Manycore Vector-Thread Architectures

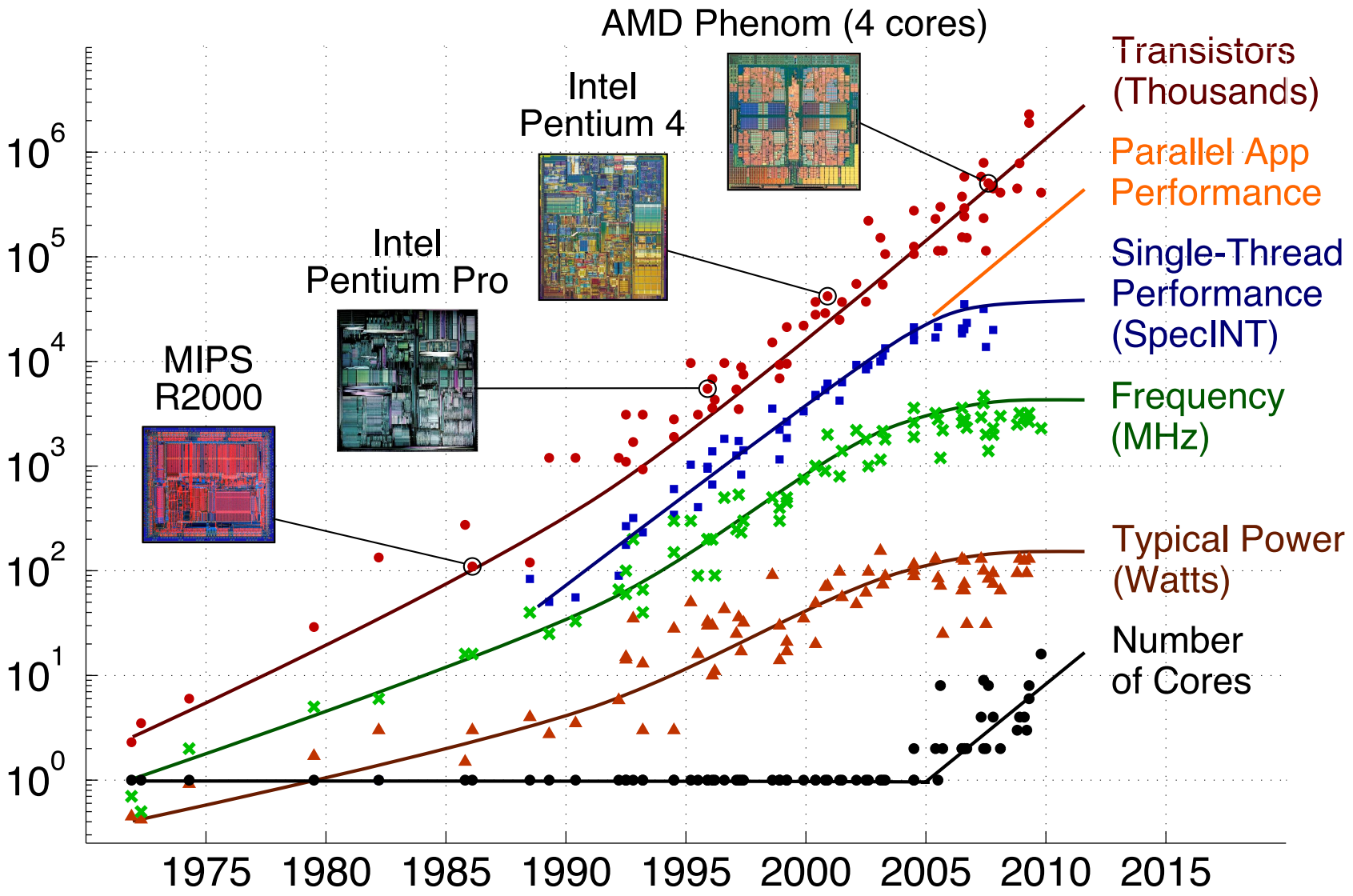
Christopher Batten

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Parallel Computing Laboratory
University of California, Berkeley

Spring 2009

The Transition to Multicore



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

Power Constrains All Modern Systems

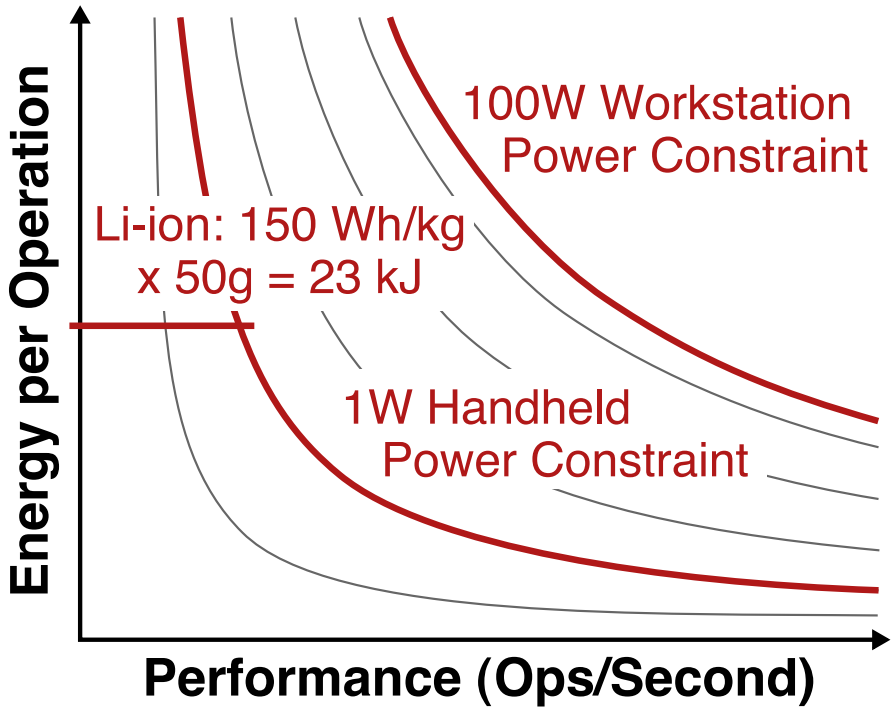
$$\text{Power} = \frac{\text{Energy}}{\text{Second}} = \frac{\text{Energy}}{\text{Op}} \times \frac{\text{Ops}}{\text{Second}}$$

Power

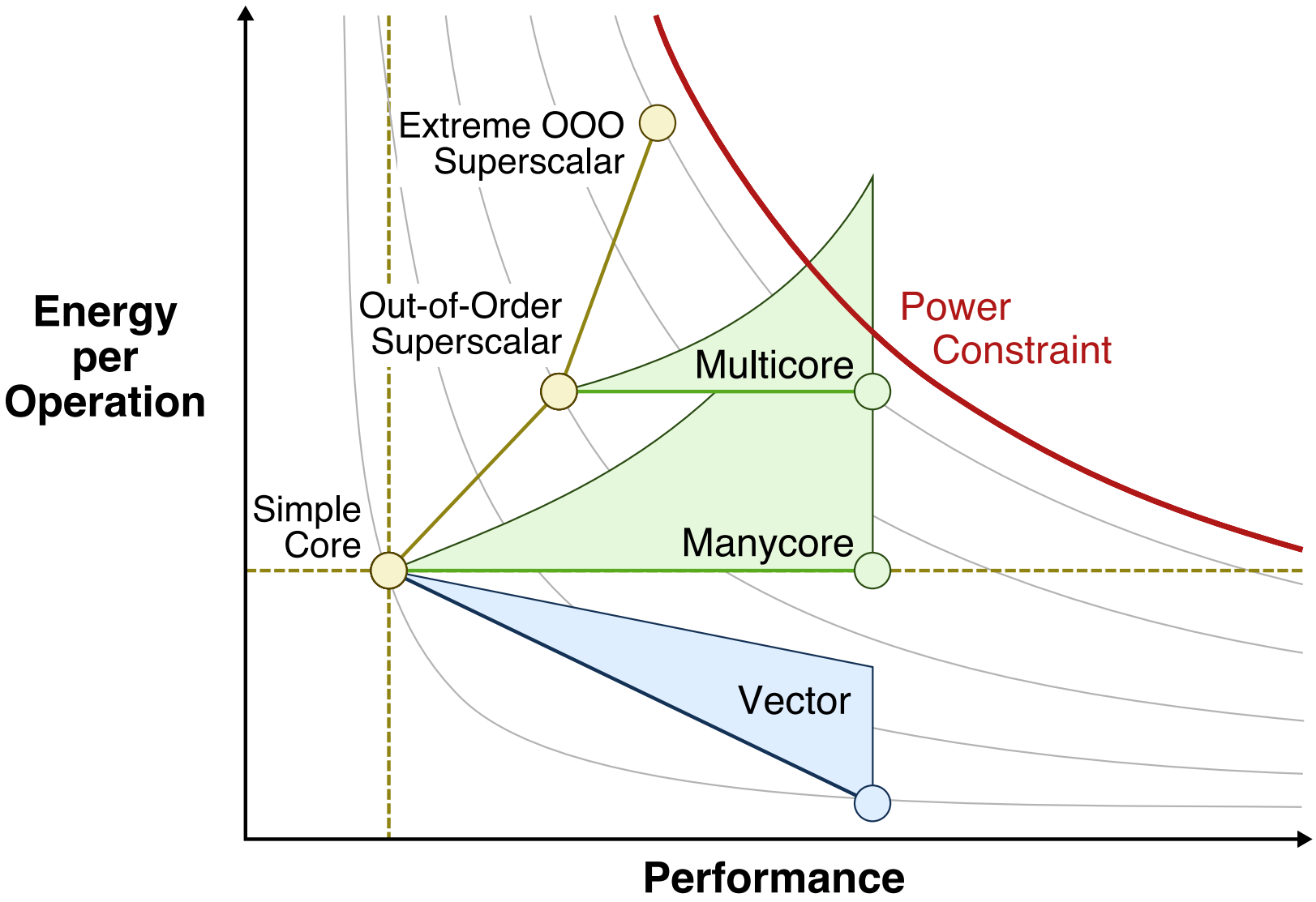
- Chip Packaging
- Chip Cooling
- System Noise
- Case Temperature
- Data-Center Air Conditioning

Energy

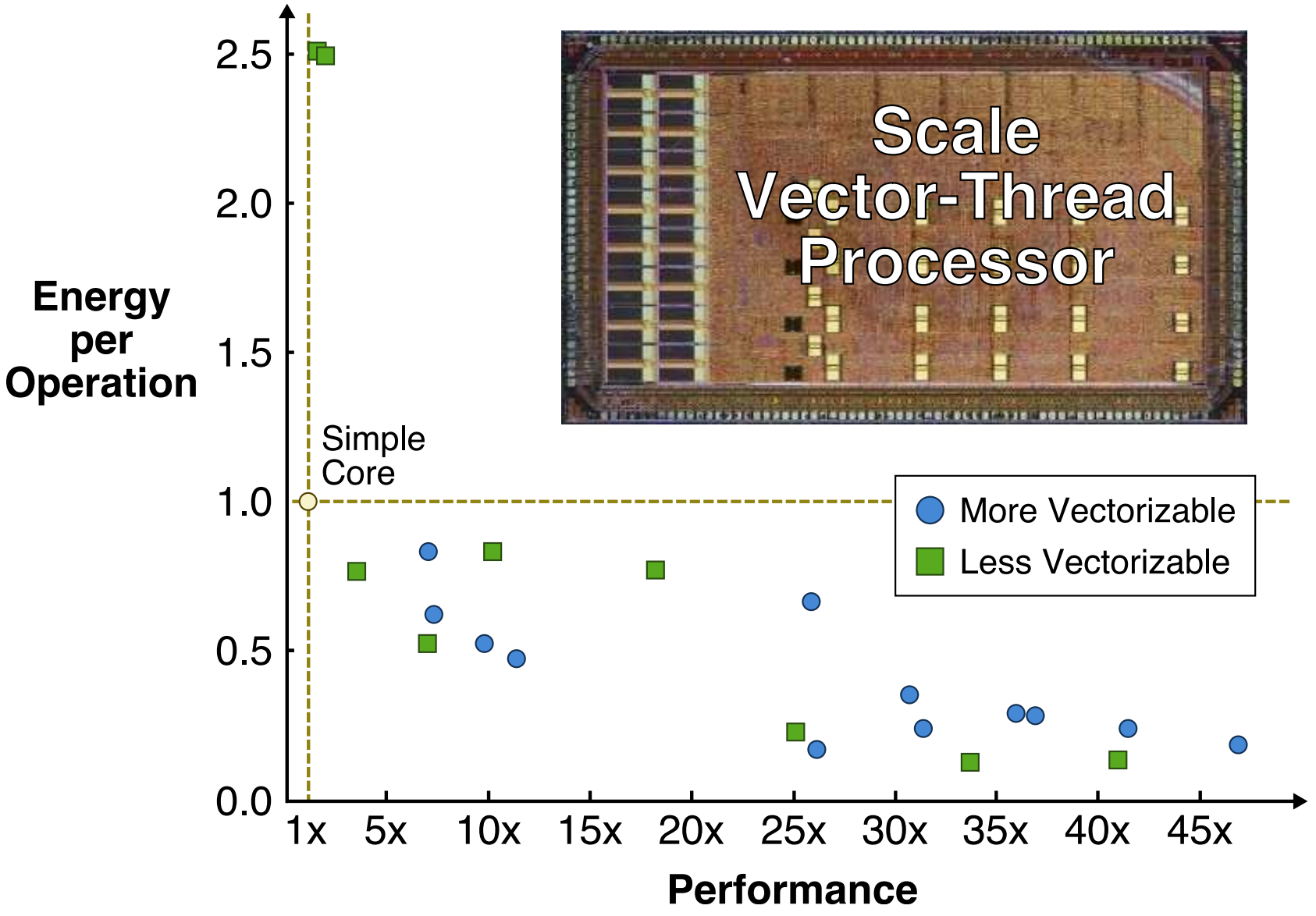
- Battery Life
- Electricity Bill
- Mobile Device Weight

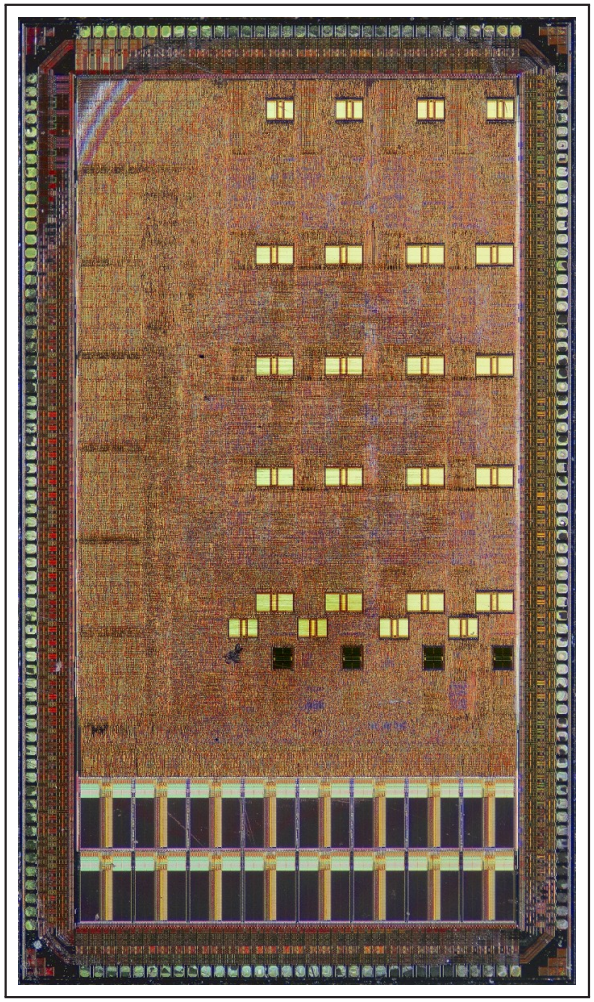


How does multicore help reduce power?



Vector-Threading Is Energy-Efficient *and* Flexible





Motivation

Vector-Thread Architectures

Scale VT Processor

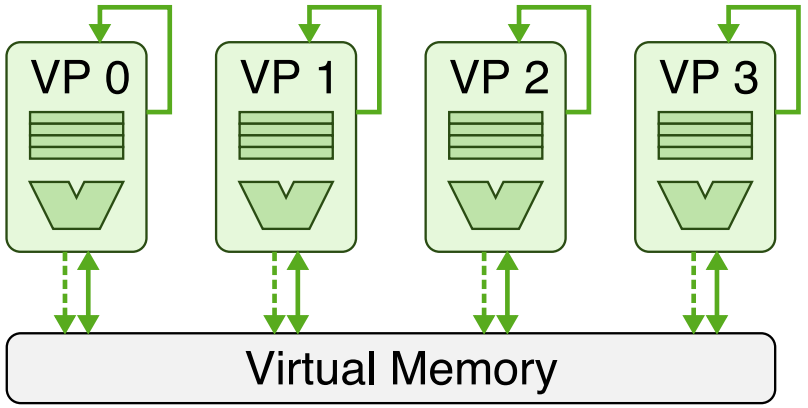
Maven VT Processor

Future Research Directions

Multithreaded Architectures

Assembly Programmer's View

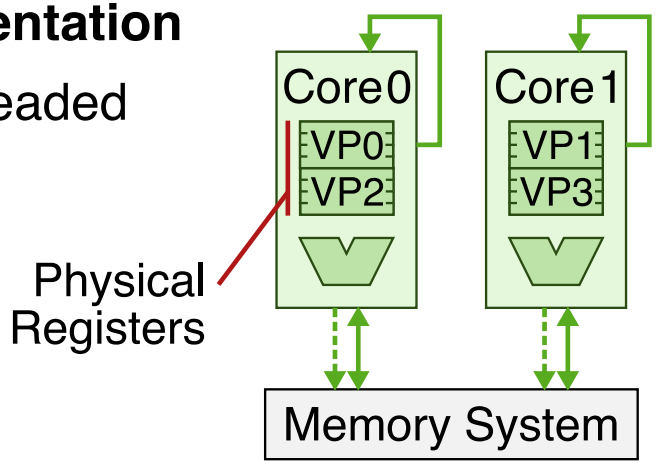
Thread Contexts
or
Virtual Processors

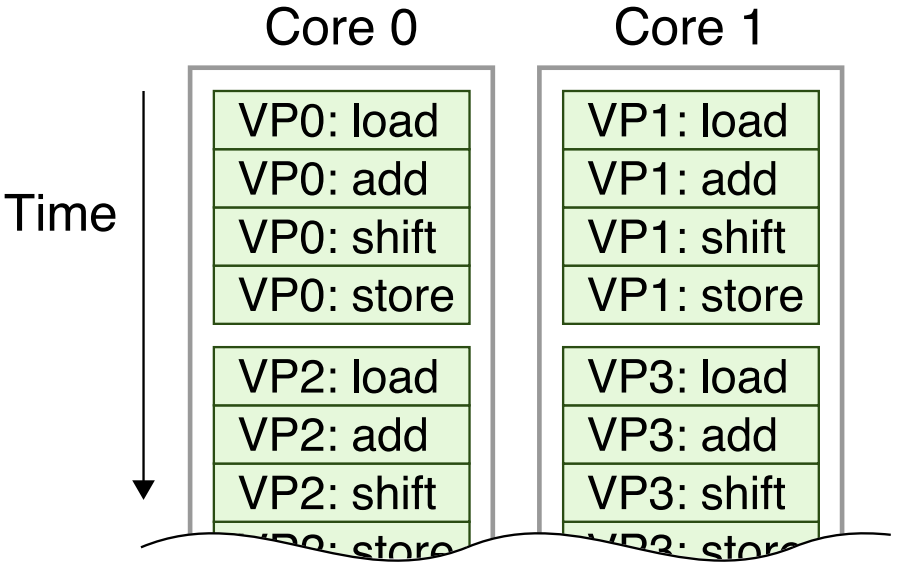



```
loop:
  load
  add
  shift
  store
  branch
```

