# Lecture 17
# Case Studies

EEC 171 Parallel Architectures

John Owens
UC Davis

# Credits

- © John Owens / UC Davis 2007–17.
- Thanks to many sources for slide material: Computer Organization and Design (Patterson & Hennessy) © 2005, Computer Architecture (Hennessy & Patterson) © 2007, Inside the Machine (Jon Stokes) © 2007, © Dan Connors / University of Colorado 2007, © Kathy Yelick / UCB 2007, © Wen-Mei Hwu/David Kirk, University of Illinois 2007, © David Patterson / UCB 2003–7, © John Lazzaro / UCB 2006, © Mary Jane Irwin / Penn State 2005, © John Kubiatowicz / UCB 2002, © Krste Asinovic/Arvind / MIT 2002, © Morgan Kaufmann Publishers 1998.

# Credits

- Additional material in this lecture courtesy Brucek Khailany and Bill Dally, Stream Processors Inc. (SPI Storm), Steve Keckler, The University of Texas (TRIPS), Anant Agarwal, MIT (RAW), Anjul Patney, UC Davis (Larrabee), and Matt Pharr, Intel (ISPC). Blue Gene, Titan, and ARM slides procured from their manufacturers. Haswell material is from an excellent David Kanter article in Ars Technica.

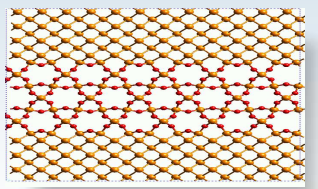
# Examples of Applications Running on Blue Gene

## Developed on L, P; many ported to Q

| Application | Owner | Application | Owner | Application | Owner |
|---|---|---|---|---|---|
| CFD Alya System | Barcelona SC | DFT iGryd | Jülich | BM: SPEC2006, SPEC openmp | SPEC |
| CFD (Flame) AVBP | CERFACS Consortium | DFT KKRnano | Jülich | BM: NAS Parallel Benchmarks | NASA |
| CFD dns3D | Argonne National Lab | DFT ls3df | Argonne National Lab | BM: RZG (AIMS,Gadget,GENE, GROMACS,NEMORB,Octopus, Vertex) | RZG |
| CFD OpenFOAM | SGI | DFT PARATEC | NERSC / LBL | Coulomb Solver - PEPC | Jülich |
| CFD NEK5000, NEKTAR | Argonne, Brown U | DFT CPMD | IBM/Max Planck | MPI PALLAS | UCB |
| CFD OVERFLOW | NASA, Boeing | DFT QBOX | LLNL | Mesh AMR | CCSE, LBL |
| CFD Saturne | EDF | DFT VASP | U Vienna & Duisburg | PETSC | Argonne National Lab |
| CFD LBM | Erlanger-Nuremberg | Q Chem GAMESS | Ames Lab/Iowa State | MpiBlast-pio Biology | VaTech / ANL |
| MD Amber | UCSF | Nuclear Physics GFMC | Argonne National Lab | RTM – Seismic Imaging | ENI |
| MD Dalton | Univ Oslo/Argonne | Neutronics SWEEP3D | LANL | Supernova Ia FLASH | Argonne National Lab |
| MD ddcMD | LLNL | QCD CPS | Columbia U/IBM | Ocean HYCOM | NOPP / Consortium |
| MD LAMMPS | Sandia National Labs | QCD MILC | Indiana University | Ocean POP | LANL/ANL/NCAR |
| MD MP2C | Jülich | Plasma GTC | PPPL | Weather/Climate CAM | NCAR |
| MD NAMD | UIUC/NCSA | Plasma GYRO (Tokamak) | General Atomics | Weather/Climate Held-Suarez Test | GFDL |
| MD Rosetta | U Washington | KAUST Stencil Code Gen | KAUST | Climate HOMME | NCAR |
| DFT GPAW | Argonne National Lab | BM:sppm,raptor,AMG,IRS,sphot | Livermore | Weather/Climate WRF, CM1 | NCAR, NCSA |

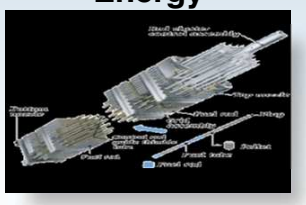
***Accelerating Discovery and Innovation in:***

**Materials Science**



Silicon Design

**Energy**

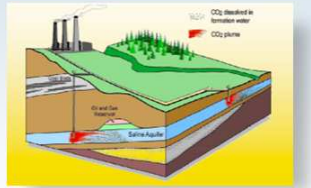


Next Gen Nuclear

**Engineering**

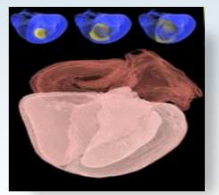
High Efficiency Engines

**Climate & Environment**



Oil Exploration

**Life Sciences**

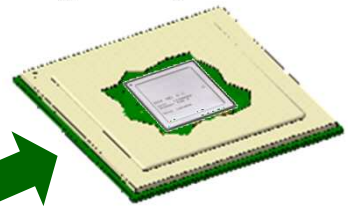


Whole Organ Simulation

# Blue Gene/Q

1. Chip:
16+2 $\mu$P
cores

2. Single Chip Module



3. Compute card:
One chip module,
16 GB DDR3 Memory,
Heat Spreader for $H_2O$ Cooling

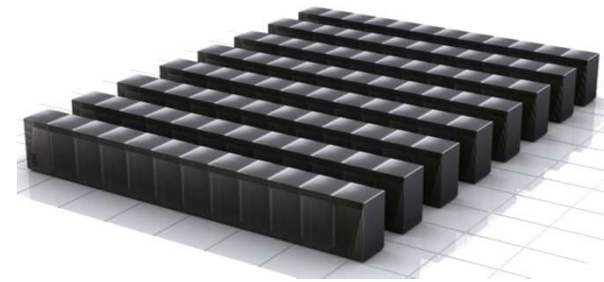
4. Node Card:
32 Compute Cards,
Optical Modules, Link Chips; 5D Torus

5a. Midplane:
16 Node Cards

5b. IO drawer:
8 IO cards w/16 GB
8 PCIe Gen2 x8 slots
3D I/O torus

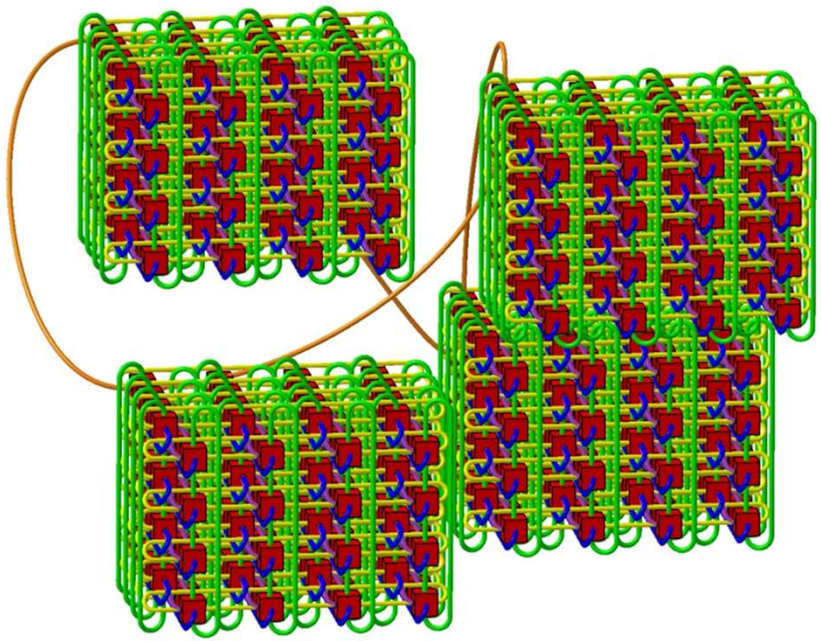
6. Rack: 2 Midplanes

7. System:
96 racks, 20PF/s



**•Sustained single node perf: 10x P, 20x L**
**• MF/Watt: (6x) P, (10x) L (~2GF/W, Green 500 criteria)**
**• Software and hardware support for programming models for exploitation of node hardware concurrency**

# Inter-Processor Communication



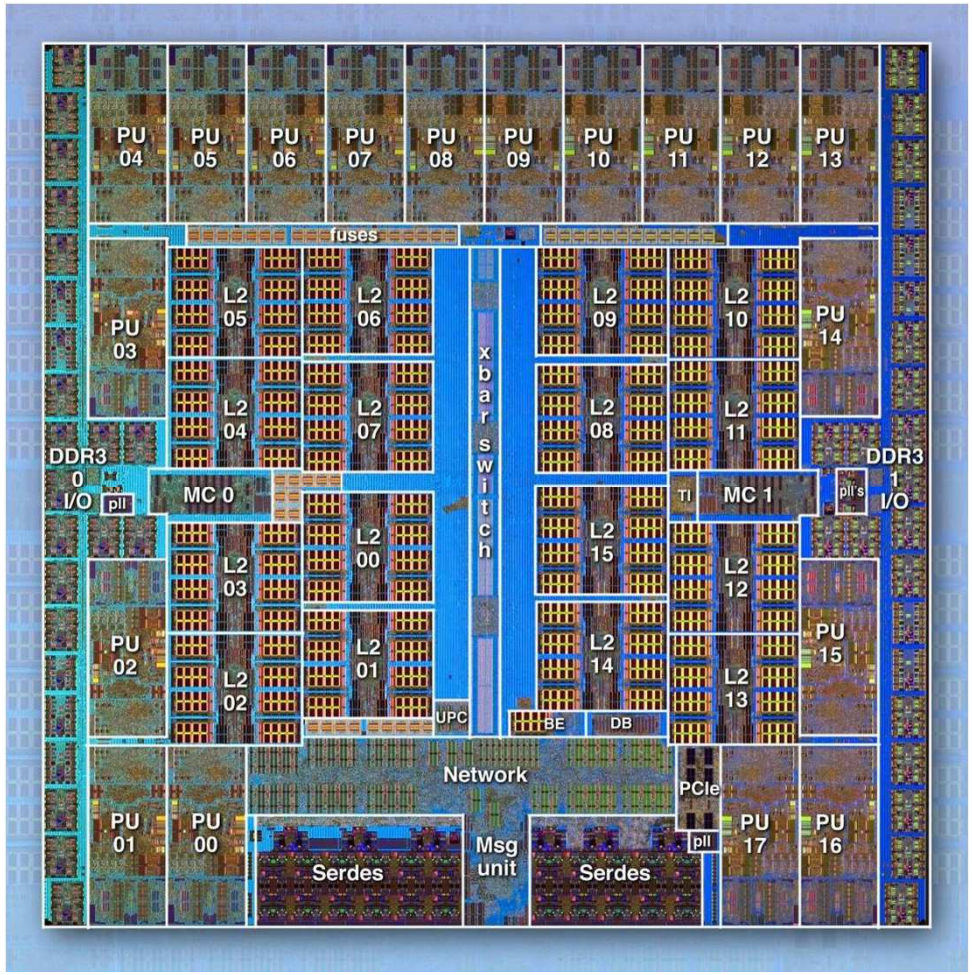
- **Integrated 5D torus**
  - Virtual Cut-Through routing
  - Hardware assists for collective & barrier functions
  - FP addition support in network
  - RDMA
    - Integrated on-chip Message Unit
- **2 GB/s raw bandwidth on all 10 links**
  - each direction -- i.e. 4 GB/s bidi
  - 1.8 GB/s user bandwidth
    - protocol overhead
- **5D nearest neighbor exchange measured at 1.76 GB/s per link (98% efficiency)**
- **Hardware latency**
  - Nearest: 80ns
  - Farthest: 3us
    (96-rack 20PF system, 31 hops)
- **Additional 11th link for communication to IO nodes**
  - BQC chips in separate enclosure
  - IO nodes run Linux, mount file system
  - IO nodes drive PCIe Gen2 x8 (4+4 GB/s)
    ↔ IB/10G Ethernet ↔ file system & world

## Network Performance

- **All-to-all: 97% of peak**
- **Bisection: > 93% of peak**
- **Nearest-neighbor: 98% of peak**
- **Collective: FP reductions at 94.6% of peak**

# BlueGene/Q Compute chip

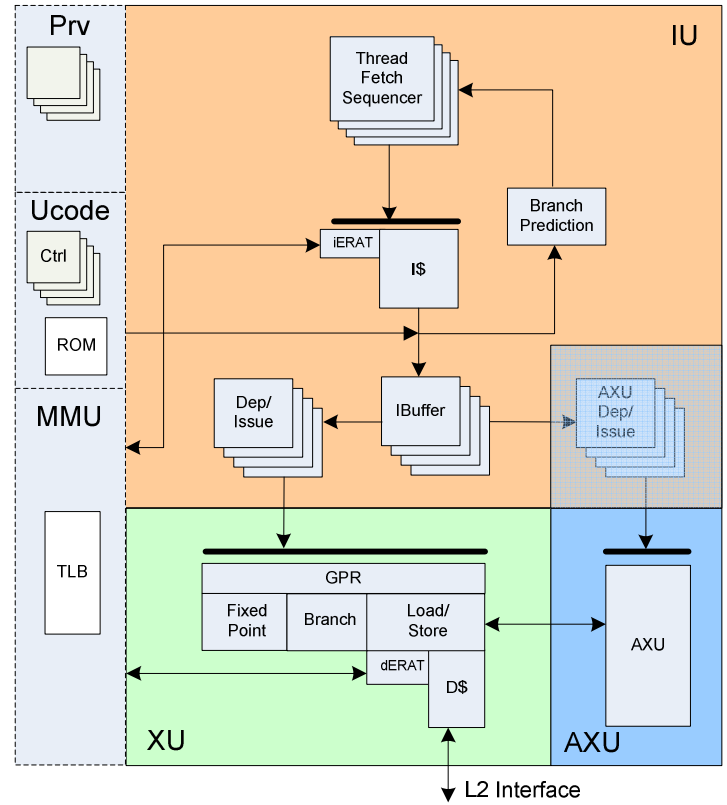
## System-on-a-Chip design : integrates processors, memory and networking logic into a single chip



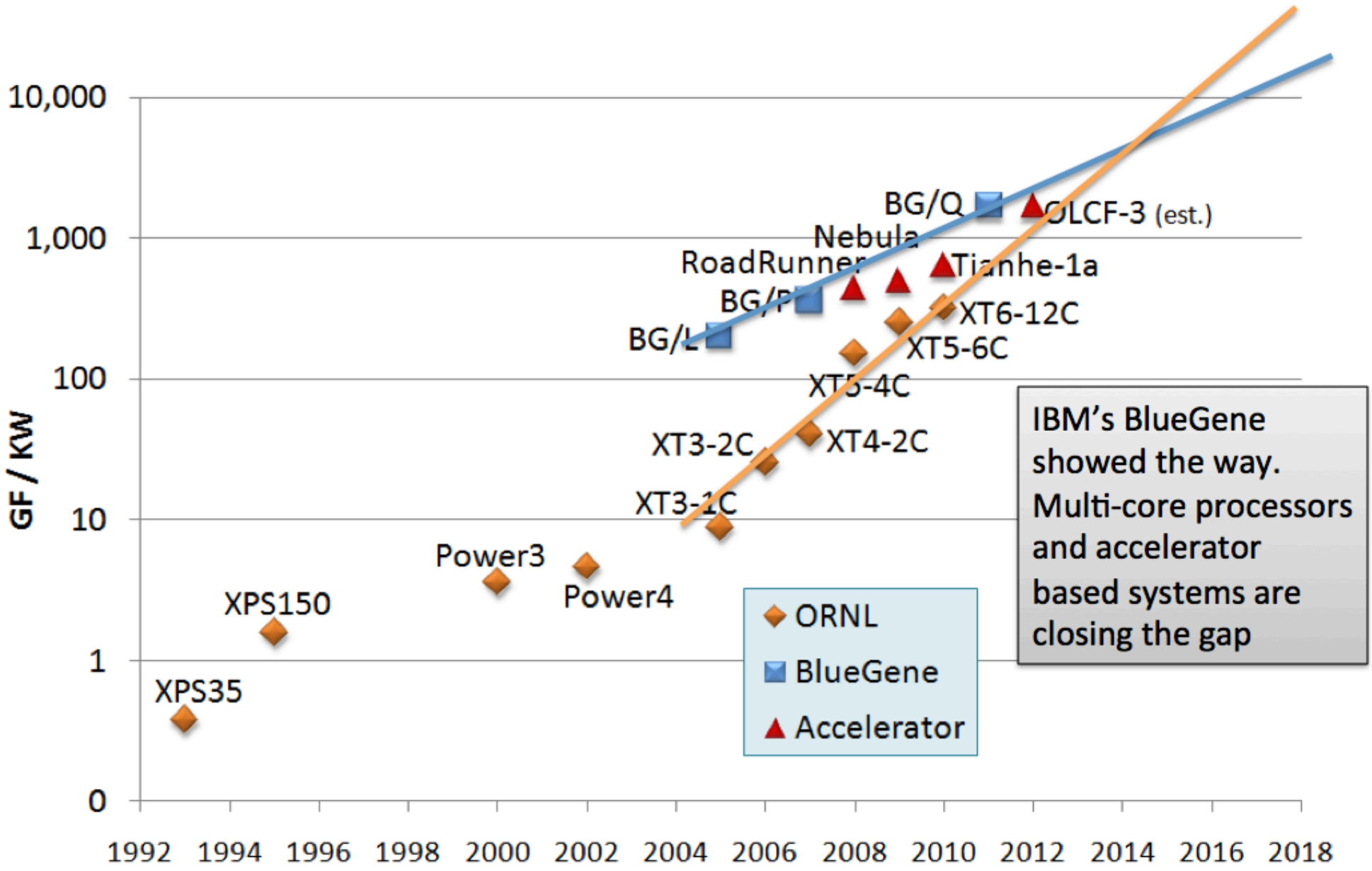

- 360 mm² Cu-45 technology (SOI)
- 16 user + 1 service PPC processors
  - plus 1 redundant processor
  - all processors are symmetric
  - 11 metal layer
  - each 4-way multi-threaded
  - 64 bits
  - 1.6 GHz
  - L1 I/D cache = 16kB/16kB
  - L1 prefetch engines
  - each processor has Quad FPU (4-wide double precision, SIMD)
  - peak performance 204.8 GFLOPS @ 55 W
- Central shared L2 cache: 32 MB
  - eDRAM
  - multiversioned cache – supports transactional memory, speculative execution.
  - supports scalable atomic operations
- Dual memory controller
  - 16 GB external DDR3 memory
  - 42.6 GB/s DDR3 bandwidth (1.333 GHz DDR3) (2 channels each with chip kill protection)
- Chip-to-chip networking
  - 5D Torus topology + external link
    → 5 x 2 + 1 high speed serial links
  - each 2 GB/s send + 2 GB/s receive
  - DMA, remote put/get, collective operations
- External (file) IO -- when used as IO chip.
  - PCIe Gen2 x8 interface (4 GB/s Tx + 4 GB/s Rx)
  - re-uses 2 serial links
  - interface to Ethernet or Infiniband cards

# BG/Q processor unit (A2 core)

- **Mostly same design as in PowerEN™ chip: Simple core, designed for excellent power efficiency and small footprint.**
- **Implemented 64-bit PowerISA™ v2.06**
- **1.6 GHz @ 0.8V.**
- **32x4x64 bit GPR**
- **4-way Simultaneous Multi- Threading**
- **2-way concurrent issue 1 XU + 1 AXU**
- **AXU port allows for unique BGQ style floating point**
- **In-order execution**
- **Dynamic branch prediction**

