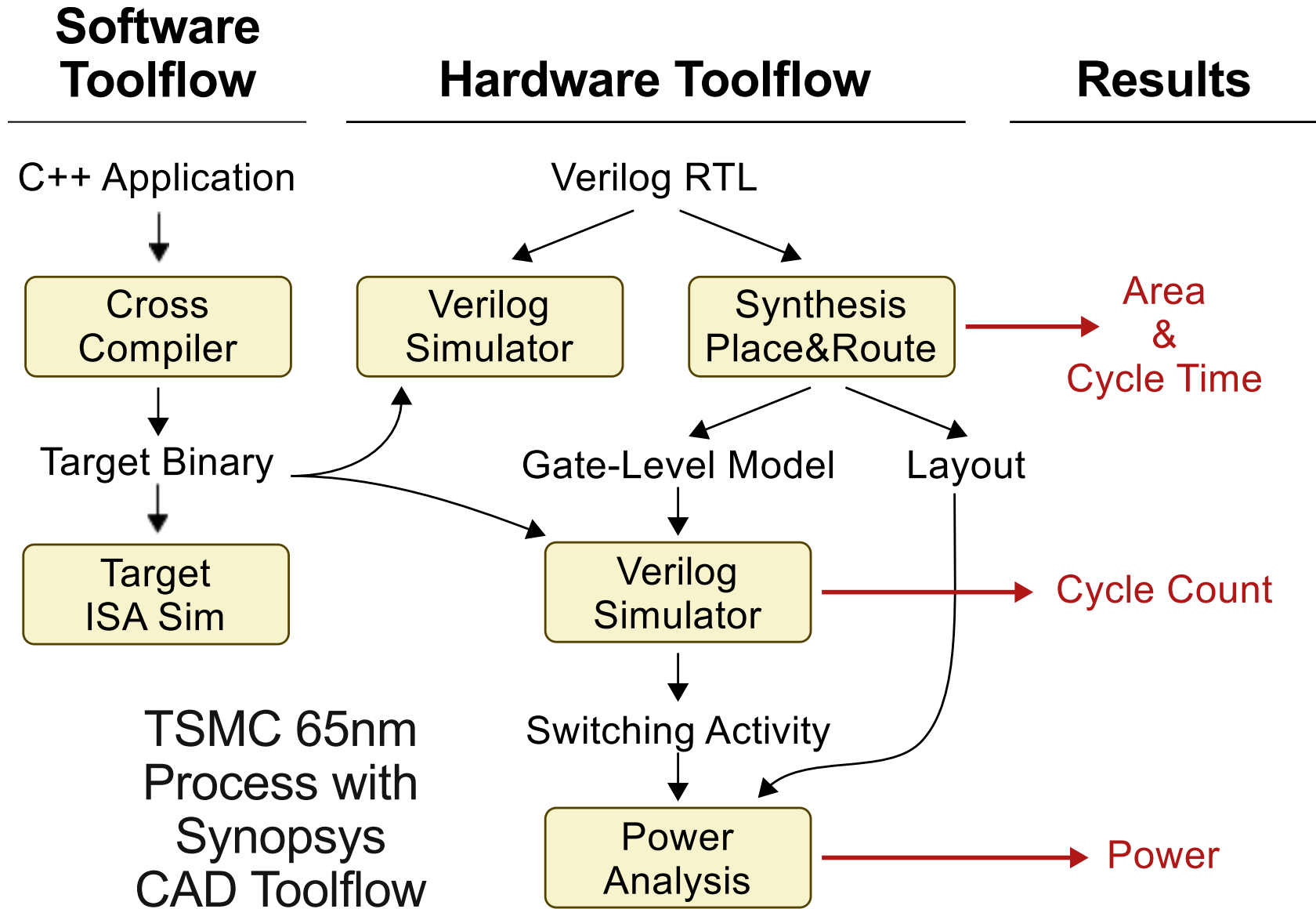
Microbenchmarks

Name	Explanation	Irregularity
vvadd	1000 element FP vector-vector add	Regular
bsearch	1000 look-ups into a sorted array	Very Irregular
bsearch-cmv	inner-loop rewritten with cond. mov	Somewhat Irregular

Application Kernels

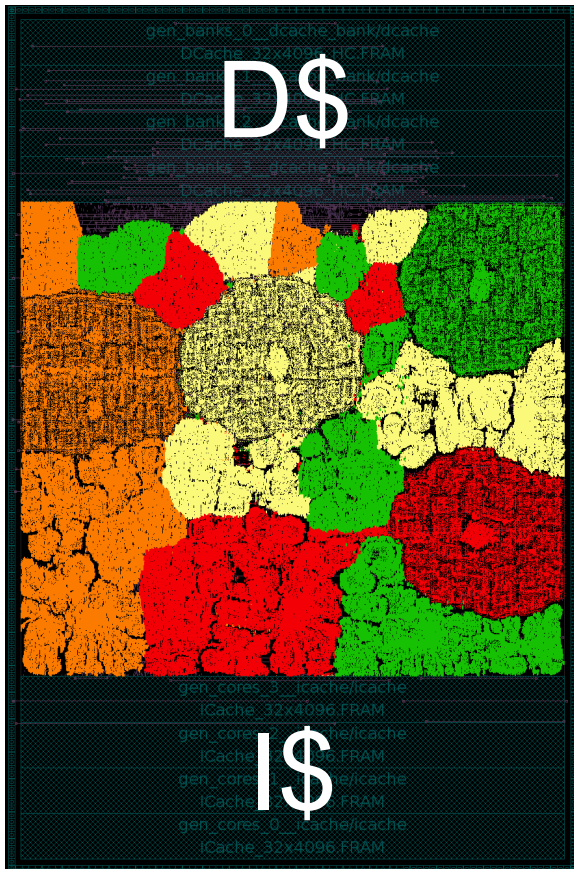
Name	Explanation	Irregularity
viterbi	Decode frames using Viterbi alg.	Regular
rsort	Radix sort on an array of integers	Slightly Irregular
kmeans	K-means clustering algorithm	Slightly Irregular
dither	Floyd-Steinberg dithering	Somewhat Irregular
physics	Newtonian physics simulation	Very Irregular
strsearch	Knuth-Morris-Pratt algorithm	Very Irregular

Evaluation Methodology

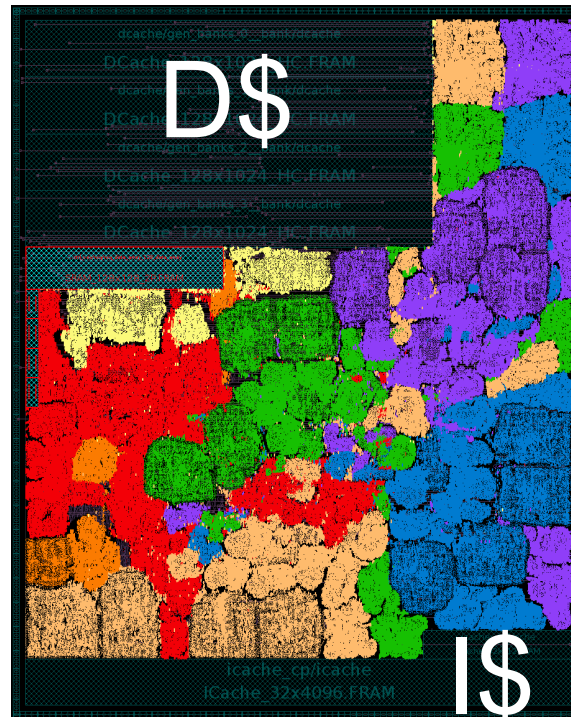


Three Example Layouts

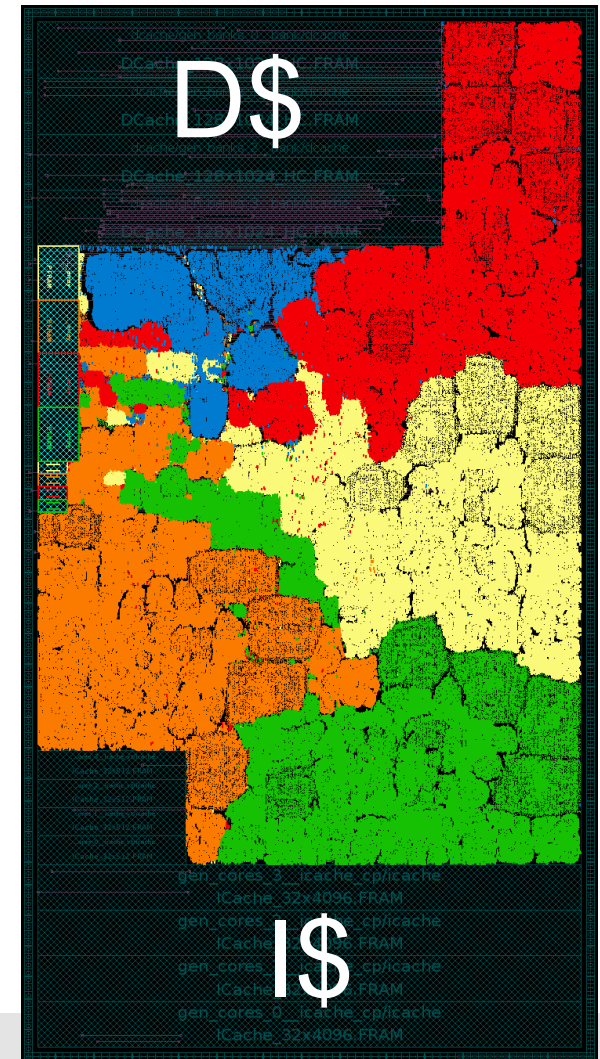
MIMD Tile



1 Core x 4 Lanes
Maven Tile



4 Cores x 1 Lane
Maven Tile



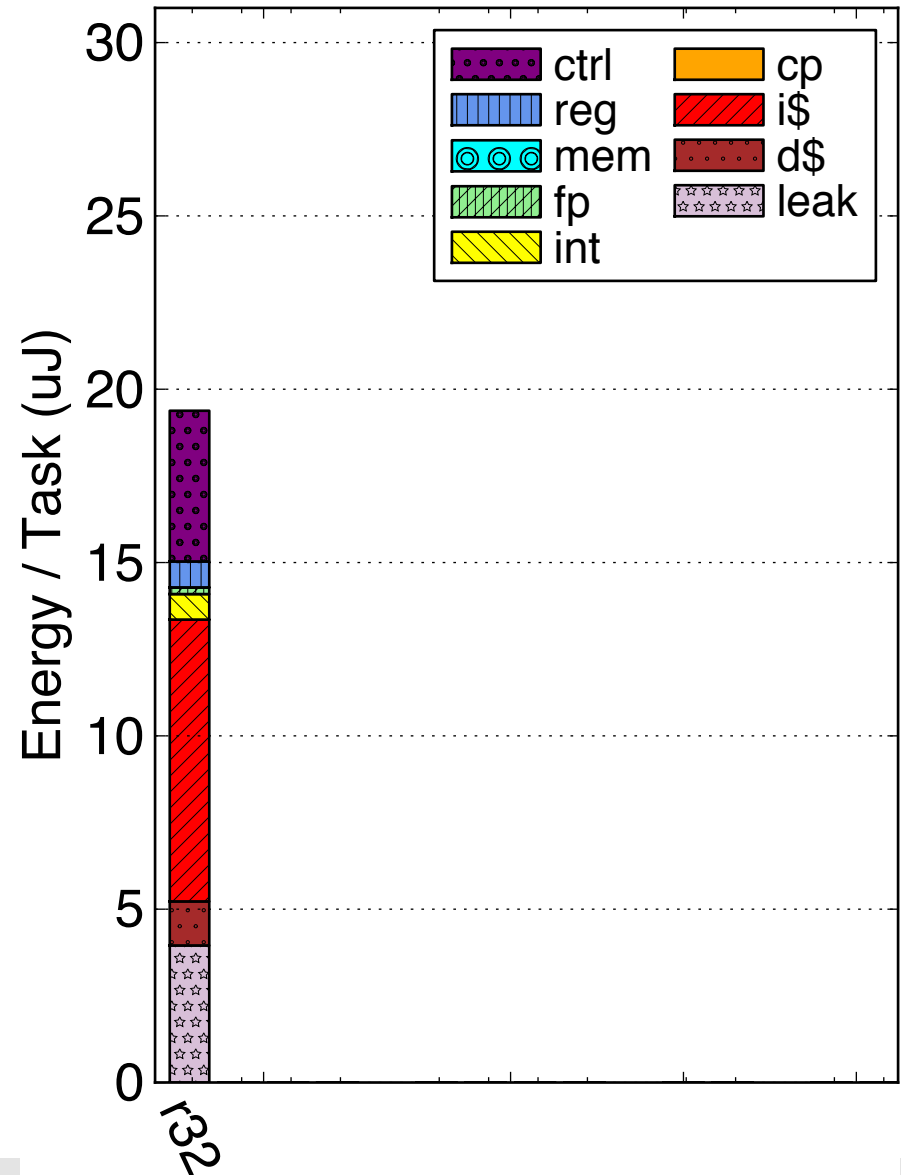
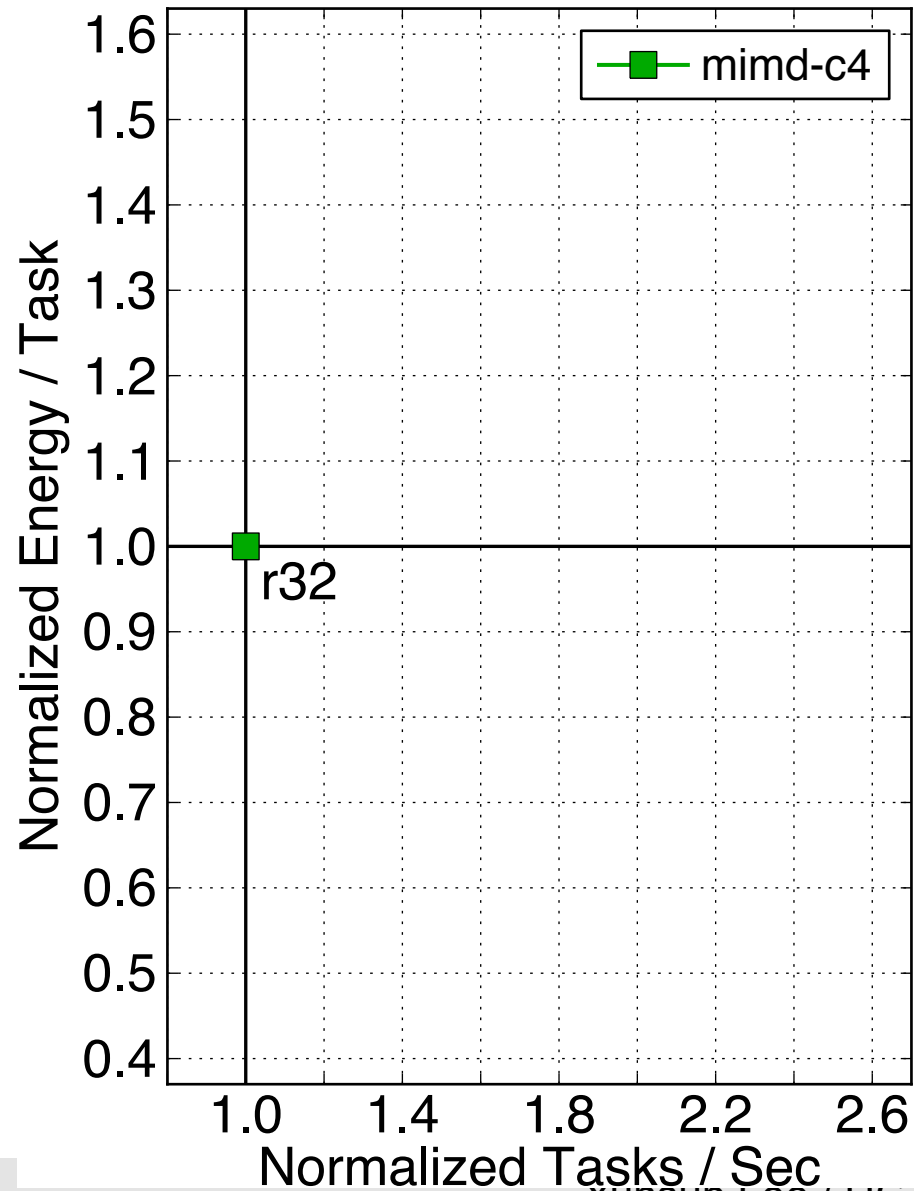
Need Gate-level Activity for Accurate Energy Numbers