Implementation

Multithreaded Cores

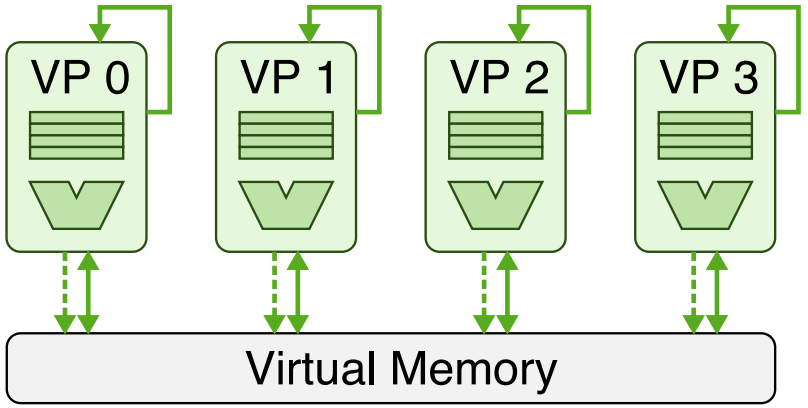




Multithreaded Architectures

Assembly Programmer's View

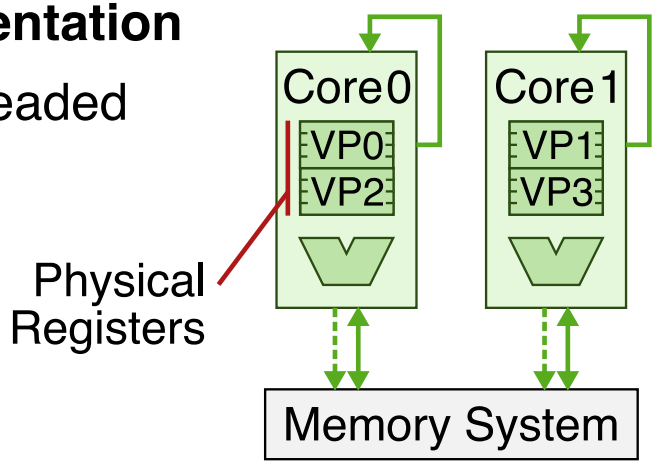
Thread Contexts
or
Virtual Processors

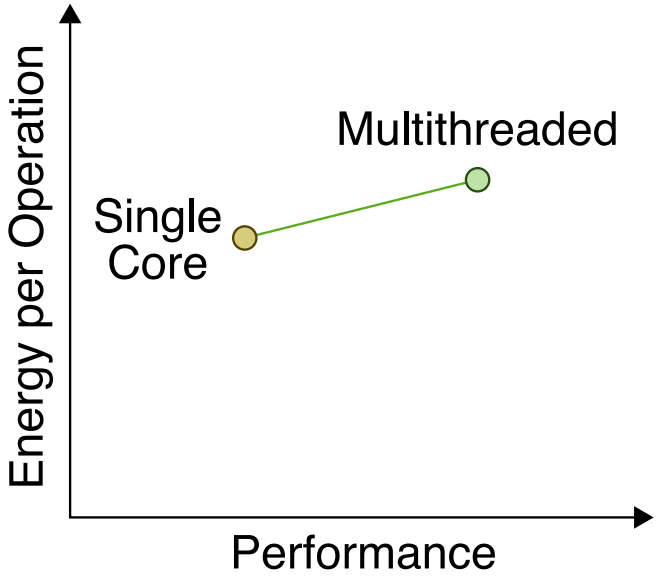



```
loop:
  load
  add
  shift
  store
  branch
```

Implementation

Multithreaded Cores



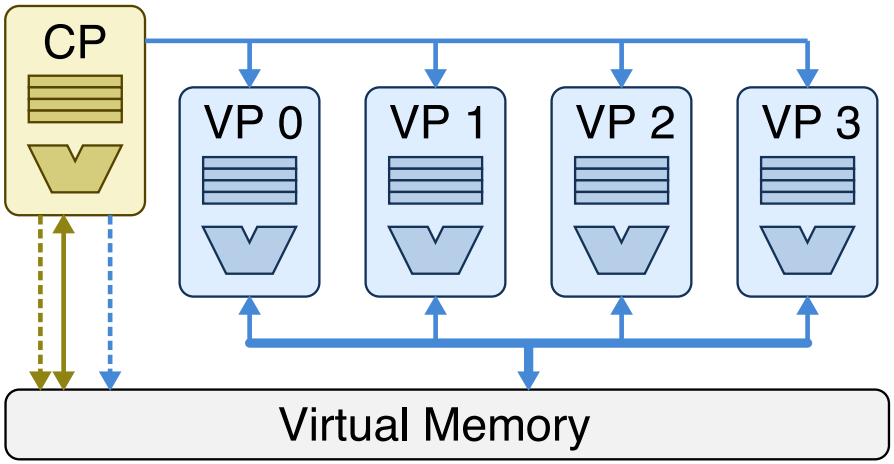


Vector Architectures

Assembly Programmer's View

Control Processor

Vector of Virtual Processors

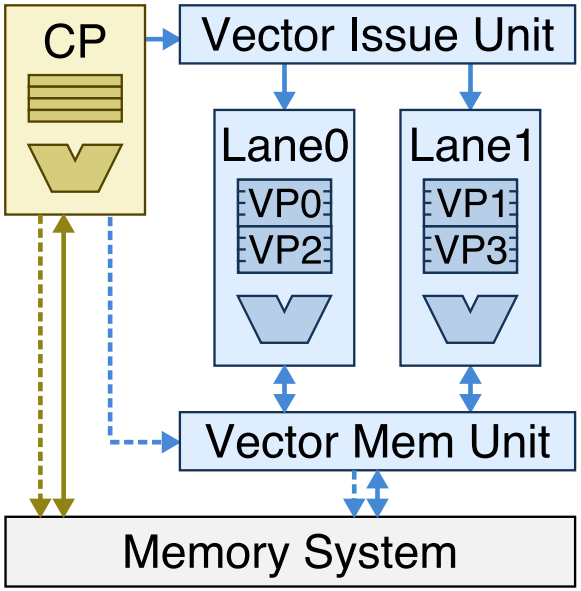


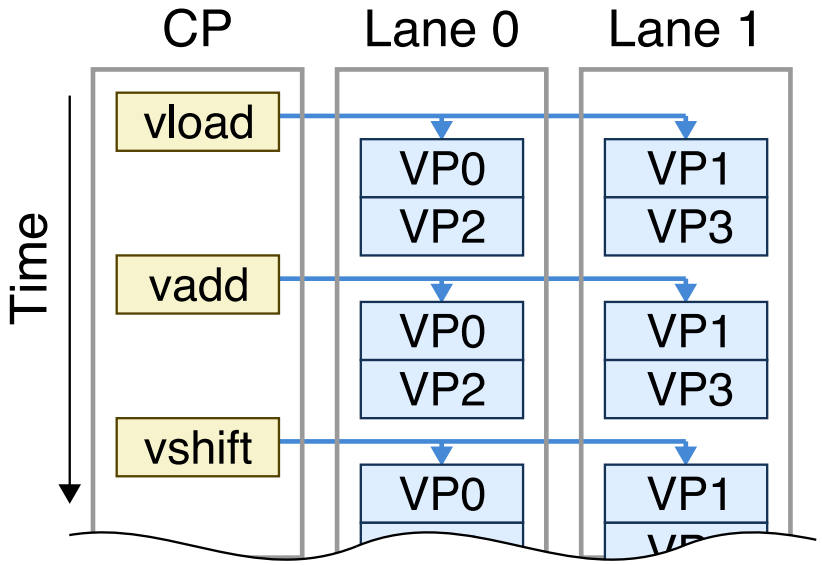

```
loop:
  vload
  vadd
  vshift
  vstore
  branch
```

Implementation

Control Processor

Vector Lanes



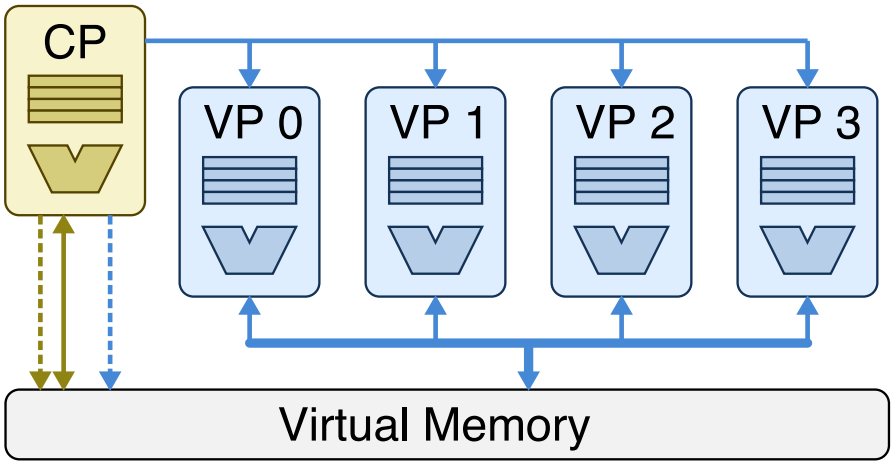


Vector Architectures

Assembly Programmer's View

Control Processor

Vector of Virtual Processors

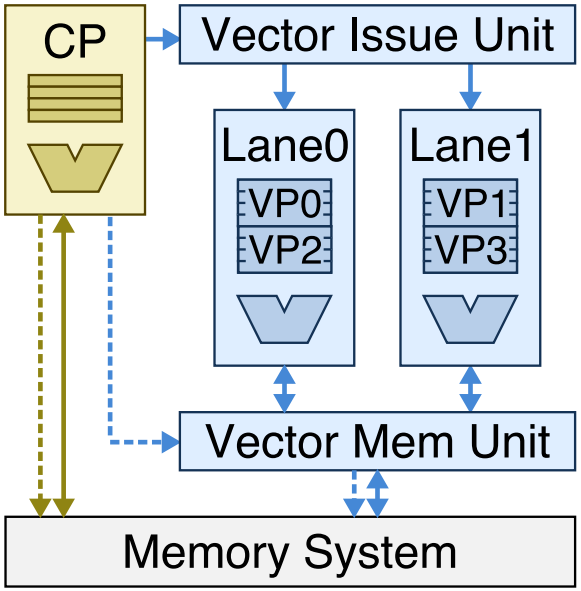


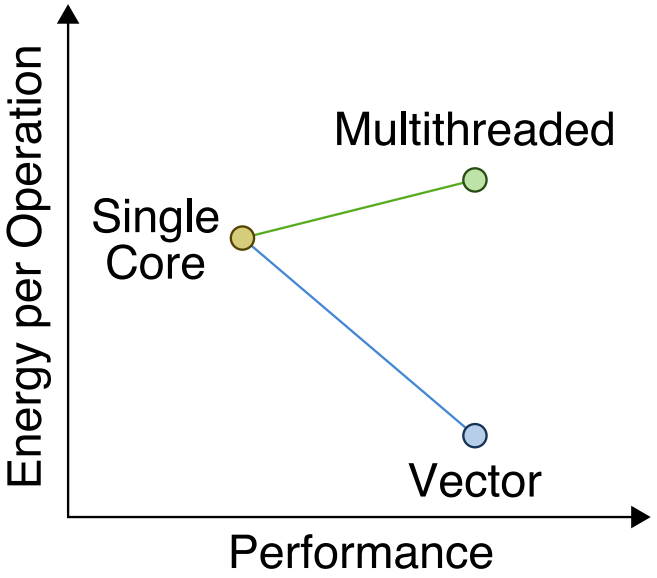

```
loop:
  vload
  vadd
  vshift
  vstore
  branch
```

Implementation

Control Processor

Vector Lanes



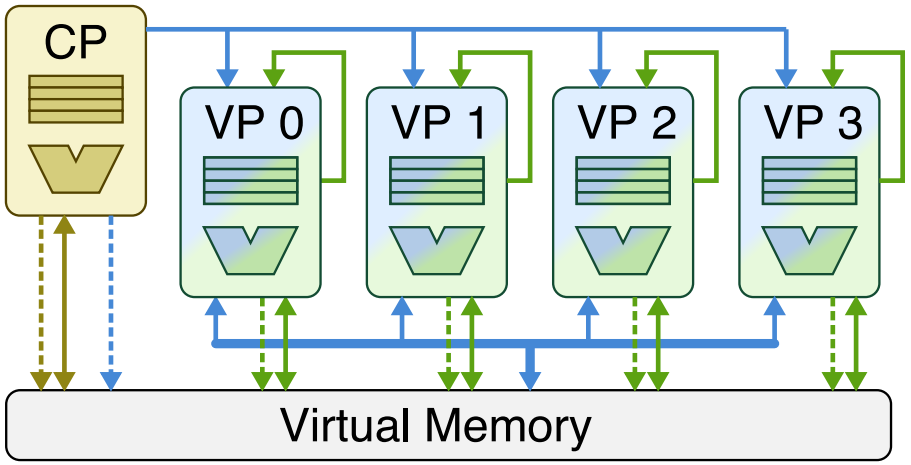


Vector-Thread Architectures

Assembly Programmer's View

Control Processor

Vector of Virtual Processors

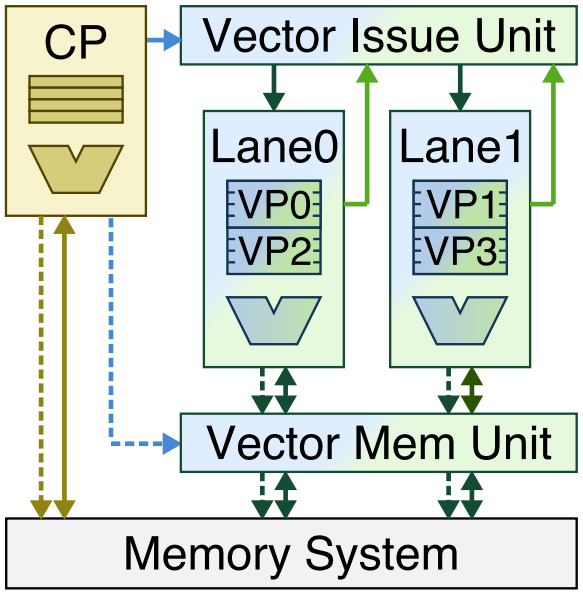


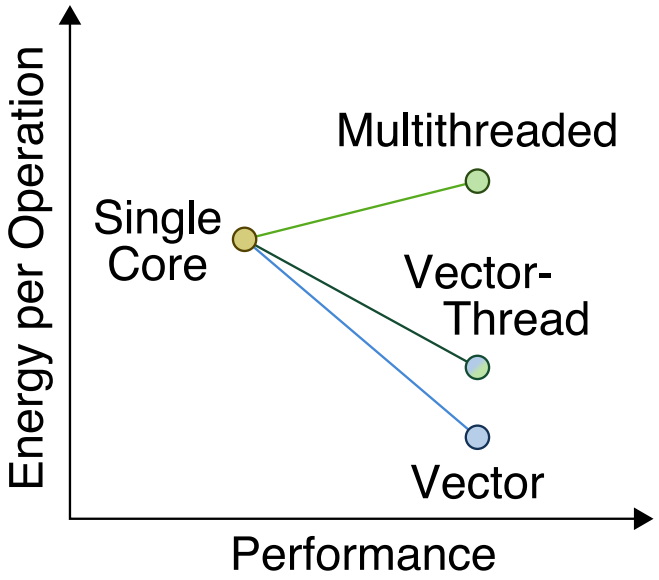

```
loop:
  vload
  vfetch foo
  vstore
  branch
foo:
  add
  shift
  vp.stop
```