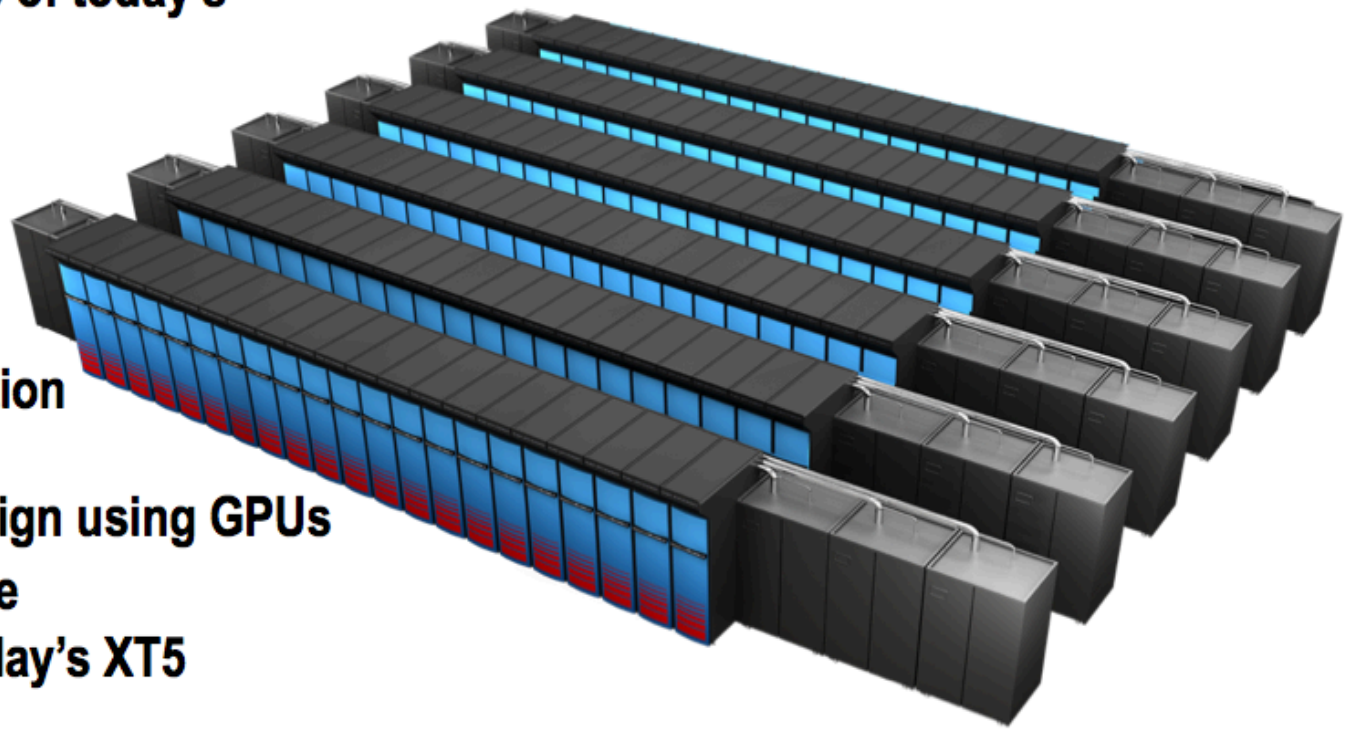


# Trends in power efficiency



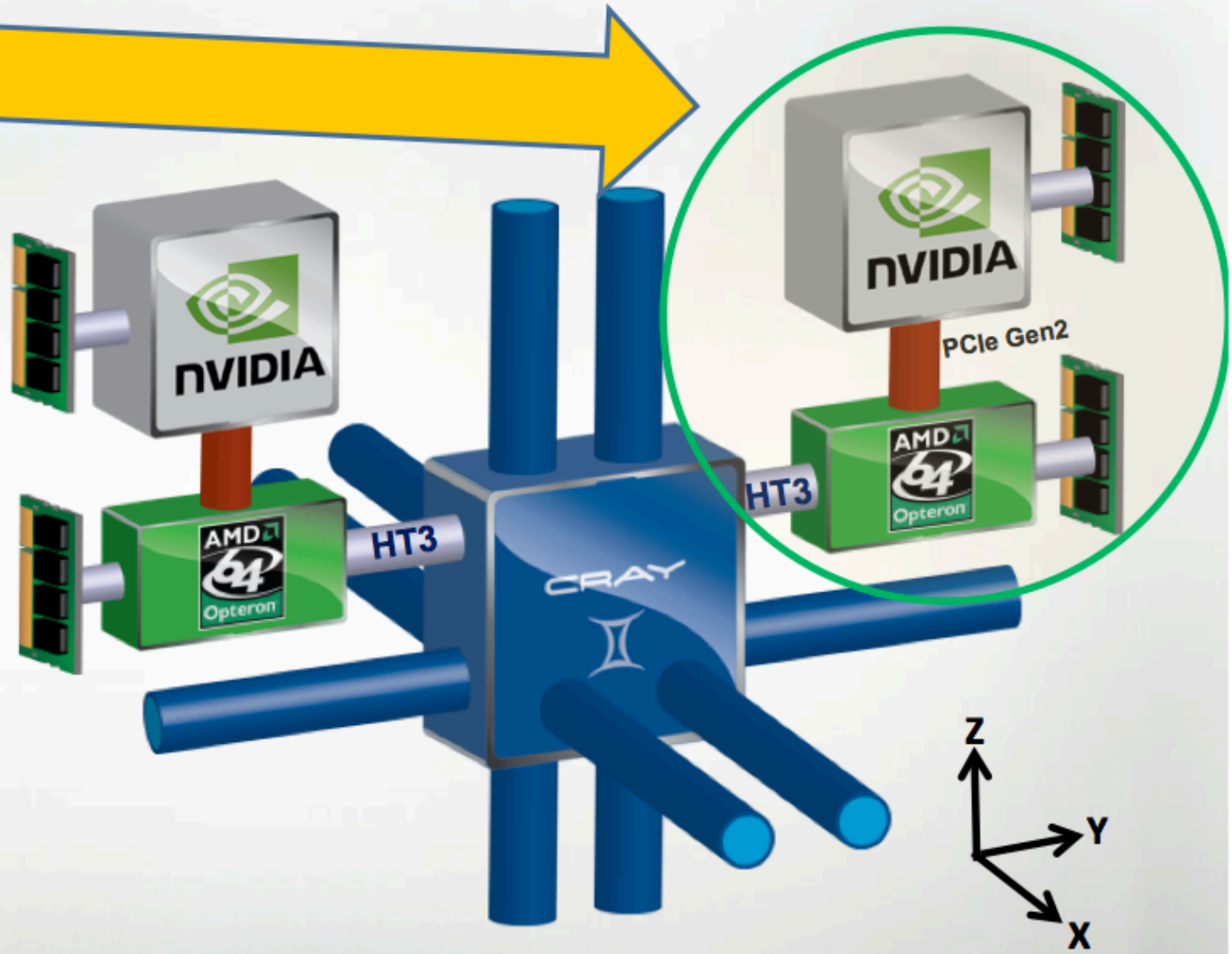
# ORNL's "Titan" 20 PF System Goals

- Designed for science from the ground up
- Operating system upgrade of today's Linux Operating System
- Gemini interconnect
  - 3-D Torus
  - Globally addressable memory
  - Advanced synchronization features
- New accelerated node design using GPUs
- 10-20 PF peak performance
  - 9x performance of today's XT5
- Larger memory
- 3x larger and 4x faster file system



# Cray XK6 Compute Node

| XK6 Compute Node Characteristics |
| --- |
| AMD Opteron 6200 Interlagos 16 core processor |
| Tesla X2090 @ 665 GF |
| Host memory 16 or 32GB 1600 MHz DDR3 |
| Tesla X090 memory 6GB GDDR5 capacity |
| Gemini high speed Interconnect |
| Upgradeable to NVIDIA's Kepler many-core processor |

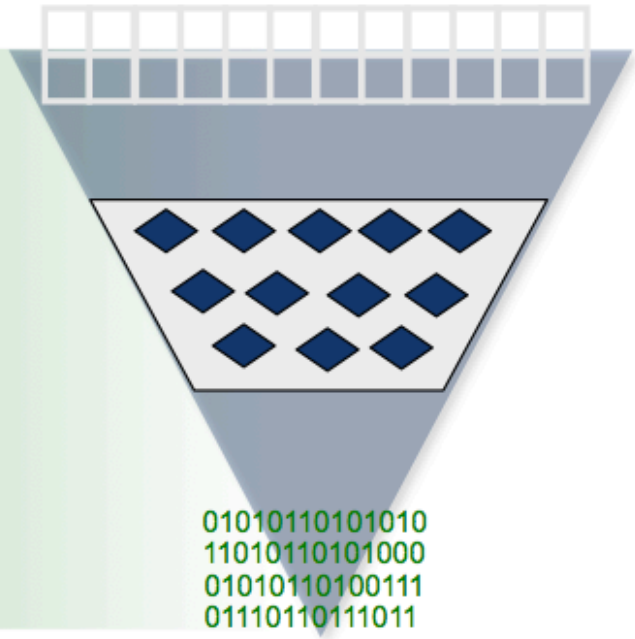


# ORNL Titan Specs

| **Titan System Configuration** | |
|---|---|
| Architecture: | Cray XK7 |
| Processor: | 16-Core AMD |
| Cabinets: | 200 |
| Nodes: | 18,688 AMD Opterons |
| Cores/node: | 16 |
| Total cores: | 299,008 Opteron Cores |
| Memory/node: | 32GB + 6GB |
| Memory/core: | 2GB |
| Interconnect: | Gemini |
| GPUs: | 18,688 K20 Keplers |
| Speed: | 20+ PF |
| Square Footage | 4,352 sq feet |

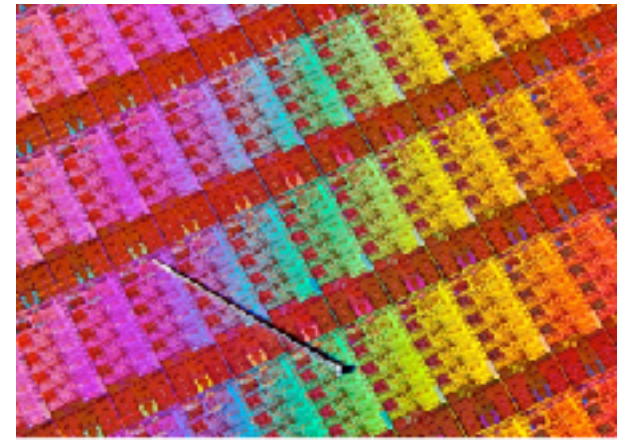
# Hierarchical Parallelism

- MPI parallelism between nodes (or PGAS)
- On-node, SMP-like parallelism via threads (or subcommunicators, or…)
- Vector parallelism
  - SSE/AVX on CPUs
  - GPU threaded parallelism




- **Exposure of unrealized parallelism is essential to exploit all near-future architectures.**
- **Uncovering unrealized parallelism and improving data locality improves the performance of even CPU-only code.**

# Intel Haswell



- Compared to Ivy Bridge (expected):
  - Twice the vector processing performance.
  - A 10% increase in sequential CPU performance (8 execution ports per core versus 6).
  - Up to double the performance of the integrated GPU. (Haswell GT3e vs Ivy Bridge HD4000)

http://arstechnica.com/gadgets/2013/05/a-look-at-haswell/

# Intel Haswell

- Instruction set extensions:
  - AVX2 widens integer SIMD (Single Instruction Multiple Data, a form of vectors) to 256-bit vectors, and that adds a gather instruction for irregular memory access.
  - The fused multiply-add (FMA) instructions improve performance for floating point (FP) workloads.
  - For cryptography, networking, and certain search operations, there are new bit manipulation instructions.
  - Lastly, Haswell is the first widely available product with transaction memory through the TSX extension.

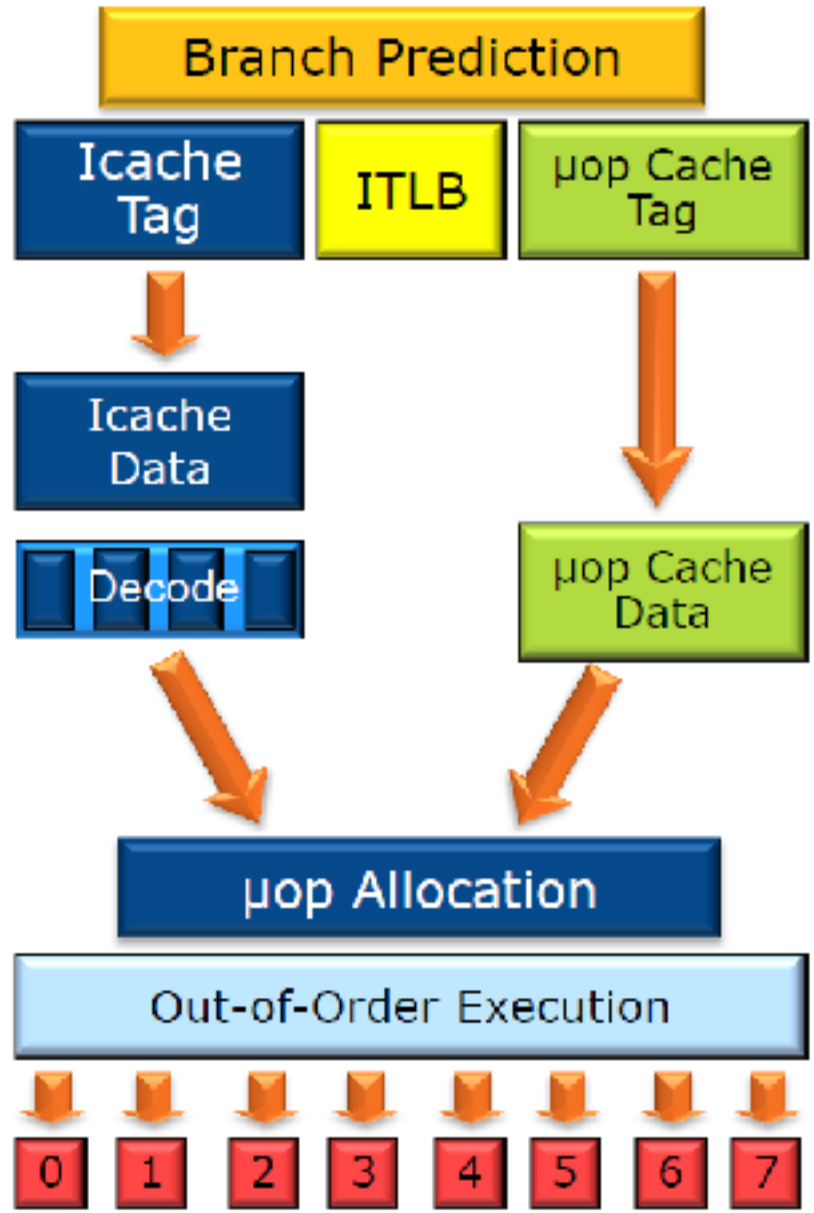
```
def transfer_money(from_account, to_account, amount):
    with transaction():
        from_account -= amount
        to_account += amount
```

http://arstechnica.com/gadgets/2013/05/a-look-at-haswell/

# Haswell front end

The decoding for Haswell is largely identical to Sandy Bridge. There are four legacy decoders that take in x86 instructions and emit simpler uops. The first is a complex one that can emit 1–4 fused uops and three simple decoders that can emit one fused uop each. Like Sandy Bridge, there is compare+jump fusion and stack pointer elimination. The Haswell uop cache is also identical, with 32 sets of eight cache lines. Each cache line holds up to six uops.

The Haswell uop queue was redesigned to improve single threaded performance. Sandy Bridge had two 28 entry uop queues, one for each thread. However, in Ivy Bridge the uop queues were combined into a single 56 entry structure. The chief advantage is that when a single thread is executing on Ivy Bridge or Haswell, the entire 56 entry uop buffer is available for loop caching and queuing.



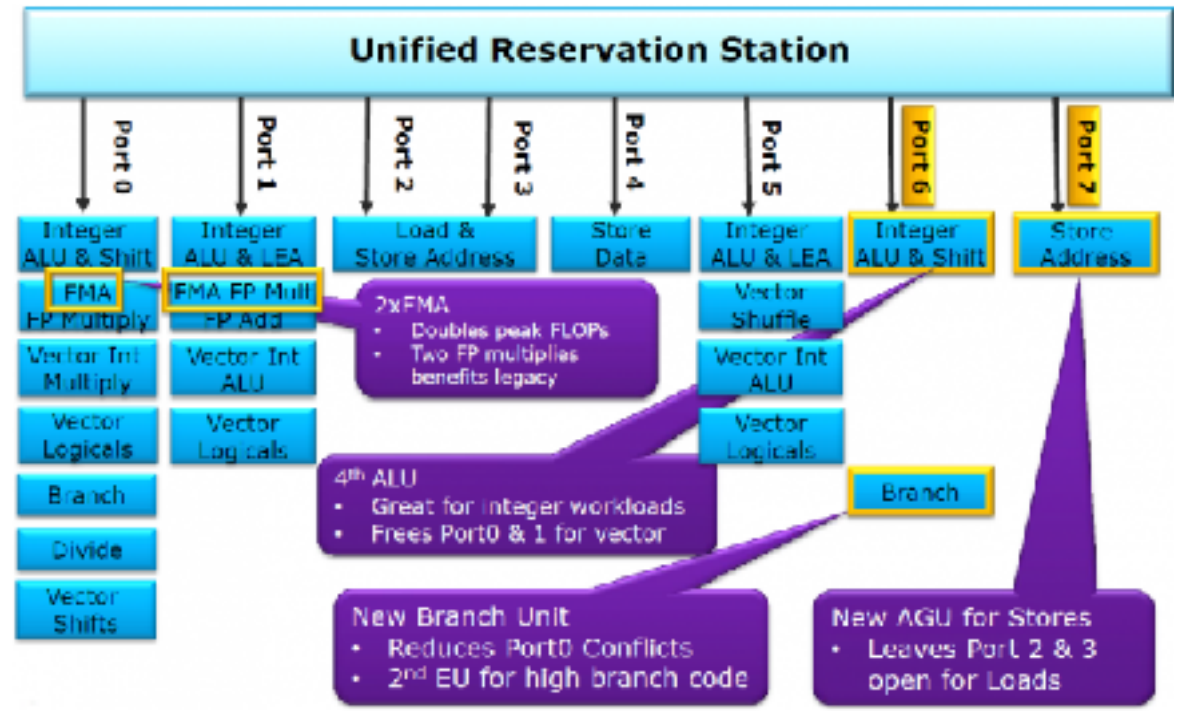
# Haswell core

- “Haswell is substantially wider than Sandy Bridge with more resources for dynamic scheduling, but it retains the same overall organization and function.”

| | Nehalem | Sandy Bridge | Haswell |
|---|---|---|---|
| Out-of-order Window | 128 | 168 | 192 |
| In-flight Loads | 48 | 64 | 72 |
| In-flight Stores | 32 | 36 | 42 |
| Scheduler Entries | 36 | 54 | 60 |
| Integer Register File | N/A | 160 | 168 |
| FP Register File | N/A | 144 | 168 |
| Allocation Queue | 28/thread | 28/thread | 56 |

# Haswell core (2)

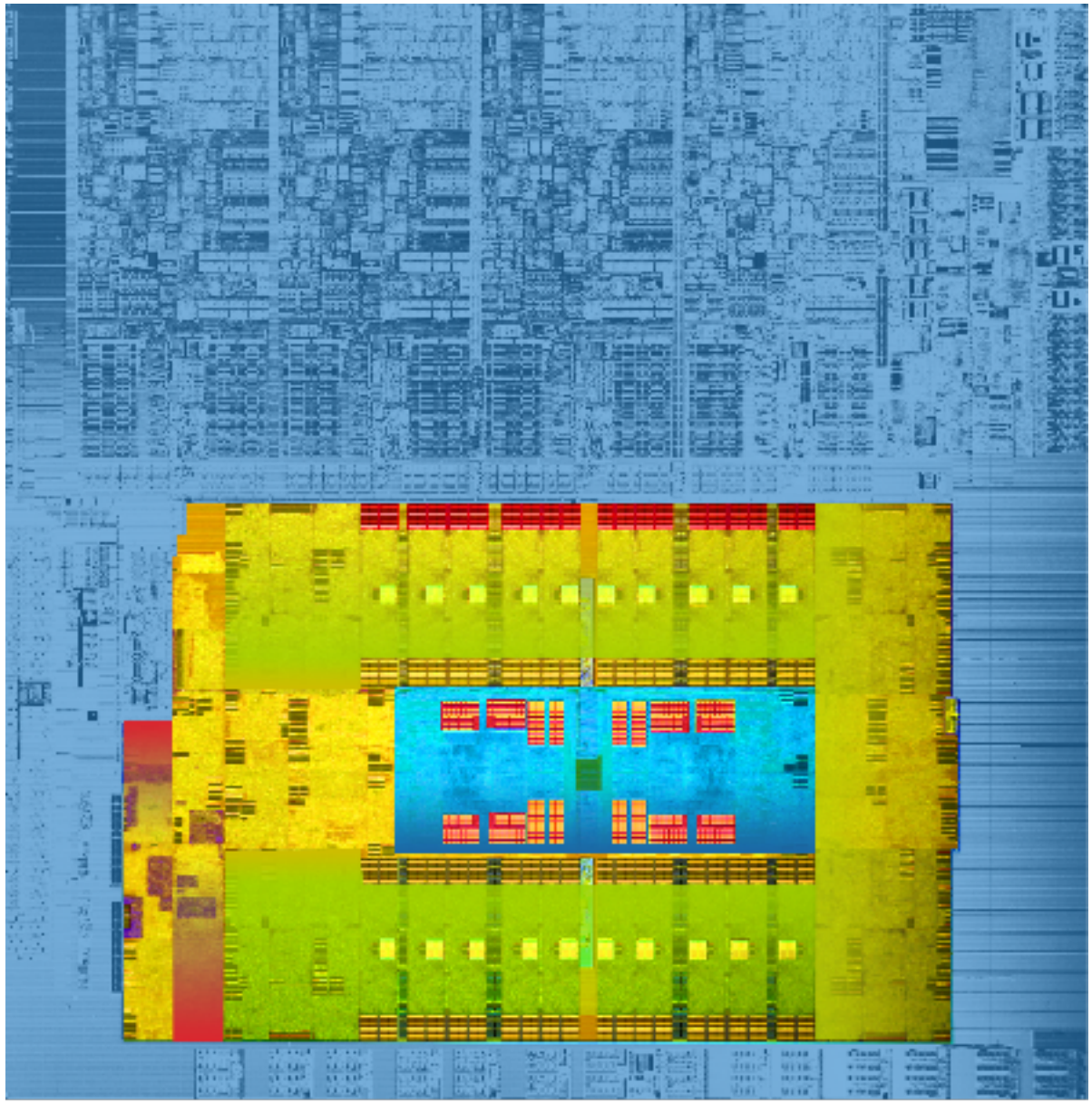
The execution units in Haswell are tremendously improved over Sandy Bridge… Haswell adds an integer dispatch port and a new memory port, bringing the execution to eight uops/cycle. But the biggest changes are to the vector execution units. On the integer SIMD side, the hardware has been extended to single cycle 256-bit execution. For floating point vectors, the big change is 256-bit fused multiply add units for two of the execution ports. As a result, the theoretical peak performance for Haswell is more than double that of Sandy Bridge.




Every cycle, up to eight uops are sent from the unified scheduler to the dispatch ports. As shown in Figure 3, computational uops are dispatched to ports 0, 1, 5, and 6 and executed on the associated execution units. The execution units include three types: integer, SIMD integer, and FP (both scalar and SIMD).

Port 6 on Haswell is a new scalar integer port. It only accesses the integer registers and handles standard ALU (Arithmetic Logic Unit) operations, including shifts and branches that were previously on port 5 (in Sandy Bridge). One of the advantages of the new integer port is that it can handle many instructions while the SIMD dispatch ports are fully utilized.