Configuration	Post Place&Route Statistical (mW)	Simulated Gate-level Activity (mW)
MIMD 1	149	137-181
MIMD 2	216	130-247
MIMD 3	242	124-261
MIMD 4	299	221-298
Multi-core Vector-SIMD	396	213-331
Multi-lane Vector-SIMD	224	137-252
Multi-core Vector-Thread 1	428	162-318
Multi-core Vector-Thread 2	404	147-271
Multi-core Vector-Thread 3	445	172-298
Multi-core Vector-Thread 4	409	225-304
Multi-core Vector-Thread 5	410	168-300
Multi-lane Vector-Thread 1	205	111-167
Multi-lane Vector-Thread 2	223	118-173

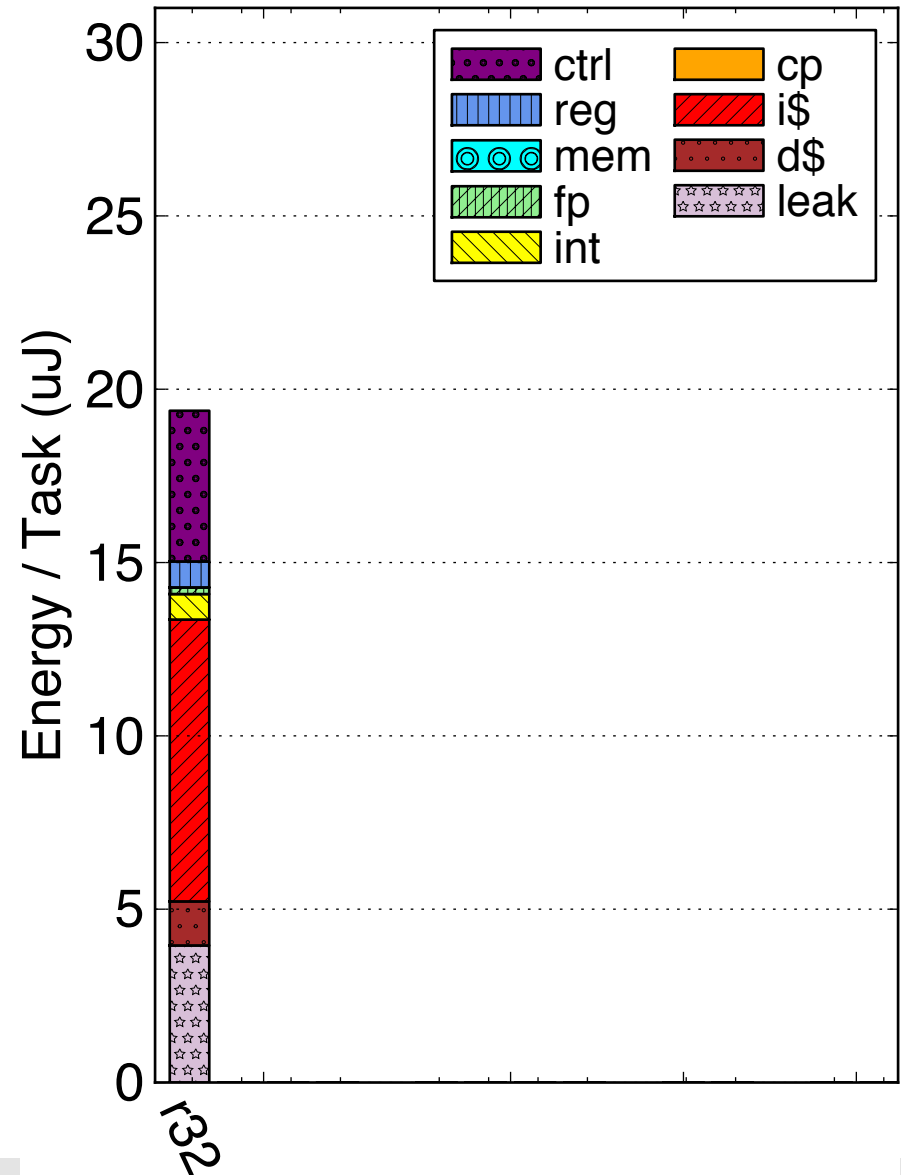
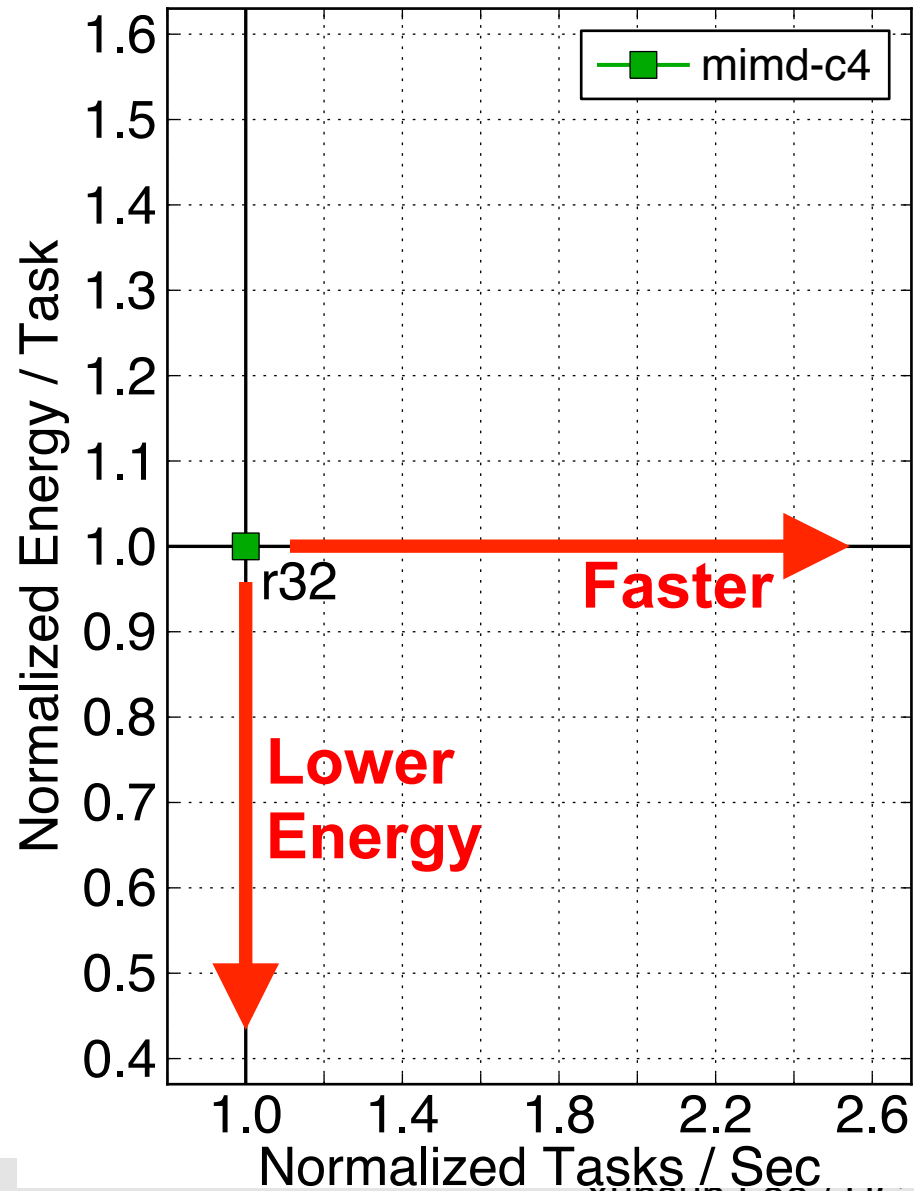
Outline

- Data Parallel Architectural Design Patterns
- Microarchitectural Components
- Evaluation Framework
- **Evaluation Results**

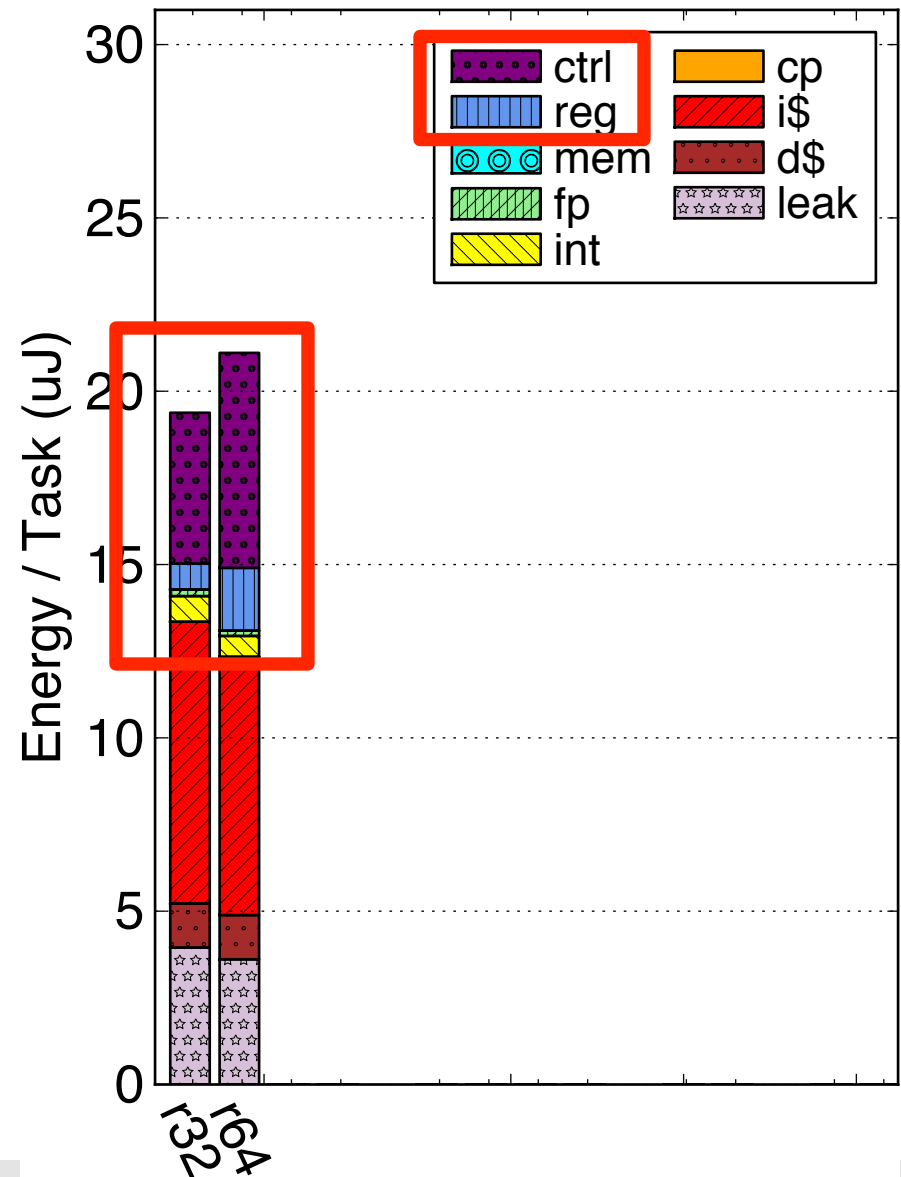
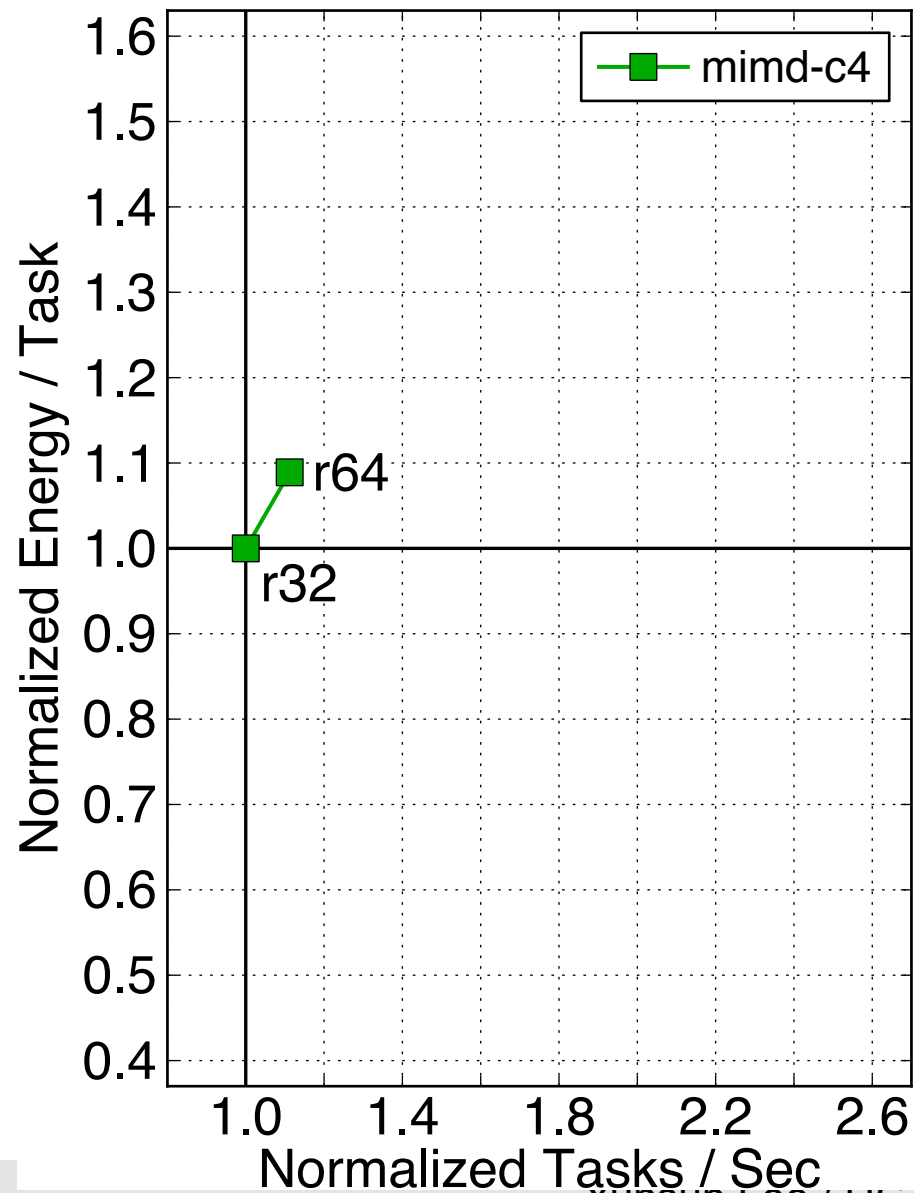
Efficiency vs. Number of uTs running *bsearch-cmv*