Implementation

Control Processor

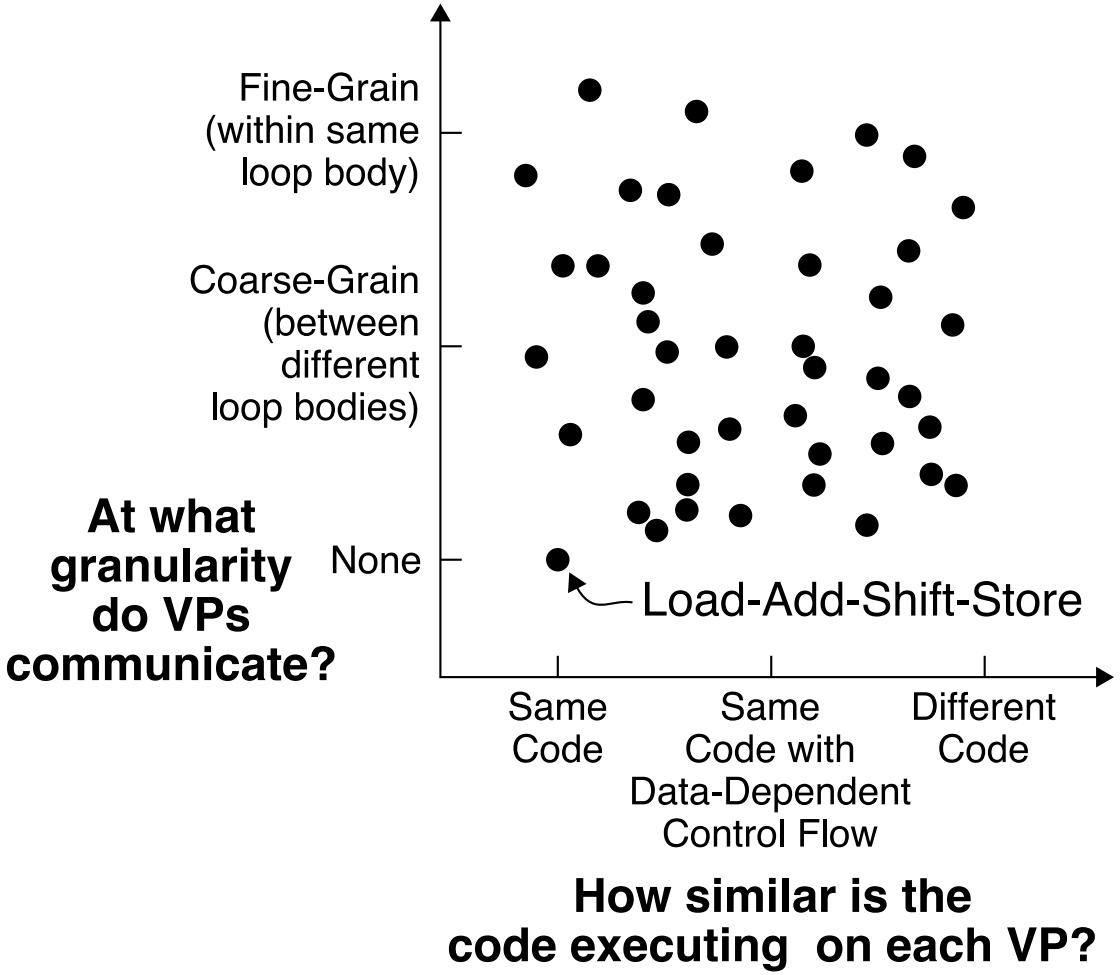
Vector Lanes



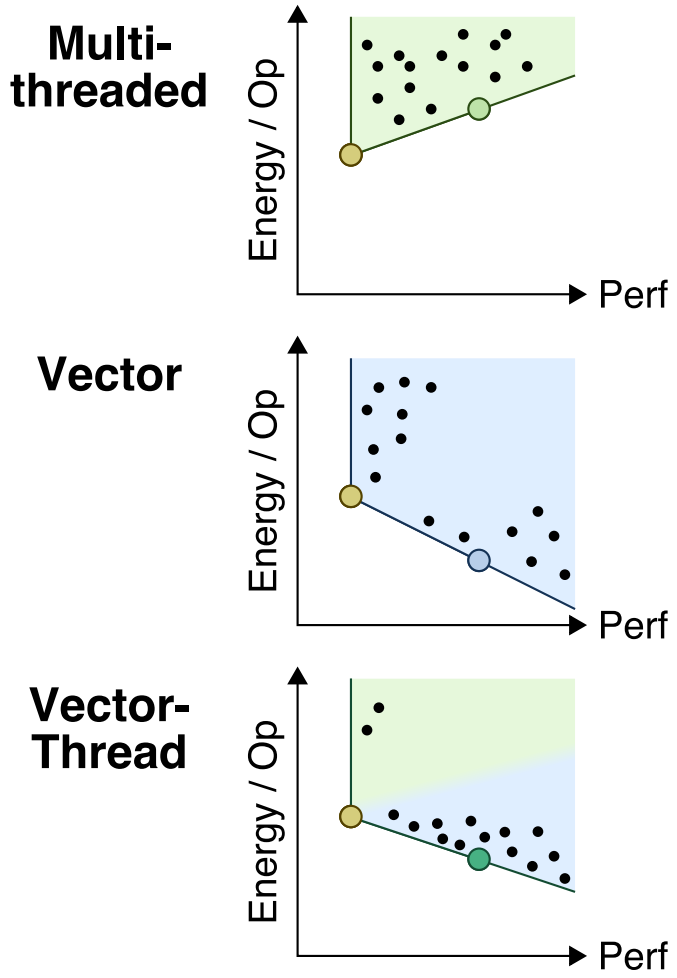


Energy-Efficiency for Various Kinds of Kernels

Flexibility

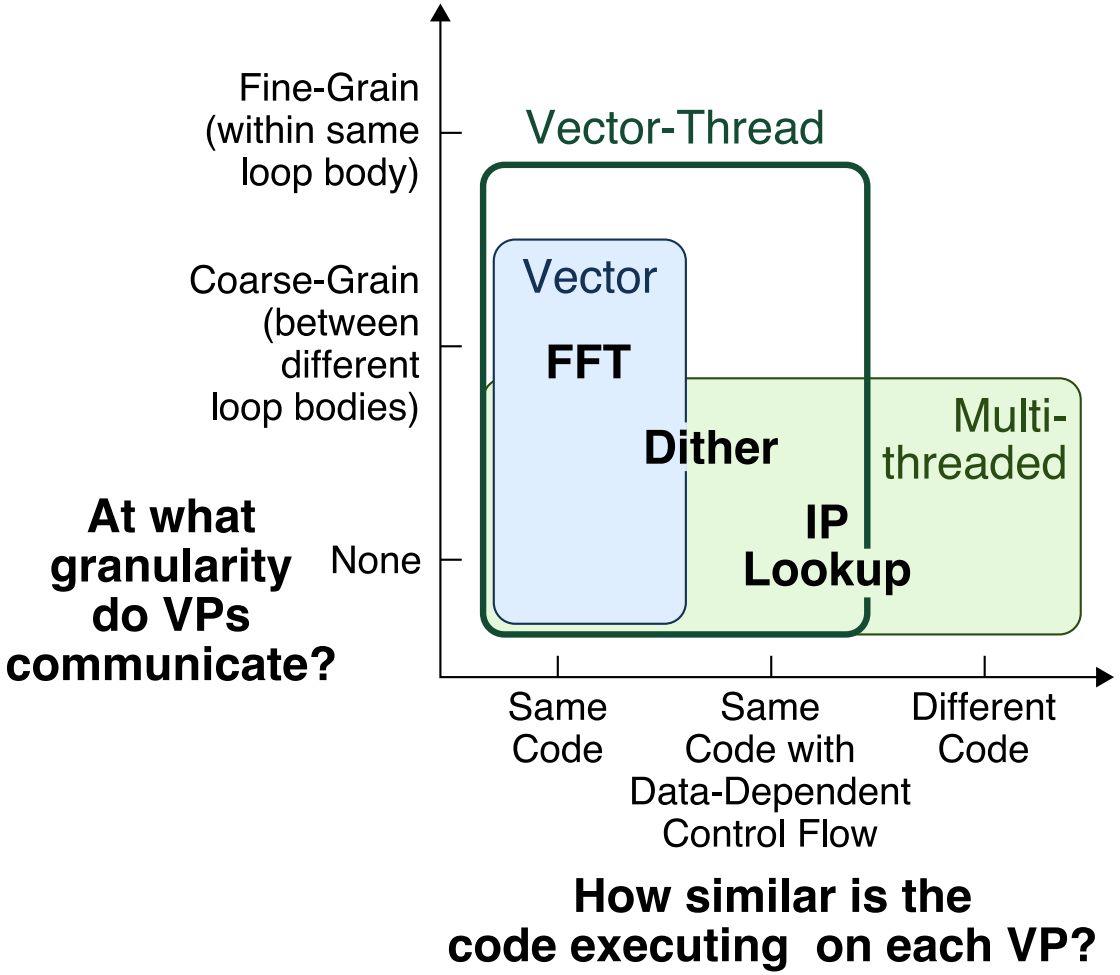


Energy-Efficiency

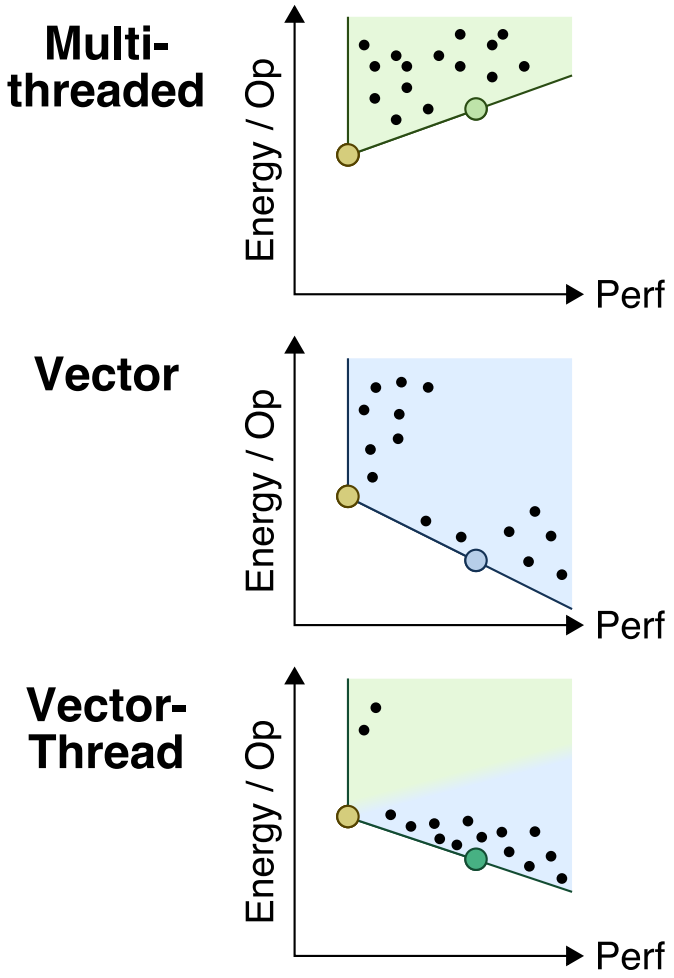


Energy-Efficiency for Various Kinds of Kernels

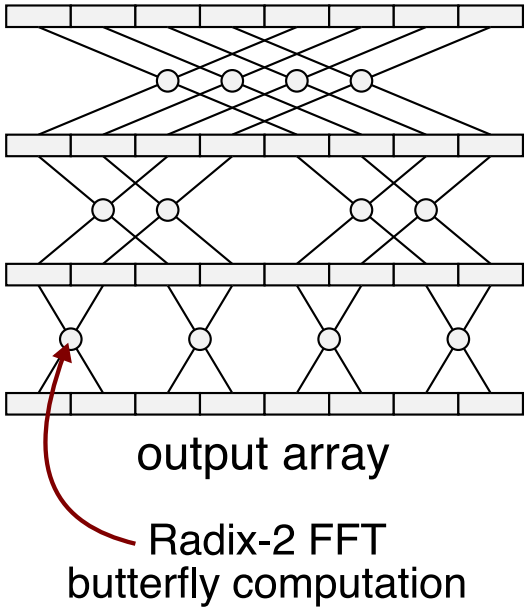
Flexibility



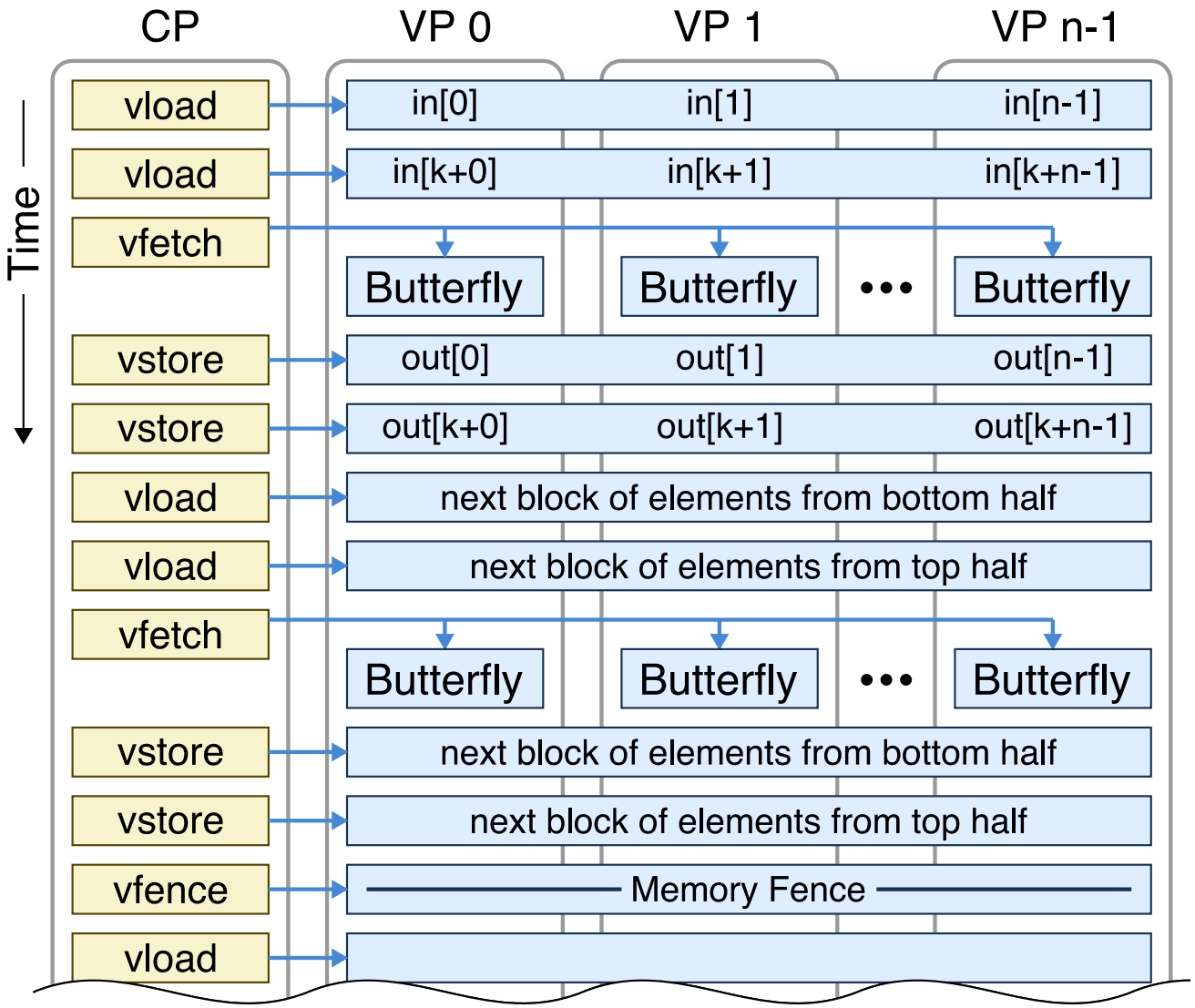
Energy-Efficiency



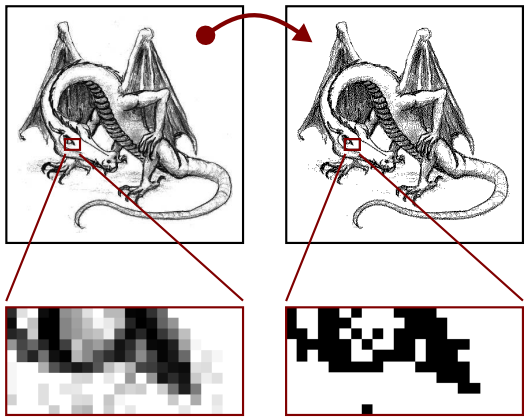
Mapping FFT to VT Programming Model



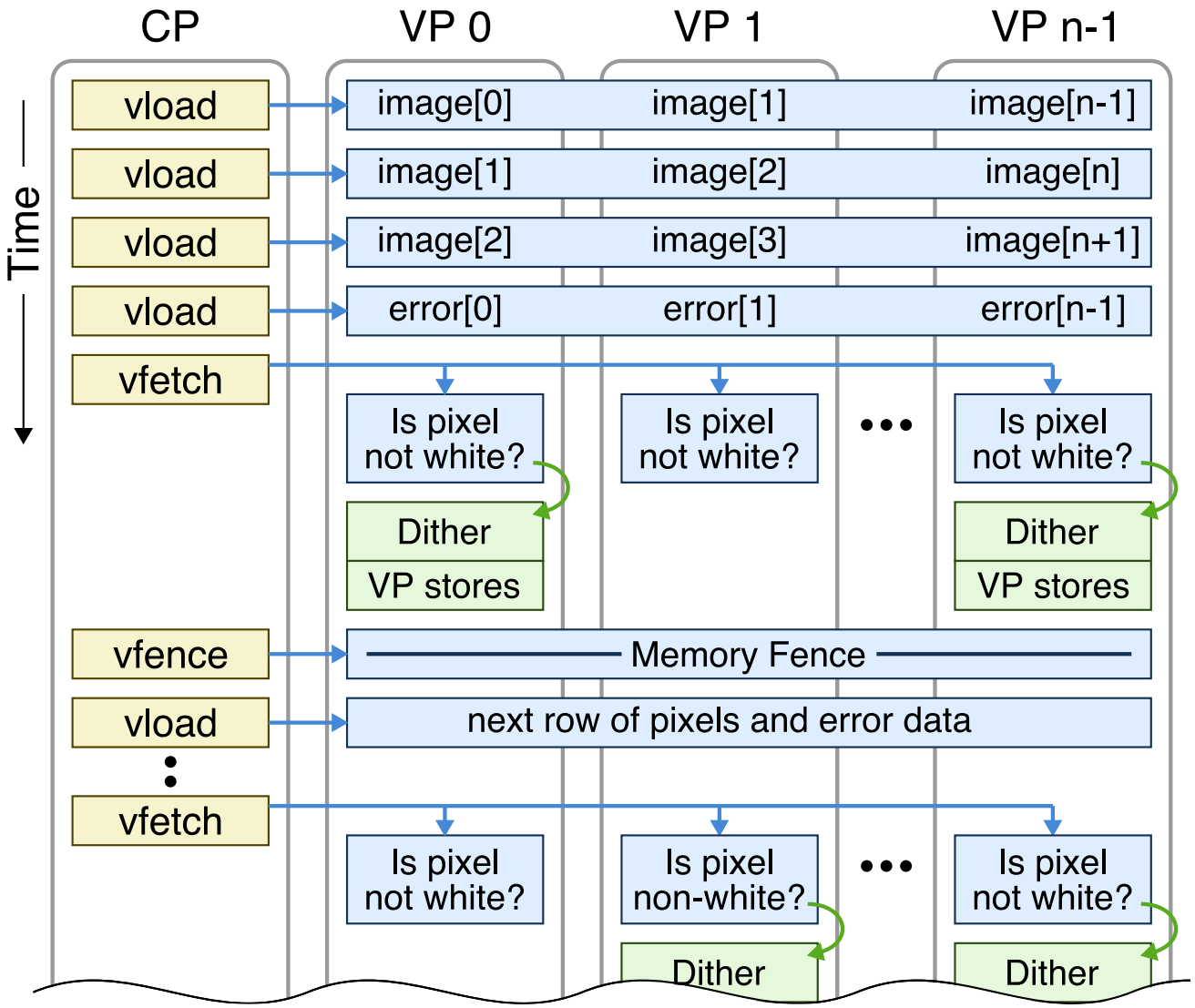

```
for each stage
 for num_elements/2
  load two elements
  butterfly
  store two elements
```



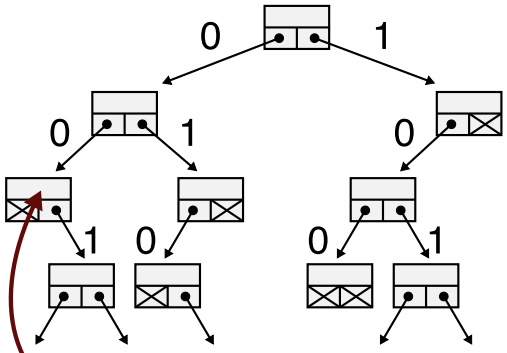
Mapping Dither to VT Programming Model



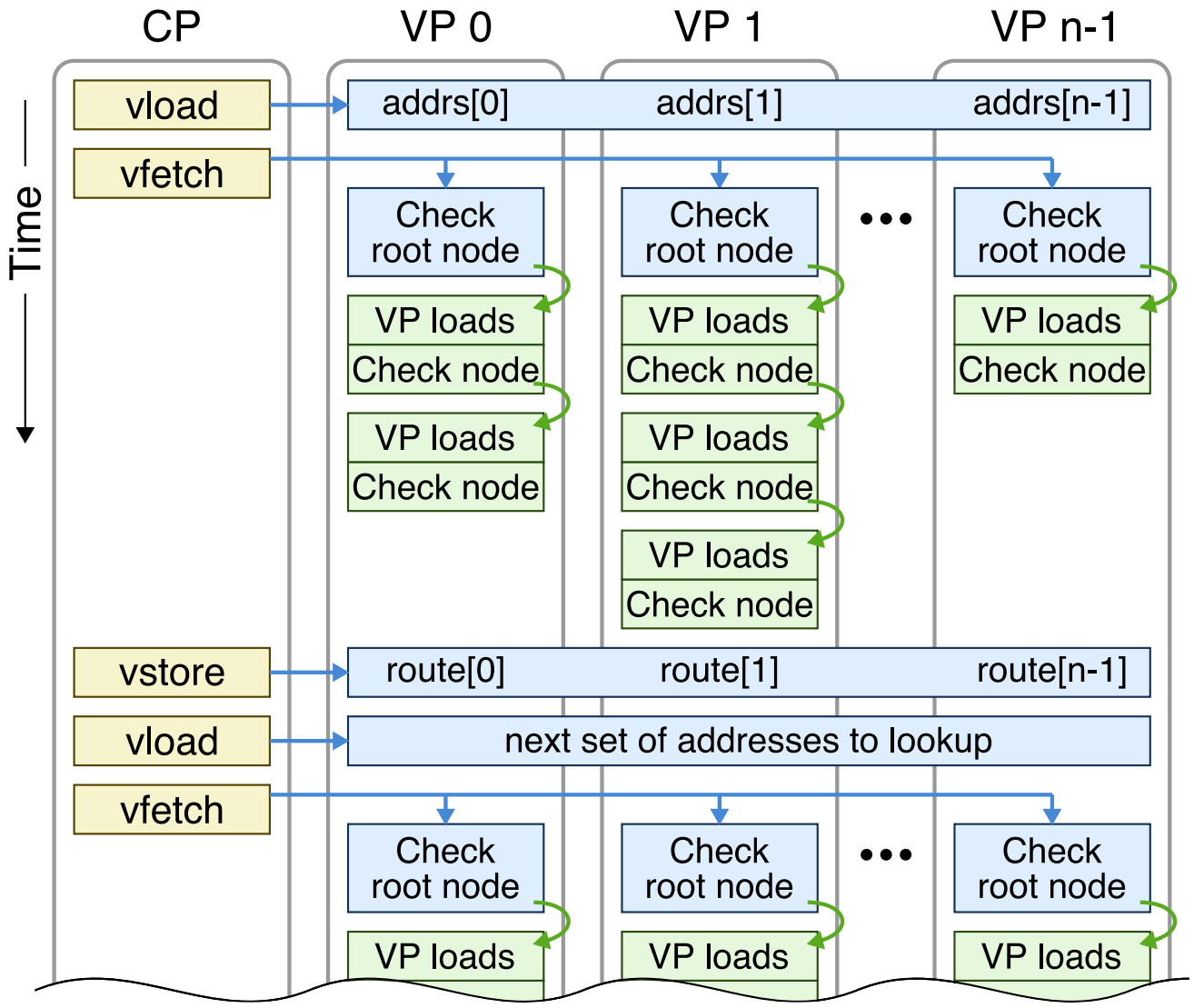
```
for each row
 for each column
  load 3 pixels
  load error value
  if ( pixel != white )
  {
   quantize to 1 bit
   calculate error
   store output pixel
   store error values
  }
```