# “Crystalwell” die photo



# Haswell conclusions (Kanter)

There is much to like in Haswell: a bevy of new instructions, a more powerful microarchitecture, and lower power. For vector friendly workloads, AVX2 brings integer SIMD to 256-bit width—dead even with FP vectors—while doubling the number of FP operations through fused multiply-add (FMA). Crucially, Haswell also includes gather instructions to fetch non-contiguous data from memory, which makes it easier for compilers and programmers to use the x86 SIMD extensions.

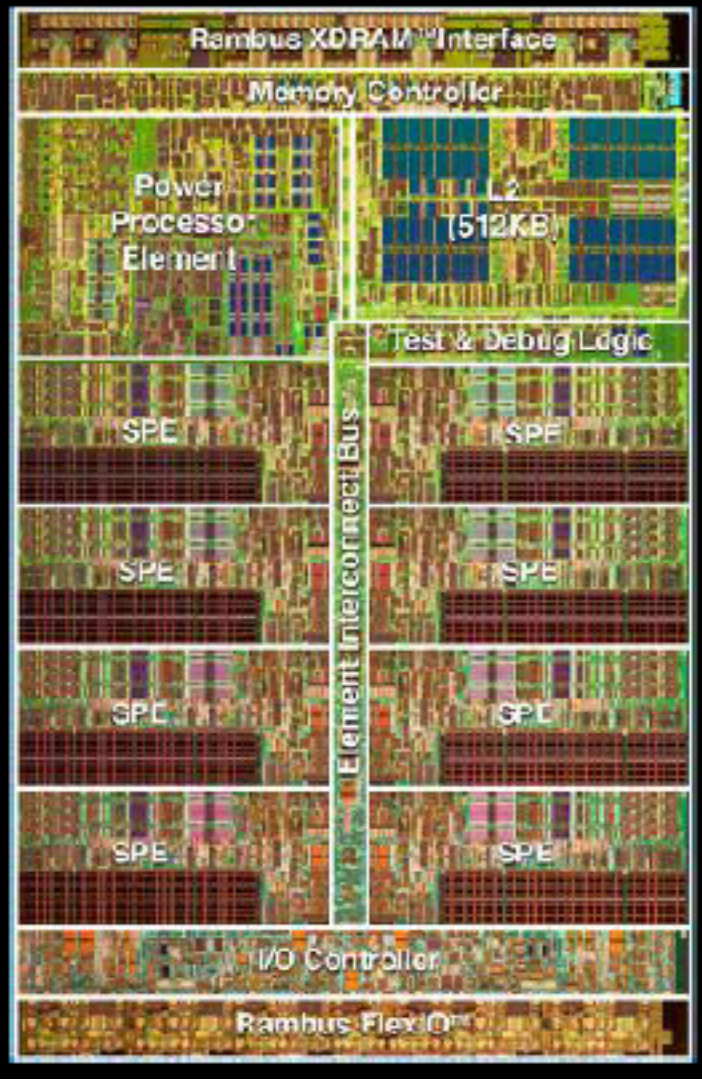
Intel's transactional memory extensions have a smaller impact on performance but have more potential in the long run. The Hardware Lock Elision separates correctness from performance. Software can use simpler techniques such as coarse-grained locks, and the hardware will automatically optimize for performance. Alternatively, Restricted Transactional Memory is an entirely new way to write concurrent code that is far easier and more intuitive, with higher performance to boot.

The Haswell microarchitecture has a modestly larger out-of-order window, with a 33 percent increase in dispatch ports and execution resources. Compared to previous generations, the theoretical FLOPs and integer operations have doubled for each core, primarily due to wider vectors. More significantly, the cache hierarchy can sustain twice the bandwidth, and it has fewer utilization bottlenecks.

As tablets have become more robust and capable, in many respects Intel's competitive focus has shifted away from AMD to the ARM ecosystem: Qualcomm, Samsung, Nvidia, and others. Haswell will be the first high performance x86 core that can really fit in a tablet, albeit in high-powered models around the 10W mark rather than the 4W devices. For consumers, Haswell will offer a heady combination of Windows 8 (and compatibility with the x86 software base) with excellent performance and tablet-like power characteristics. Compared to AMD or ARM-based solutions, the performance will be dramatically higher, so the biggest question will be power efficiency. There, Intel is claiming some impressive advances while keeping the details close, particularly about actual products. But with Computex coming soon, we should soon see Haswell's true colors.

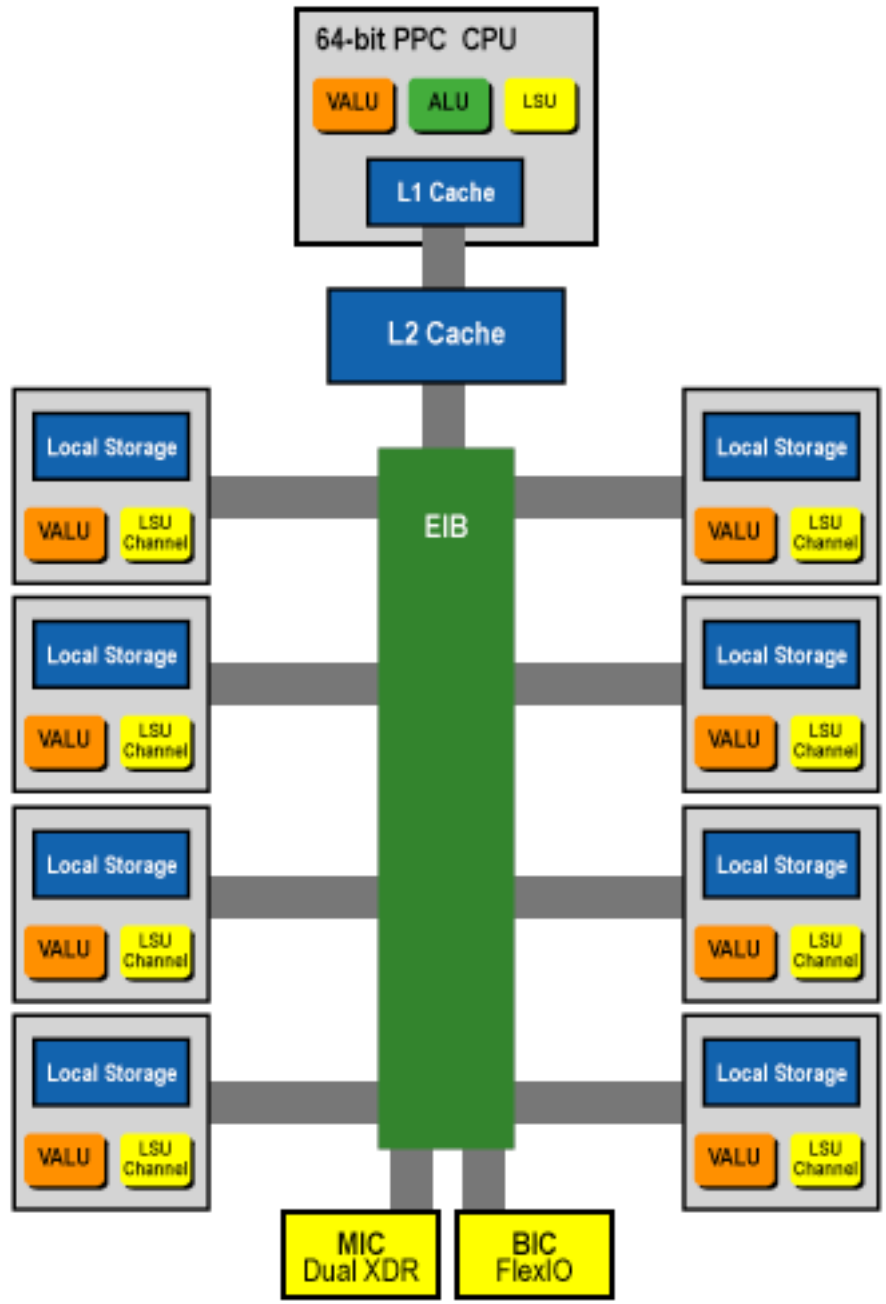
# Highlights (3.2 GHz)

- 241M transistors
- 235mm2
- 9 cores, 10 threads
- >200 GFlops (SP)
- >20 GFlops (DP)
- Up to 25 GB/s memory B/W
- Up to 75 GB/s I/O B/W
- >300 GB/s EIB
- Top frequency >4GHz
  (observed in lab)



# Cell Summary

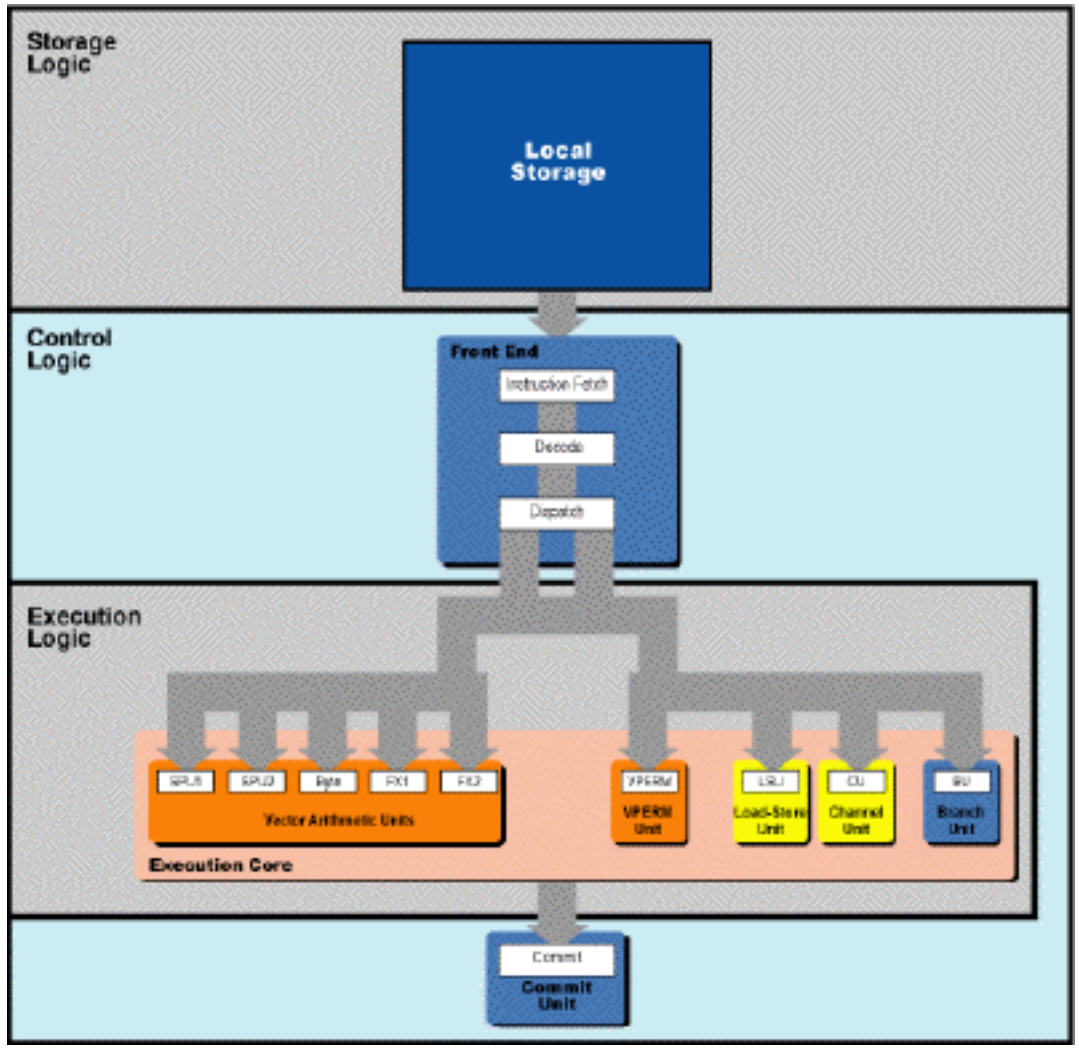
- Target: memory wall
- PPC core
  - 2-way SMT
- 8x SPU
  - User-managed L1
- Element interconnect bus
  - 4x16B rings, 96 B/cycle
  - >100 outstanding requests



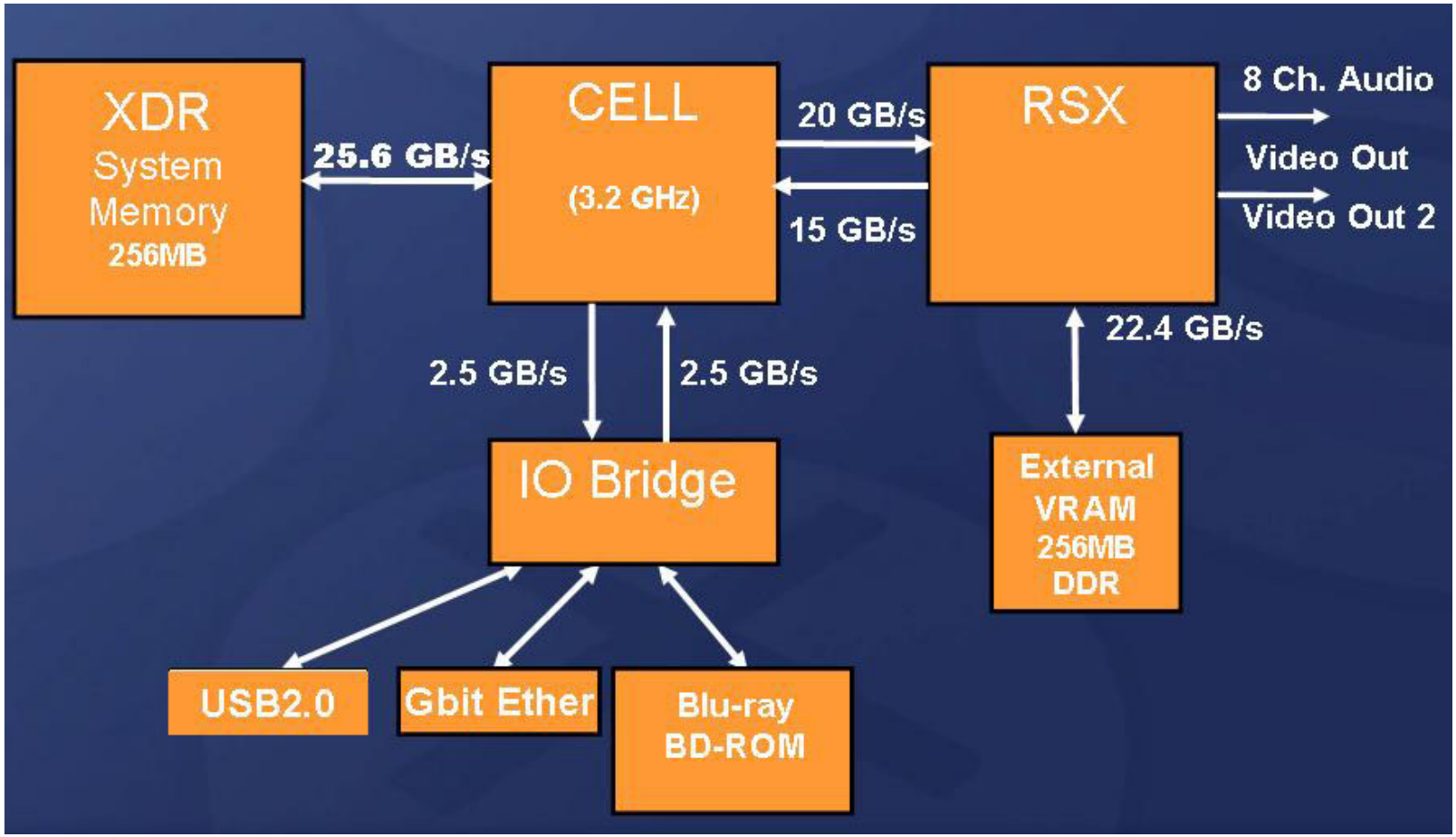

The CELL Architecture

# SPE Detail

- Dual-issue, statically scheduled
  - No OOO, no renaming, no reordering
  - 128 registers
  - 8 port RF (6R, 2W)
- Most instructions operate on 128b 4x SIMD data
- L1 replaced with 256 kB user-managed memory
  - DMA hw primitives
  - 21M xtors (14M SRAM, 7M logic)



# Sony PS3




- CPUs are good at creating & manipulating data structures
- GPUs are good at accessing & updating data structures

http://www.watch.impress.co.jp/game/docs/20060329/3dps303.jpg

# Things that work extremely well today (up to 100x)

- **Problem can be re-coded**
- **Predictable non-trivial memory access pattern**
  - Can build scatter-gather lists
- **Problem can benefit from SIMD**
- **Focus on 32b float, or <=32b integer**

- **Examples:**
  - FFTw ( best result about 100 GFlops )
  - Terrain Rendering Engine
  - Volume rendering

- **Typical code is double-buffered gather-compute-scatter**

# Question marks

- **Can a compiler based approach, without restructuring code specifically for the SPEs result in a chip-level advantage?**
  - About 3-4x more SPEs in same area or power
  - But, have to compiler manage local store
- **Interesting benchmarks: SpecFP, MediaBench, EEMBC, etc.**
  - New more explicitly parallel benchmarks?
- **Would you ever use an SPE for a SpecInt-type workload?**

# The Raw Tiled Processor Architecture

Anant Agarwal

MIT

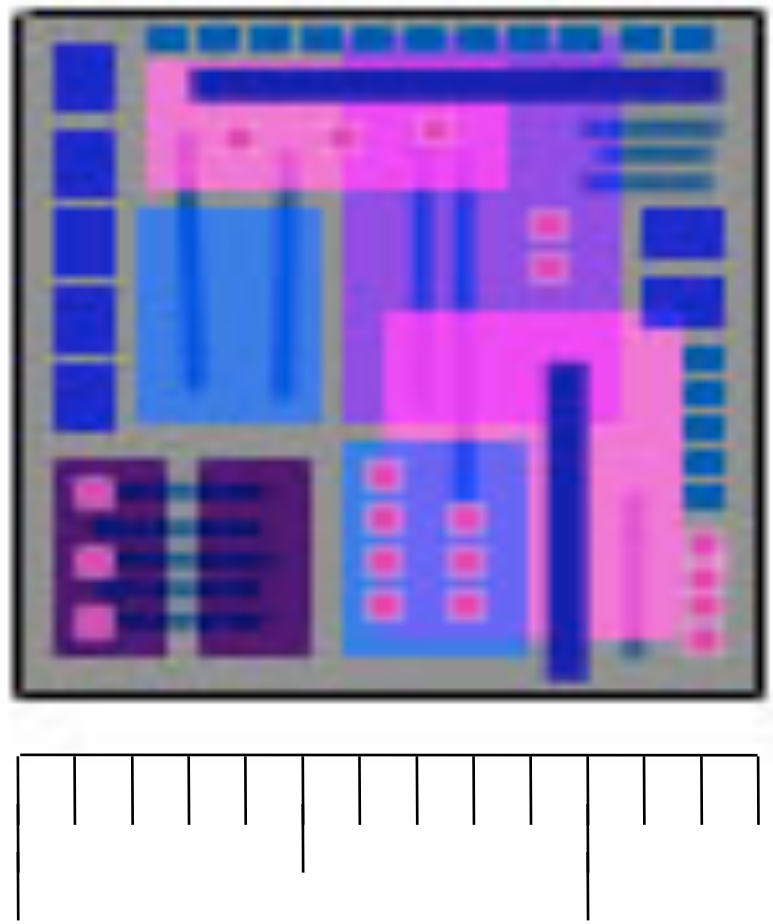
http://www.cag.csail.mit.edu/raw

# A tiled processor architecture prototype: the Raw microprocessor


October 02

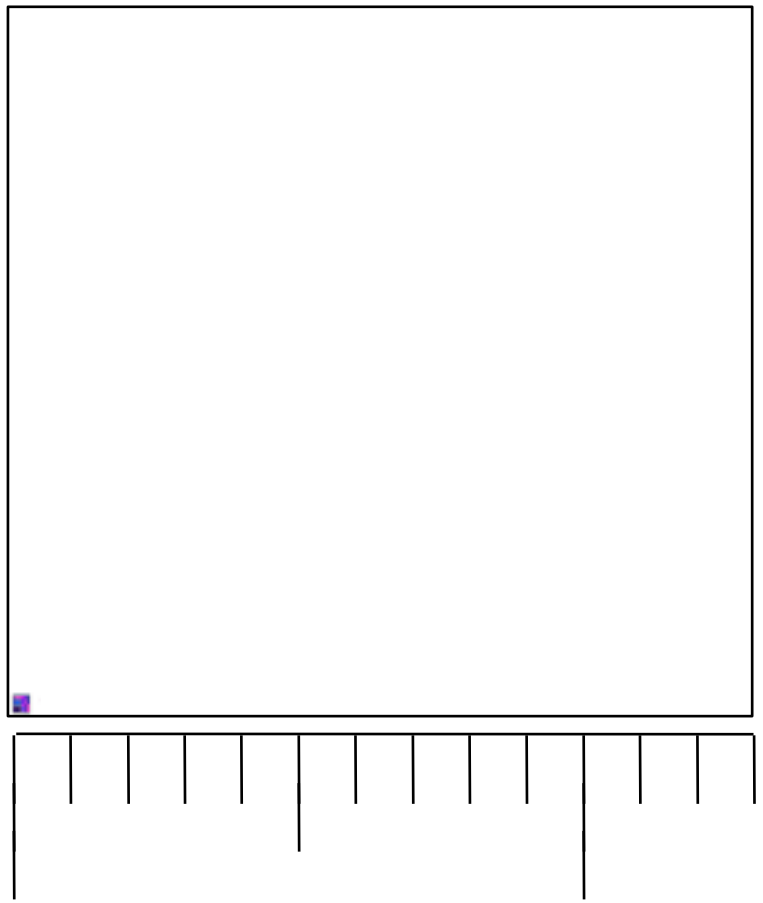
# The opportunity

20MIPS cpu
1987



# The opportunity

2007



The billion transistor chip

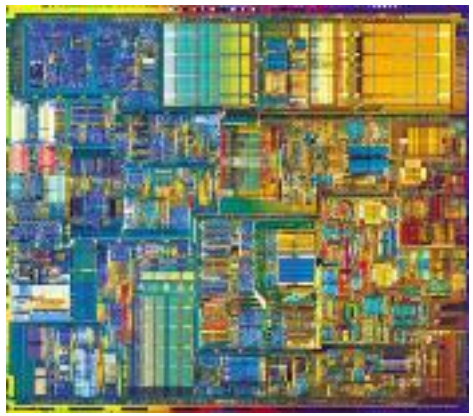
# Enables seeking out new application domains