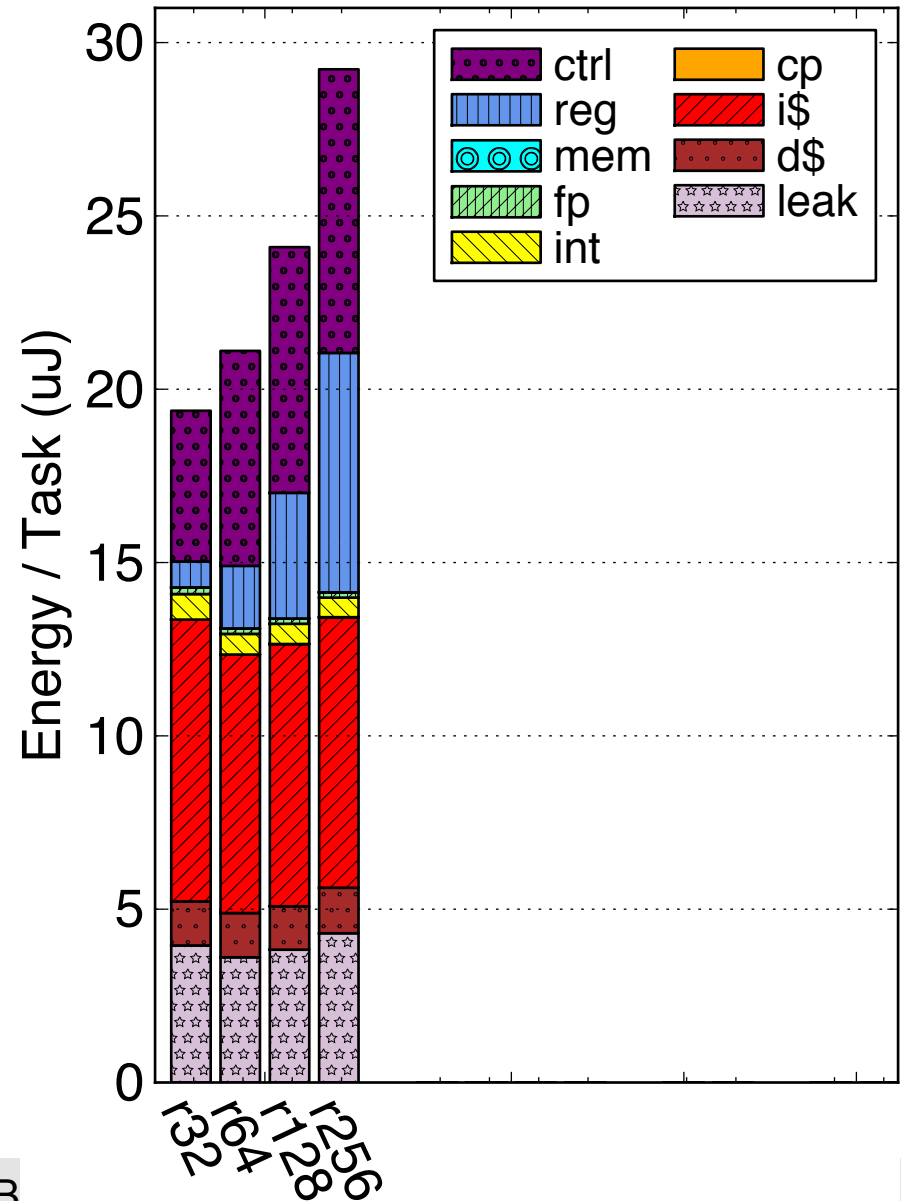
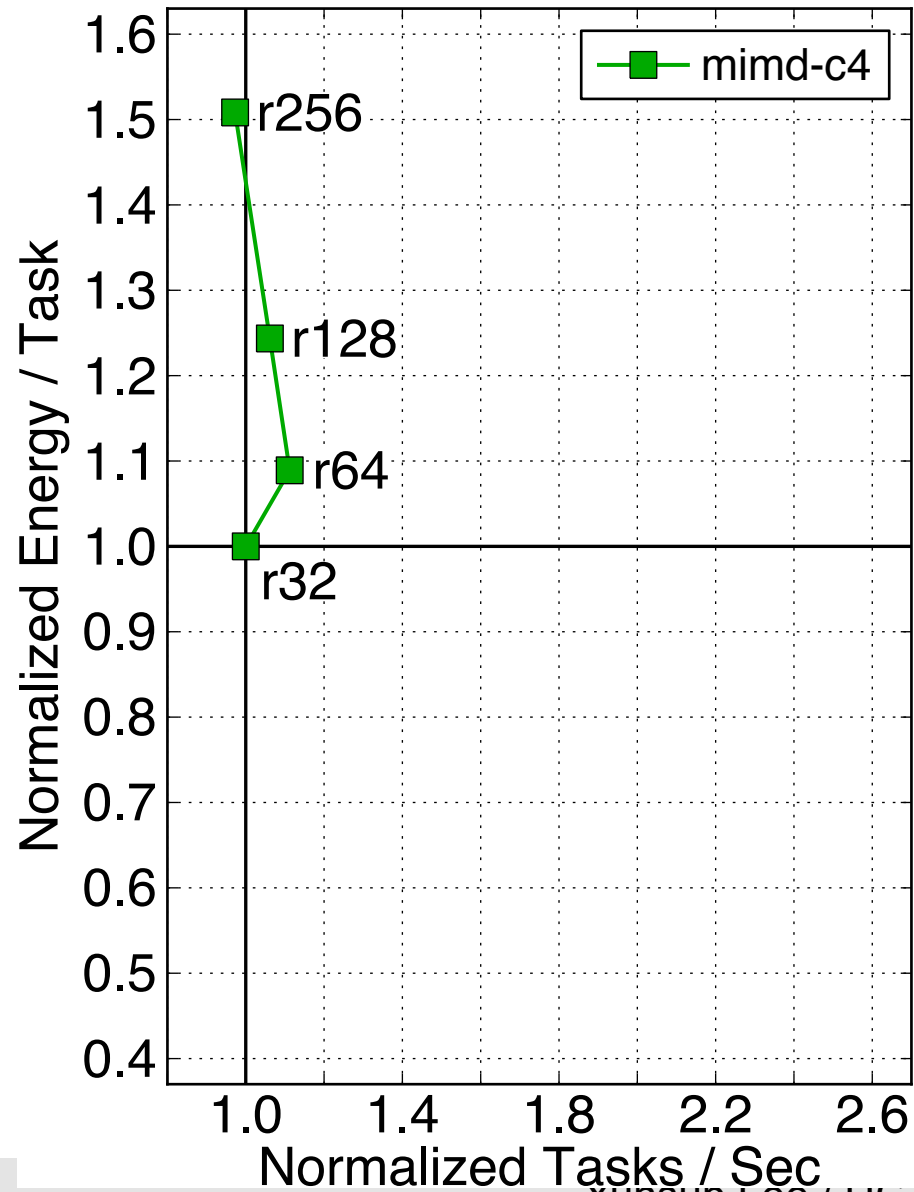
Efficiency vs. Number of uTs running *bsearch-cmv*



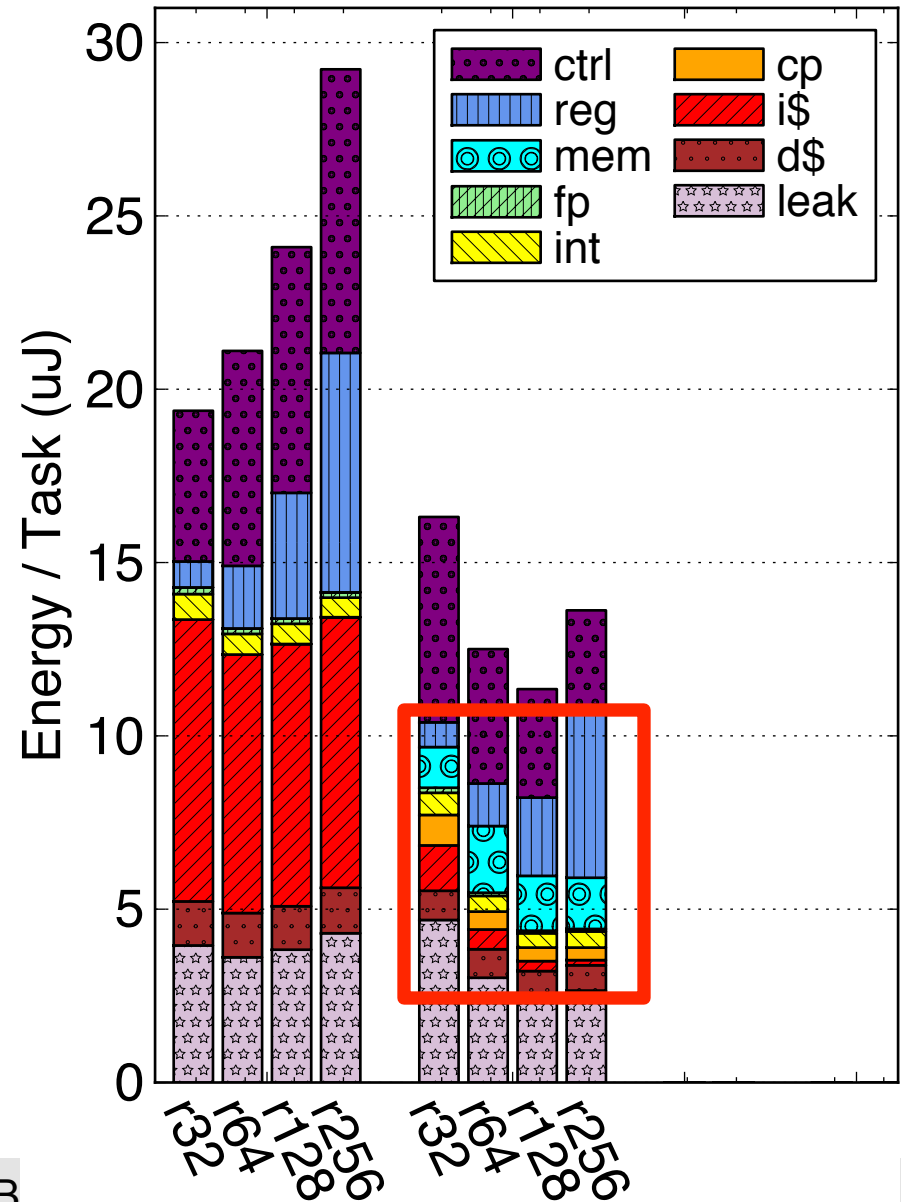
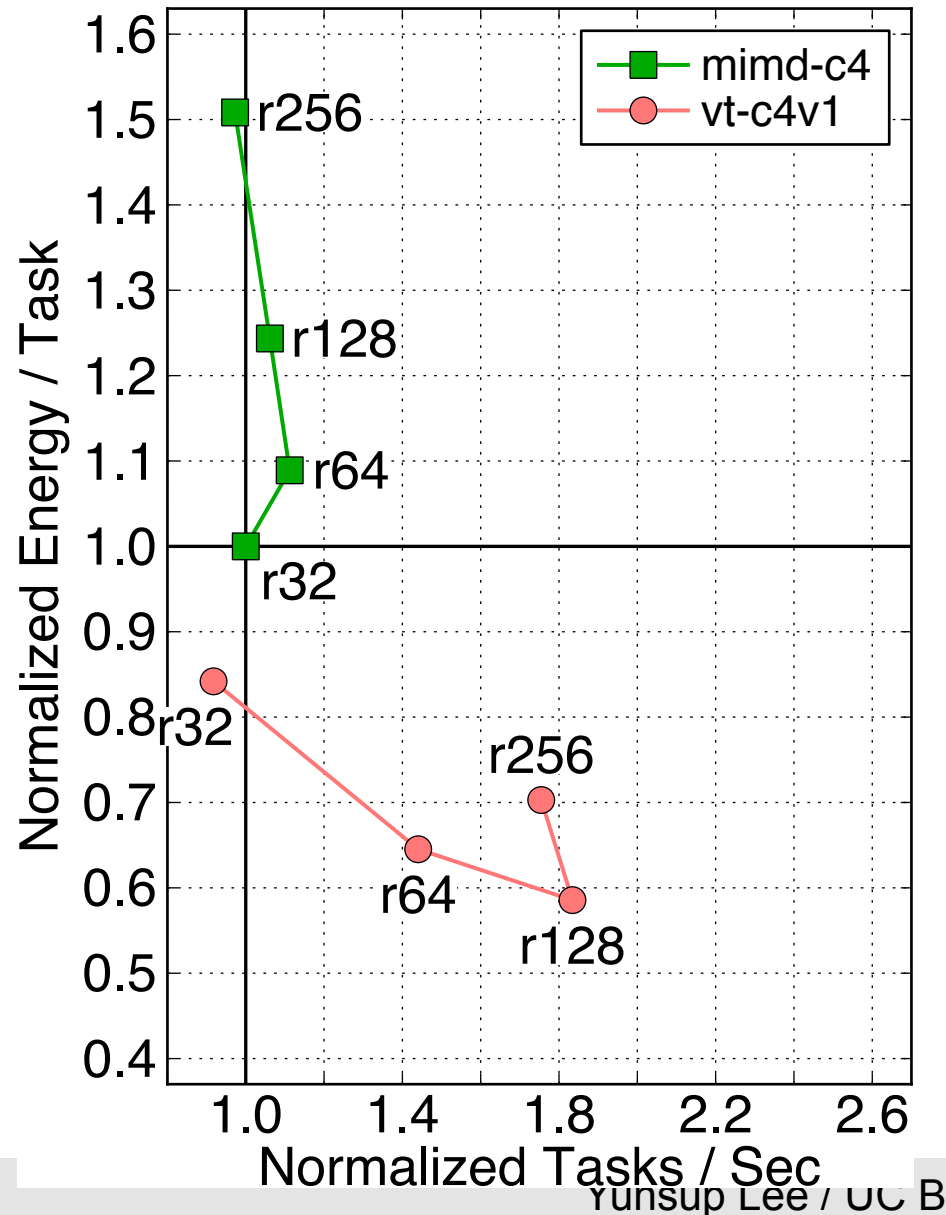
Efficiency vs. Number of uTs running *bsearch-cmv*