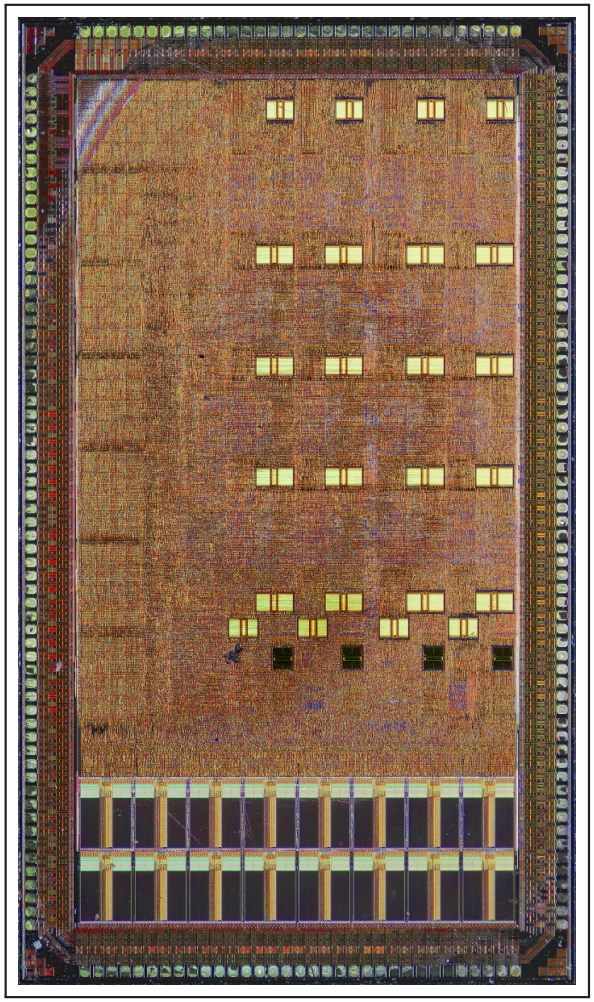


Mapping IP Lookup to VT Programming Model




```
for each ipaddr
 while ( !found )
 {
  route = node->route
  node =
   (ipaddr[bit++] == 0)
     ? node->left
     : node->right
  found = (node != NULL)
 }
 store route
```





Motivation

Vector-Thread Architectures

Scale VT Processor

Maven VT Processor

Future Research Directions

Motivation for Scale VT Processor

Goal: Evaluate energy-efficiency and flexibility of vector-threading by building a prototype chip with one control processor and several lanes

Initial implementation targets modern embedded systems

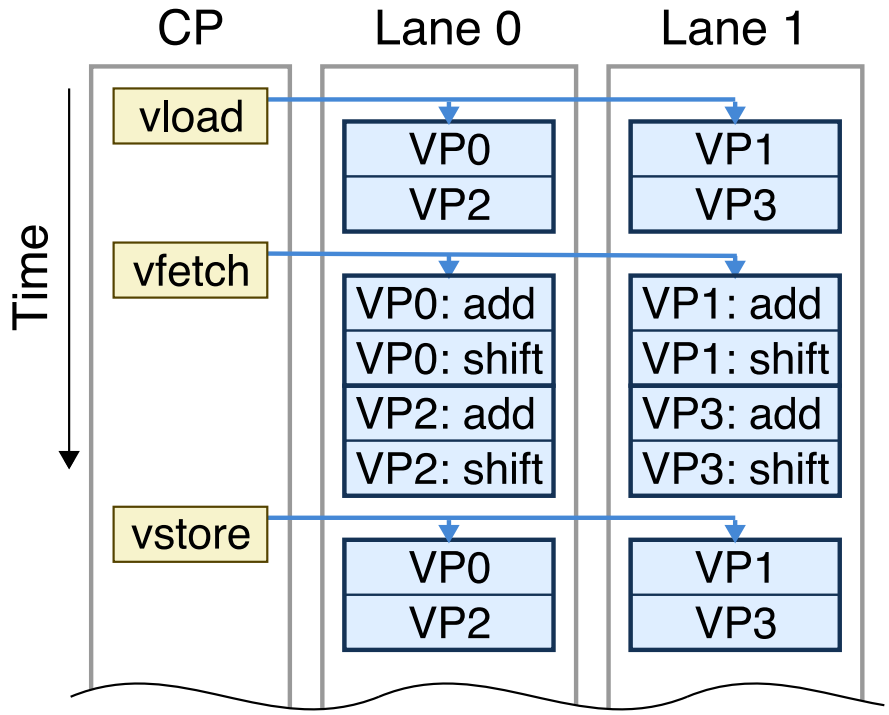
- Heavily power constrained
- Increasingly require high-performance
- Support growing number of features

set-top boxes • game consoles
media players • network routers
wireless base stations • smart phones

Atomic Instruction Block Interleaving

```
loop:
 vload   vr0, in_ptr
 vfetch  foo
 vstore  vr2, out_ptr
 add     in_ptr, vlen
 add     out_ptr, vlen
 sub     n, vlen
 bnez    n, loop
foo:
 add     r1, r0, r3
 shift   r2, r1, 16
```

Atomic Instruction Block (AIB)



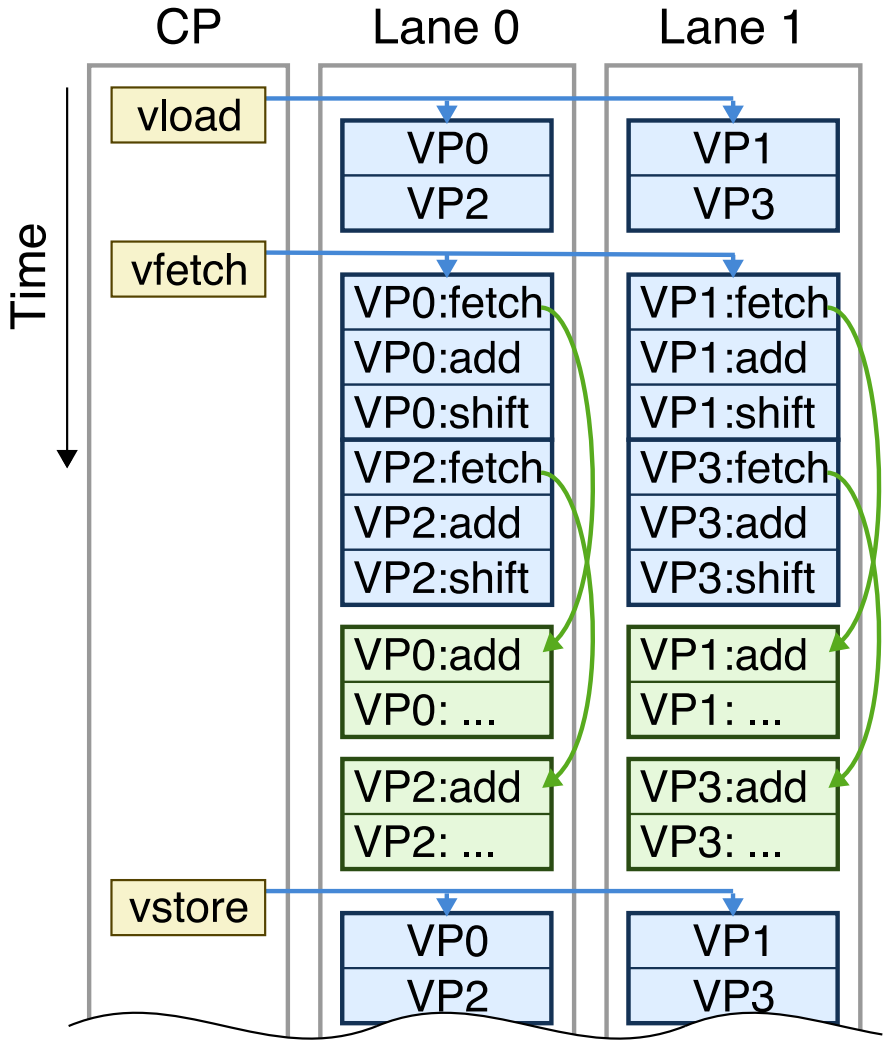
AIBs enable usage of temporary state
to reduce energy
and increase vector lengths

Explicit Thread Fetches

```
loop:
 vload   vr0, in_ptr
 vfetch  foo
 vstore  vr2, out_ptr
 add     in_ptr, vlen
 add     out_ptr, vlen
 sub     n, vlen
 bneqz   n, loop
foo:
 fetch   bar
 add     r1, r0, r3
 shift   r2, r1, 16
bar:
 add     ...
```

Atomic Instruction Block (AIB)

Explicit thread fetches reuse much of the hardware required for vector fetched AIBs



Scale Features

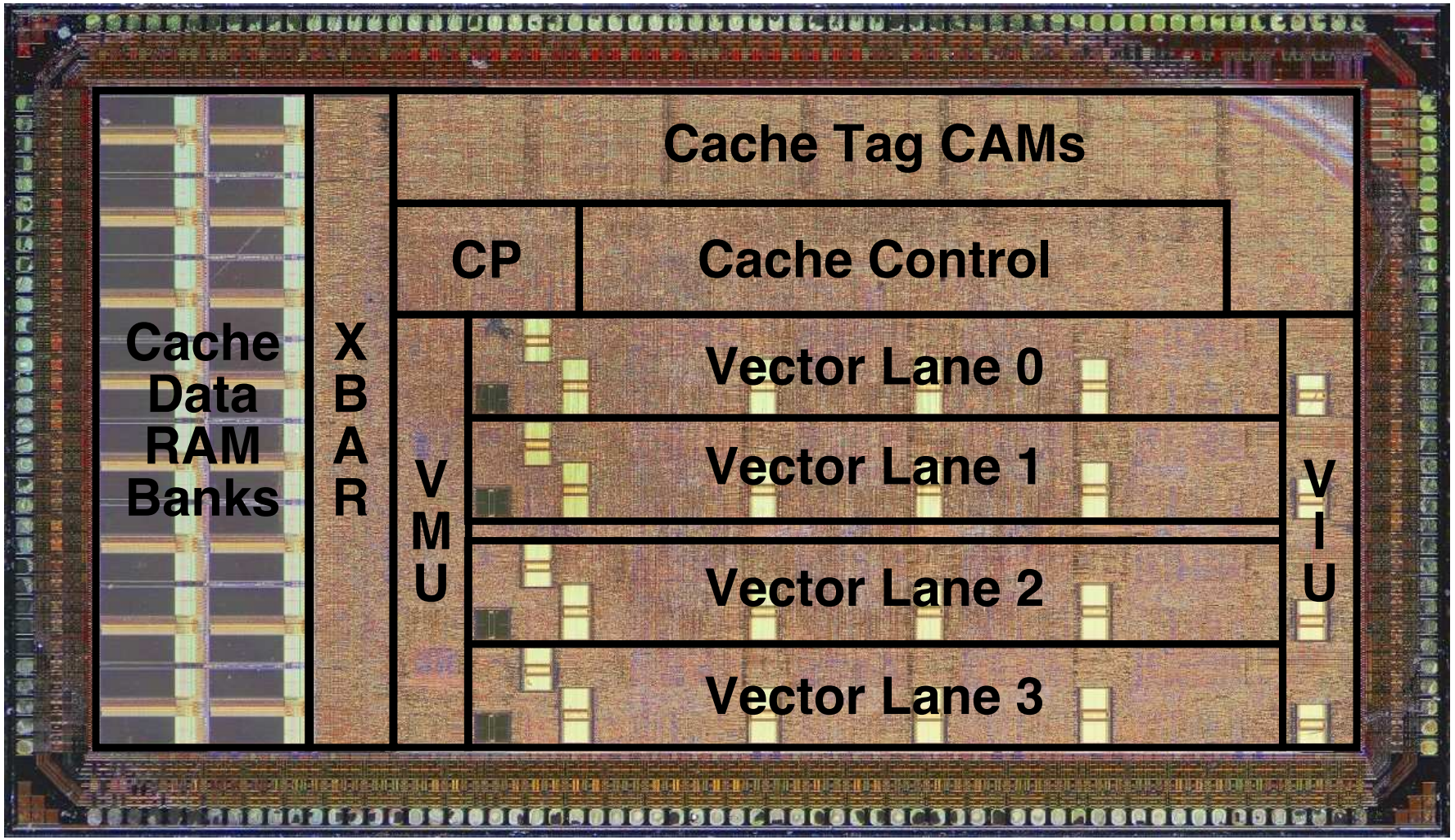
- Atomic Instruction Blocks
- Explicit Thread Fetches

- Clustering – reduce vector register file area, energy
- Cross-VP Ring Network – loop carried dependencies
- Configurable Vector Length – more elements, fewer vector registers
- Shared Vector Registers – temporaries, reductions, and constants
- AIB Caches – reduce fetch energy and amplify fetch issue bandwidth
- Vector Refill/Access Decoupling – better hide memory latencies
- Vector Segment Accesses – two-dimensional access patterns

International Symposium on Computer Architecture (ISCA), 2004
International Symposium on Microarchitecture (MICRO), 2004