- Redefine our notion of a "general purpose processor"

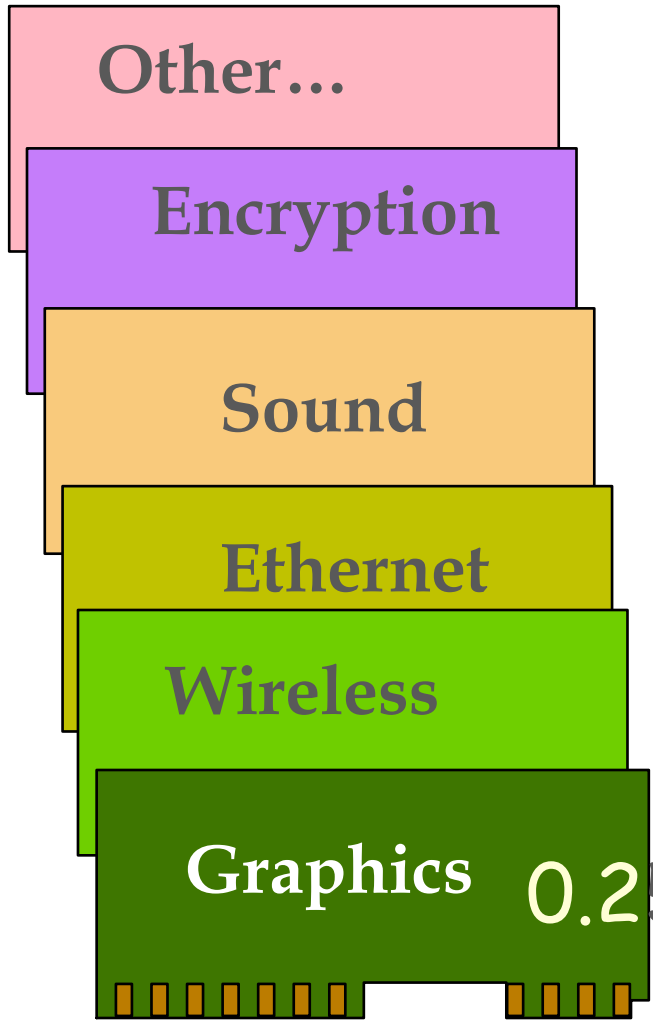
- Imagine a single-chip handheld that is a speech driven cellphone, camera, PDA, MP3 player, video engine
- Imagine a single-chip PC that is also a 10G router, wireless access point, graphics engine

- While running the gamut of existing desktop binaries

# But, where is general purpose computing today?



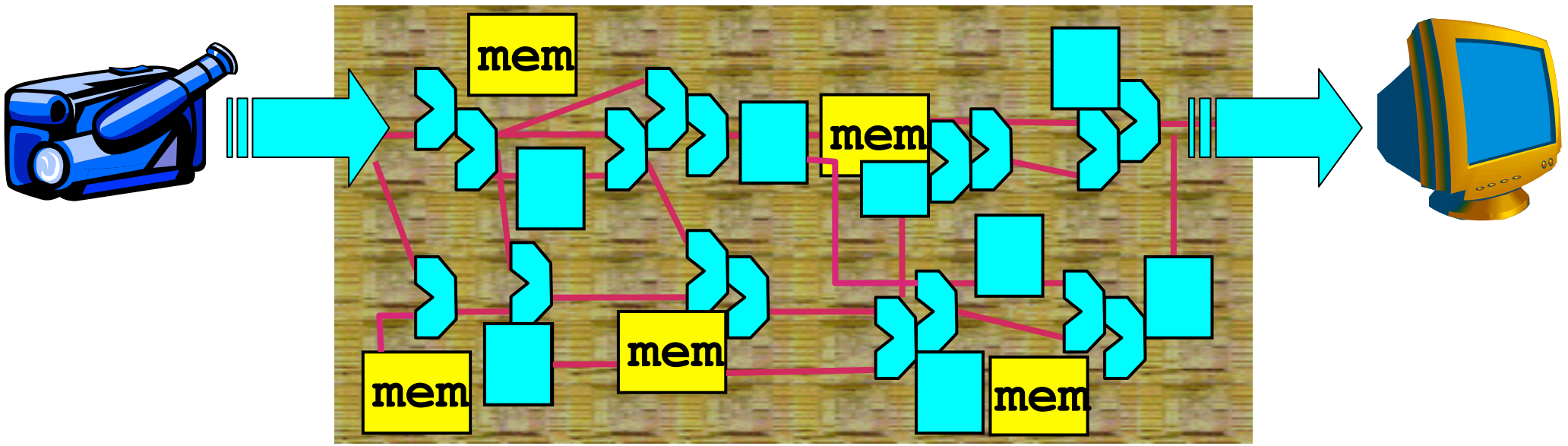
X86
Pentium IV




ASICs

0.25TFLOPS

# How does the ASIC do it?




- Lots of ALUs, lots of registers, lots of small memories
- Hand-routed, short wires

- Lower power (everything close by)
- Stream data model for high throughput

But, not general purpose

# Our challenge

How to exploit ASIC-like features

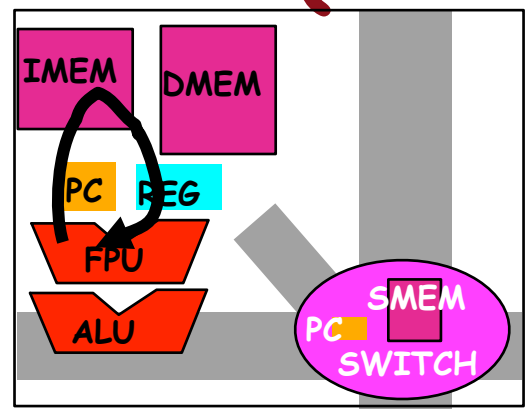
- Lots of resources like ALUs and memories
- Application-specific routing of short wires

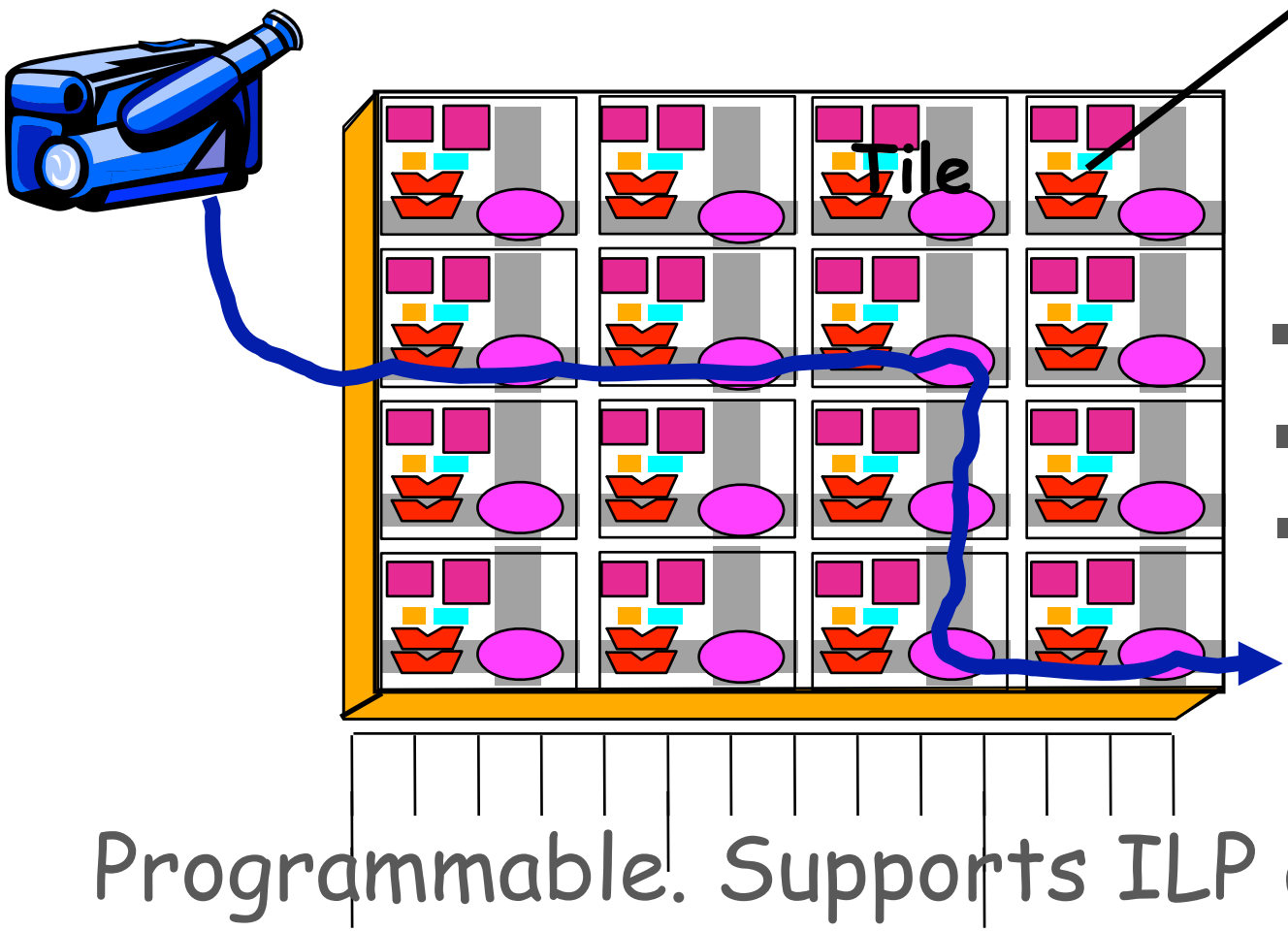
While being "general purpose"

- Programmable
- And even running ILP-based sequential programs

One Approach: Tiled Processor Architecture (TPA)

# Tiled Processor Architecture (TPA)



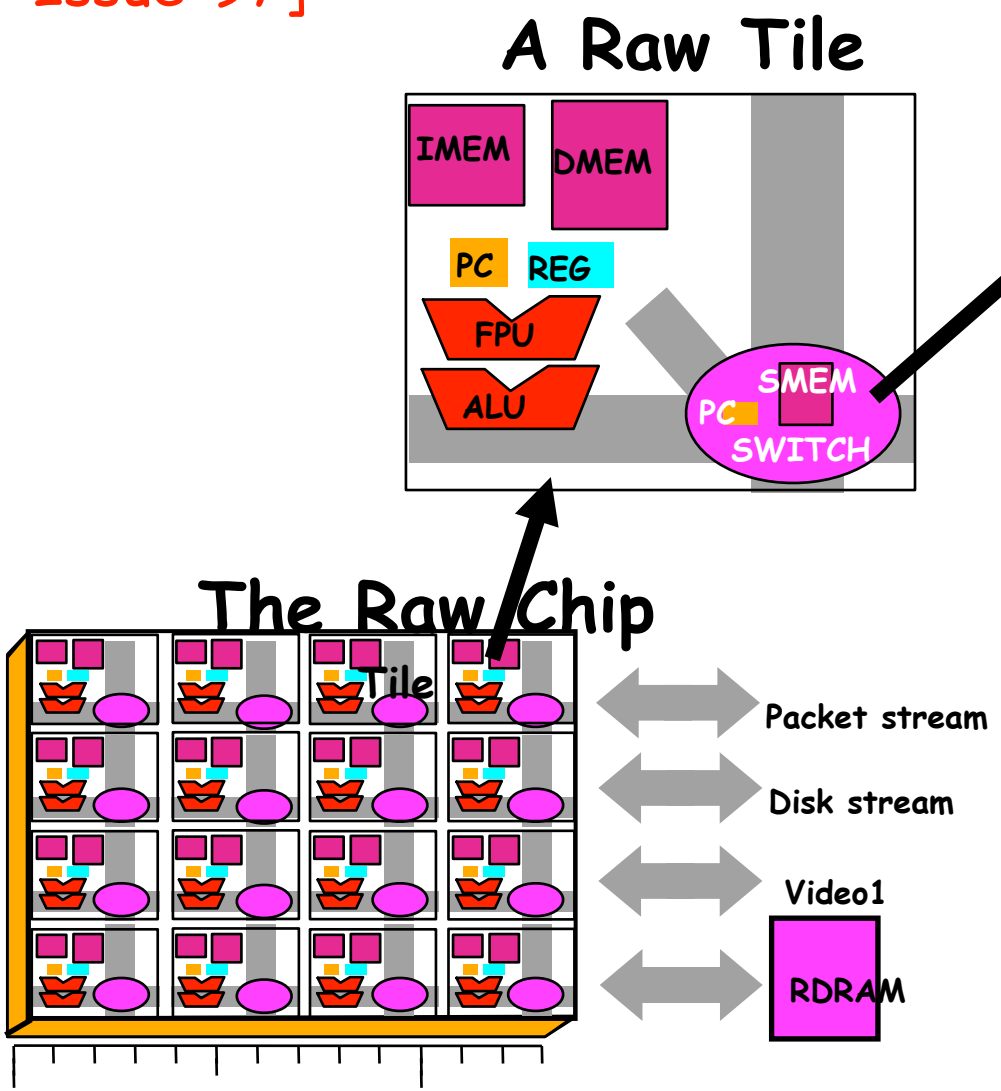



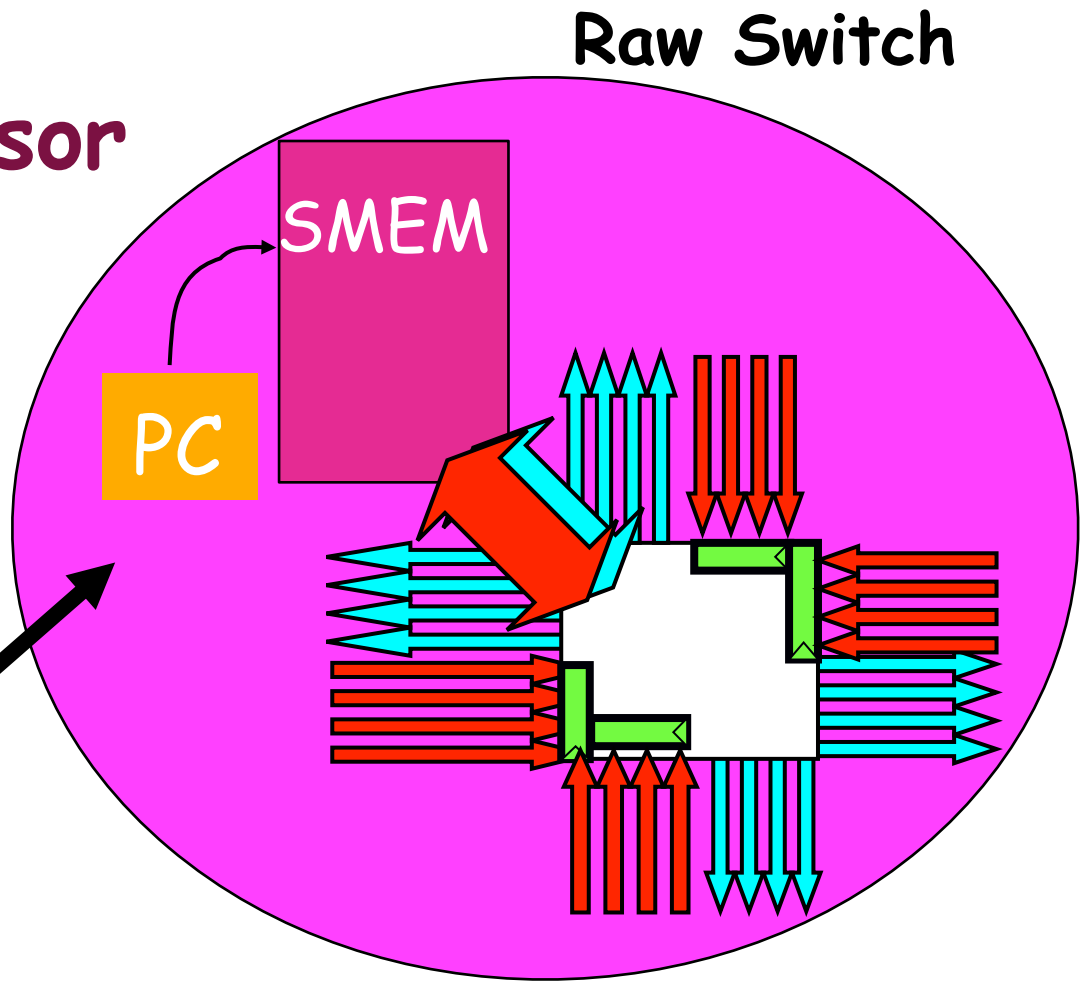

- Lots of ALUs and regs
- Short, prog. wires
- Lower power

Programmable. Supports ILP and Streams

# A Prototype TPA: The Raw Microprocessor

[Billion transistor IEEE Computer Issue '97]

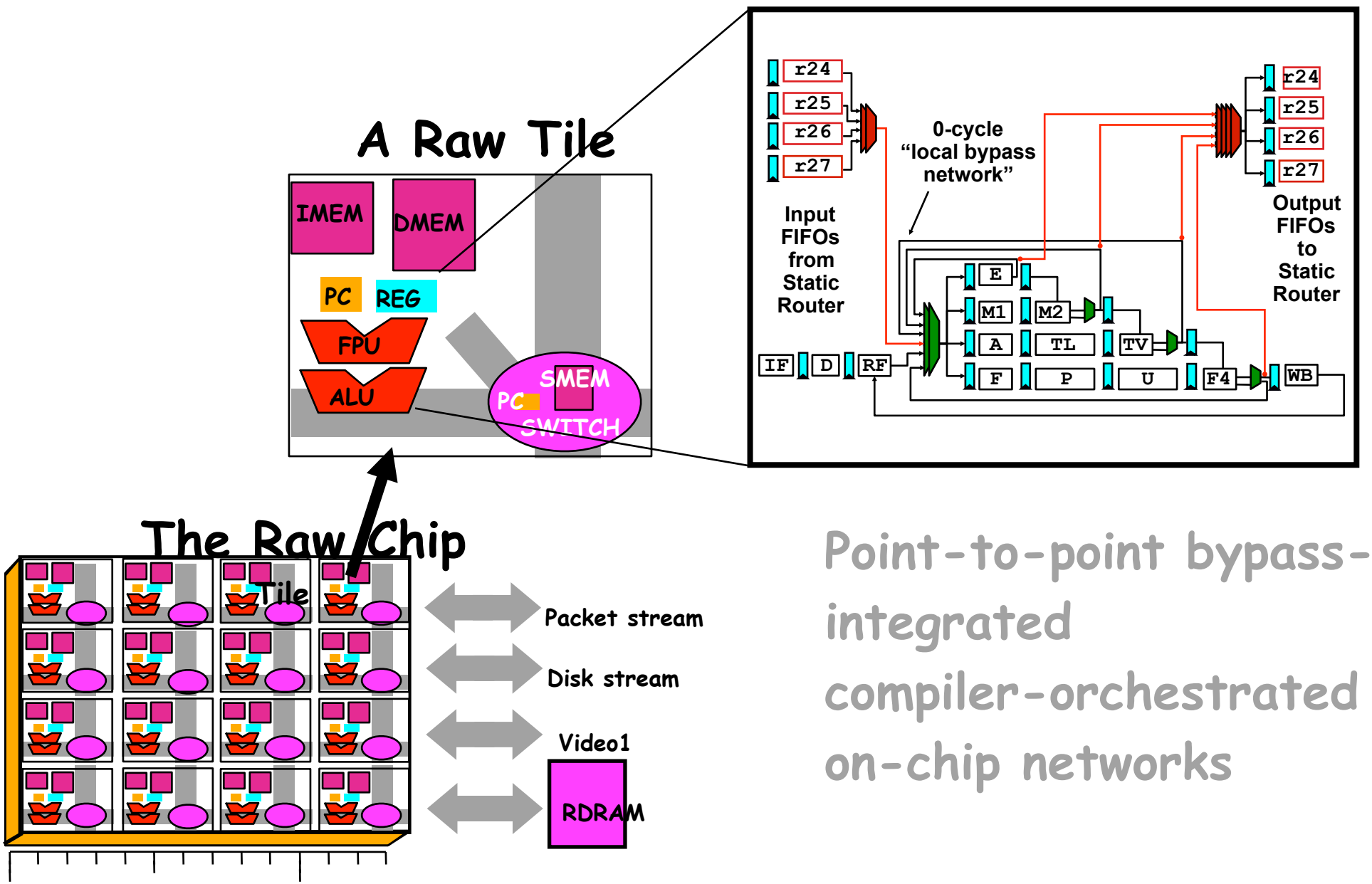



**Software-scheduled interconnects**

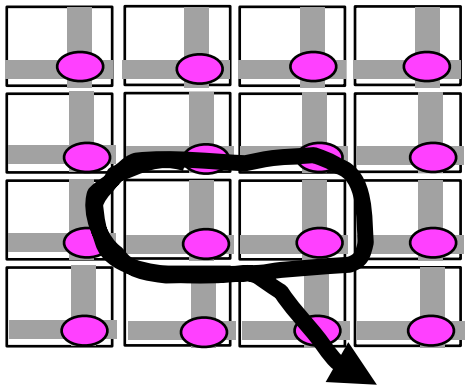
(can use static or dynamic routing - but compiler determines instruction placement and routes)