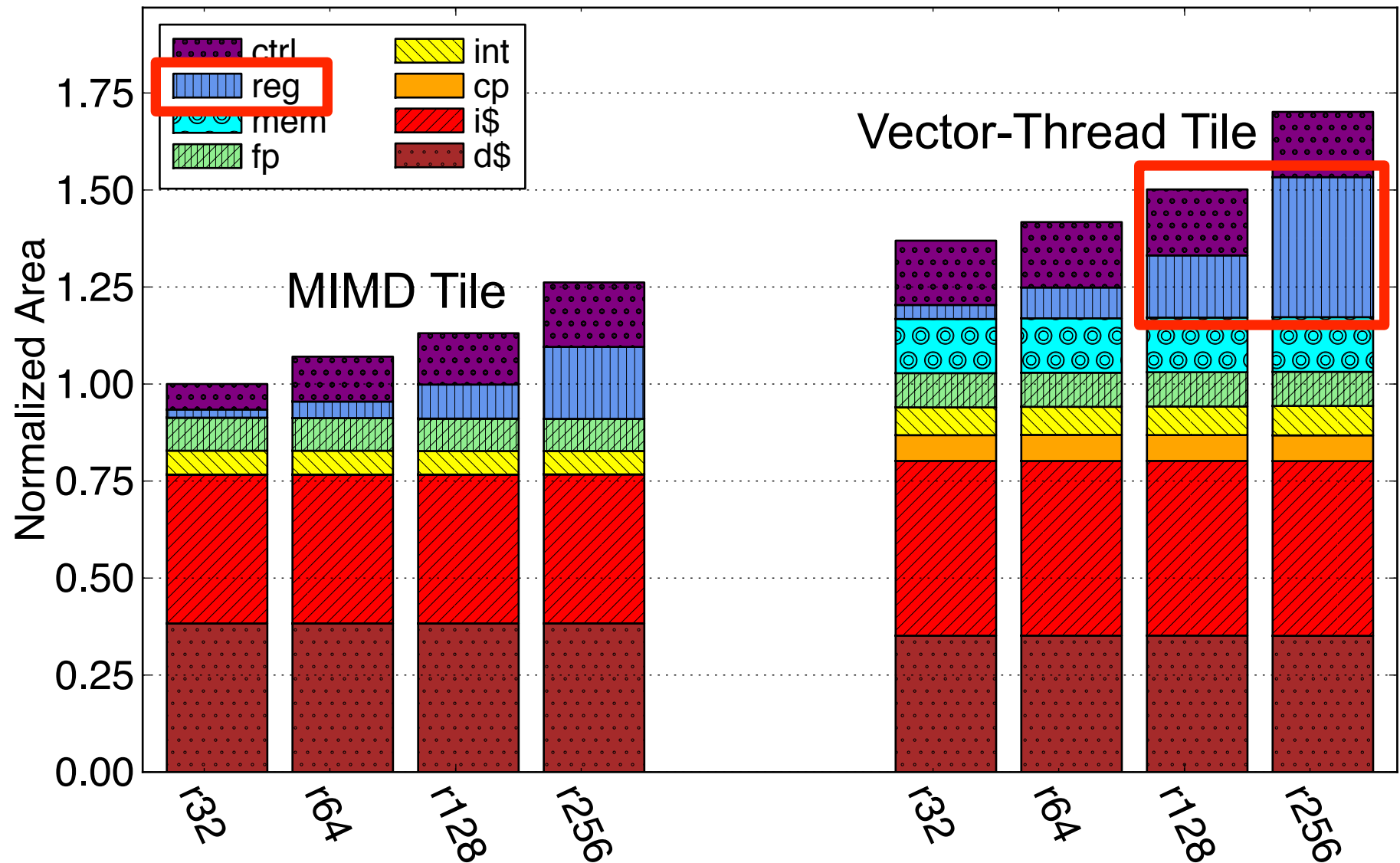
Efficiency vs. Number of uTs running *bsearch-cmv*



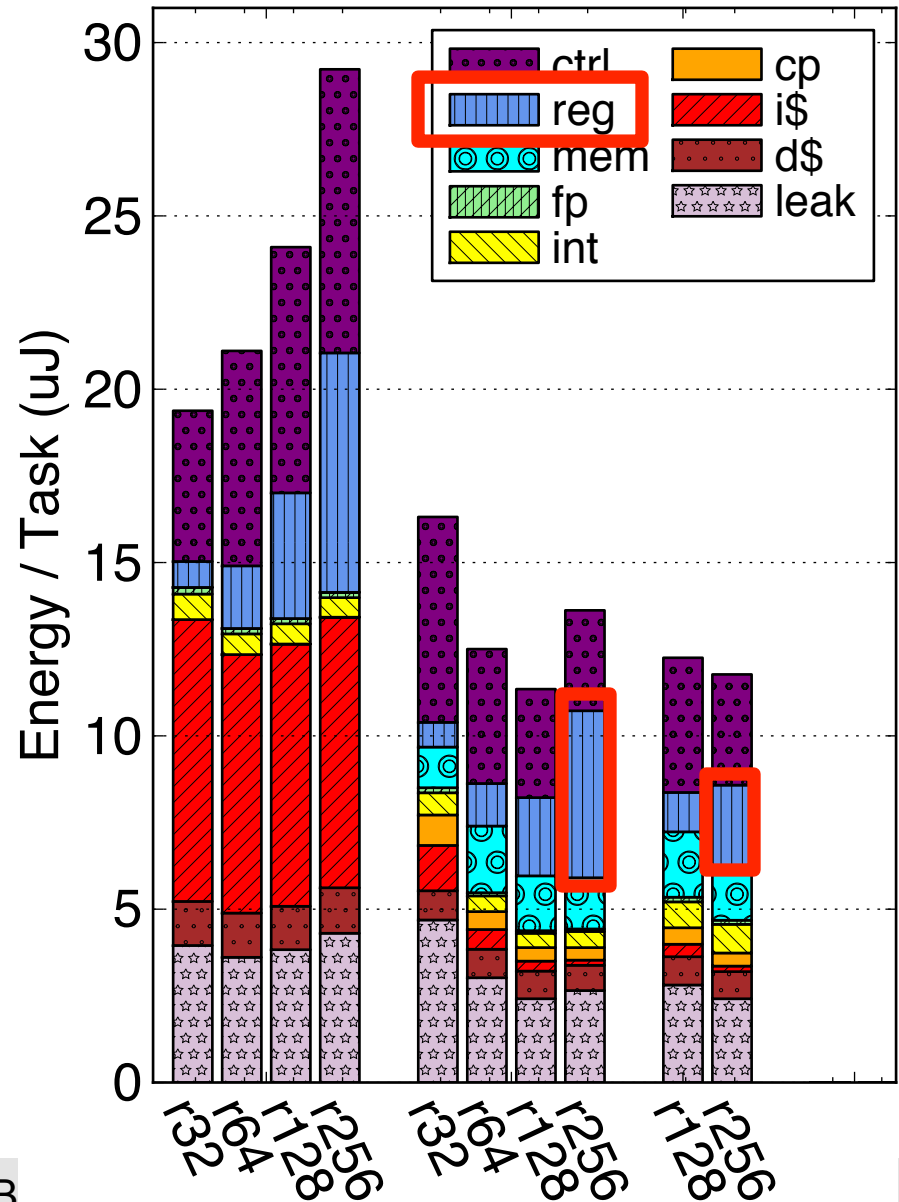
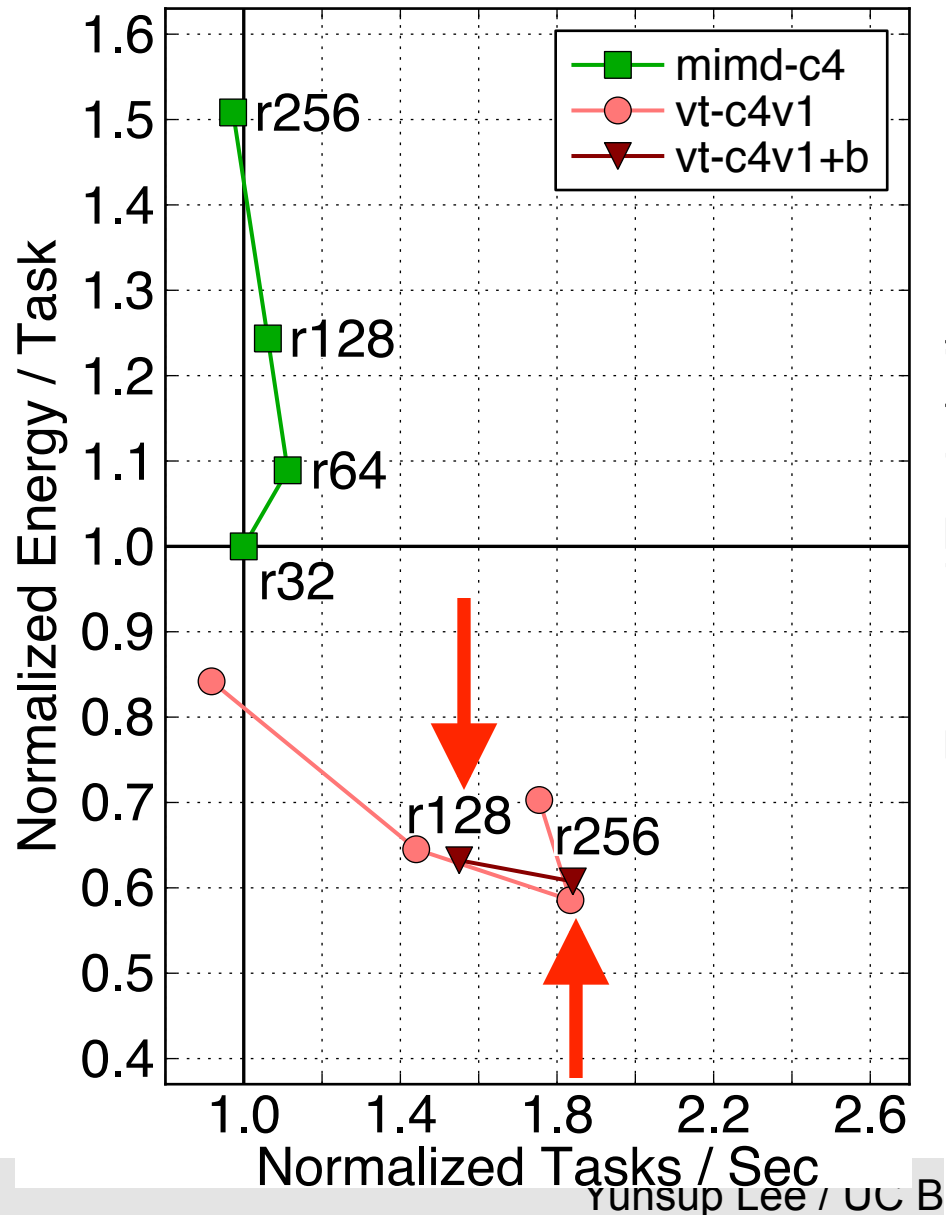
Efficiency vs. Number of uTs running *bsearch-cmv*