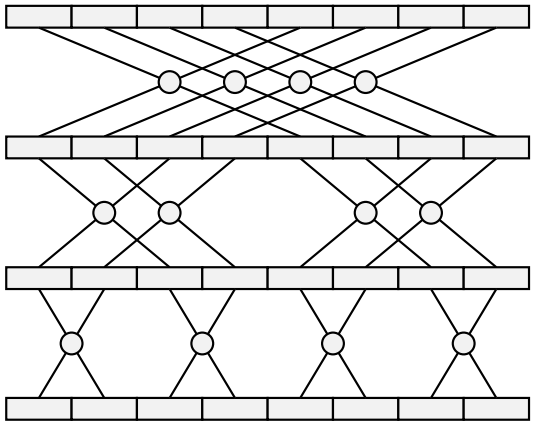
Scale VT Processor Prototype Chip

TSMC 0.18µm • 7.14 Million Transistors • 16.6 mm^2 Core Area

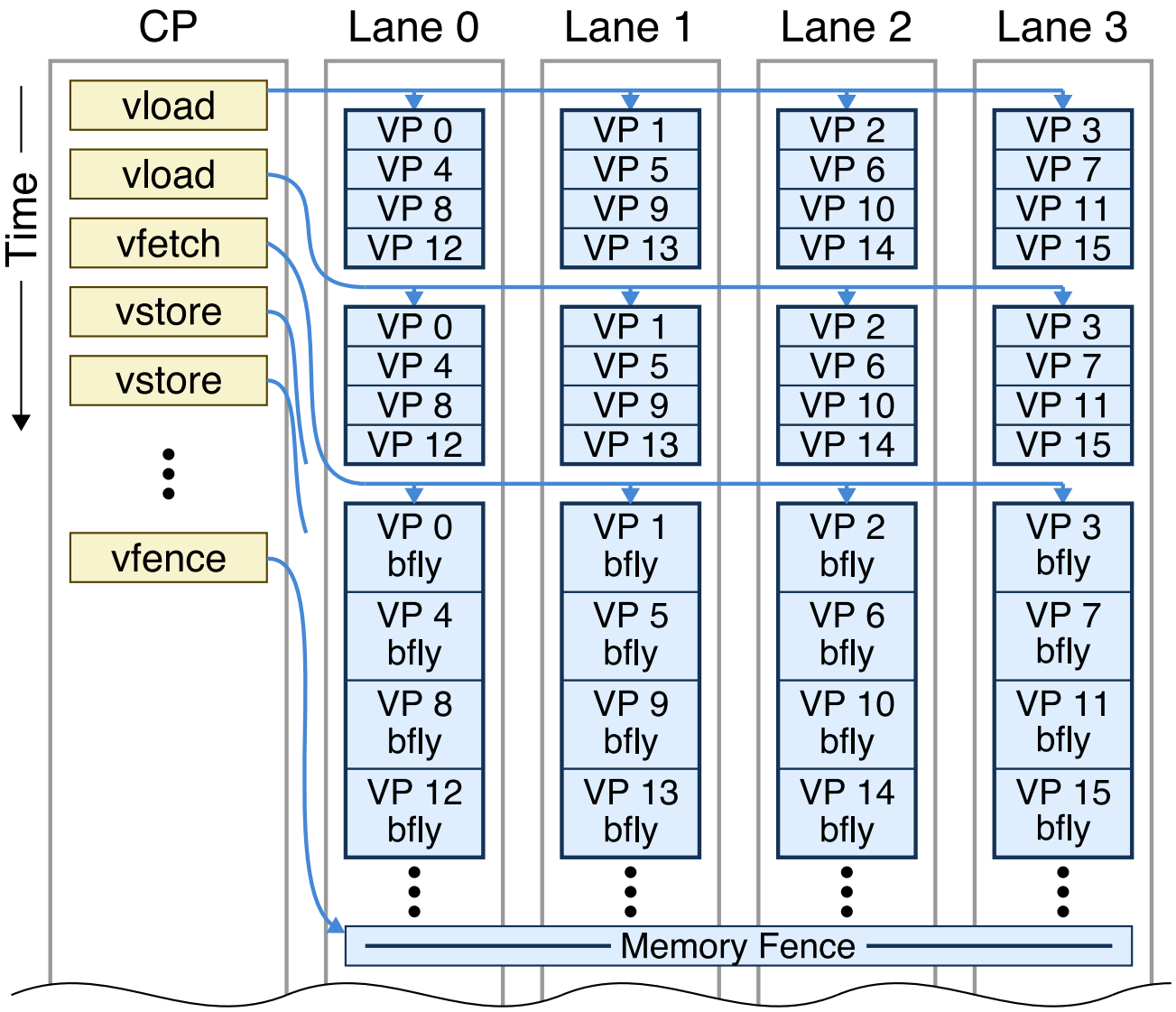


For more information see Trans. on Design Automation of Electronic Systems, 2008

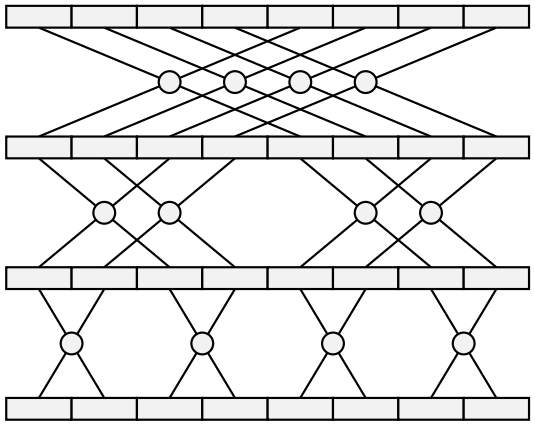
Executing FFT on Scale



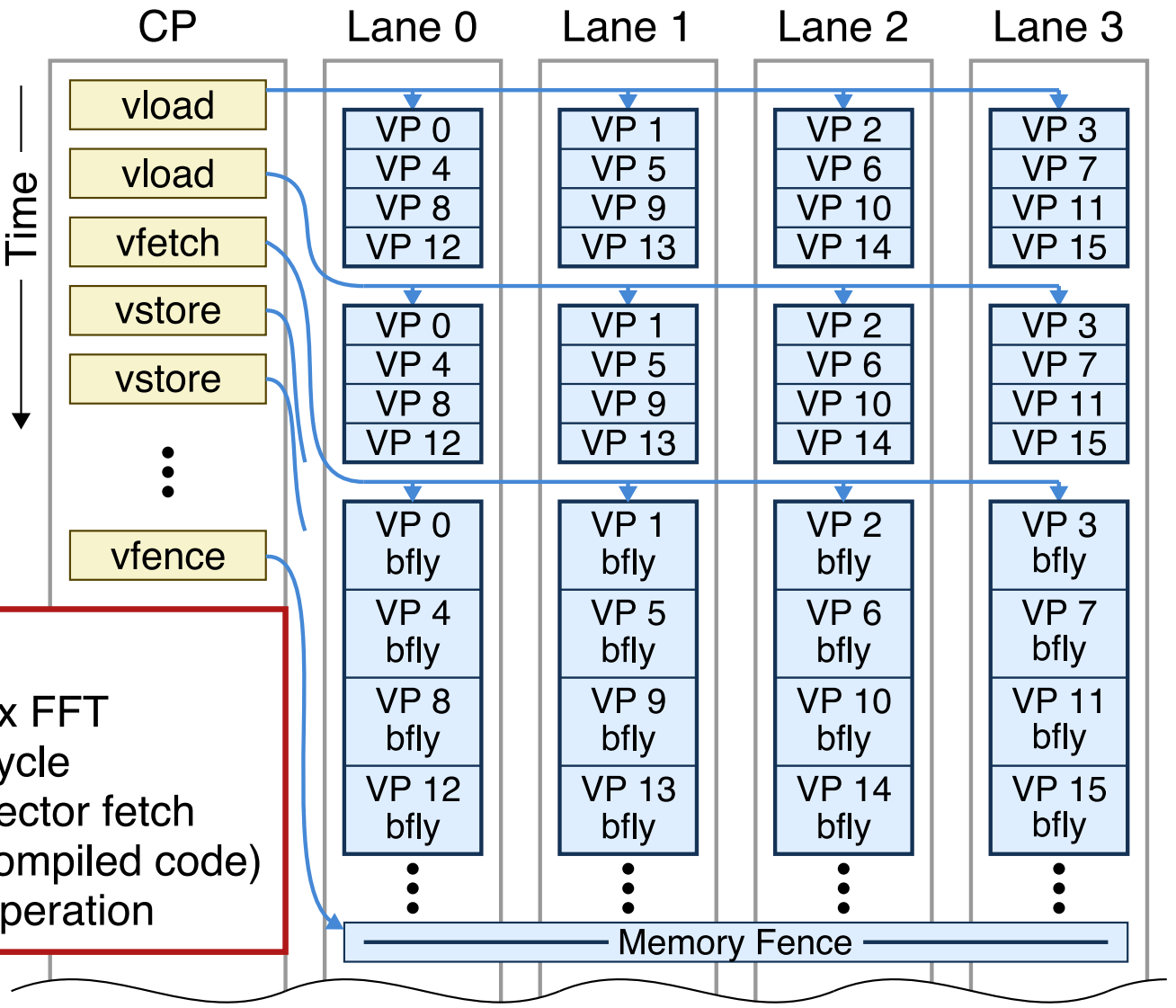
```
for each stage
 for num_elements/2
  load two elements
  butterfly
  store two elements
```



Executing FFT on Scale



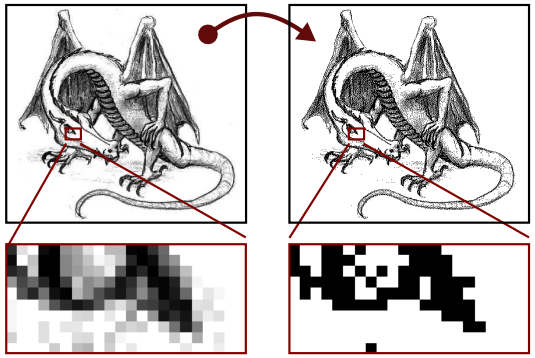
```
for each stage
 for num_elements/2
  load two elements
  butterfly
```



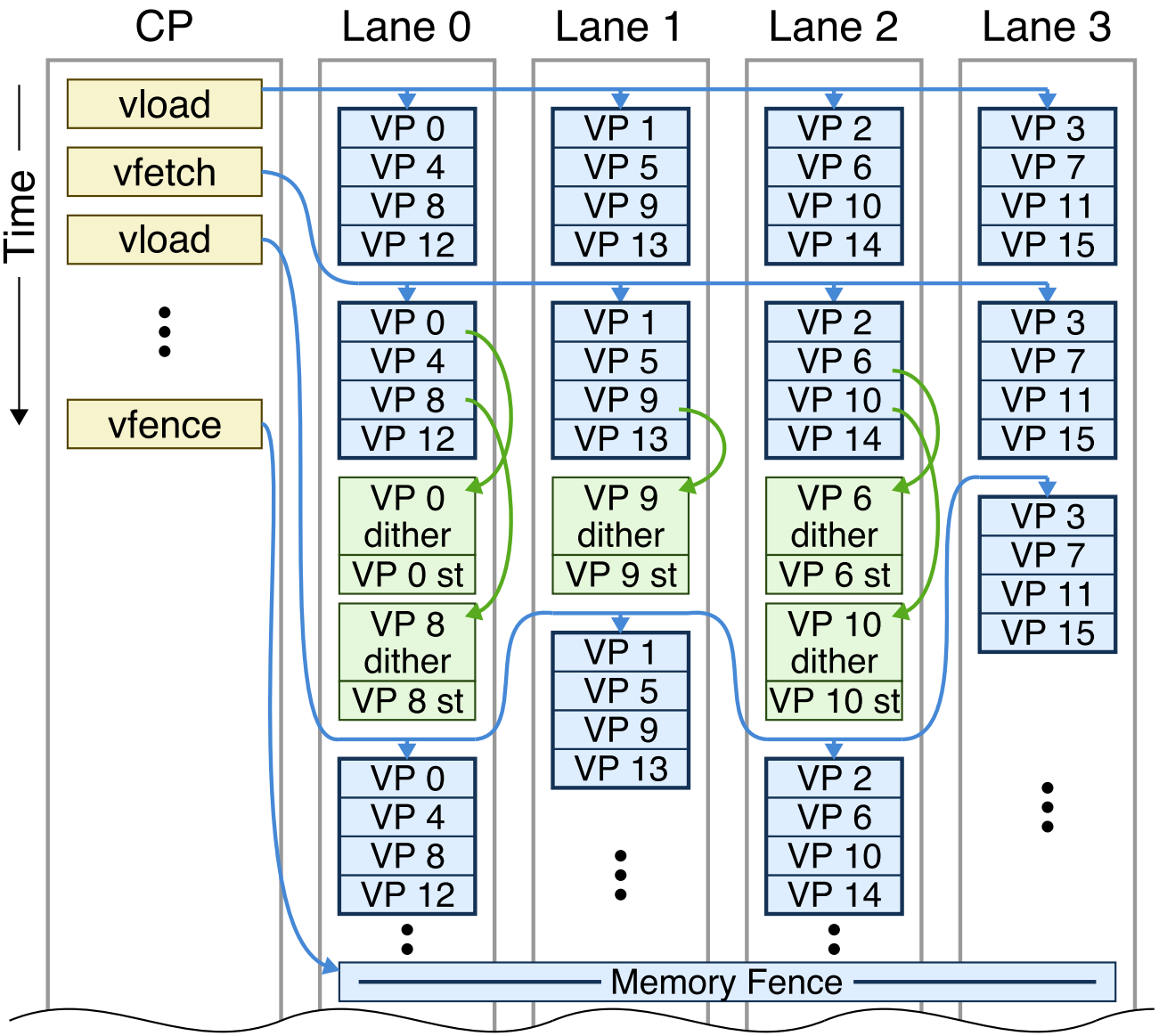
Results

- **256** point 16b complex FFT
- **~7** instructions per cycle
- **76** instructions per vector fetch
- **37x** faster than CP (compiled code)
- **3.5x** less energy per operation

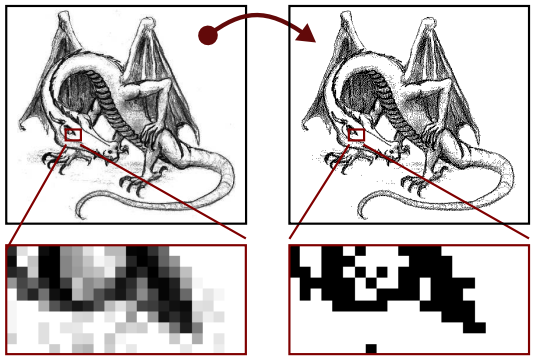
Executing Dither on Scale



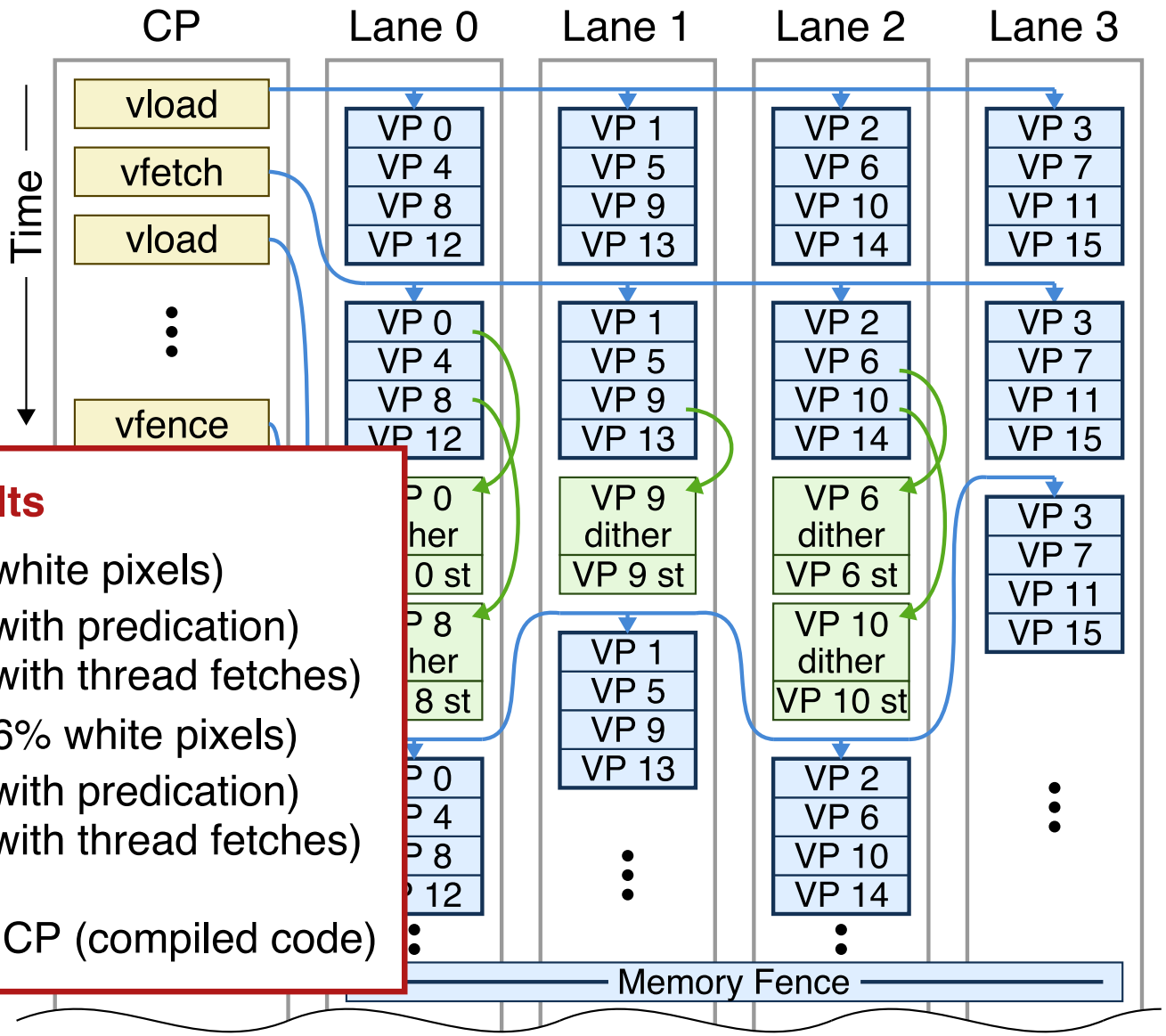
```
for each row
 for each column
  load 3 pixels
  load error value
  if ( pixel != white )
  {
   quantize to 1 bit
   calculate error
   store output pixel
   store error values
  }
```



Executing Dither on Scale



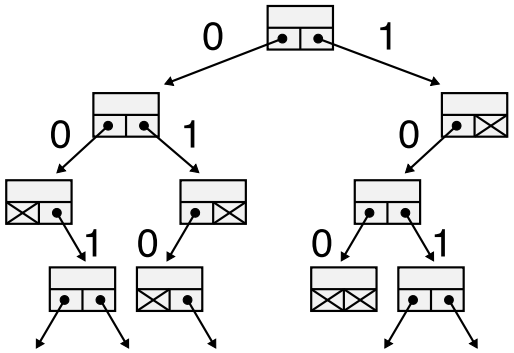
```
for each row
 for each column
```



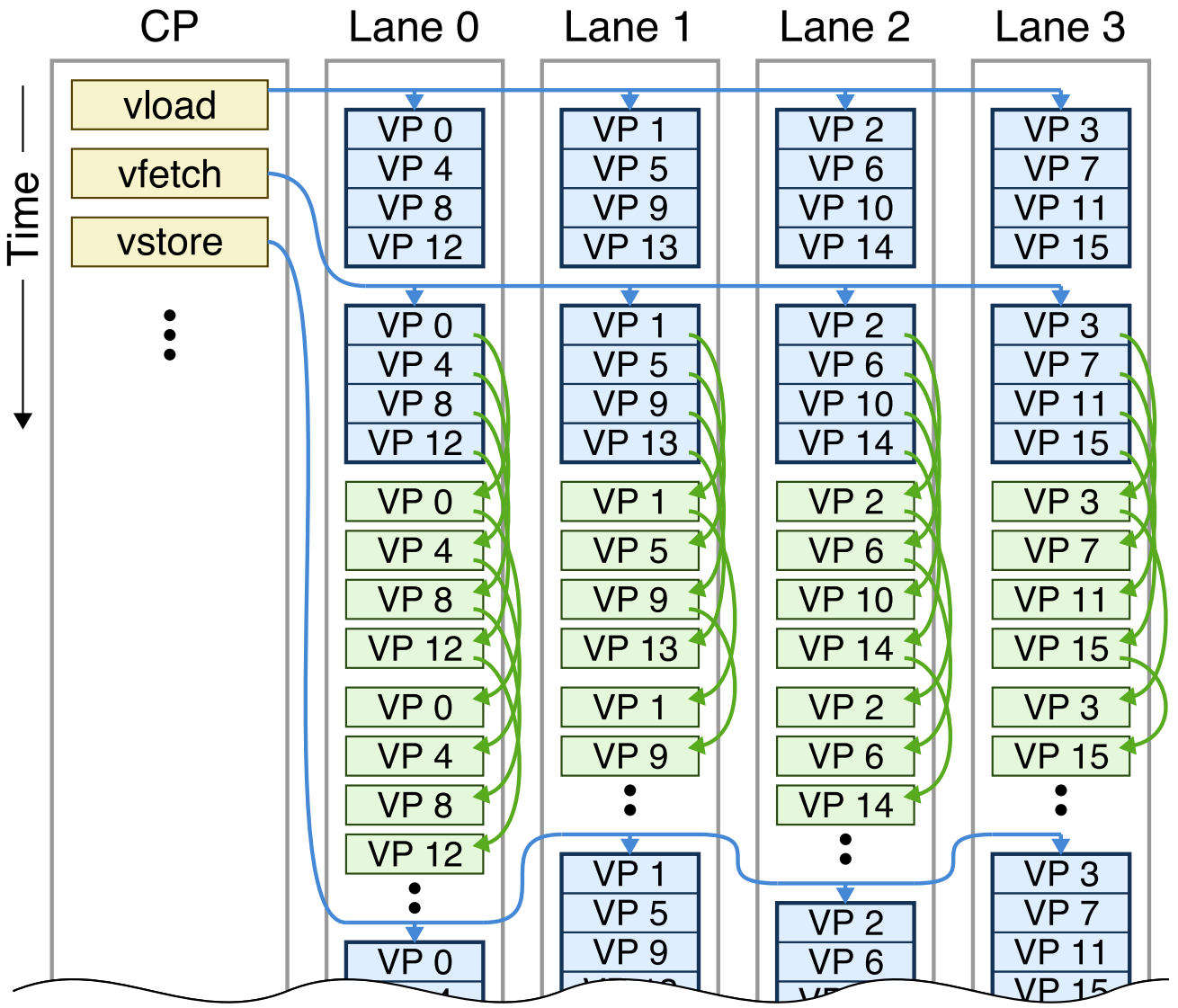
Results

- **Dragon Image** (64% white pixels)
  - **27** ops per pixel (with predication)
  - **12.8** ops per pixel (with thread fetches)
- **Test Pattern Image** (6% white pixels)
  - **27** ops per pixel (with predication)
  - **28.3** ops per pixel (with thread fetches)
  - **6.5** ops per cycle
  - **~7x** speed up over CP (compiled code)