# Tight integration of interconnect



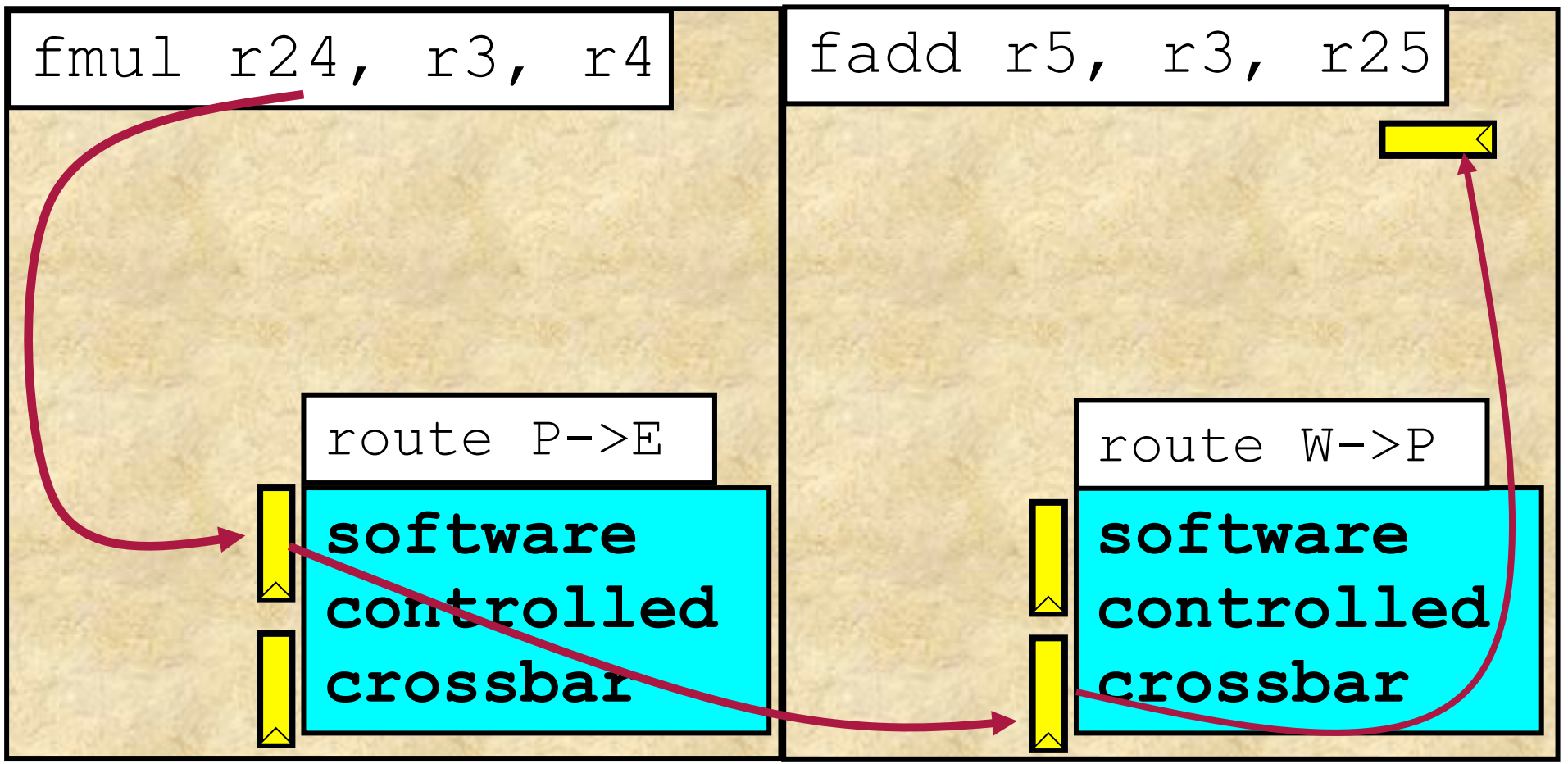

Point-to-point bypass-integrated compiler-orchestrated on-chip networks

# How to "program the wires"

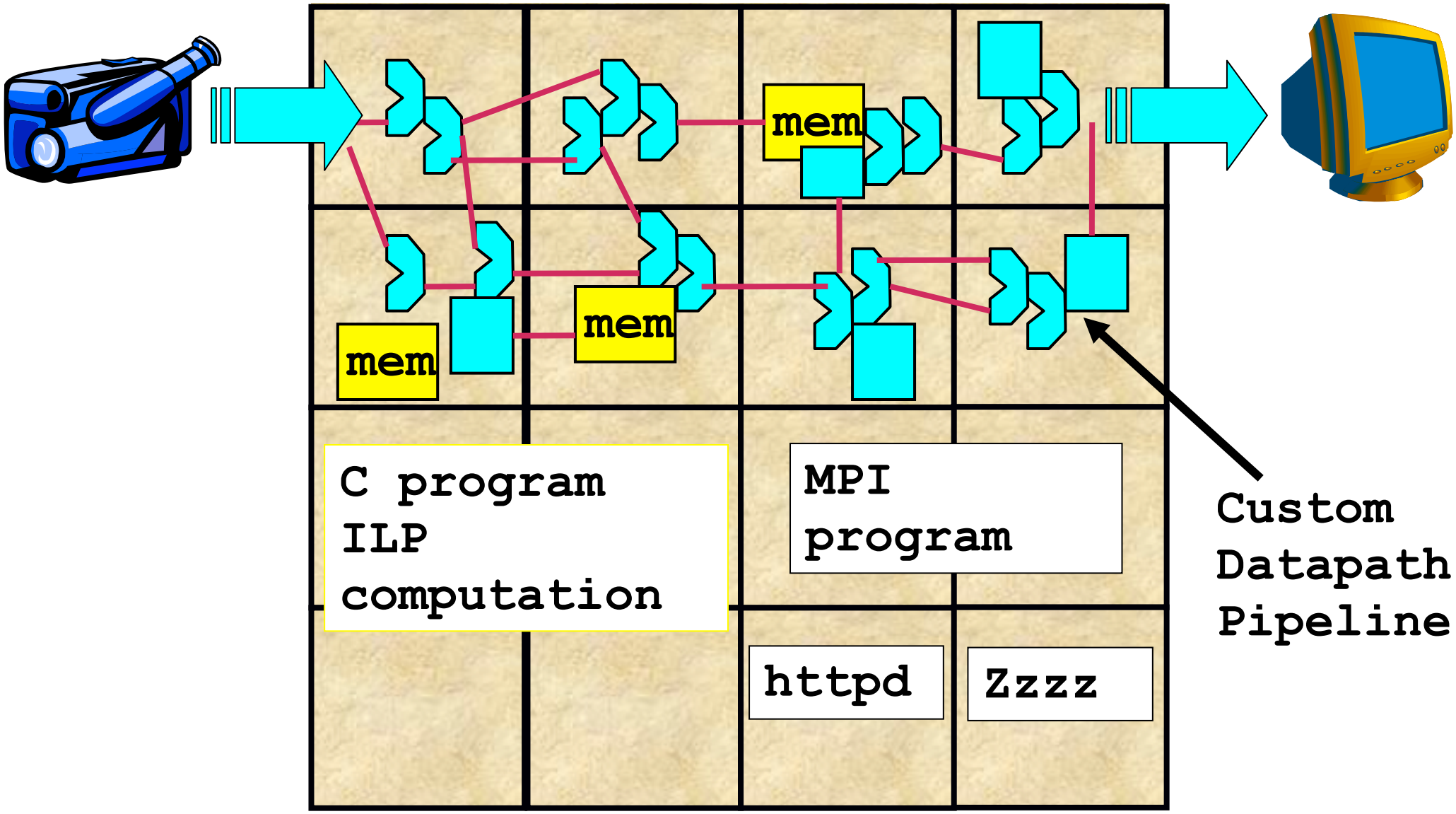


Tile 10

Tile 11



# The result of orchestrating the wires



# Perspective

We have replaced

Bypass paths, ALU-reg bus, FPU-Int. bus, reg-cache-bus, cache-mem bus, etc.

With a general, point-to-point, routed interconnect called:

**Scalar operand network (SON)**

**Fundamentally new kind of network optimized for both scalar and stream transport**

# Scalar (ILP) program mapping

E.g., Start with a C program, and several transformations later:

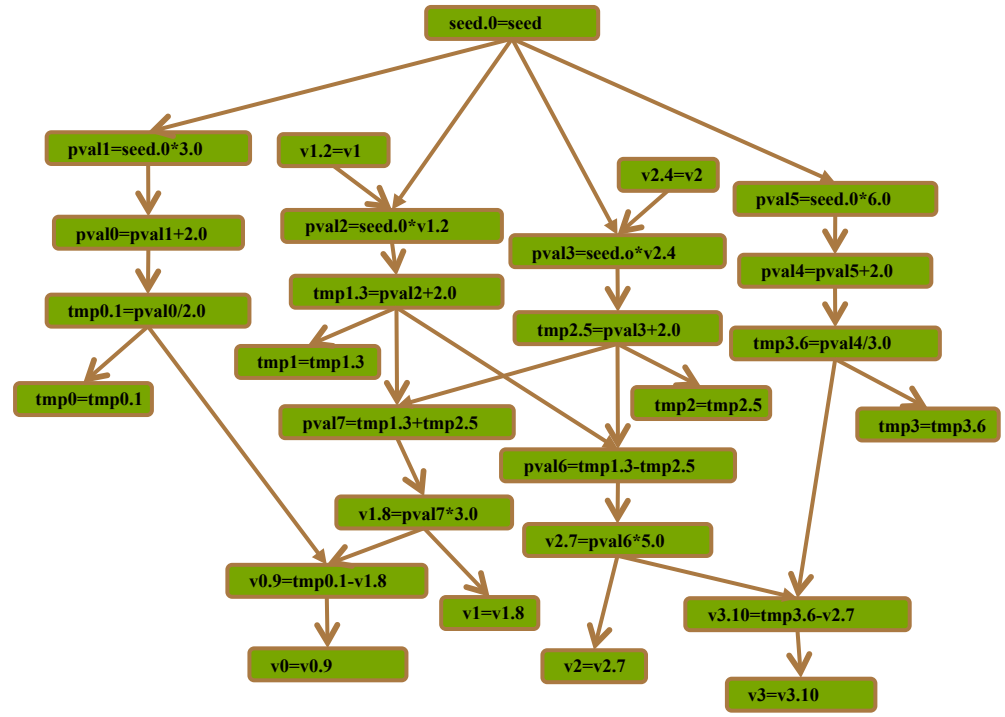
```
v2.4 = v2
seed.0 = seed
v1.2 = v1
pval1 = seed.0 * 3.0
pval0 = pval1 + 2.0
tmp0.1 = pval0 / 2.0
pval2 = seed.0 * v1.2
tmp1.3 = pval2 + 2.0
pval3 = seed.0 * v2.4
tmp2.5 = pval3 + 2.0
pval5 = seed.0 * 6.0
pval4 = pval5 + 2.0
tmp3.6 = pval4 / 3.0
pval6 = tmp1.3 - tmp2.5
v2.7 = pval6 * 5.0
pval7 = tmp1.3 + tmp2.5
v1.8 = pval7 * 3.0
v0.9 = tmp0.1 - v1.8
v3.10 = tmp3.6 - v2.7
tmp2 = tmp2.5
v1 = v1.8;
tmp1 = tmp1.3
v0 = v0.9
tmp0 = tmp0.1
v3 = v3.10
tmp3 = tmp3.6
v2 = v2.7
```

Existing languages will work

Lee, Amarasinghe et al, "Space-time scheduling", ASPLOS '98

# Scalar program mapping

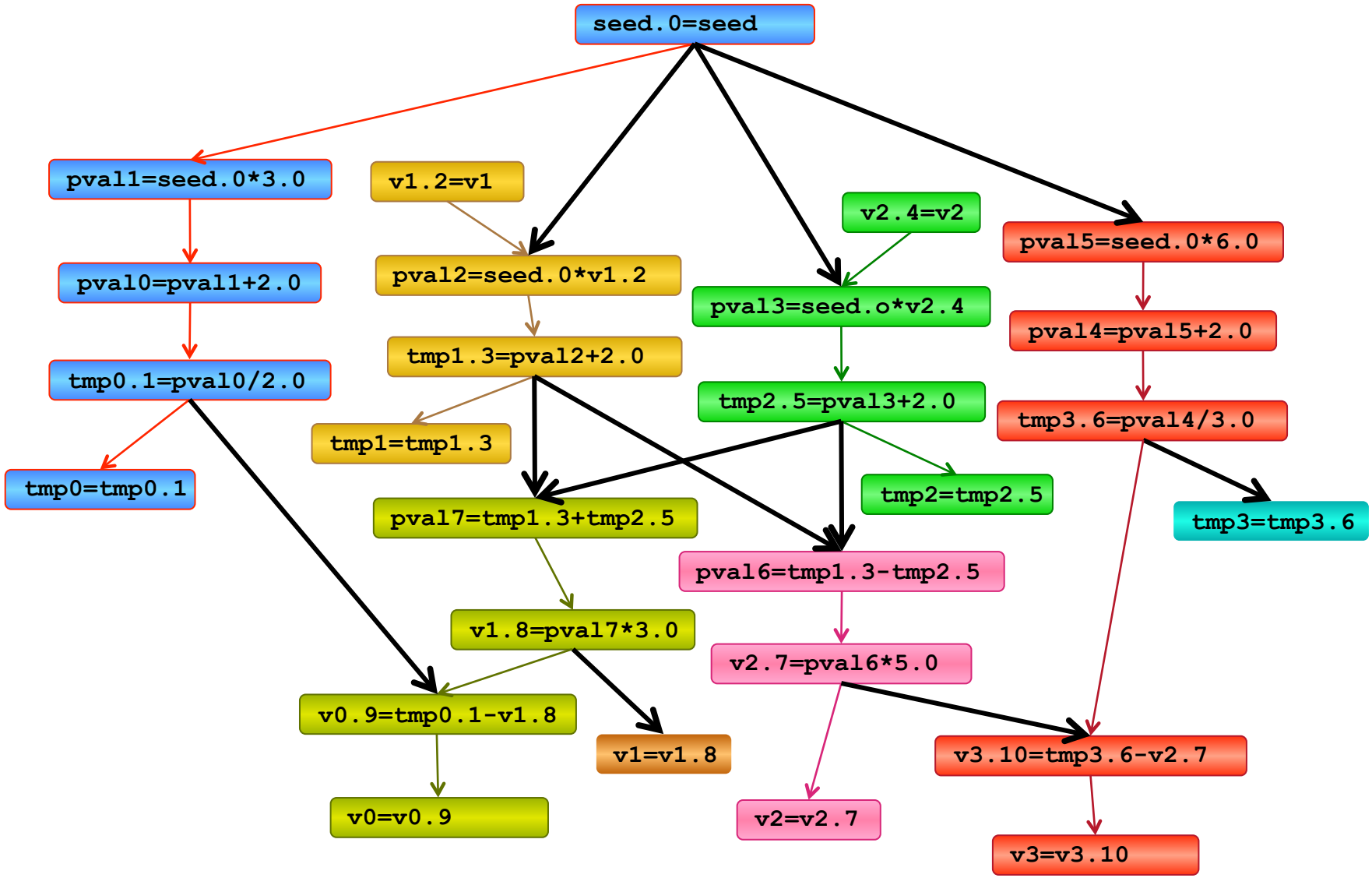
```
v2.4 = v2
seed.0 = seed
v1.2 = v1
pval1 = seed.0 * 3.0
pval0 = pval1 + 2.0
tmp0.1 = pval0 / 2.0
pval2 = seed.0 * v1.2
tmp1.3 = pval2 + 2.0
pval3 = seed.0 * v2.4
tmp2.5 = pval3 + 2.0
pval5 = seed.0 * 6.0
pval4 = pval5 + 2.0
tmp3.6 = pval4 / 3.0
pval6 = tmp1.3 - tmp2.5
v2.7 = pval6 * 5.0
pval7 = tmp1.3 + tmp2.5
v1.8 = pval7 * 3.0
v0.9 = tmp0.1 - v1.8
v3.10 = tmp3.6 - v2.7
tmp2 = tmp2.5
v1 = v1.8;
tmp1 = tmp1.3
v0 = v0.9
tmp0 = tmp0.1
v3 = v3.10
tmp3 = tmp3.6
v2 = v2.7
```