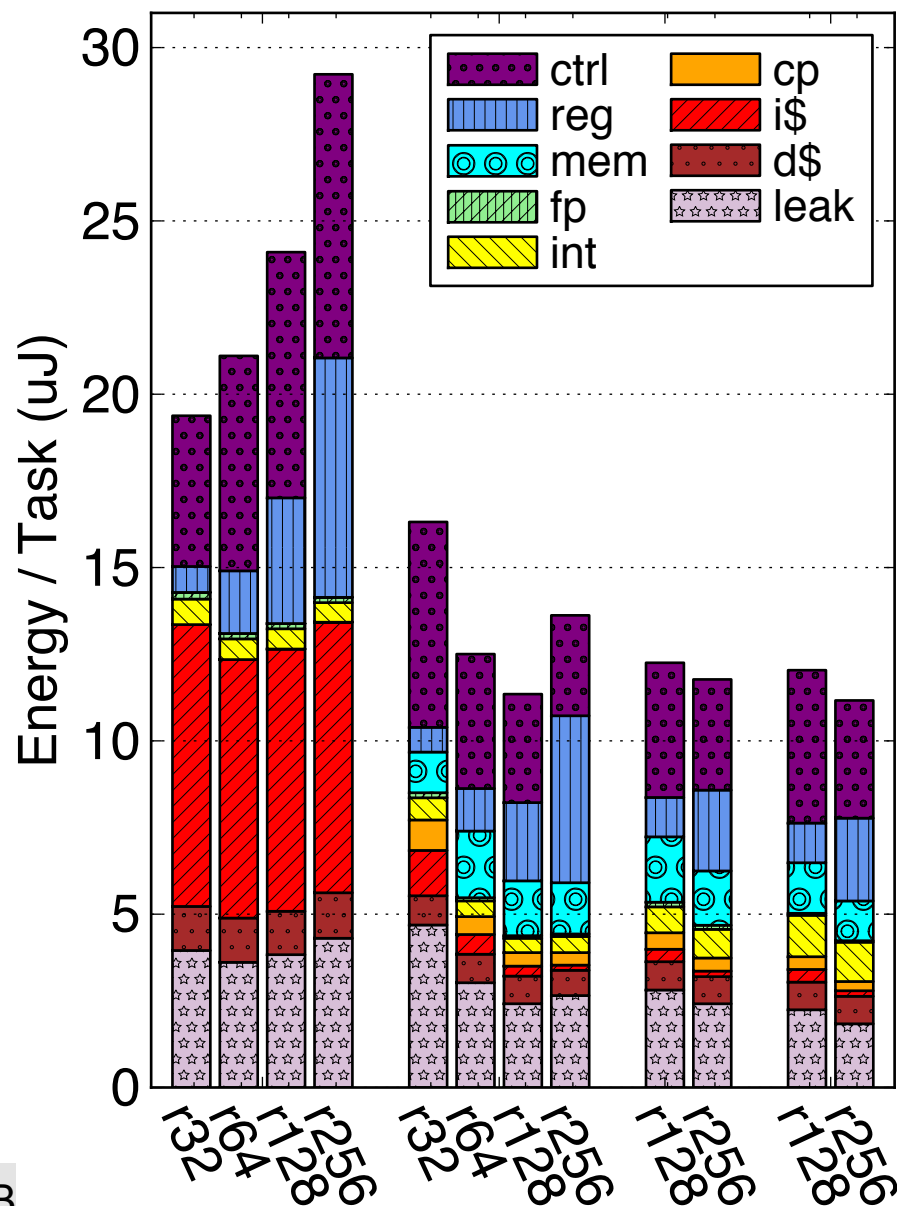
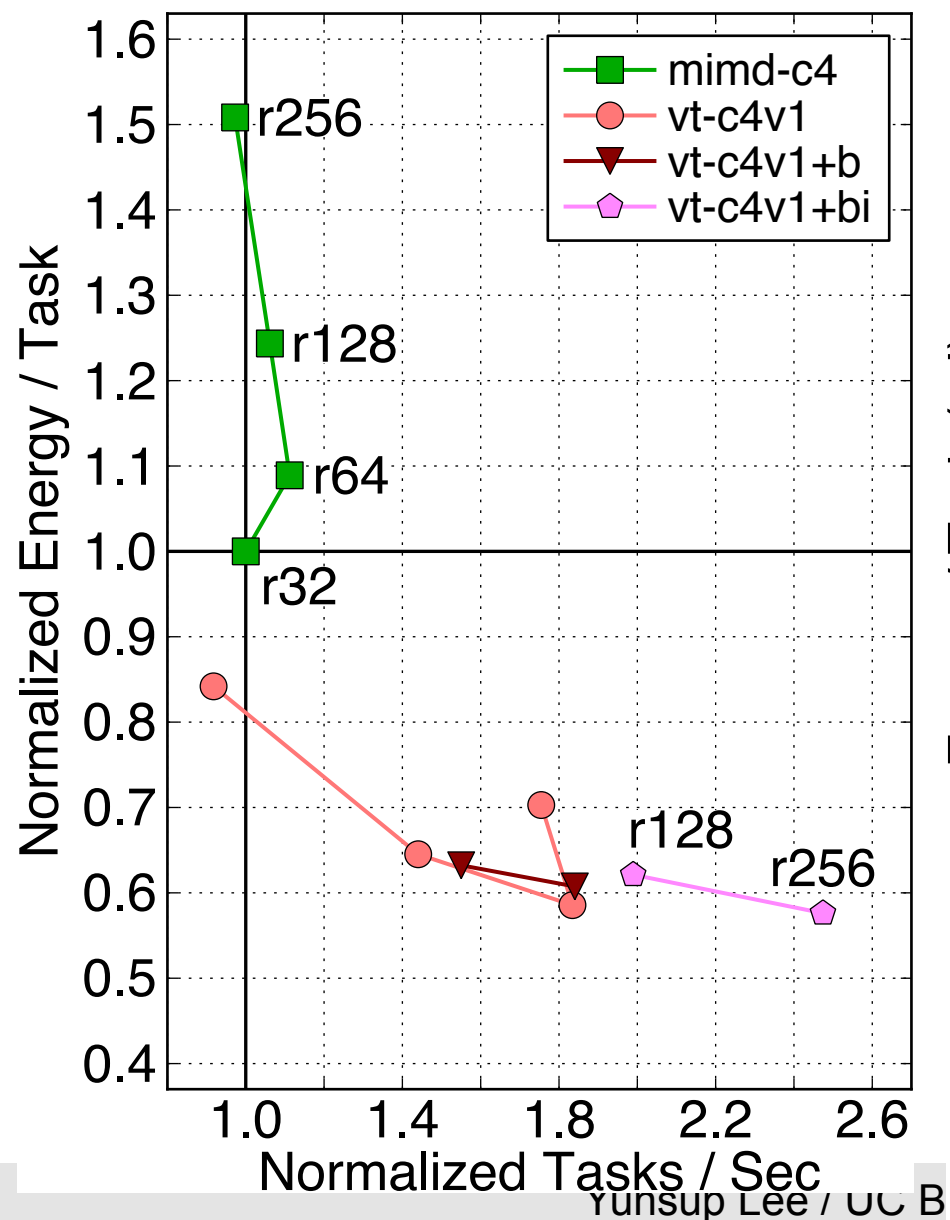
6r3w Vector Register File is Area Inefficient



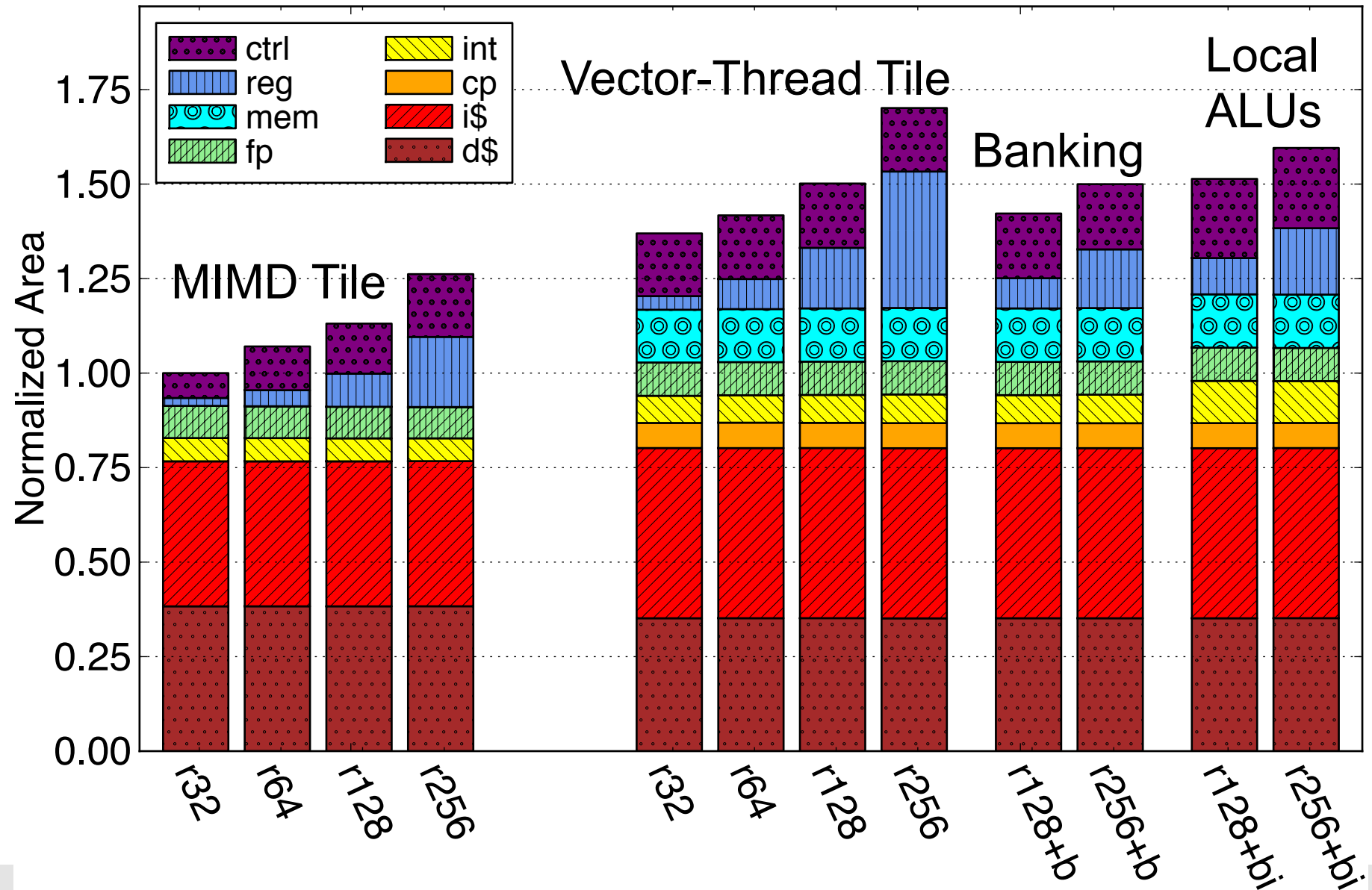
Efficiency vs. Number of uTs with Banking running *bsearch-cmv*



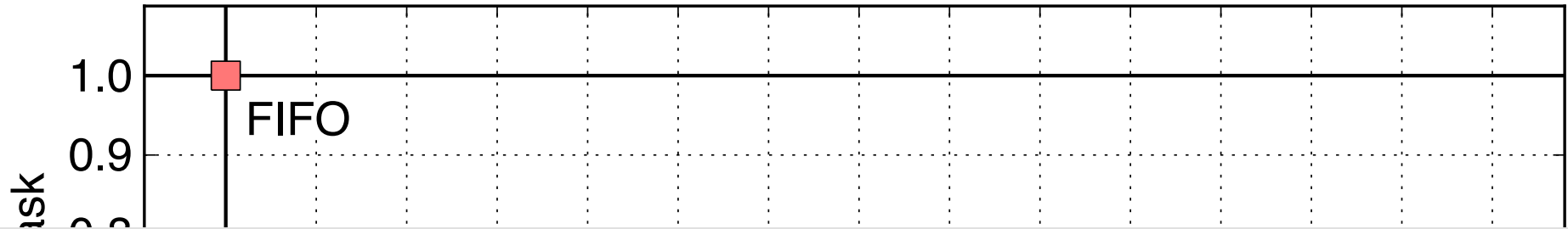
Efficiency vs. Number of uTs with Per-Bank Integer ALU running *bsearch-cmv*



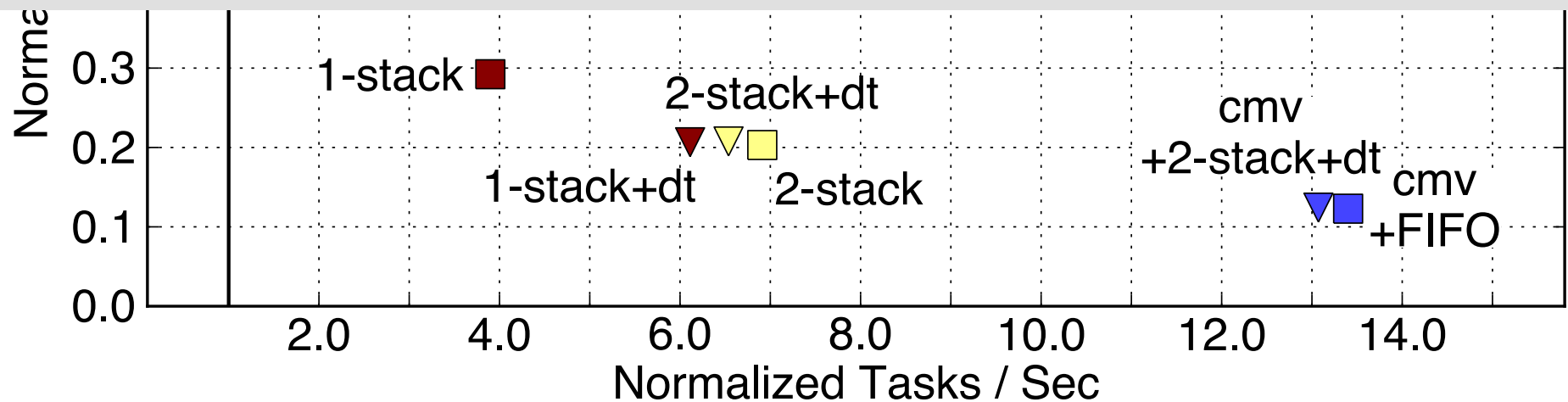
Bank Vector Register File Per-Bank Integer ALUs