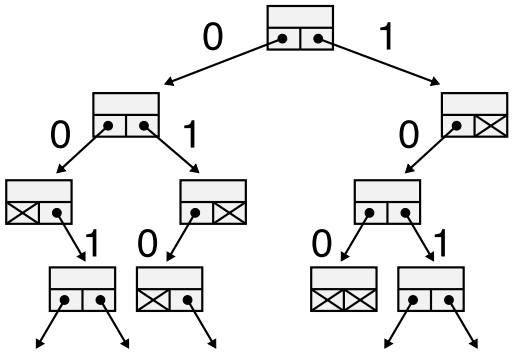
Executing IP Lookup on Scale



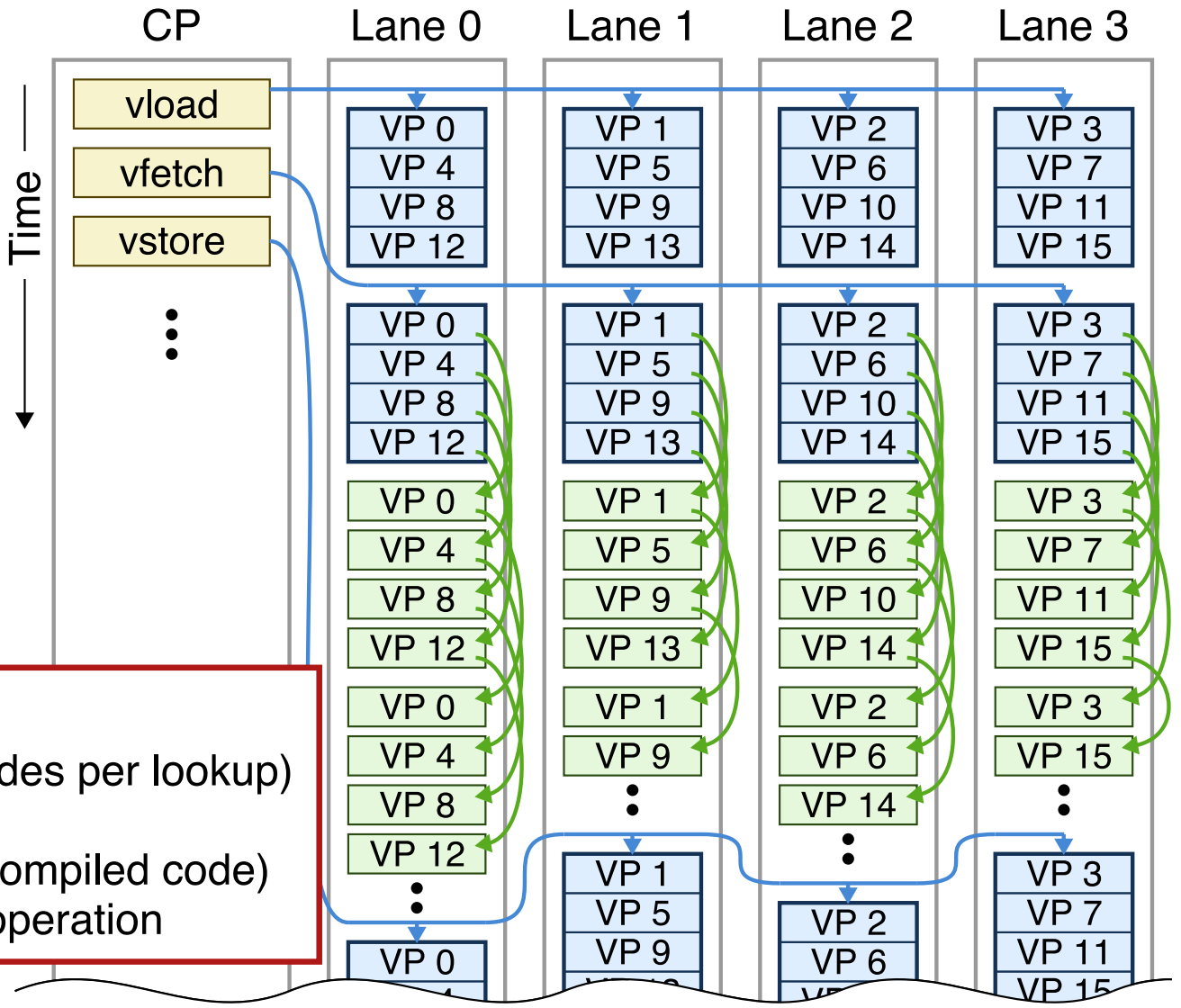

```
for each ipaddr
 while ( !found )
 {
  route = node->route
  node =
   (ipaddr[bit++] == 0)
     ? node->left
     : node->right
  found = (node != NULL)
 }
 store route
```



Executing IP Lookup on Scale




```
for each ipaddr
 while ( !found )
 {
  route = node->route
  node =
   (ipaddr[bit++] == 0)
     ? node->left
```



Results

- **2000** lookups (5-12 nodes per lookup)
- **7.4** ops per cycle
- **~7x** faster than CP (compiled code)
- **16%** less energy per operation

Benchmarks

Image Processing

V	**rgbcmyk**	Color conversion
V	**rgbyiq**	Color conversion
V	**hpg**	Grayscale high-pass filter
VT	**dither**	Floyd-Steinberg dithering
V	**rotate**	Rotate binary image
V	**fdct**	Forward DCT 8x8
V	**idct**	Inverse DCT 8x8
V	**quantize**	JPEG quantization 8x8
T	**jpegenc**	JPEG compression 8x8

Audio Processing

T	**adpcme**	ADPCM speech encoding
T	**adpcmd**	ADPCM speech decoding

Network Processing

T	**iplookup**	IP address lookups
T	**pktflow**	IP packet processing
T	**ospf**	Dijkstra shortest path first

Text & Data Processing

T	**text**	Control language parsing
T	**qsort**	Quick sort of strings
VT	**li**	Lisp interpreter
T	**pntrch**	Searching linked list

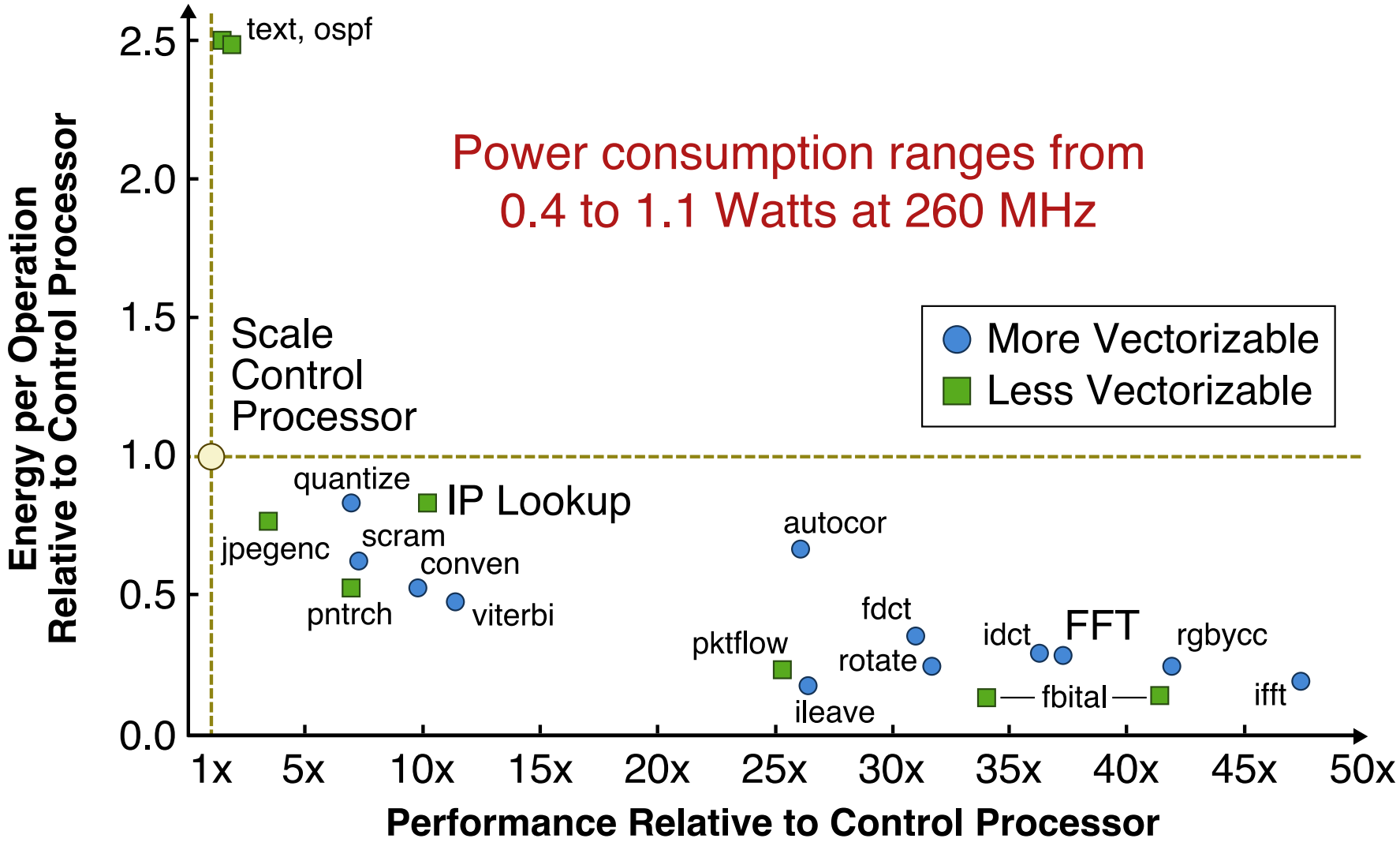
Security Processing

V	**rijndael**	AES encryption
VT	**sha**	Secure hash algorithm

Telcom & Digital Signal Processing

V	**fir**	Finite impulse resp. filter
T	**fbital**	DSL modem bit allocation
V	**fft**	256pt fixed-point FFT
V	**viterbi**	Viterbi decoder
V	**autocor**	Fixed-point autocorrelation
V	**conven**	802.11A convolutional enc
V	**scram**	802.11A scrambler
V	**ileave**	802.11A bit interleaver
V	**mapper**	802.11A OFDM mapper
V	**ifft**	802.11A 64pt inverse FFT

Energy-Efficiency of the Scale VT Processor



The Scale VT Processor Team at MIT

Project Supervisor

Professor Krste Asanović

Scale Architecture and Chip Implementation

Christopher Batten, Ronny Krashinsky

VLSI Toolflow Development and Test Board Bring-Up

Ken Barr, Asif Khan, Albert Ma, Jaime Quinonez

Compiler Development

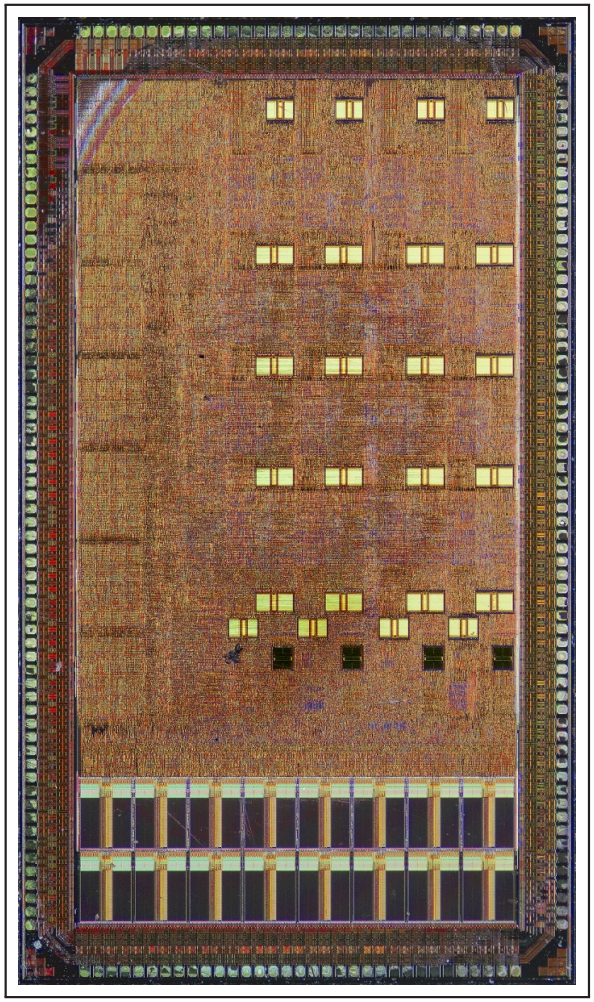
Mark Hampton

Benchmark and Test Case Development

Jared Casper, Jeff Cohen, Steve Gerding

DRAM Model Development

Brian Pharris



Motivation

Vector-Thread Architectures

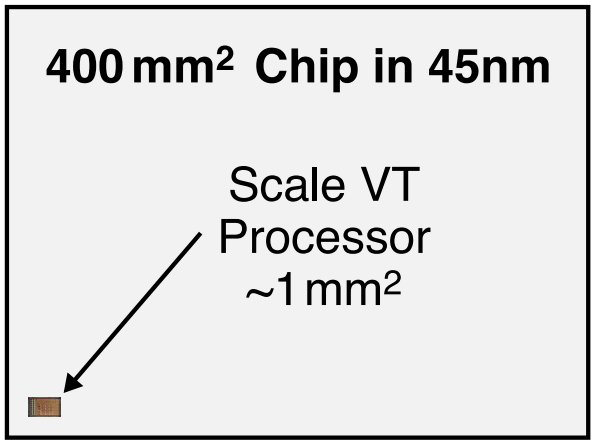
Scale VT Processor

Maven VT Processor

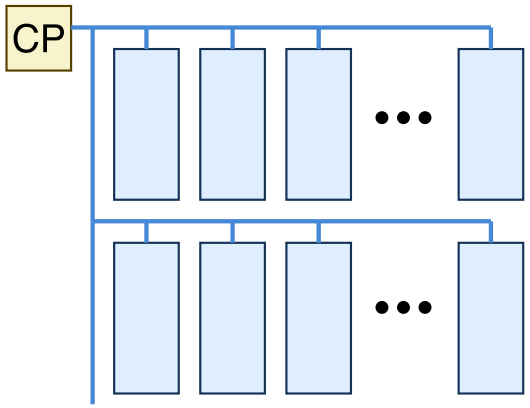
Future Research Directions

Motivation for the Maven VT Processor

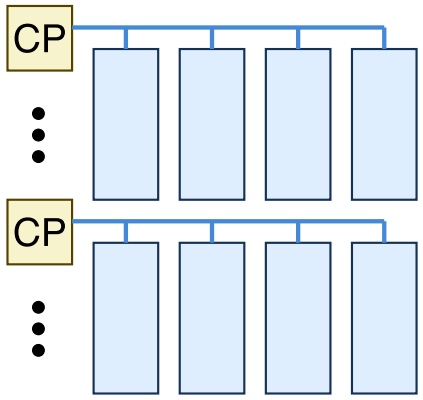
How can we extend VT to more general-purpose systems with larger dies and more execution resources?



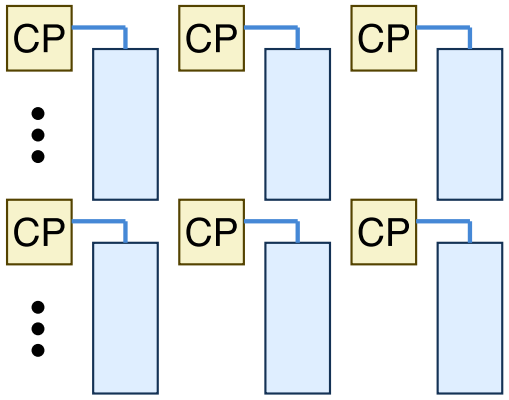
One Control Processor with 10-100s of Lanes



Several Control Processors Each with a Couple Lanes



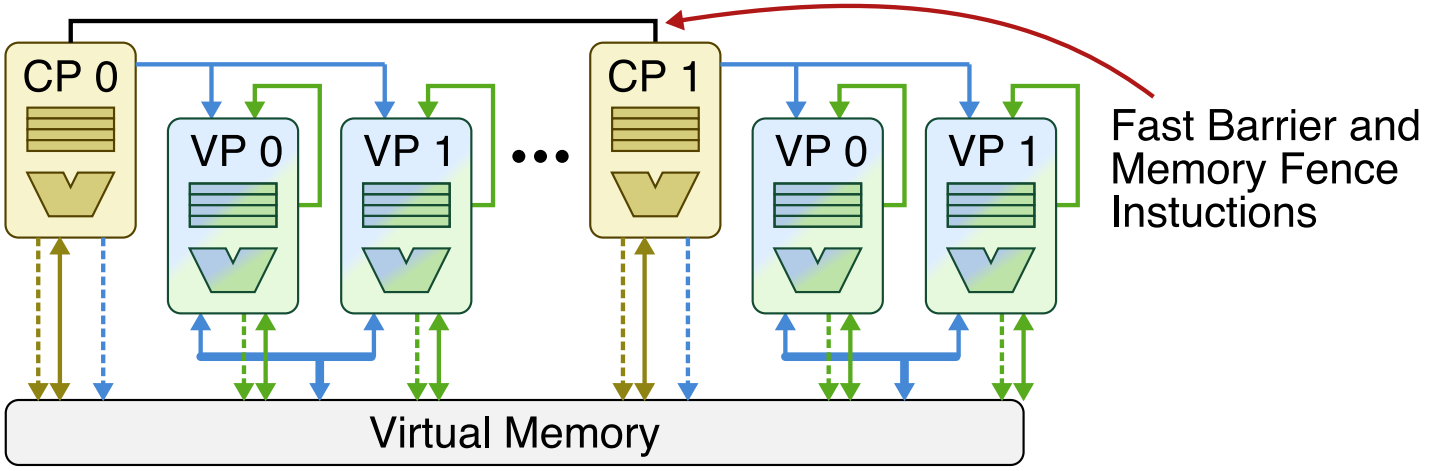
Many Control Processors Each with a Single Lane



A Manycore Vector-Thread Architecture

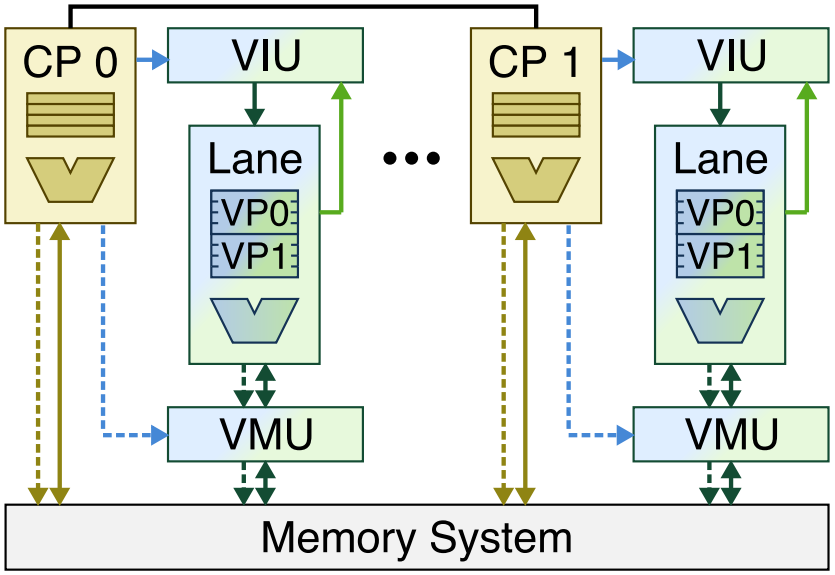
Assembly Programmer's View

Multiple CP Threads Each With a Vector of VPs



Implementation

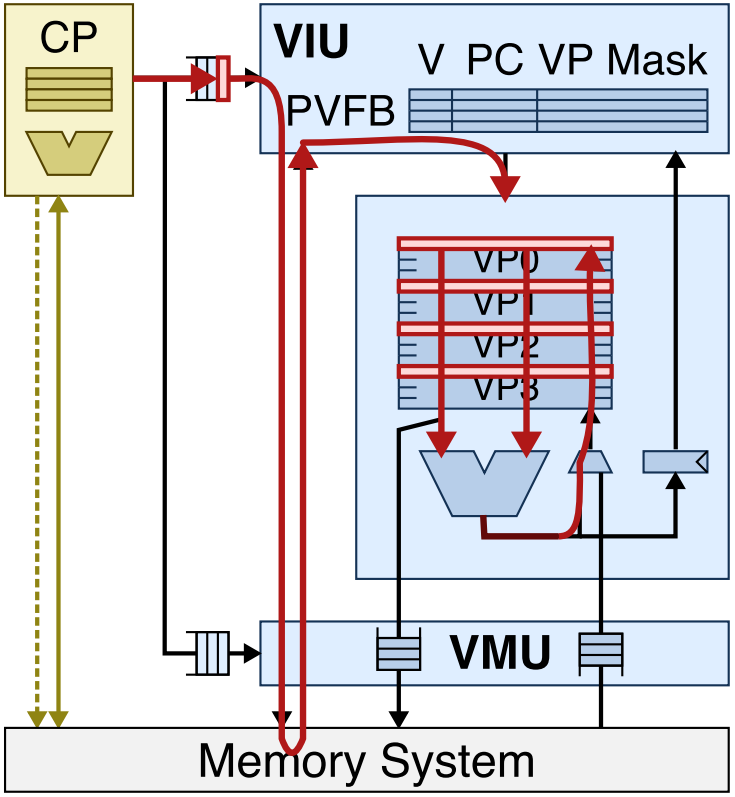
Sea of Single-Lane Vector-Thread Units

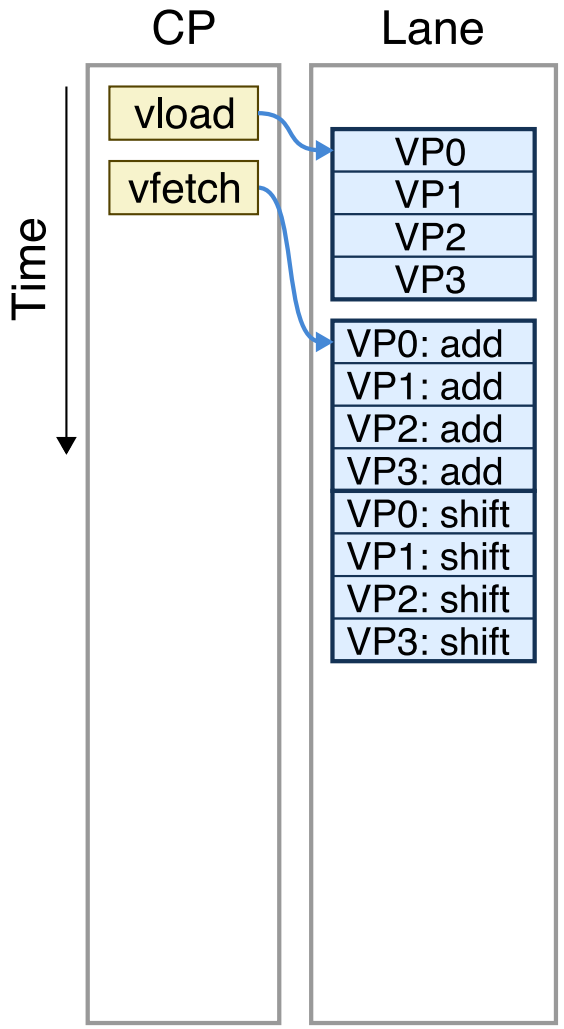


Maven Lane Microarchitecture

```
loop:
 vload   vr0, in_ptr
 vfetch  foo
 vstore  vr2, out_ptr

 add     in_ptr, vlen
 add     out_ptr, vlen
 sub     n, vlen
 bneqz   n, loop
foo:
 add     r1, r0, r3
 shift   r2, r1, 16
 vp.stop
```