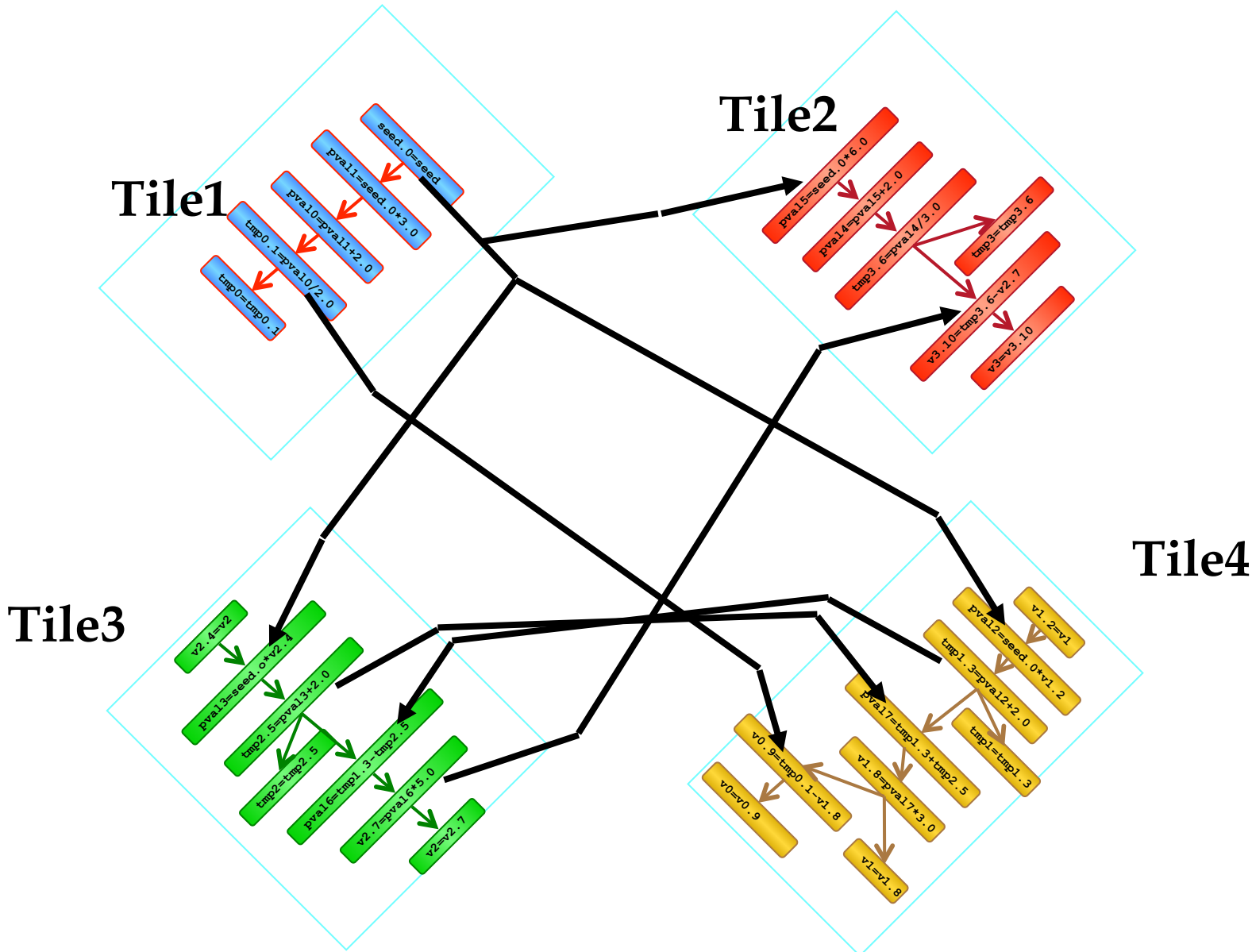

Graph

Program code

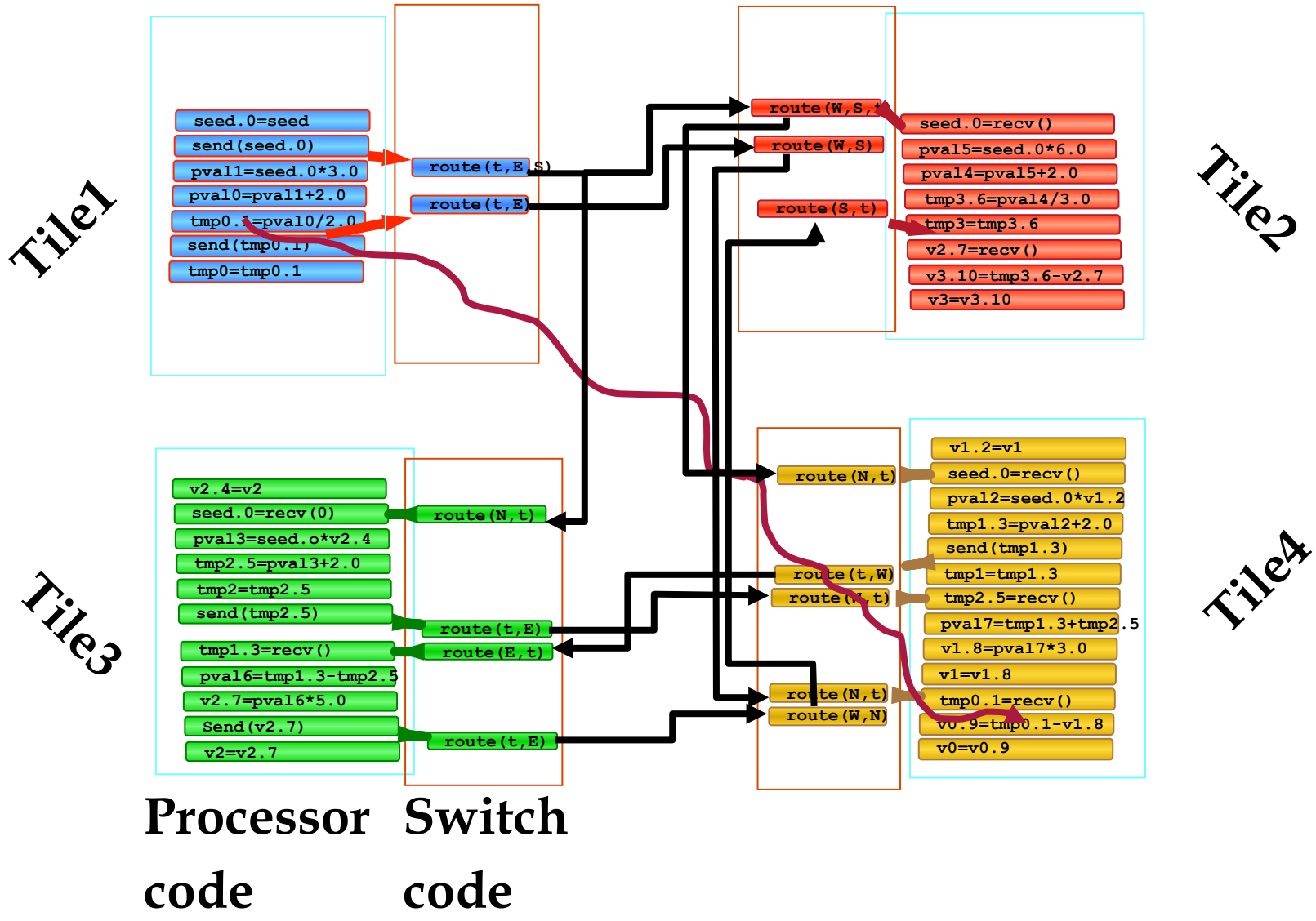
# Program graph clustering



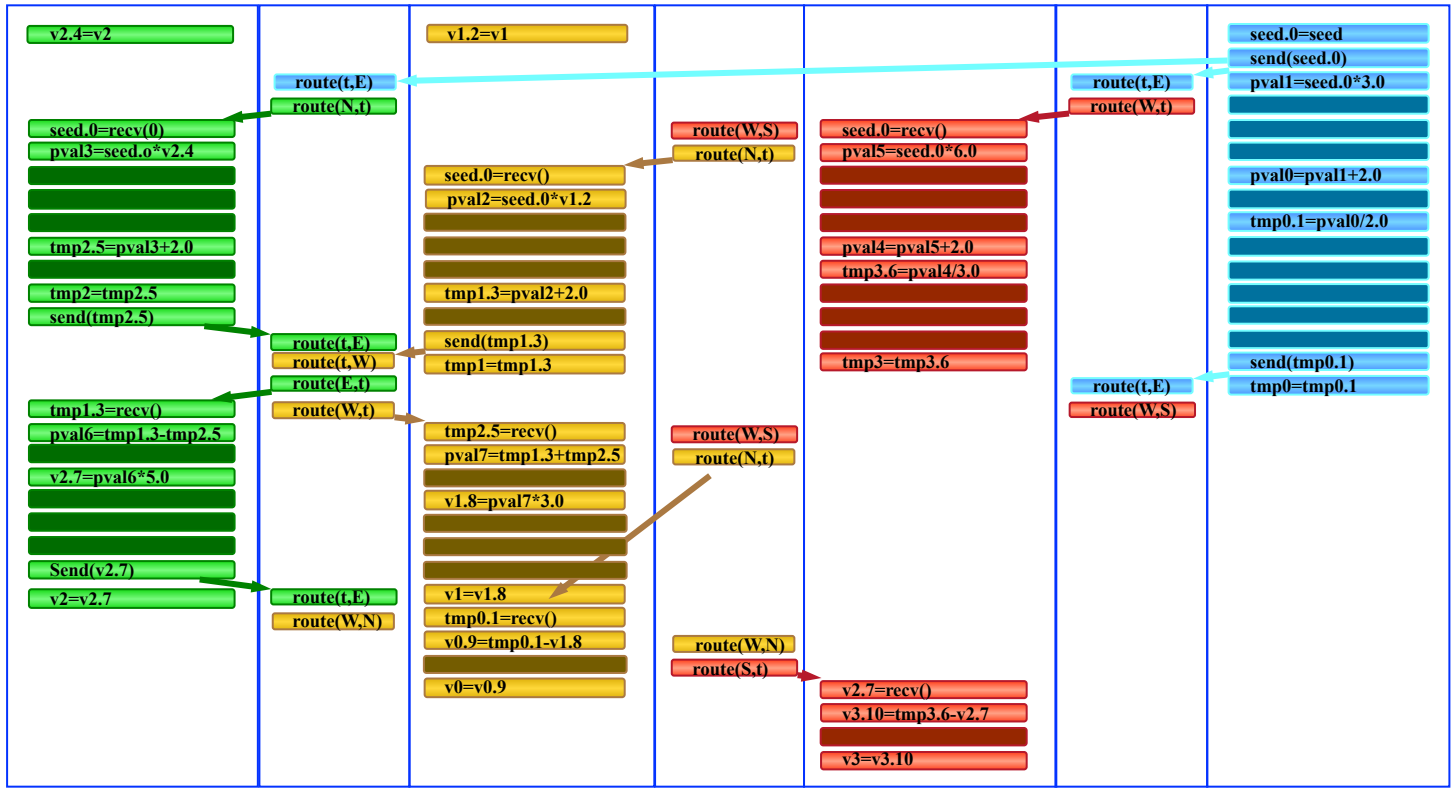
# Placement


Tile1
Tile2
Tile3
Tile4
seed.0=seed
pval1=seed.0*3.0
pval0=pval1+2.0
tmp0.1=pval0/2.0
tmp0=tmp0.1
pval5=seed.0*6.0
pval4=pval5+2.0
tmp3.6=pval4/3.0
tmp3=tmp3.6
v3.10=tmp3.6-v2.7
v3=v3.10
v2.4=v2
pval3=seed.0*v2.4
tmp2.5=pval3+2.0
tmp2=tmp2.5
pval6=tmp1.3-tmp2.5
v2.7=pval6*5.0
v2=v2.7
v1.2=v1
pval2=seed.0*v1.2
tmp1.3=pval2+2.0
tmp1=tmp1.3
pval7=tmp1.3+tmp2.5
v0.9=tmp0.1-v1.8
v1.8=pval7*3.0
v0=v0.9
v1=v1.8

# Routing


Tile1
Tile2
Tile3
Tile4
seed.0=seed
send(seed.0)
pval1=seed.0*3.0
pval0=pval1+2.0
tmp0.1=pval0/2.0
send(tmp0.1)
tmp0=tmp0.1
route(t,E,S)
route(t,E)
route(W,S,t)
route(W,S)
route(S,t)
seed.0=recv()
pval5=seed.0*6.0
pval4=pval5+2.0
tmp3.6=pval4/3.0
tmp3=tmp3.6
v2.7=recv()
v3.10=tmp3.6-v2.7
v3=v3.10
v2.4=v2
seed.0=recv(0)
pval3=seed.o*v2.4
tmp2.5=pval3+2.0
tmp2=tmp2.5
send(tmp2.5)
tmp1.3=recv()
pval6=tmp1.3-tmp2.5
v2.7=pval6*5.0
Send(v2.7)
v2=v2.7
route(N,t)
route(t,E)
route(E,t)
route(t,E)
route(N,t)
route(t,W)
route(W,t)
route(N,t)
route(W,N)
v1.2=v1
seed.0=recv()
pval2=seed.0*v1.2
tmp1.3=pval2+2.0
send(tmp1.3)
tmp1=tmp1.3
tmp2.5=recv()
pval7=tmp1.3+tmp2.5
v1.8=pval7*3.0
v1=v1.8
tmp0.1=recv()
v0.9=tmp0.1-v1.8
v0=v0.9
Processor code
Switch code

# Instruction Scheduling

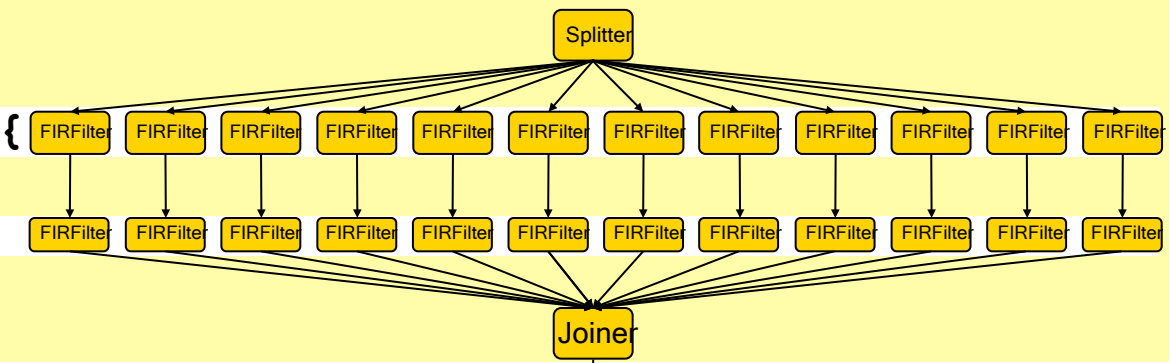


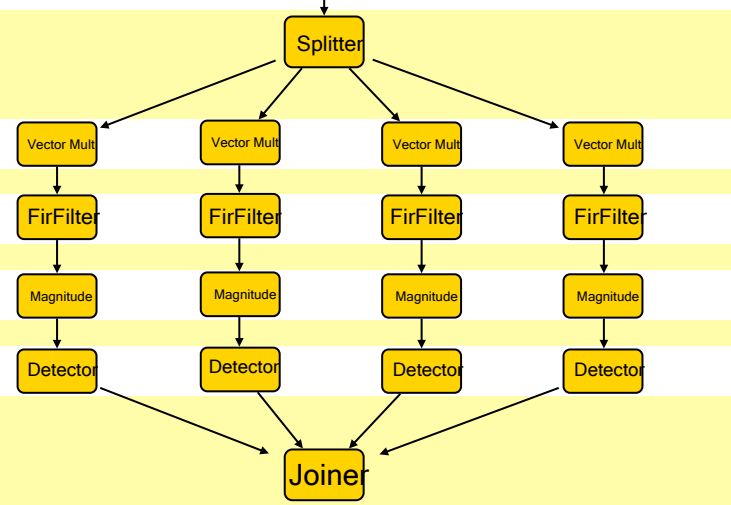
# StreamIt: Stream Language and Compiler

## Amarasinghe et al.

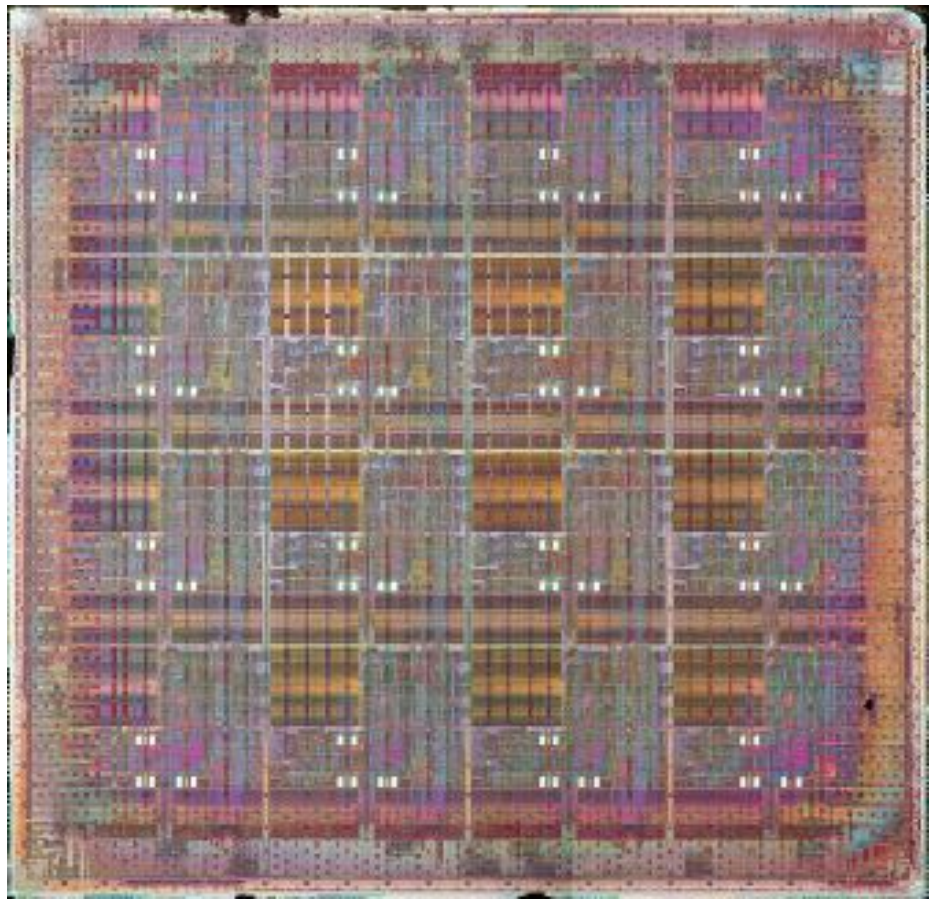
## e.g., BeamFormer

```
class BeamFormer extends Pipeline {
    void init(numChannels, numBeams) {
        add(new SplitJoin() {
            void init() {
                setSplitter(Duplicate());
                for (int i=0; i<numChannels; i++) {
                    add(new FIR1(N1));
                    add(new FIR2(N2));
                }
                setJoiner(RoundRobin());  }});
            }
        add(new SplitJoin() {
            void init() {
                setSplitter(Duplicate());
                for (int i=0; i<numBeams; i++) {
                    add(new VectorMult());
                    add(new FIR3(N3));
                    add(new Magnitude());
                    add(new Detect());
```



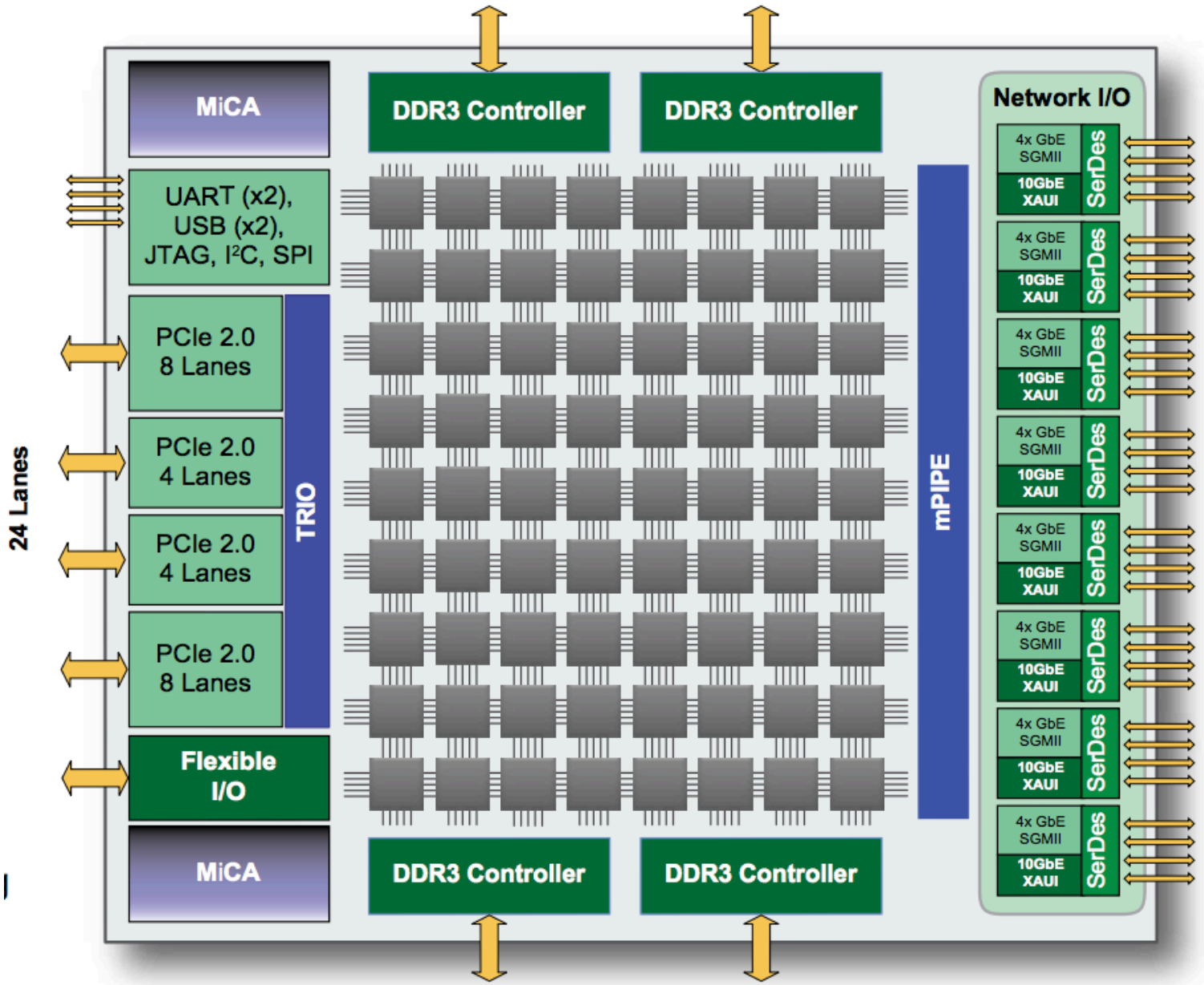
# Raw die photo



.18 micron process, 16 tiles, 425MHz, 18 Watts (vpenta)

Of course, custom IC designed by industrial design team could do much better

# TILE-Gx8072 72-core Processor




TILE-Gx8072 Processor Block Diagram

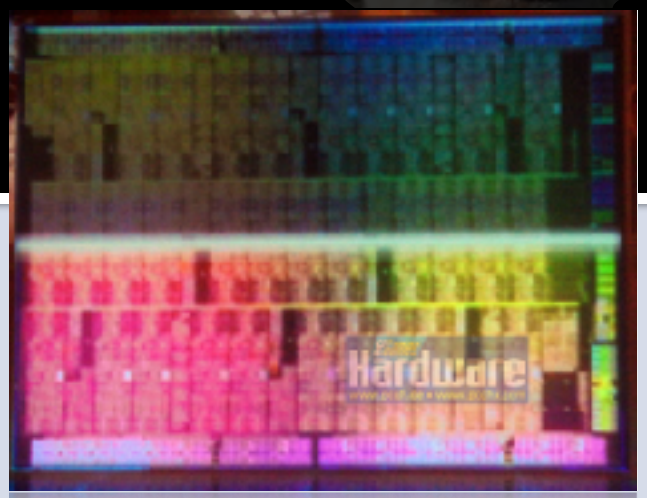
The TILE-Gx8072 is ideal for applications such as:

- 80 - 100 Gbps of networking and security dataplane offload
- 80 Gbps packet filtering and bandwidth management
- 40 Gbps SSL/IPsec security protocol processing
- High-performance computing offload across PCIe
- H.264/H.265 high-density video transcoding

http://www.tilera.com/sites/default/files/images/products/TILE-Gx8072_PB041-03_WEB.pdf