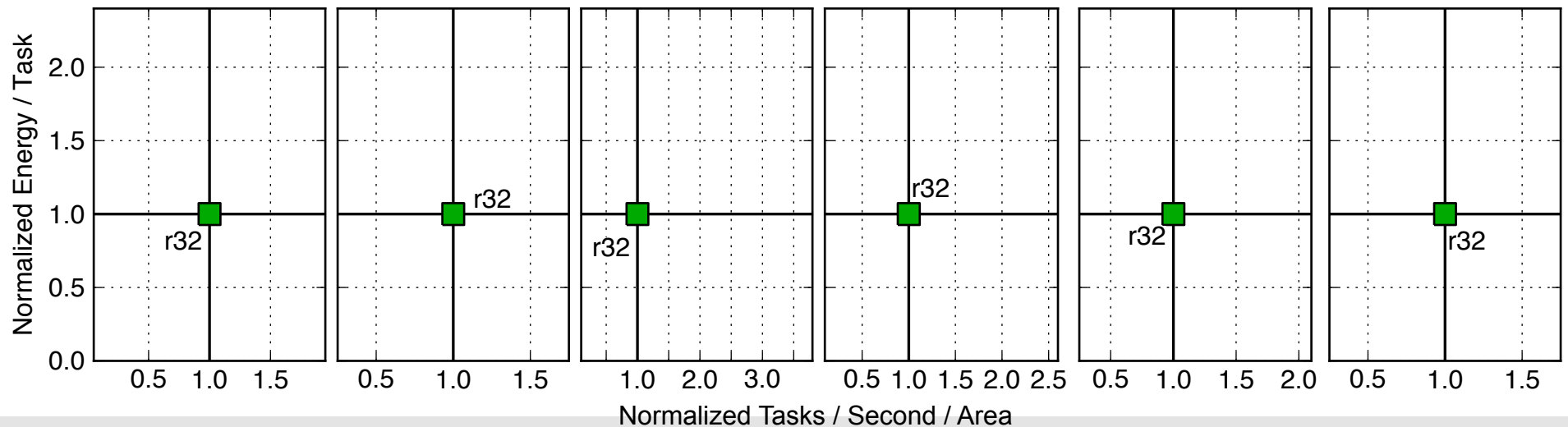
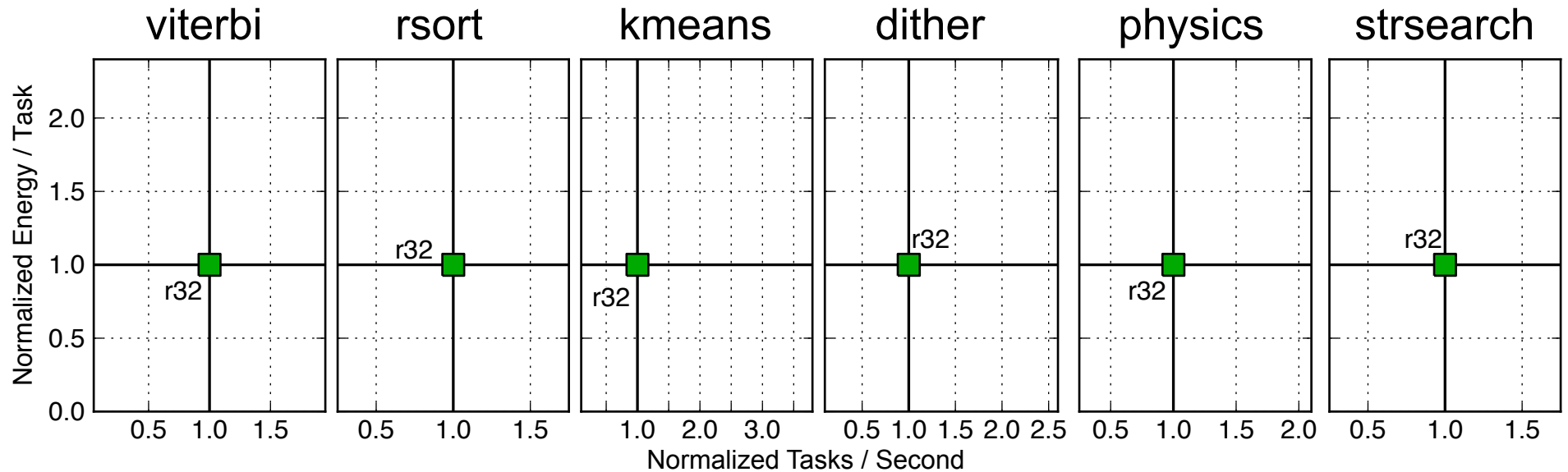
Results running *bsearch* compared to *bsearch-cmv*



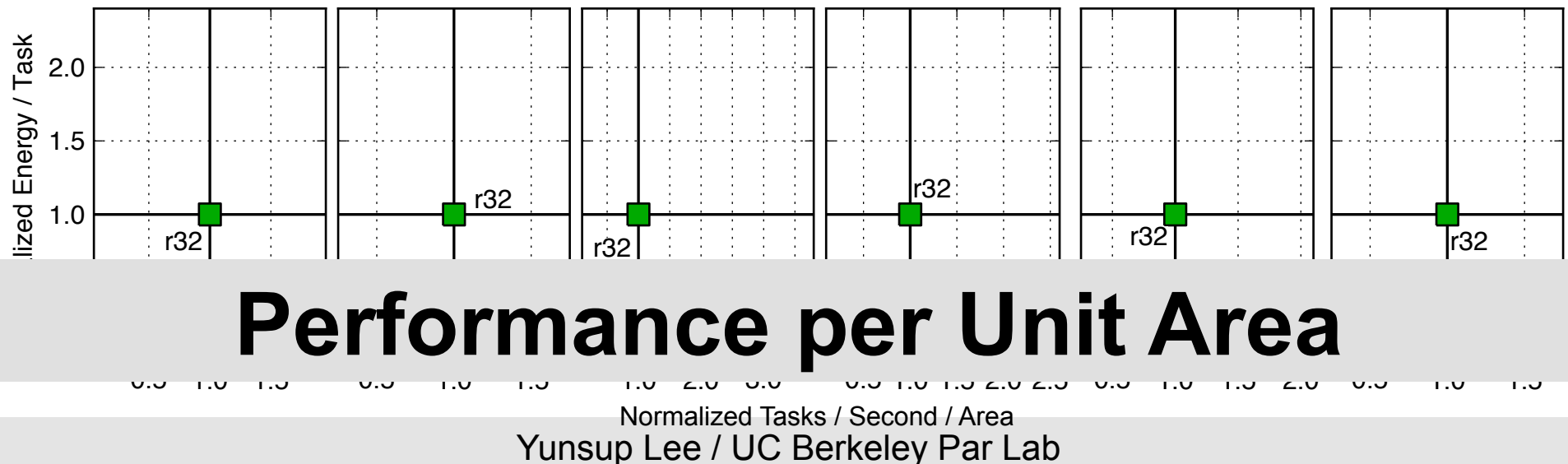
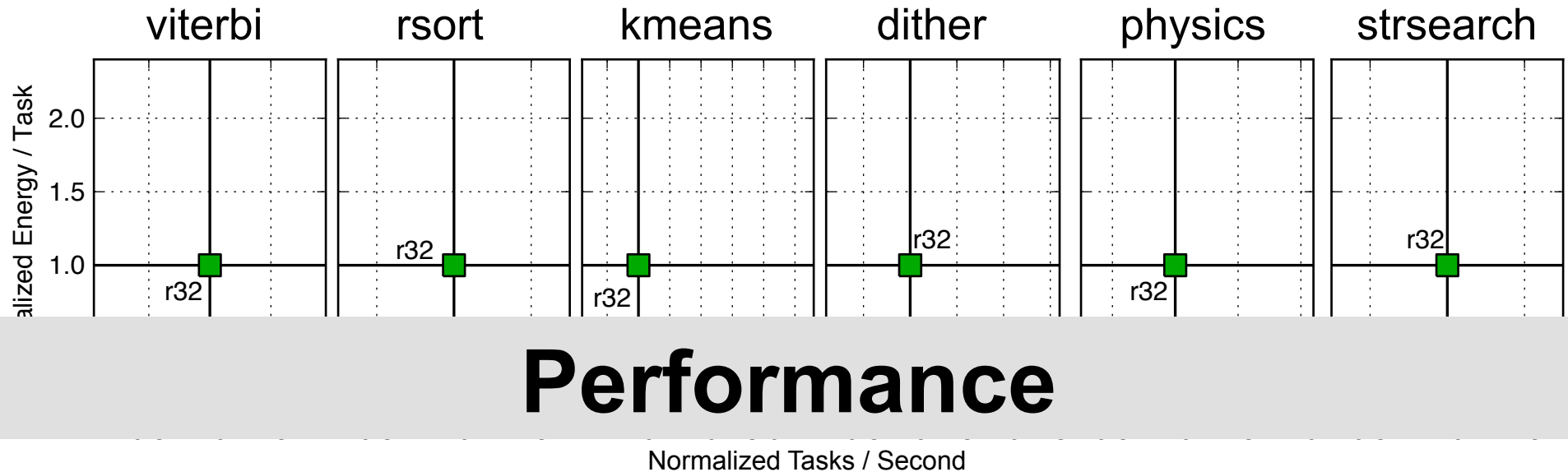
Results of Design Space Exploration Apply Density-Time Execution Convergence Scheme: 2-Stack PVFB



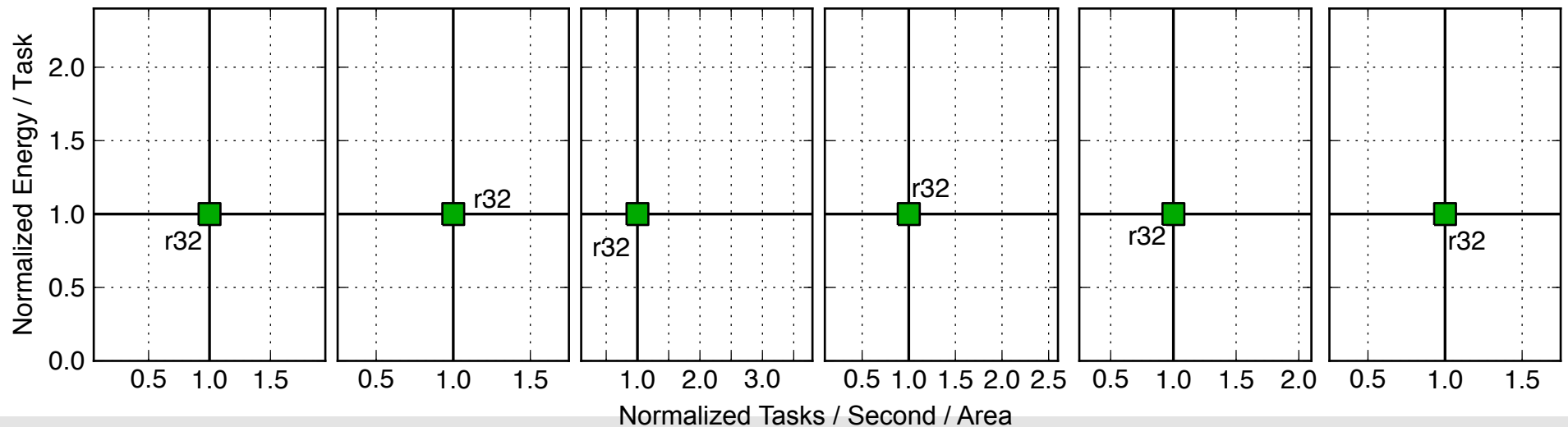
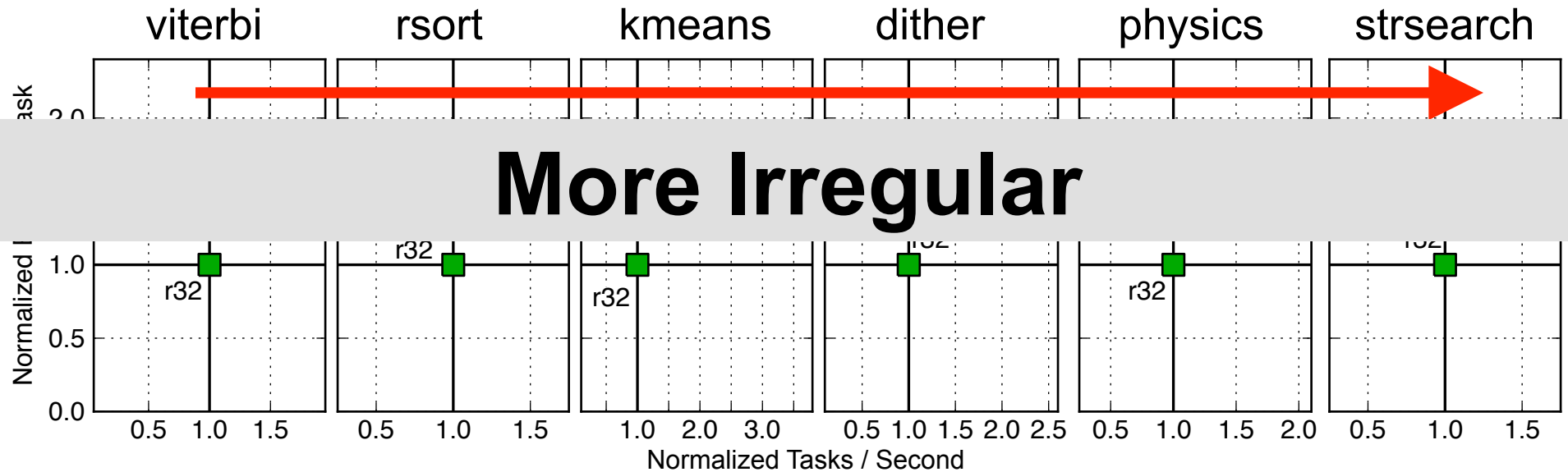
Results Running Application Kernels