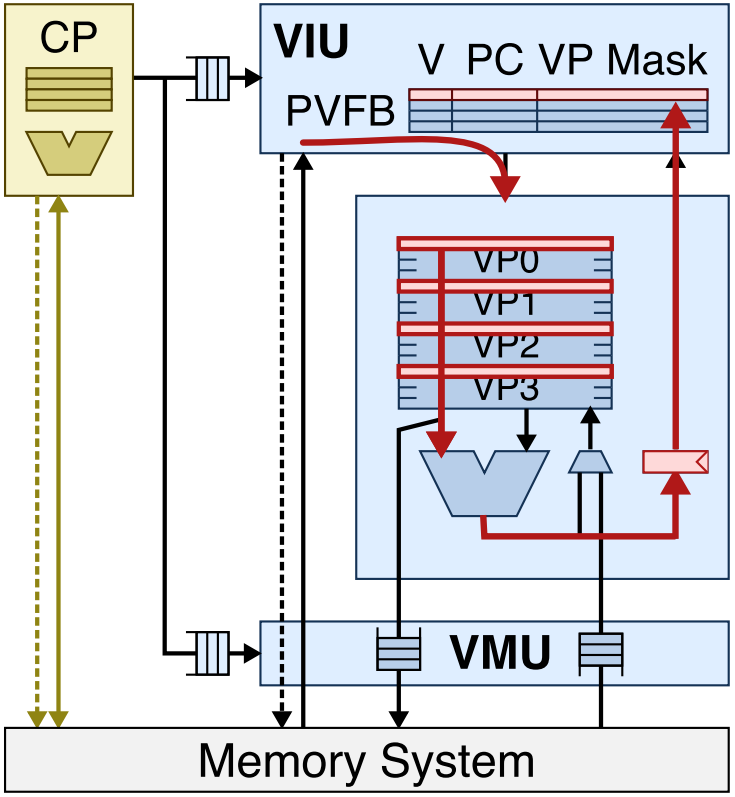


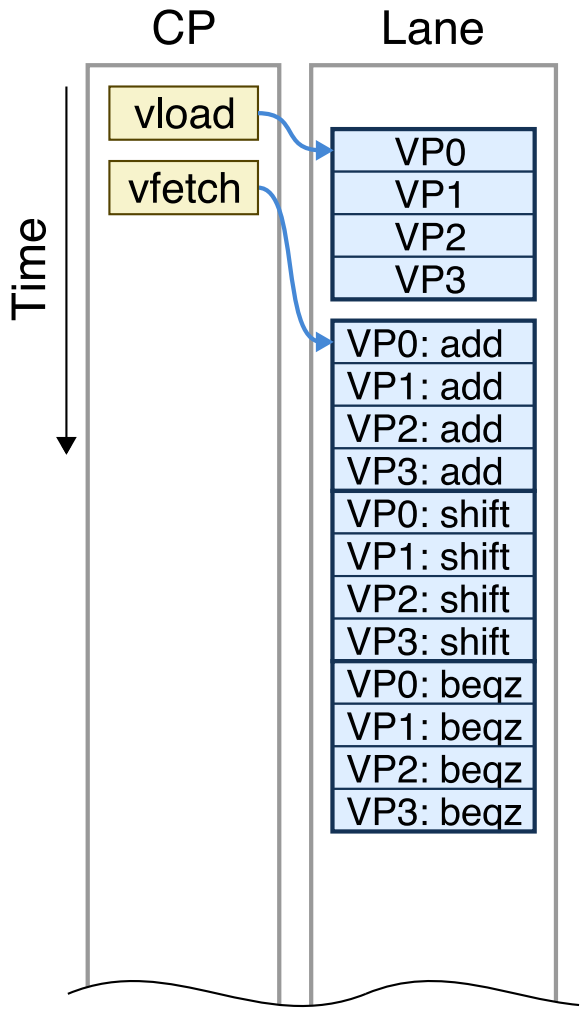


Maven Lane Microarchitecture

```
loop:
 vload   vr0, in_ptr
 vfetch  foo
 vstore  vr2, out_ptr

 add     in_ptr, vlen
 add     out_ptr, vlen
 sub     n, vlen
 bneqz   n, loop
foo:
 add     r1, r0, r3
 shift   r2, r1, 16
 beqz    r2, bar
 vp.stop
bar:
 add     ...
 vp.stop
```

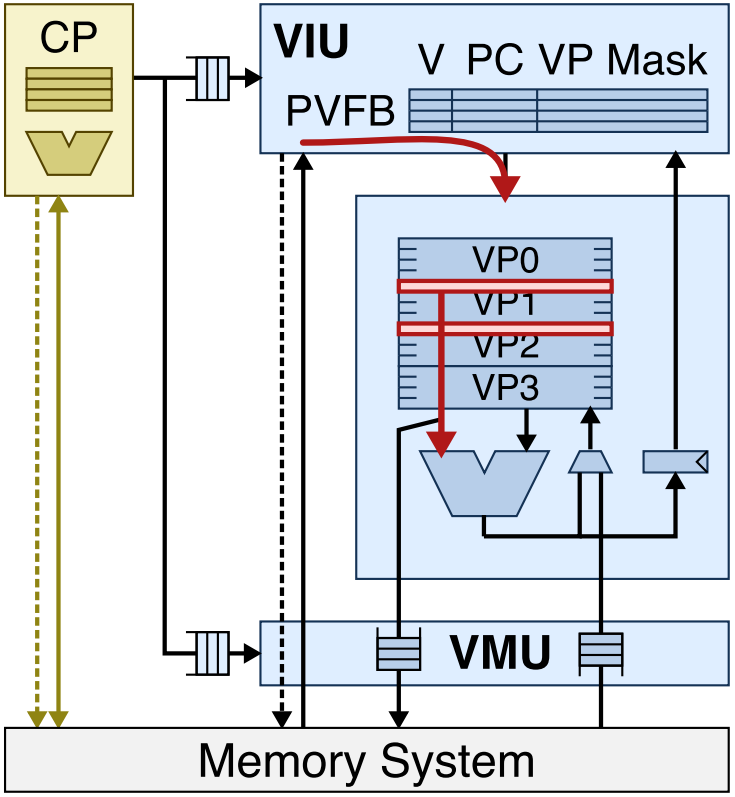


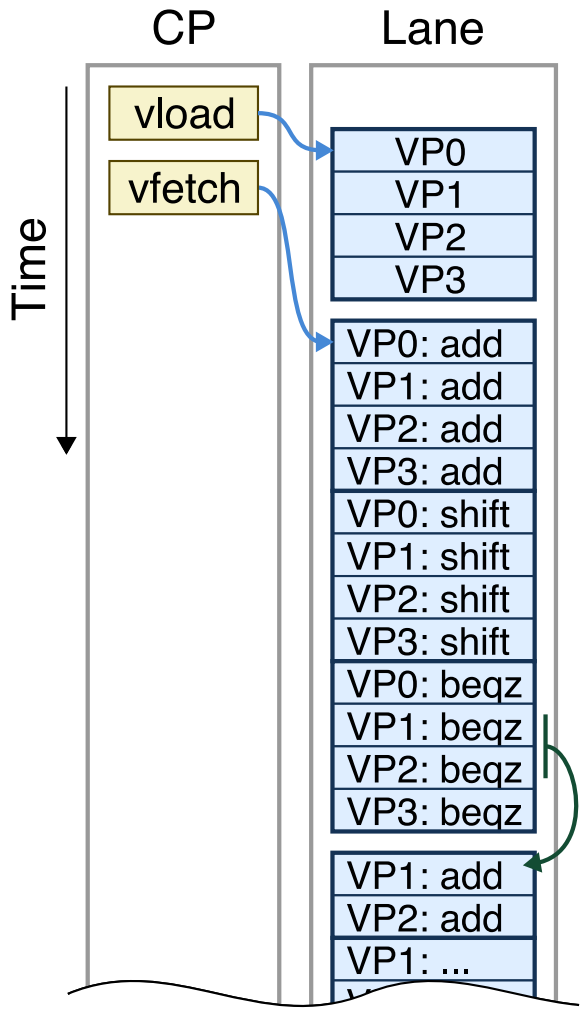


Maven Lane Microarchitecture

```
loop:
 vload   vr0, in_ptr
 vfetch  foo
 vstore  vr2, out_ptr

 add     in_ptr, vlen
 add     out_ptr, vlen
 sub     n, vlen
 bneqz   n, loop
foo:
 add     r1, r0, r3
 shift   r2, r1, 16
 beqz    r2, bar
 vp.stop
bar:
 add     ...
 vp.stop
```

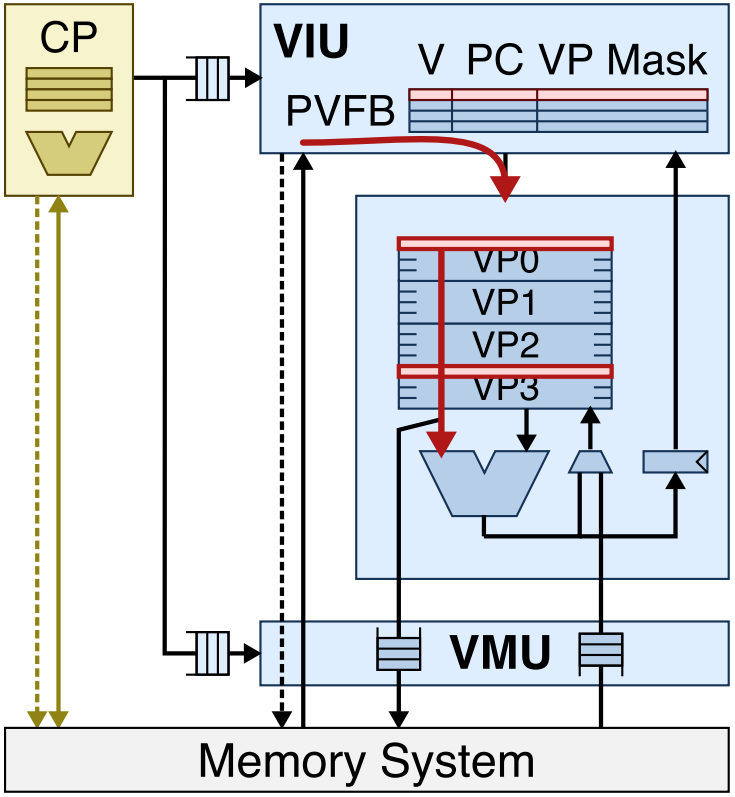


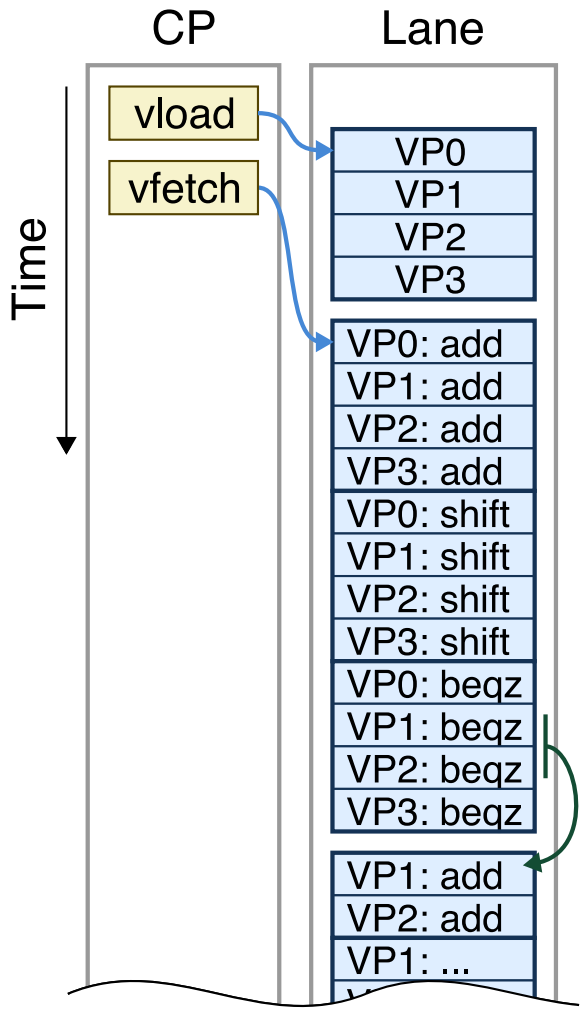


Maven Lane Microarchitecture

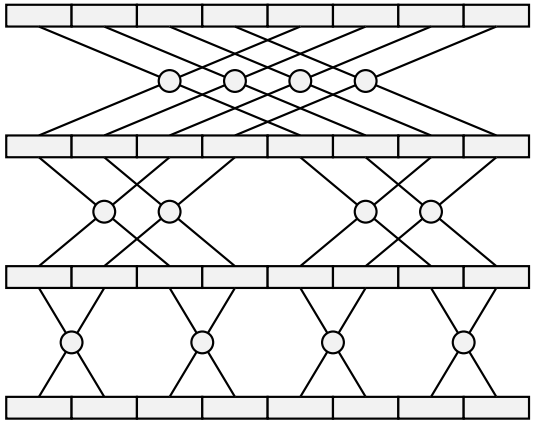
```
loop:
 vload   vr0, in_ptr
 vfetch  foo
 vstore  vr2, out_ptr

 add     in_ptr, vlen
 add     out_ptr, vlen
 sub     n, vlen
 bneqz   n, loop
foo:
 add     r1, r0, r3
 shift   r2, r1, 16
 beqz    r2, bar
 vp.stop
bar:
 add     ...
 vp.stop
```