Anjul Patney
EEC171

# Intel Larrabee Architecture




Unofficial die-shot courtesy
www.pcgameshardware.com

# Disclaimers

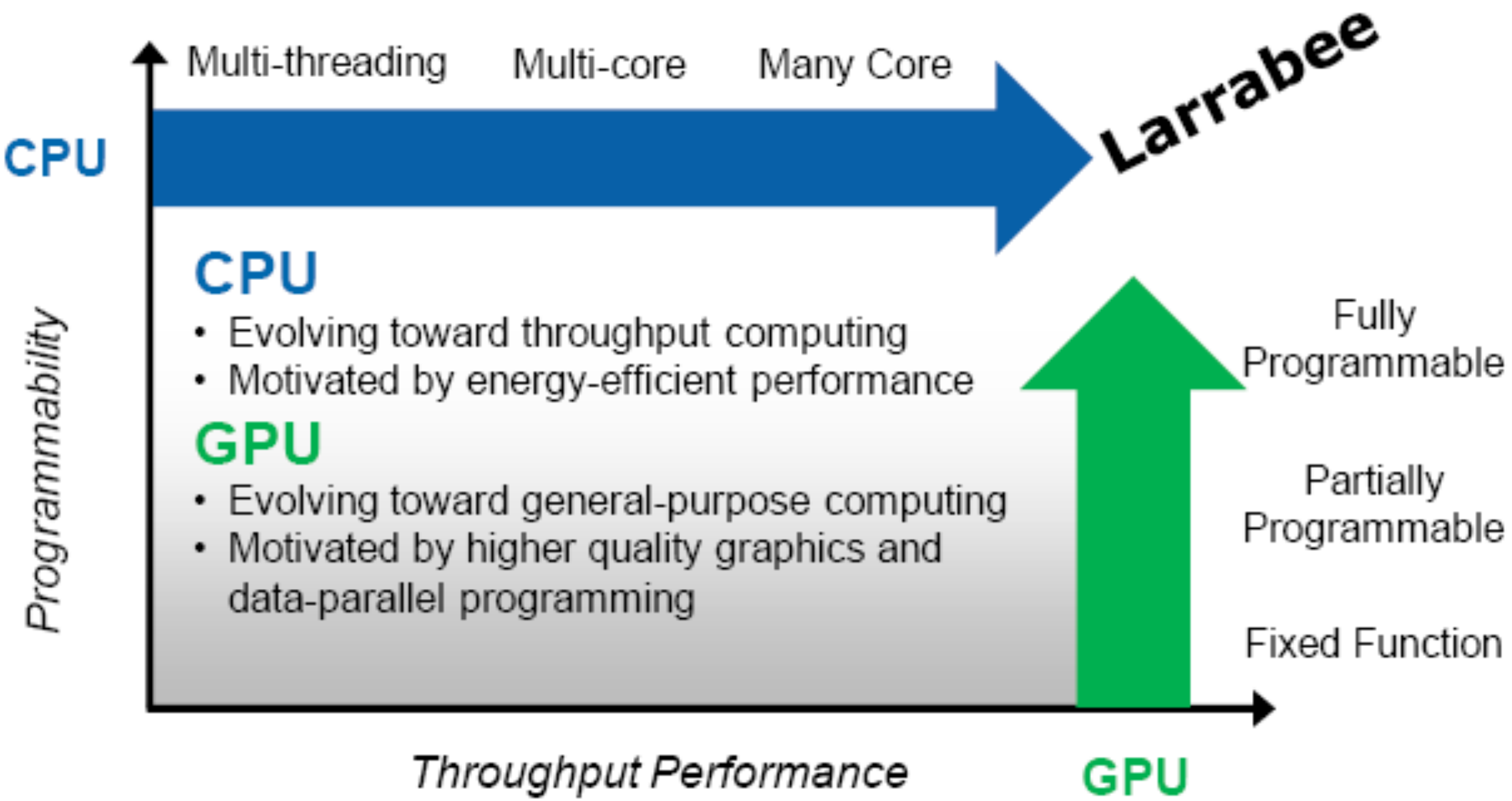
- I do not represent Intel
  - All information in this presentation is available in the public domain
  - Sources
    - Intel's paper from SIGGRAPH 2008
    - Intel's slides from Hot Chips 2008 and from GDC 2009
    - Beyond3D
- Intel emphasizes that Larrabee represents an 'architecture', not a 'product'

# Larrabee - Introduction

- Intel's first many-core architecture
- Designed as a graphics processor
  - But general purpose at heart (runs x86)
  - Aimed primarily for the gaming market
- Architecture goals
  - High peak-throughput
  - Massive parallelism

# Motivation

# Motivation (official answer)




Source: Intel Hot Chips 2008 Presentation

# Motivation (my interpretation)

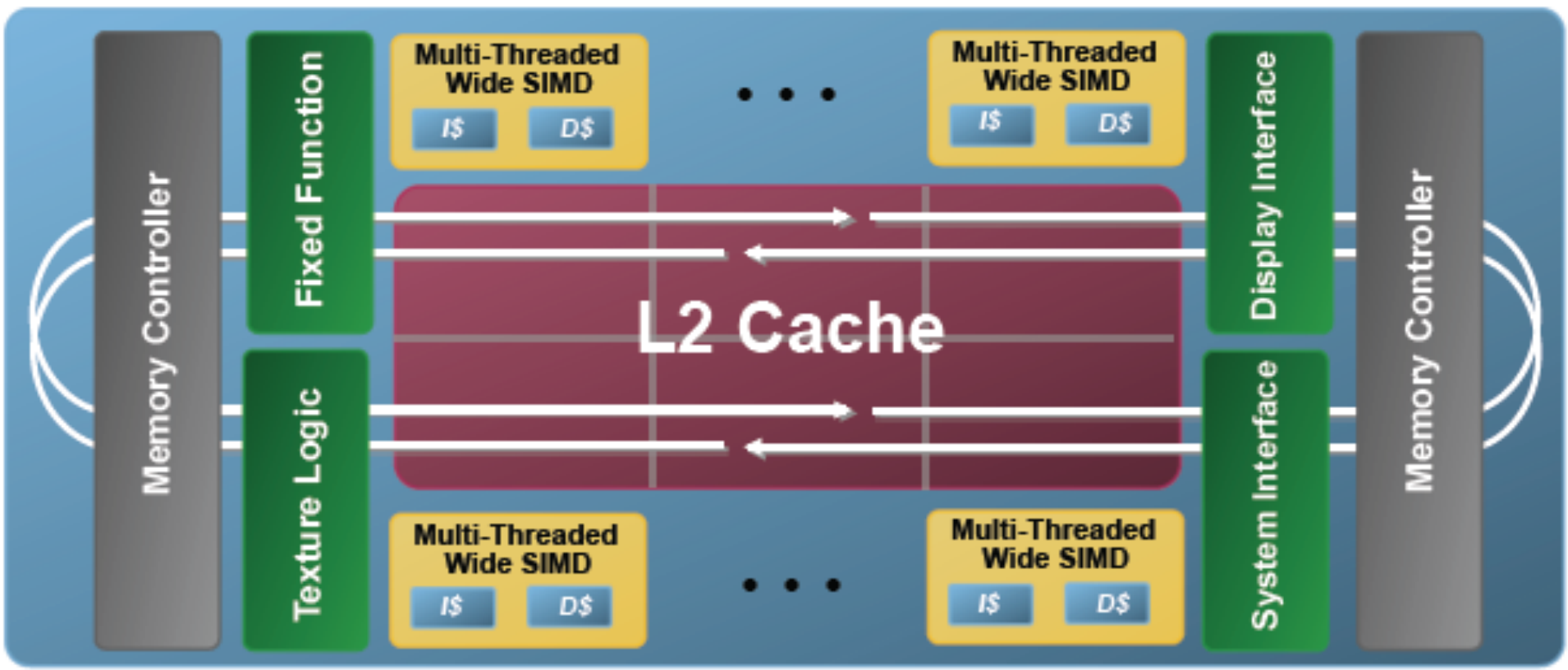
- GPUs == more $$
  - The market for desktop CPUs is saturating
  - But people still run for the latest GPU
- GPUs are eating into Intel's market(s)
  - PhysX
  - CUDA

# Motivation (ideally)

- 2/4/6/8 cores discourage healthy parallel programming
  - Optimize today for 4-cores
  - Re-optimize tomorrow for 8-cores
- Goal of Larrabee-like architectures
  - Offer massive parallelism
  - Allow the programmer to optimize for infinite cores

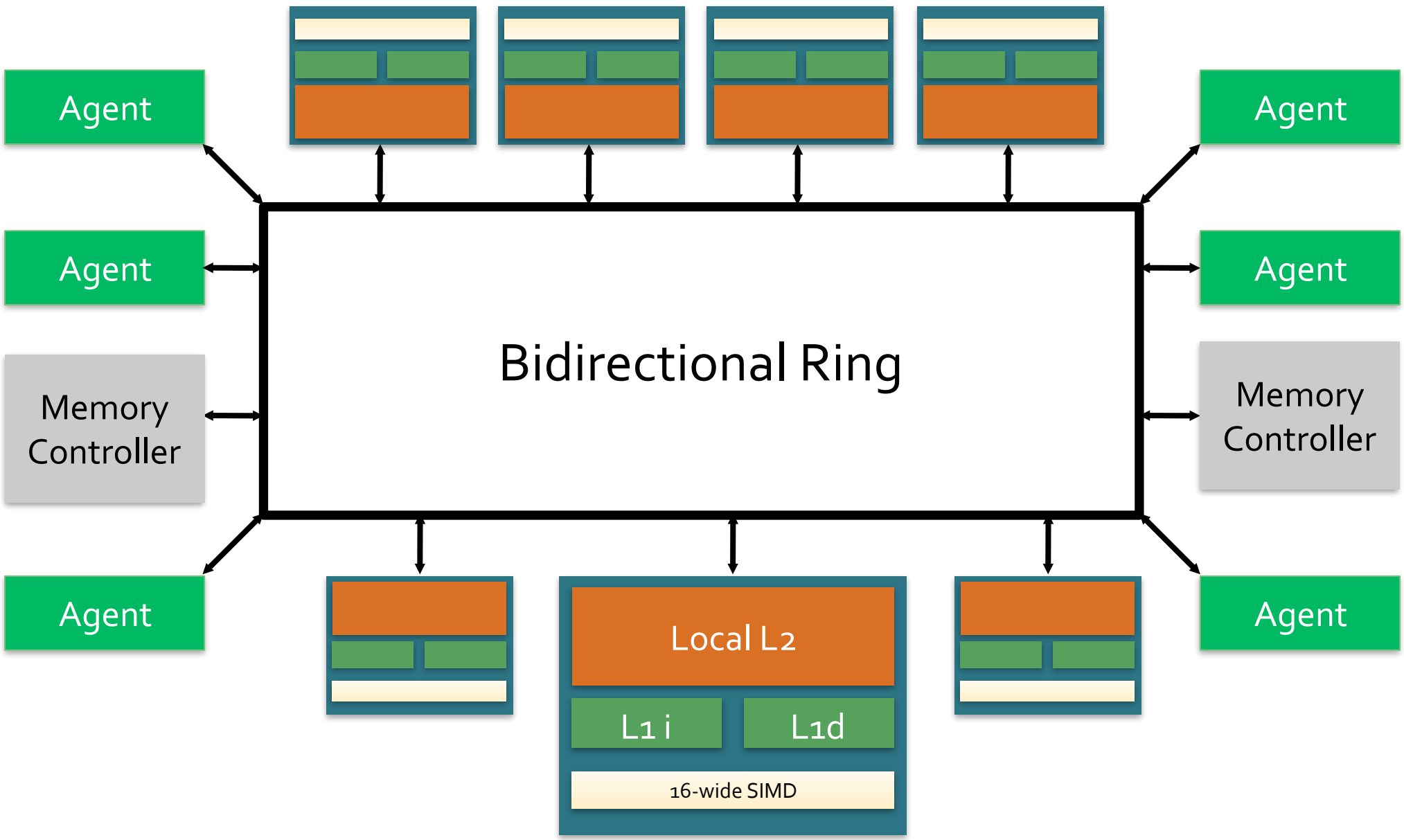
# Architecture

# Larrabee Architecture



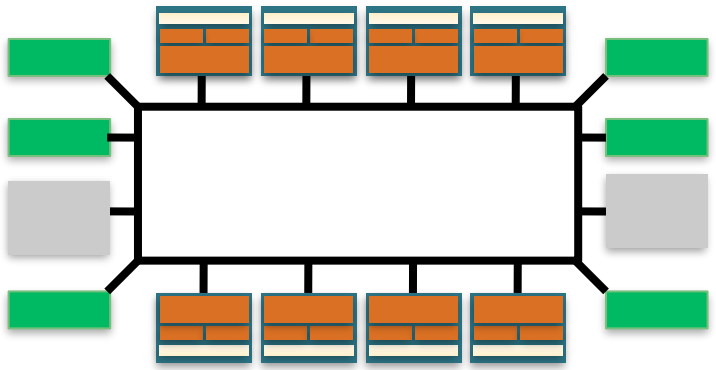

Source: Intel Hot Chips 2008 Presentation

# Larrabee Architecture (Simplified)



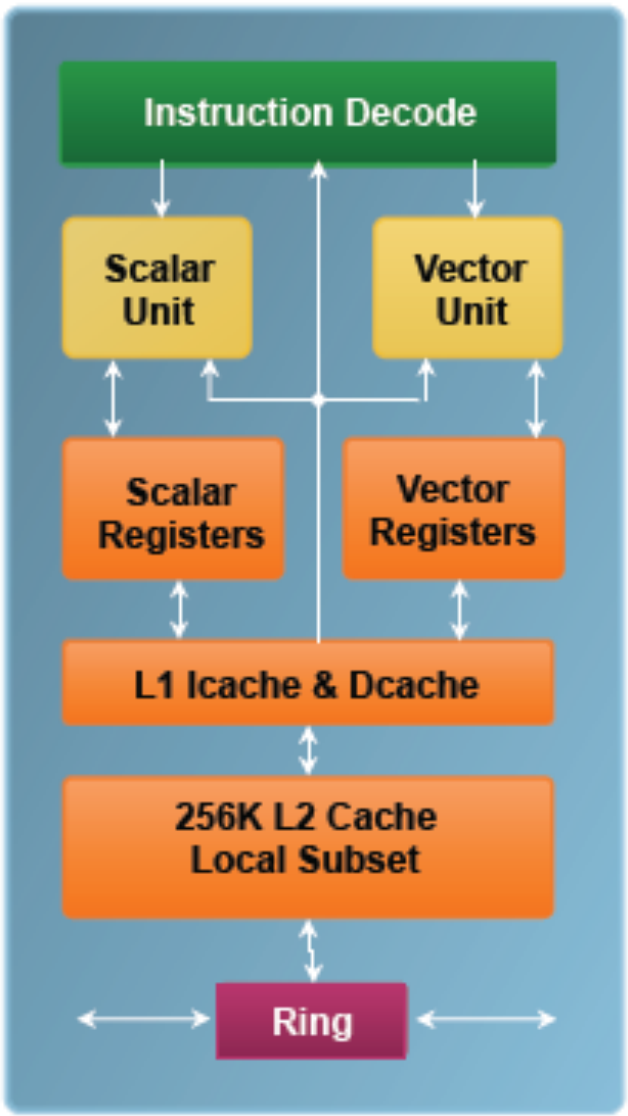
# Basic Architecture

- Ring-bus (interconnect)
  - Bidirectional
  - 1 cycle/hop—faster than DRAM!



- Everything else is a ring agent
  - Cores (many)
  - Memory controllers
  - System Interface / Command Processor
  - Fixed-function Logic (texture, video decode)

# Larrabee Core




- In-order pipeline (p54c)
- 1 scalar + 1 vector unit (16-wide)
  - 32 vector R, 8 vector mask R
- Dual-issue, 4 hardware threads
- Hardware Scatter/Gather & streaming support
- Local L1 caches (I and D)

Source: Intel Hot Chips 2008 Presentation

# Larrabee Cache Structure

- Each core has a local L1 cache (32K + 32K)
  - Small, low-latency
  - Can be used to share data amongst SIMD lanes
- … and a subset of shared L2 cache (256 K)
  - Larger, supports data sharing across cores
  - Major distinction from NVIDIA G80
- Flexible: several cache control instructions

# Larrabee Cache Coherence

- MOESI Coherence Protocol
  - Modified
  - **Owned – Reduces off-chip traffic**
  - **Exclusive – Reduces ring traffic**
  - Shared
  - Invalid
- Distributed tag directory with MOESI bits

# Larrabee Fixed-function Units

- Larrabee aims to perform all computation in software (like a CPU)
- However, some things are just too slow / power inefficient
  - Texture Sampler (10x faster than sw)
  - System / Display / Memory interfacing

# Instruction Set Architecture

# Larrabee ISA

- x86 derivative
  - >100 added instructions (mostly vector)
- Vector-specific Instructions
  - Vector Arithmetic: **vmulps** v0, v5, v6
    - May mask bits: **vmulps** v0 {k1}, v5, v6
    - May convert values: **vmulps** v0 {k1}, v5, v6 {float16}
  - Instruction support for parallel algorithms
    - Gather/Scatter: **vgatherd** v1 {k1}, [rbx + v2*4]
    - Pack-store: **vcompressd** [rbx] {k1}, v0

# Larrabee ISA

- Cache-specific instructions
  - Because caches don't lend well to streaming
  - Prefetch instructions
  - Cache eviction instructions/hints
    - Some data may never be needed again
    - Non-temporal caching

# Larrabee ISA

- CUDA Vs. Larrabee

- CUDA sample:

```
__global__ void addOddsubEven ( float *V1, float *V2, float *V3 )
{
    int idx = threadIdx.x;
    if(idx % 2 == 1){
        V1[idx] = V2[idx] + V3[idx];
    } else {
        V1[idx] = V2[idx] - V3[idx];
    }
}
```

# Larrabee ISA

- CUDA Vs. Larrabee

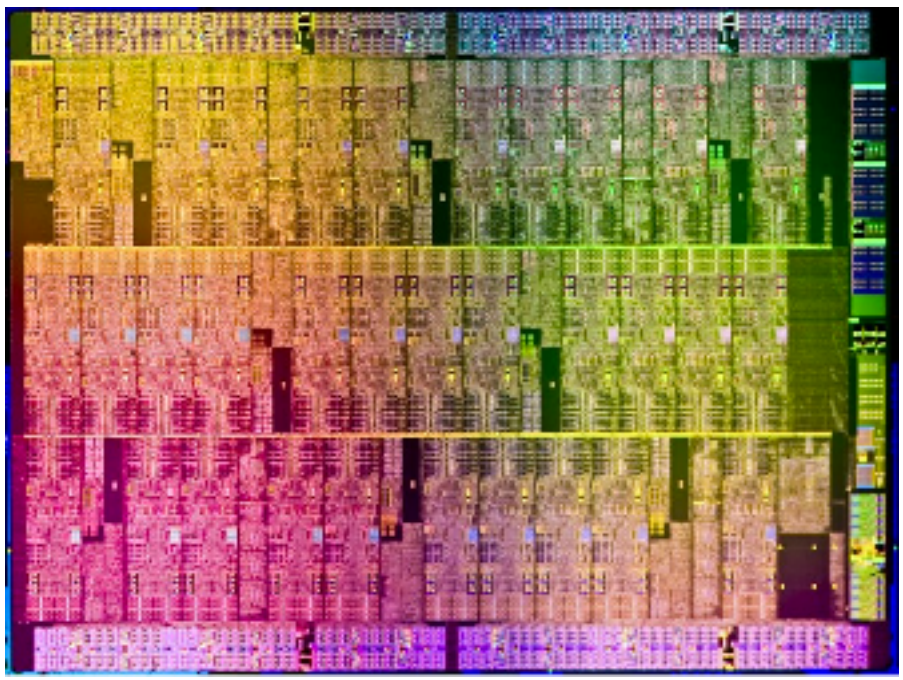
- Approximate LRBni equivalent:

```
; Load data
        vloadd v2, [V2 +  0]
        vloadd v3, [V3 +  0]
; Set mask values
        kmov k1, 0x5555          ; Slightly inaccurate
        kmov k2, 0xaaaa          ; Slightly inaccurate
; Add and subtract
        vaddps v1 {k1}, v2, v3   ; k1 = 0x5555
        vsubps v1 {k2}, v2, v3   ; k2 = 0xaaaa
; Store output
        vstored [V1 +  0], v1
```

# Summary

- Larrabee is a generalized architecture for computing on a large number of simple cores
  - Follows a GPGPU-like approach to parallel computing
- To improve throughput performance, each core is 16-wide SIMD
- Many optimizations and special hardware for graphics and games
  - Approaching graphics from a CPU standpoint
- Actual performance is still a mystery
  - But Larrabee-based products should be out soon *[Knight's X/Xeon Phi]*
  - We have a lot to see!