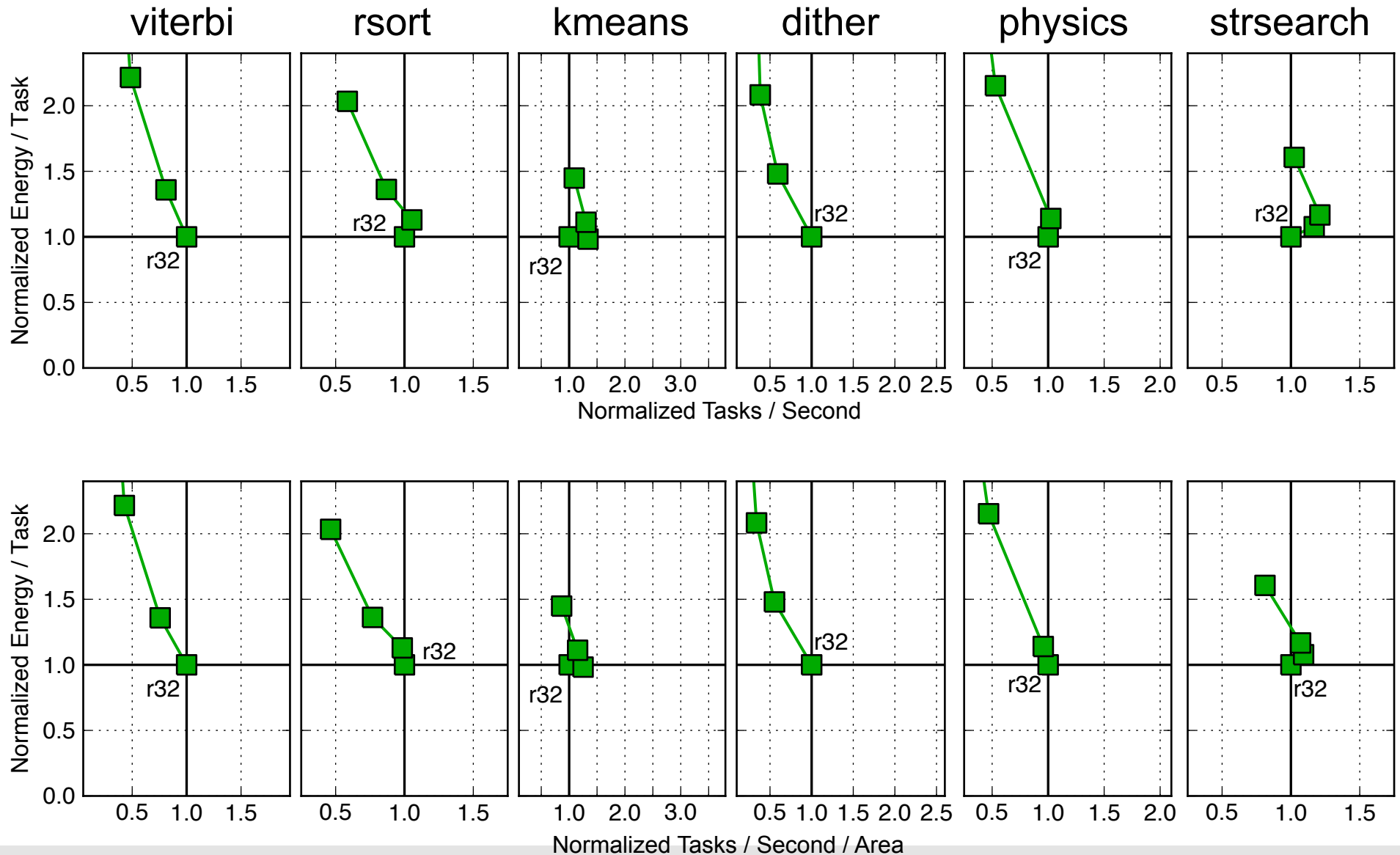
Results Running Application Kernels



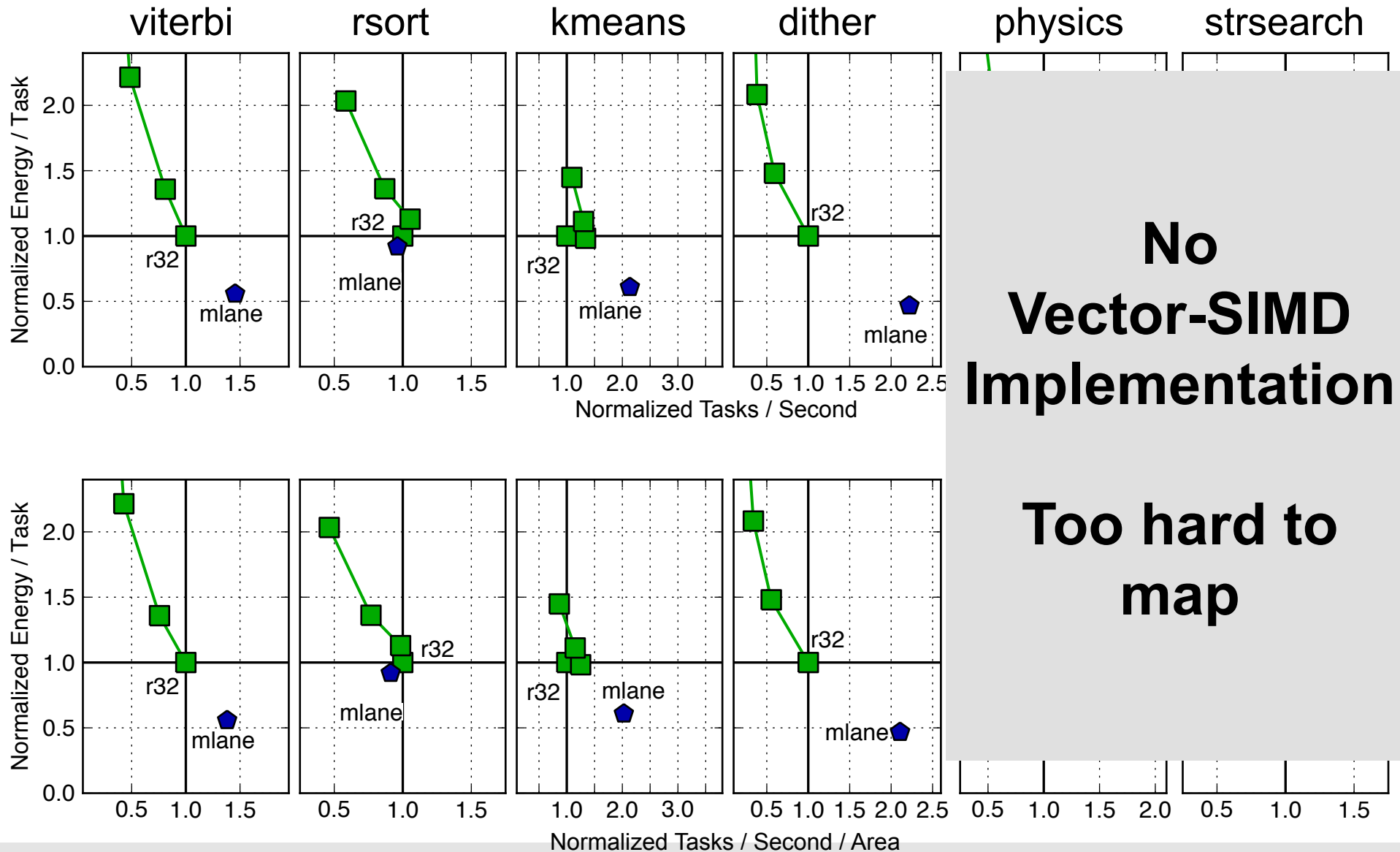
Results Running Application Kernels



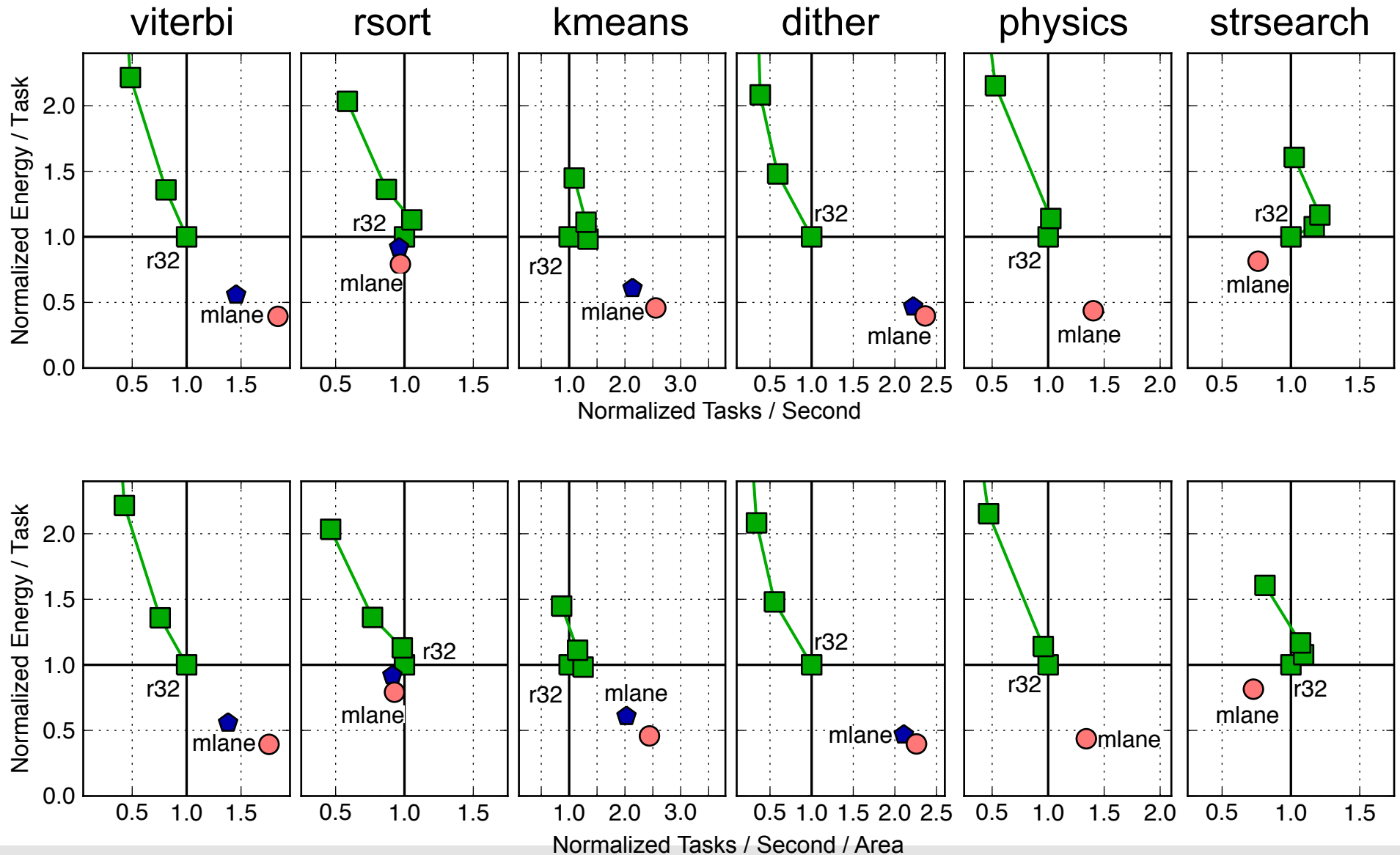
Multi-threading is not Effective on DLP Code



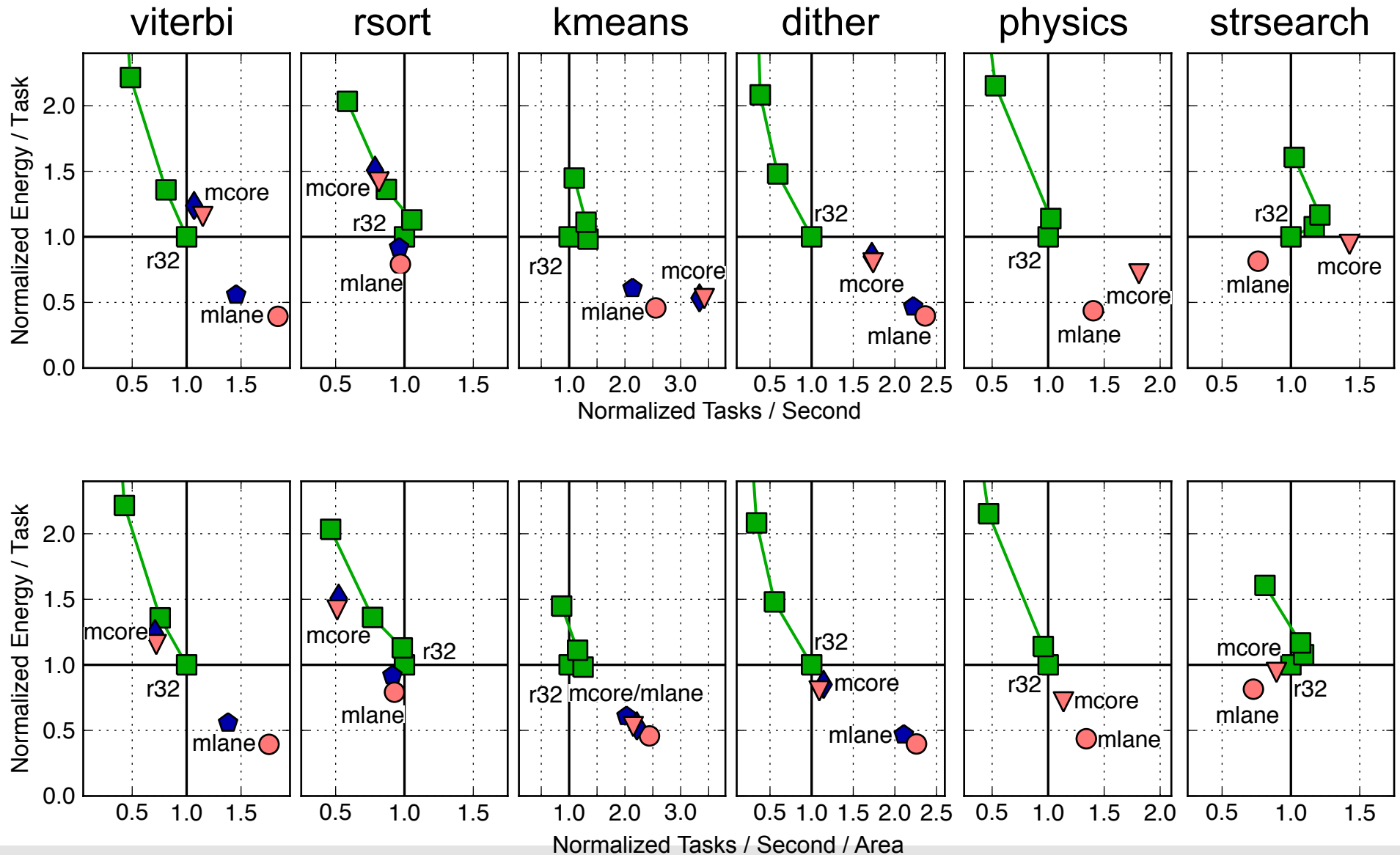
Vector-SIMD is Faster and/or More Efficient than MIMD