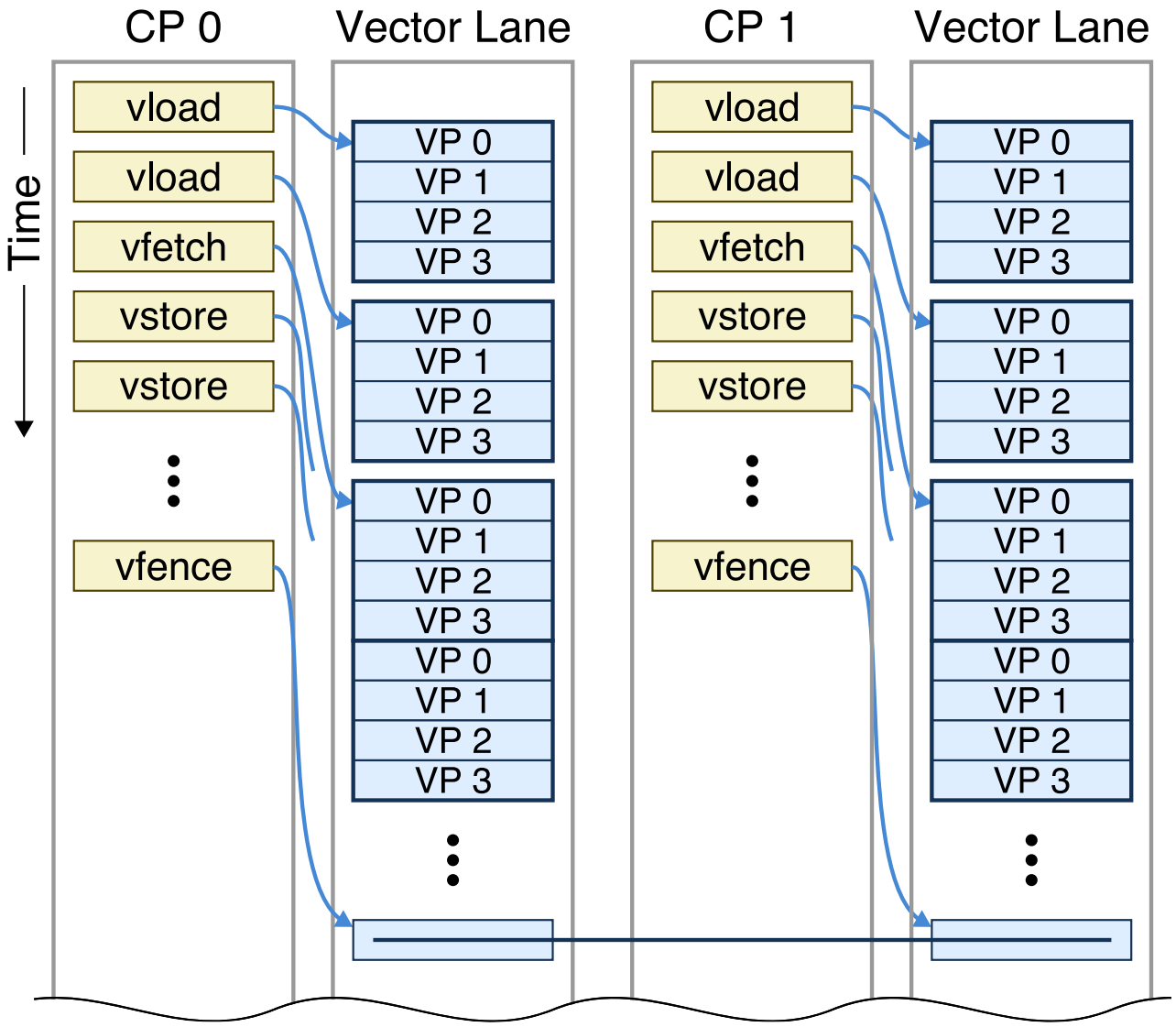




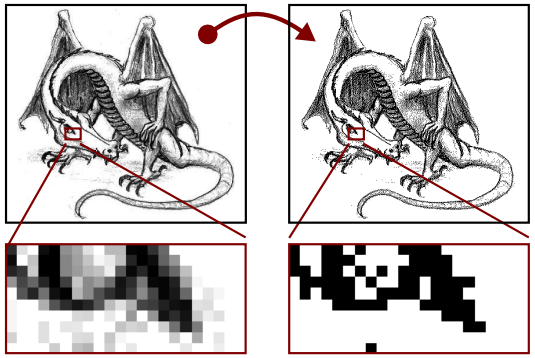
Executing FFT on Maven



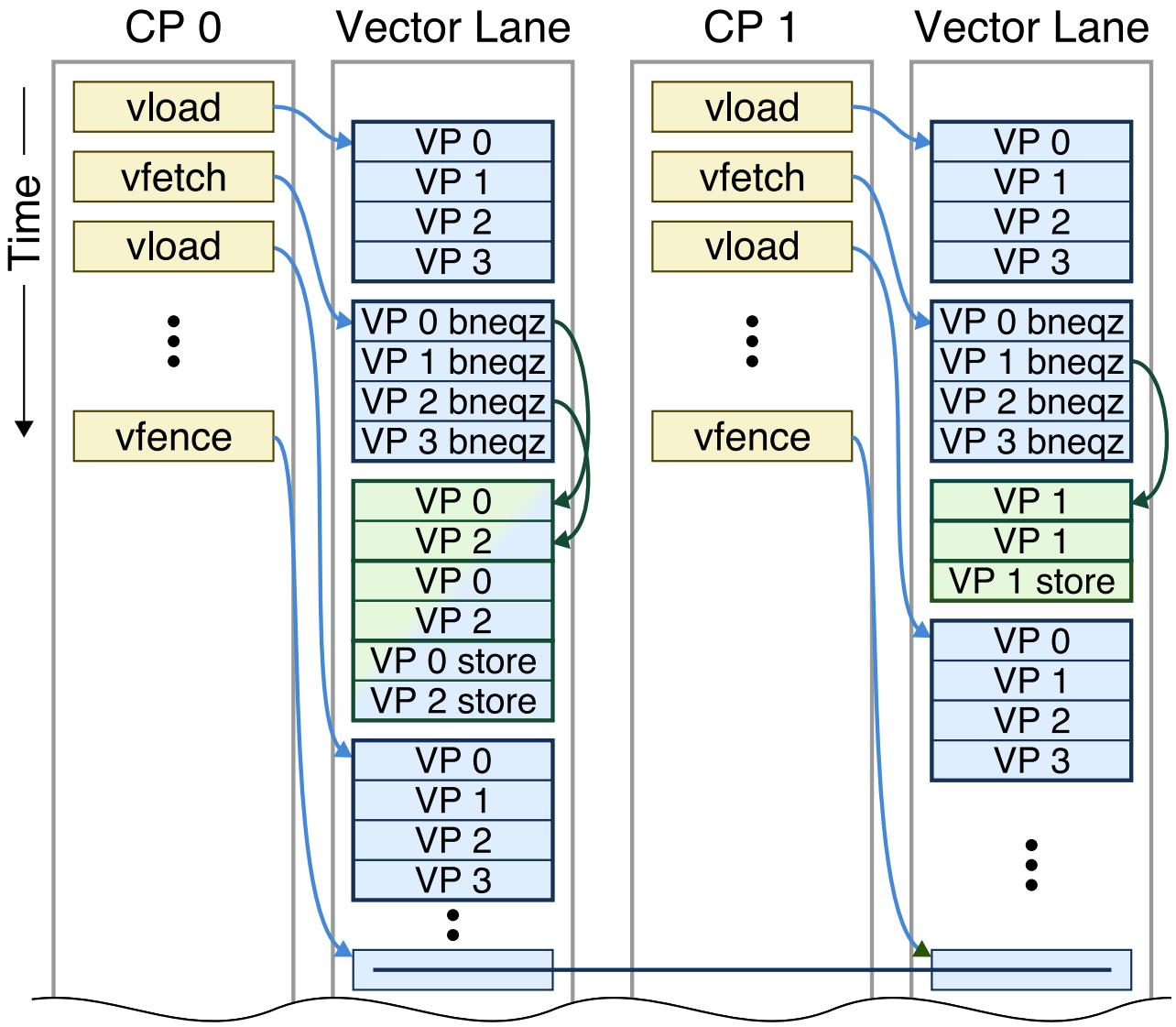
```
for each stage
 for num_elements/2
  load two elements
  butterfly
  store two elements
```



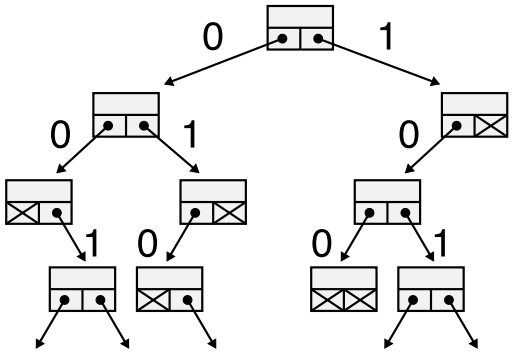
Executing Dither on Maven



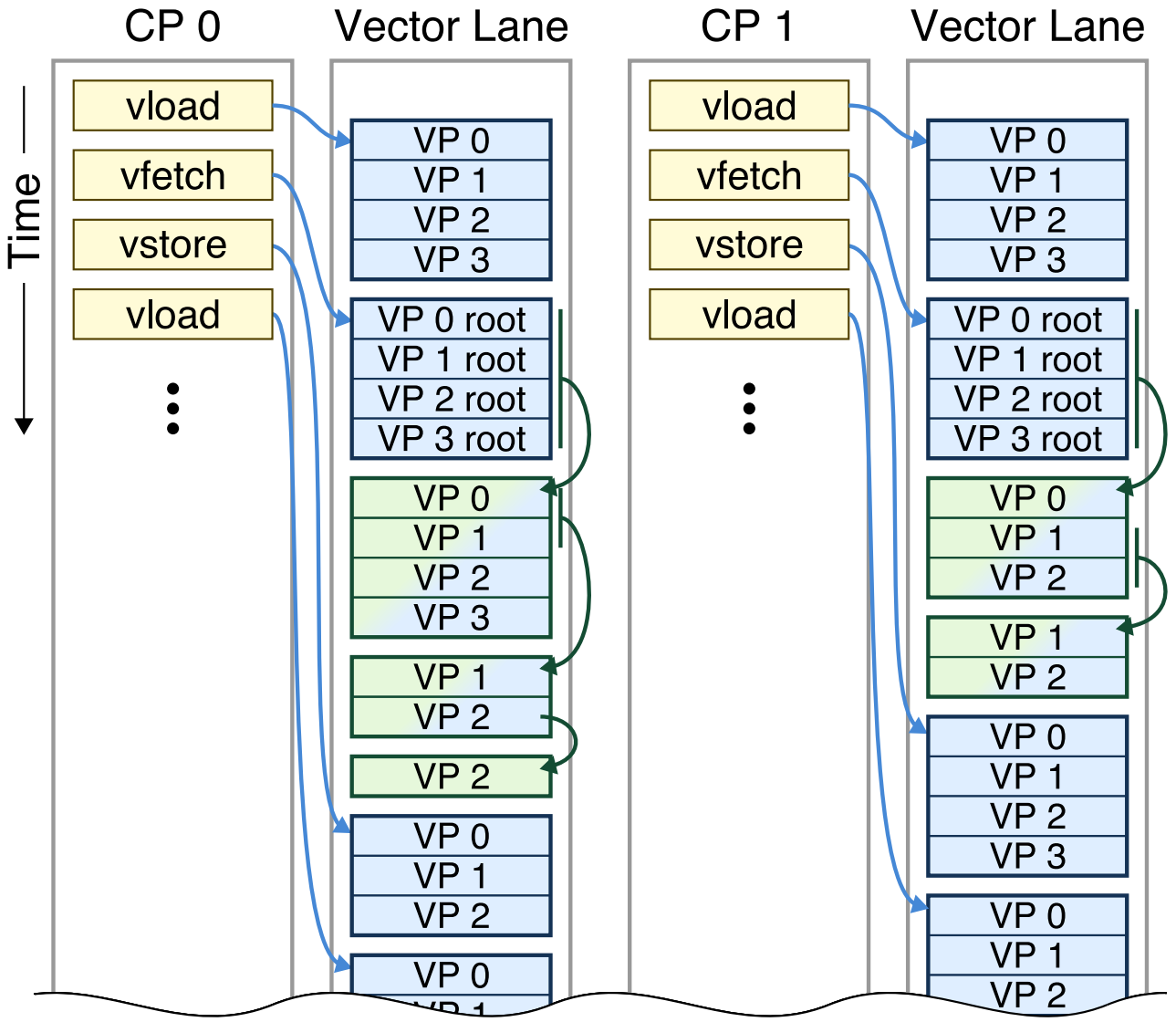
```
for each row
 for each column
  load 3 pixels
  load error value
  if ( pixel != white )
  {
   quantize to 1 bit
   calculate error
   store output pixel
   store error values
  }
```



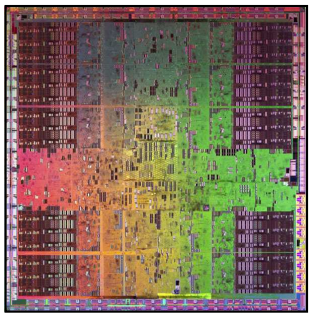
Executing IP Lookup on Maven




```
for each ipaddr
 while ( !found )
 {
  route = node->route
  node =
   (ipaddr[bit++] == 0)
     ? node->left
     : node->right
  found = (node != NULL)
 }
 store route
```

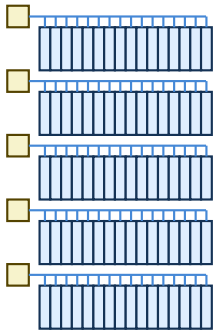


Related Work



Nvidia General-Purpose Graphics Processing Units

- Hardware support for VP divergence/reconvergence
- Instruction exec time always proportional to vector length
- Dynamically coalesce VP memory ops into vectors
- Control processor is not exposed to programmer



Intel Larrabee Graphics Processing Unit

- Tens of exposed CPs each with 16-lane vector unit
- Shared memory with hardware-coherent caches
- Vectors mapped spatially
- Multithreaded control processor
- No VP control flow other than predication

The Maven VT Processor Team at UC Berkeley

Project Supervisor

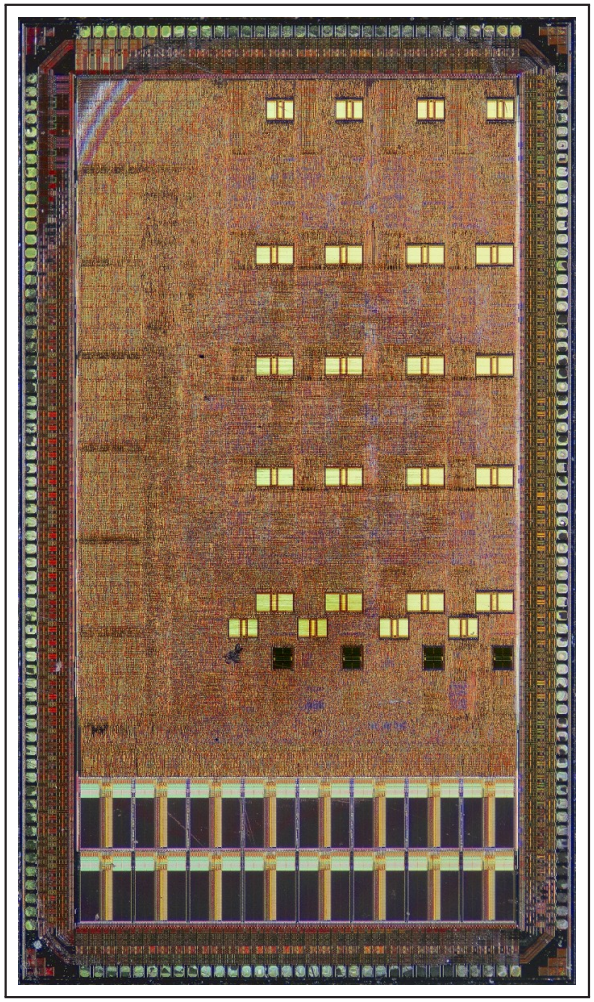
Professor Krste Asanović

Maven Architecture Development

Christopher Batten

Dr. Hidetaka Aoki (visitor from Hitachi)

Yunsup Lee



Motivation

Vector-Thread Architectures

Scale VT Processor

Maven VT Processor

Future Research Directions

Techniques which reduce the energy per operation *and* simplifying manycore parallel programming

Core Architecture

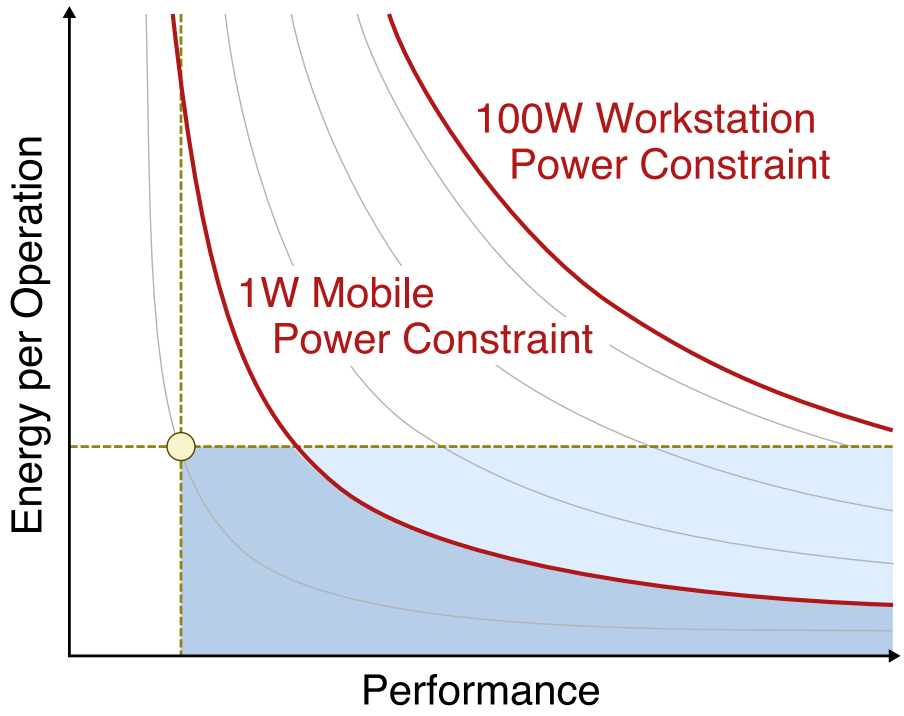
- Hardware support for managing cooperating control processors

On-Chip Network Architecture

- High-radix topologies
- Custom circuits for specialized networks (e.g. global barriers)

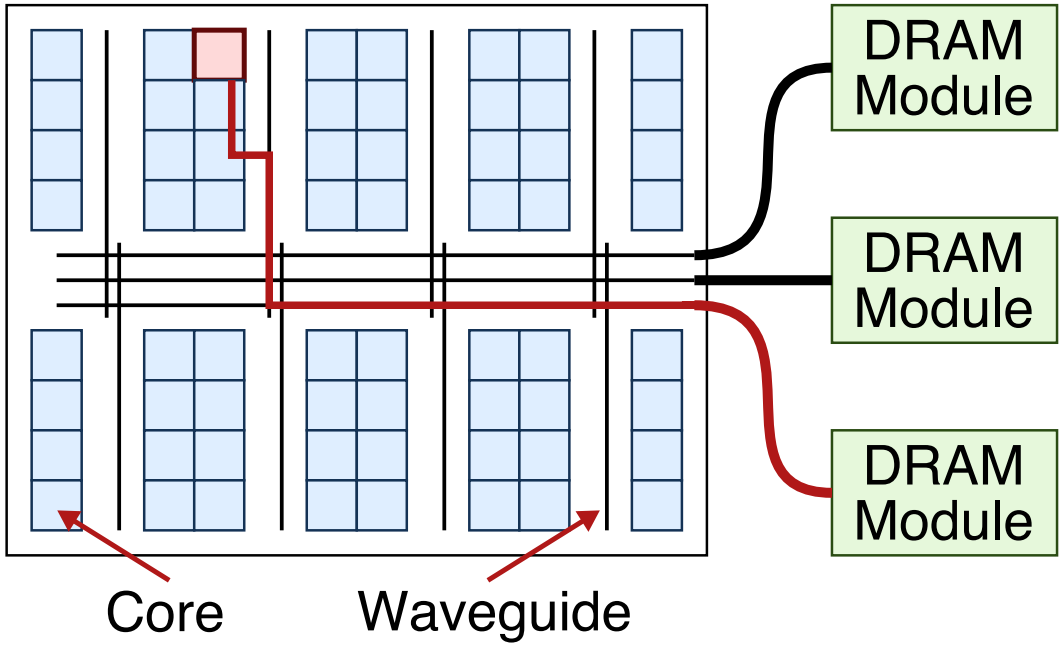
Leveraging Emerging Technologies

- 3D stacking
- Silicon photonics

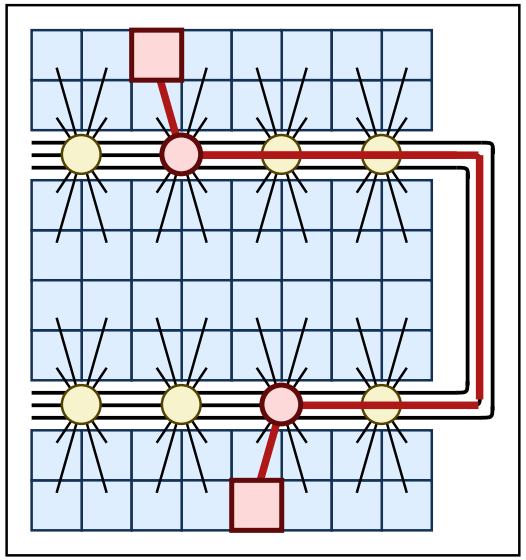


Silicon Photonic Chip-Level Networks

Silicon Photonic Core-to-DRAM Network

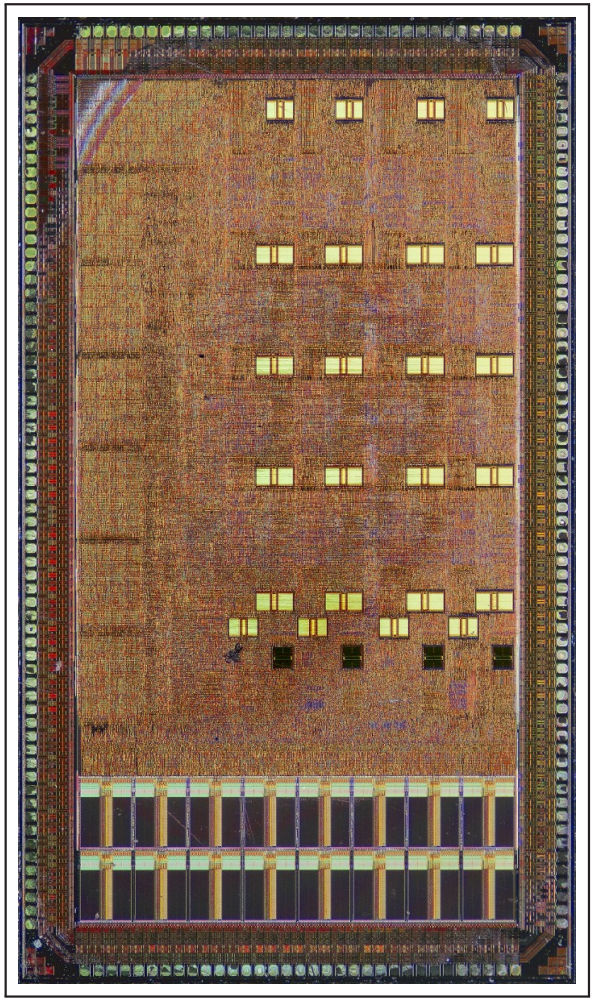


Silicon Photonic Core-to-Core Network



Symposium on High-Performance Interconnects (Hot Interconnects), 2008
International Symposium on Networks-on-Chip (NOCS), 2009

Conclusions



- Manycore is not enough – architects should work towards reducing the energy per operation while also improving performance.
- Vector-Threading combines the energy-efficiency of vector execution with the flexibility of multithreaded execution.
- The Scale VT Processor is a first implementation which illustrates the advantages of vector-threading, and the Maven VT Processor will extend vector-threading to tens or hundreds of lanes on a chip.