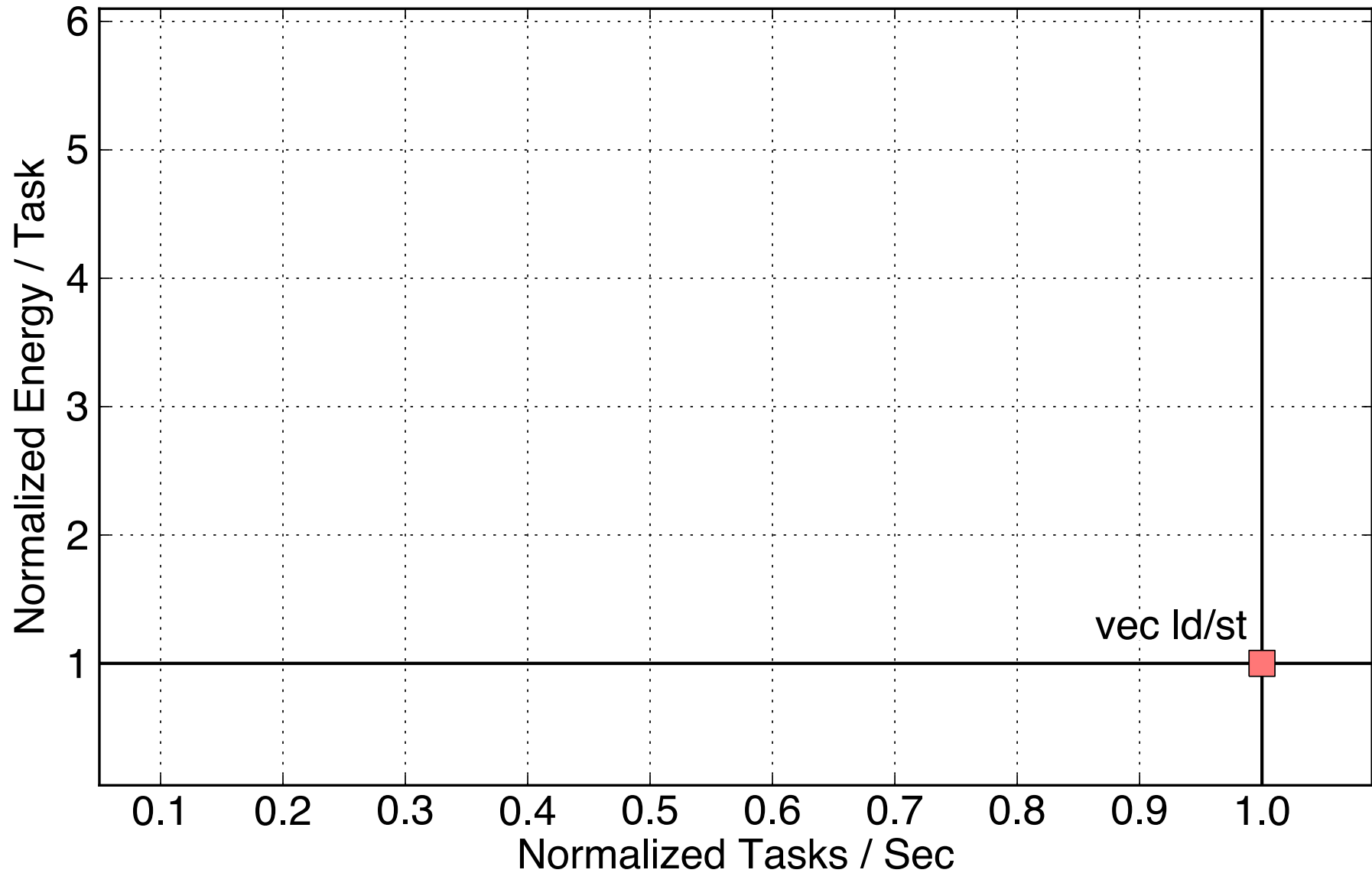
Maven Vector-Thread is More Efficient than Vector-SIMD



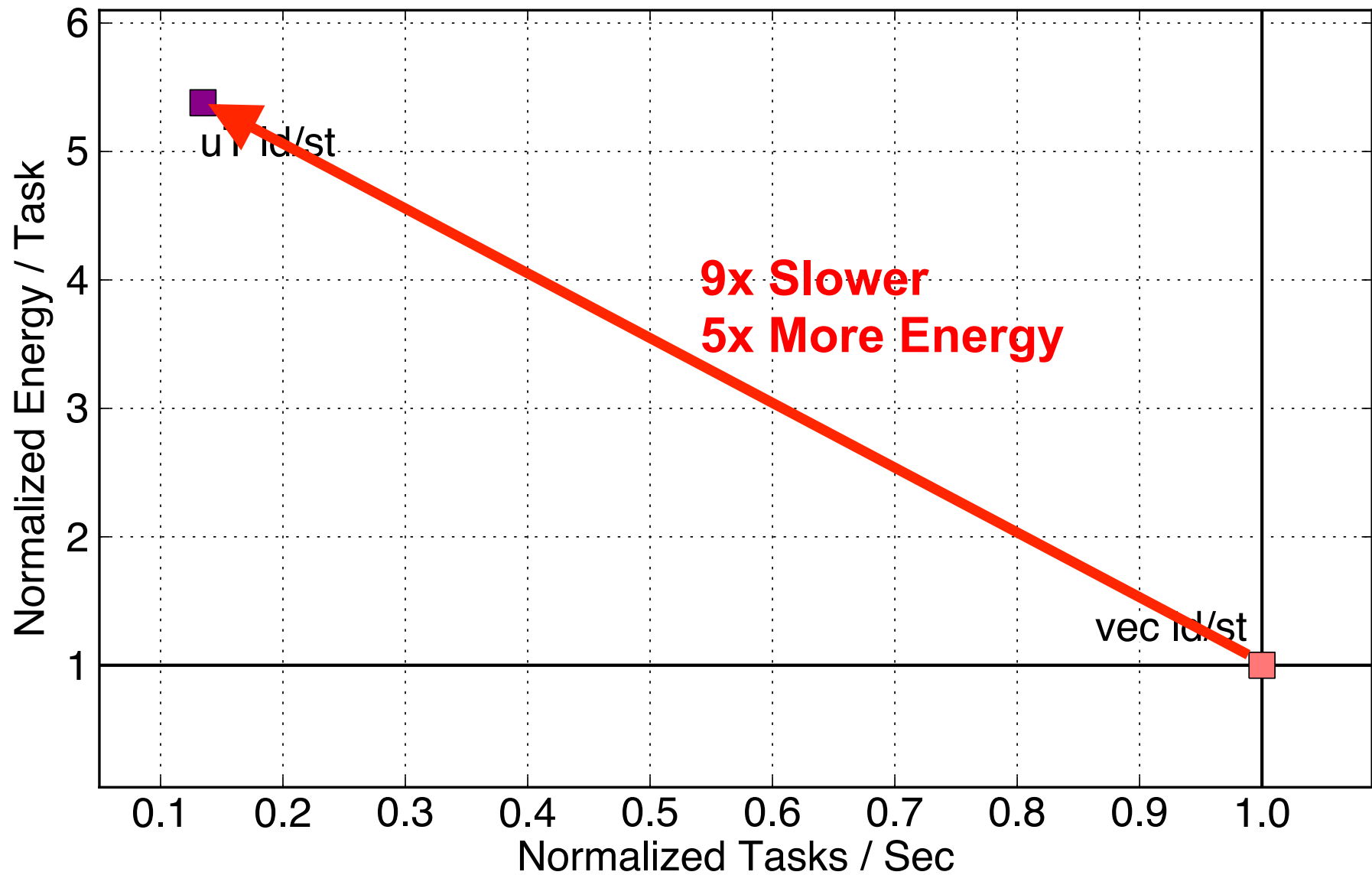
Multi-Lane Tiles are More Efficient than Multi-Core Tiles



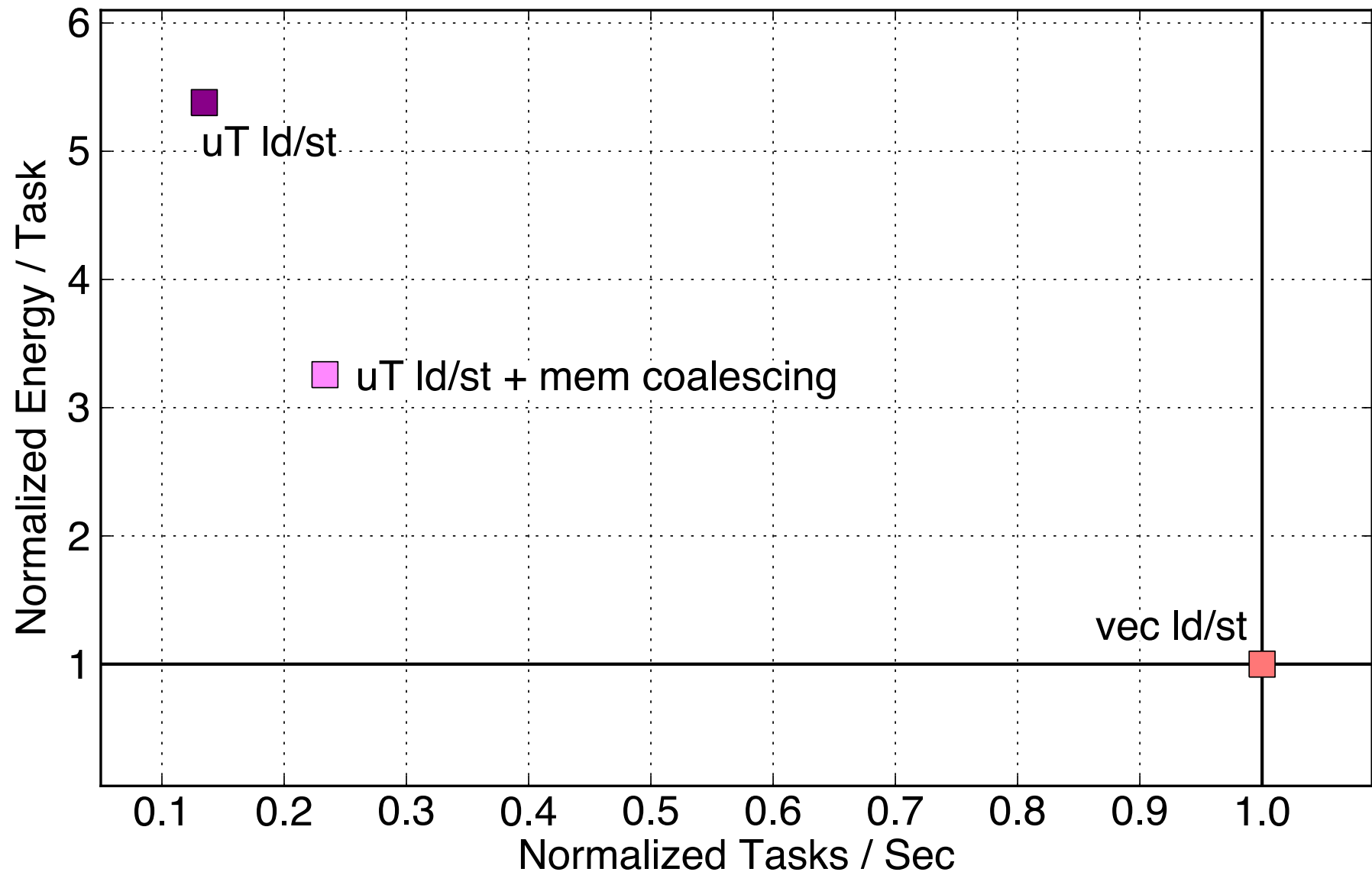
Comparing vector load/stores vs. uT load/stores running *vvadd*



uT load/stores are Inefficient



Memory Coalescing Helps, but Still Far Off



Conclusions

- Vector architectures are more area and energy efficient than MIMD architectures on regular DLP and (surprisingly) on irregular DLP
- The Maven vector-thread architecture is a promising alternative to traditional vector-SIMD architectures, providing greater efficiency and easier programmability
- Using real RTL implementations and a standard ASIC toolflow is necessary to compare energy-optimized future architectures

This work was supported in part by Microsoft (Award #024263) and Intel (Award #024894, equipment donations) funding and by matching funding from U.C. Discovery (Award #DIG07-10227)