# Knight's Landing (2015)

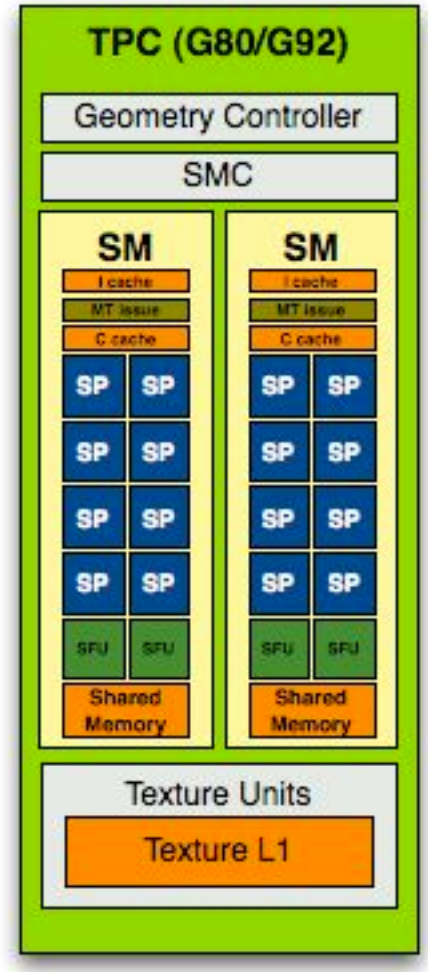


- Up to 72 cores
- 16 GB of stacked DRAM
  - 500 GB/s
- 14 nm process
- 3/6 TFLOPS SP/DP FP
- 14–16 GFLOPS/W
  - Current SC: ~4
- 13/top 500 supercomputers
  - NVIDIA Tesla: 38/500

http://www.extremetech.com/extreme/171678-intel-unveils-72-core-x86-knights-landing-cpu-for-exascale-supercomputing

# NVIDIA Tesla (G80) SM

- “Each SP can fulfill up to two single-precision operations per clock: 1 Multiply and 1 Add, using a single MAD instruction. Each SFU can fulfill up to four operations per clock: four MUL (Multiply) instructions. So one SM as a whole can execute 8 MADs (16 operations) and 8 MULs (8 operations) per clock, or 24 operations per clock, which is (relatively speaking) 3 times the number of SPs.”




http://www.geeks3d.com/20100318/tips-what-is-a-texture-processor-cluster-or-tpc/

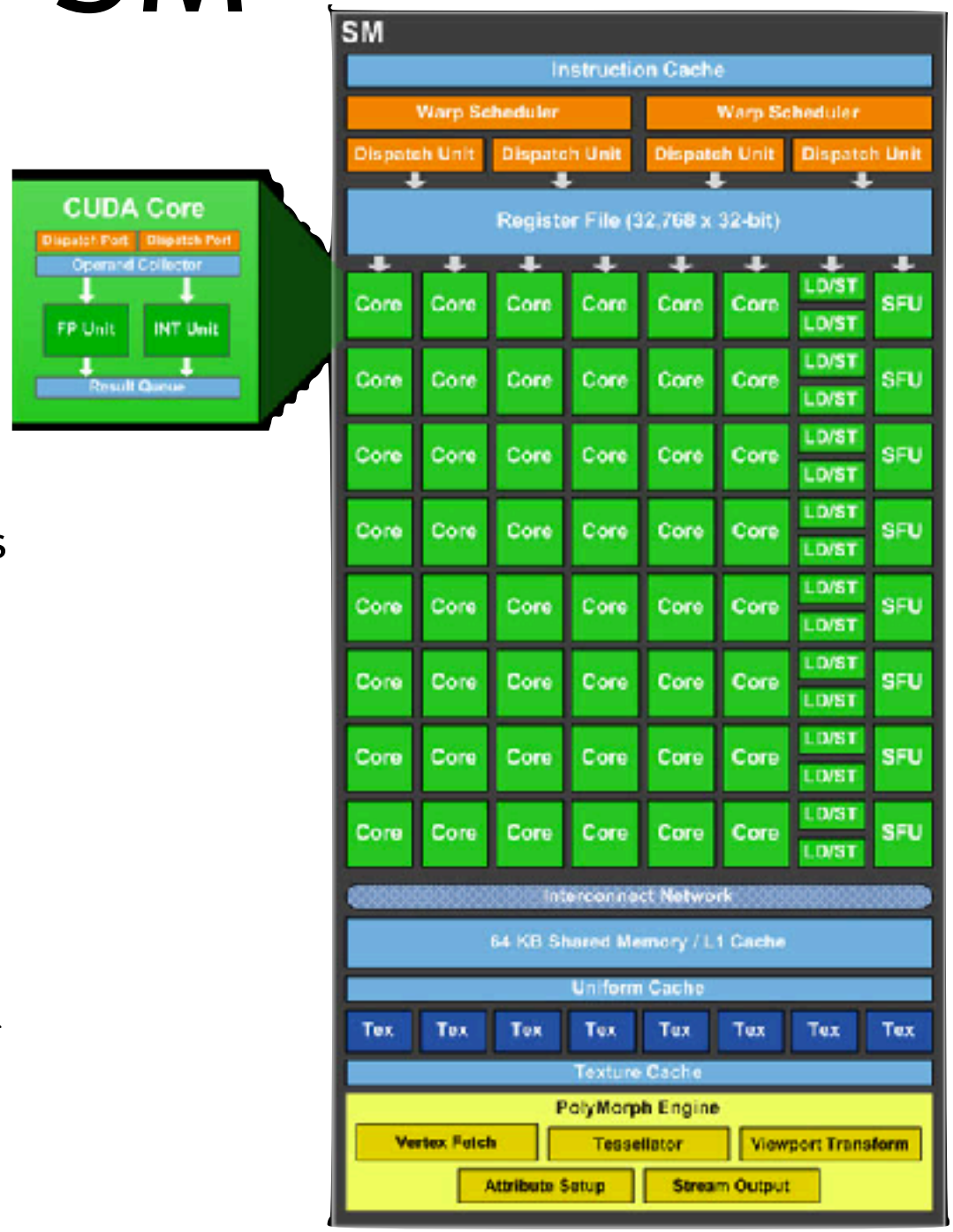
# NVIDIA Kepler SM

- (Fermi) predecessor: “In GF114 each SM contained 48 CUDA cores, with the 48 cores organized into 3 groups of 16. Joining those 3 groups of CUDA cores were 16 load/store units, 16 interpolation SFUs, 8 special function SFUs, and 8 texture units. Feeding all of those blocks was a pair of warp schedulers, each of which could issue up to 2 instructions per core clock cycle, for a total of up to 4 instructions in flight at any given time.”


http://www.anandtech.com/show/5699/nvidia-geforce-gtx-680-review/2

# NVIDIA Kepler SM

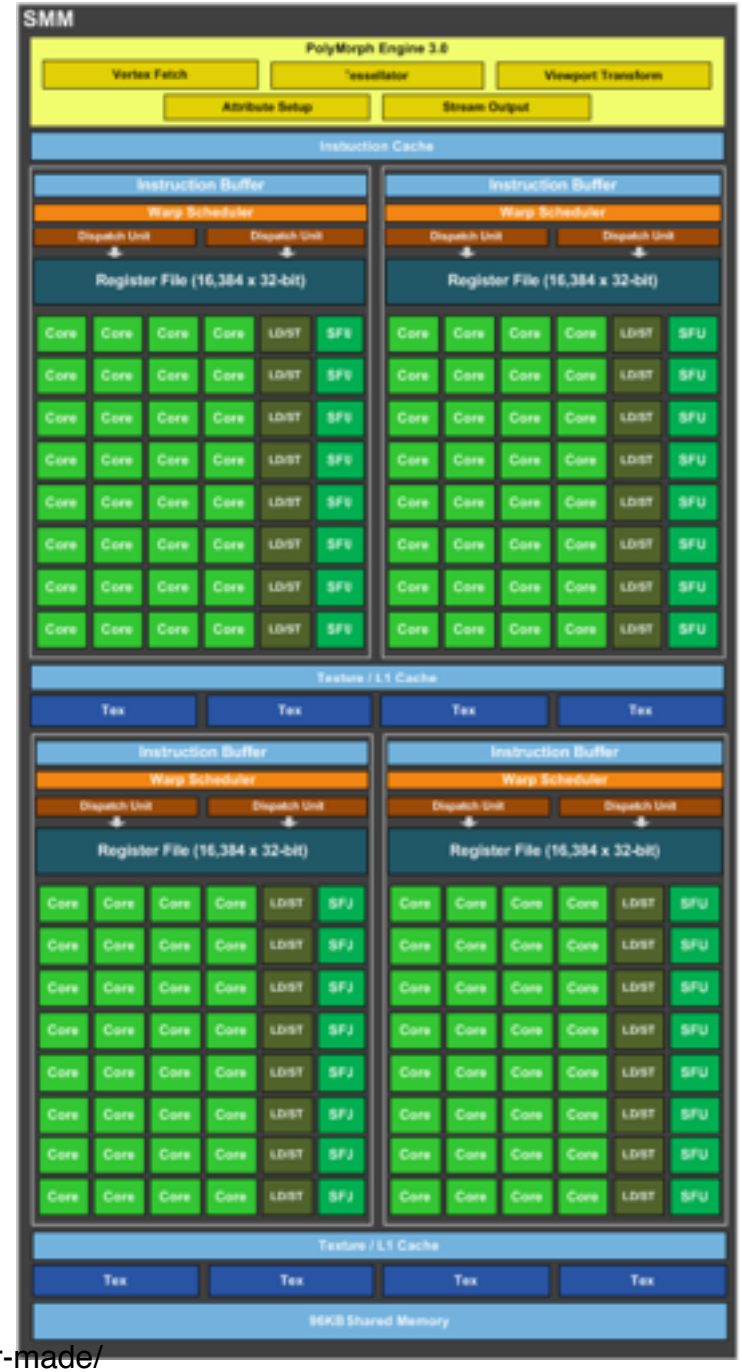
- “GK104 is built to execute a whole warp at once”
- “After doubling the size of the functional units in a SM, NVIDIA then doubled the number of functional units in each SM in order to grow the performance of the SM itself. 3 groups of CUDA cores became 6 groups of CUDA cores, 2 groups of load/store units, 16 texture units, etc. At the same time, with twice as many functional units NVIDIA also doubled the other execution resources, with 2 warp schedulers becoming 4 warp schedulers, and the register file being doubled from 32K entries to 64K entries.”




http://www.anandtech.com/show/5699/nvidia-geforce-gtx-680-review/2

# NVIDIA Maxwell SM

- “more than 40% higher delivered performance per CUDA core, and overall twice the efficiency of Kepler GK104.”
- “SMM uses a quadrant-based design with four 32-core processing blocks each with a dedicated warp scheduler capable of dispatching two instructions per clock.”




https://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/

# NVIDIA Pascal SM

- “GP100’s SM incorporates 64 single-precision (FP32) CUDA Cores. In contrast, the Maxwell and Kepler SMs had 128 and 192 FP32 CUDA Cores, respectively. The GP100 SM is partitioned into two processing blocks, each having 32 single-precision CUDA Cores, an instruction buffer, a warp scheduler, and two dispatch units. While a GP100 SM has half the total number of CUDA Cores of a Maxwell SM, it maintains the same register file size and supports similar occupancy of warps and thread blocks.”


https://devblogs.nvidia.com/parallelforall/inside-pascal/

# NVIDIA Volta SM

- “Similar to Pascal GP100, the GV100 SM incorporates 64 FP32 cores and 32 FP64 cores per SM.”
- “The GV100 SM is partitioned into four processing blocks, each with 16 FP32 Cores, 8 FP64 Cores, 16 INT32 Cores, two of the new mixed-precision Tensor Cores for deep learning matrix arithmetic, a new Lo instruction cache, one warp scheduler, one dispatch unit, and a 64 KB Register File.”

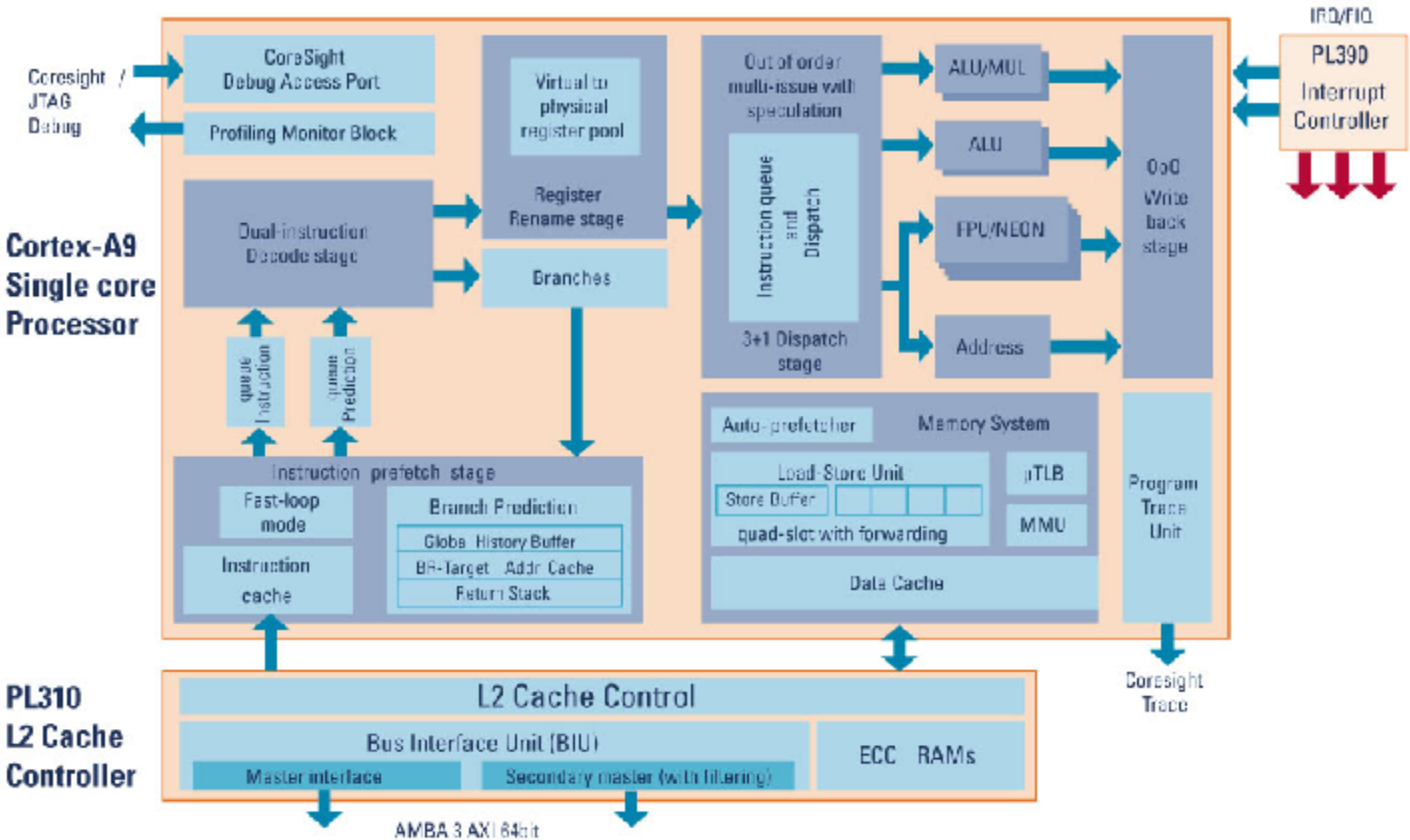

https://devblogs.nvidia.com/parallelforall/inside-volta/

# ARM Cortex A9

- Up to 4 cache-coherent Cortex-A9 cores
  - ARM v7 instruction set (afaik, introduced Thumb)
- Can run up to 2 GHz
- Apple A5 (iPad 2) has 2 A9 cores + 2 PowerVR cores, 512 MB DRAM, 1 GHz
  - iSuppli estimates $14

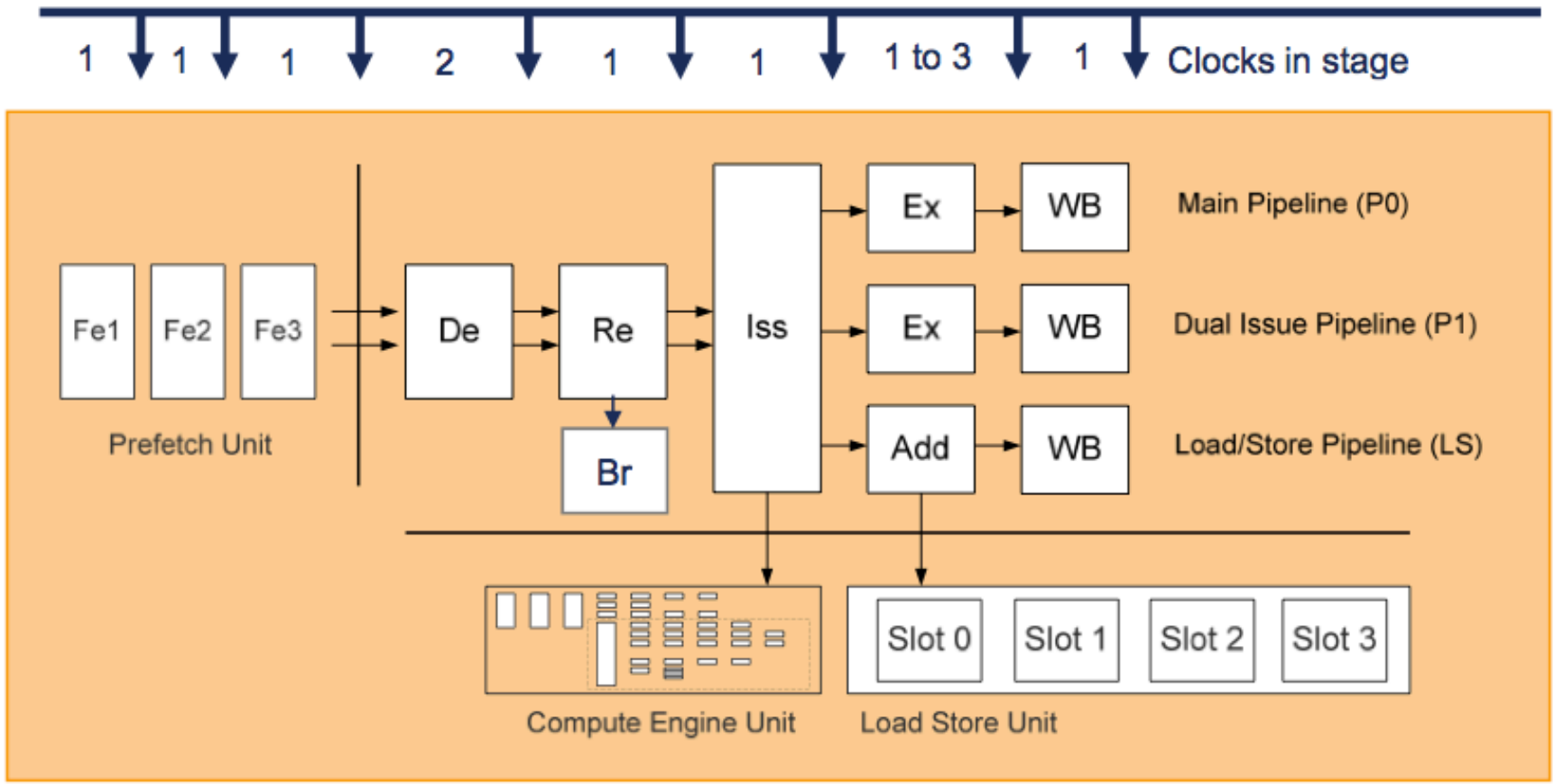
http://www.arm.com/products/processors/cortex-a/presentations.php

# A9 Single Core




http://www.linuxfordevices.com/files/misc/arm_cortex_a9_single_core_diag.gif

# ARM Cortex A9 Core

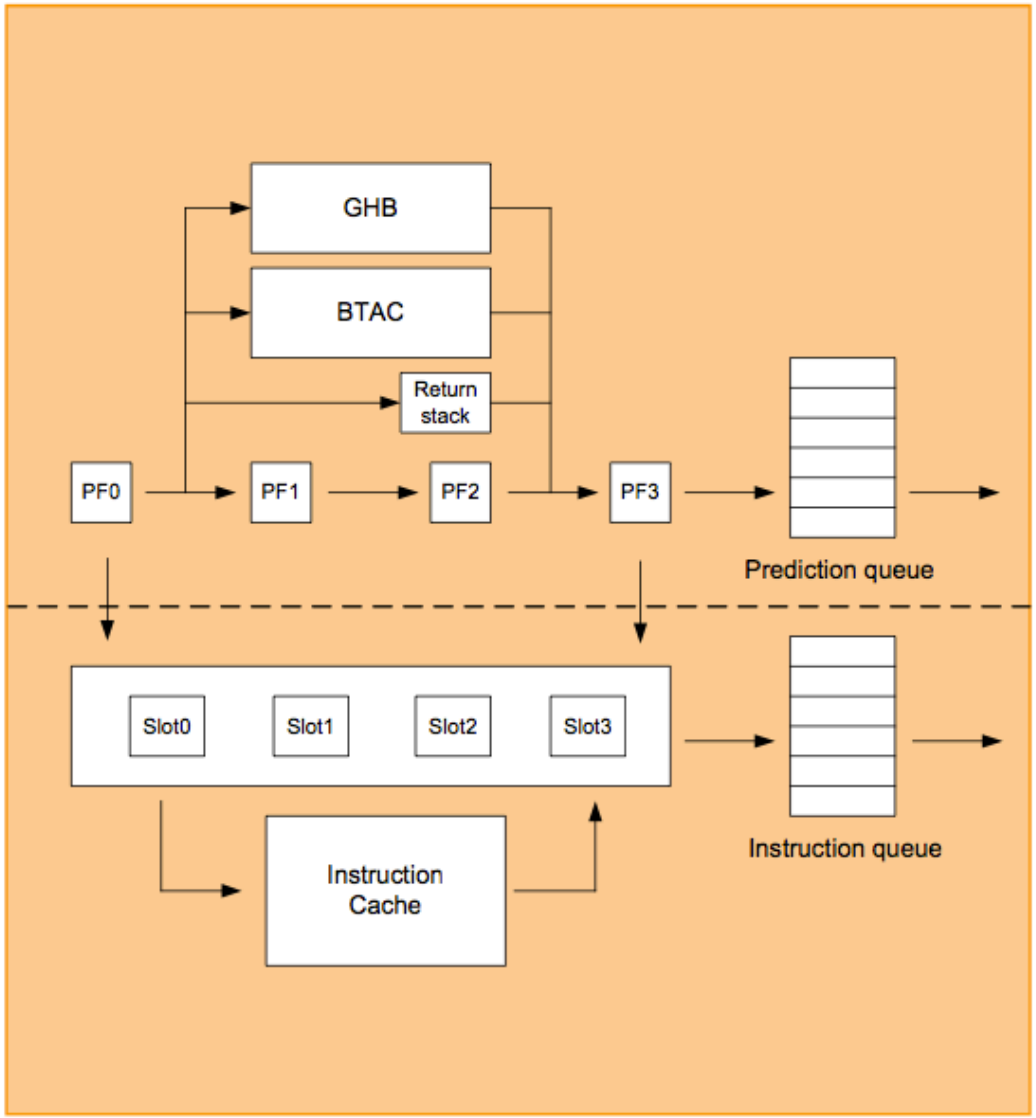



- Dispatching up to 4 instructions per clock cycle
  - With out of order dispatch from the issue queue to ensure maximum utilization of subsequent pipeline stages
  - With up to 7 instructions completing per cycle (including FPU/NEON)
- Compute engine (FPU/NEON) executing in parallel

# More ARM A9 Details

- Variable length, out of order, 8 stage superscalar pipeline
  - Advanced prefetch with parallel branch pipeline enabling early branch prediction and resolution
  - Multi-issued into:
    - Primary data processing pipeline
    - Secondary full data processing pipeline
    - Load-store pipeline
    - Compute engine (FPU/NEON) pipeline
- Speculative execution
  - Supporting virtual renaming of ARM physical registers removing pipeline stalls due to data dependencies
    - Removing dependency on compiler instruction scheduling
    - Increasing processor utilization and hiding of memory latencies
    - Increased performance by hardware unrolling of code loops
  - Reducing interrupt latency by speculative entry to ISR

# ARM A9 Branch Prediction



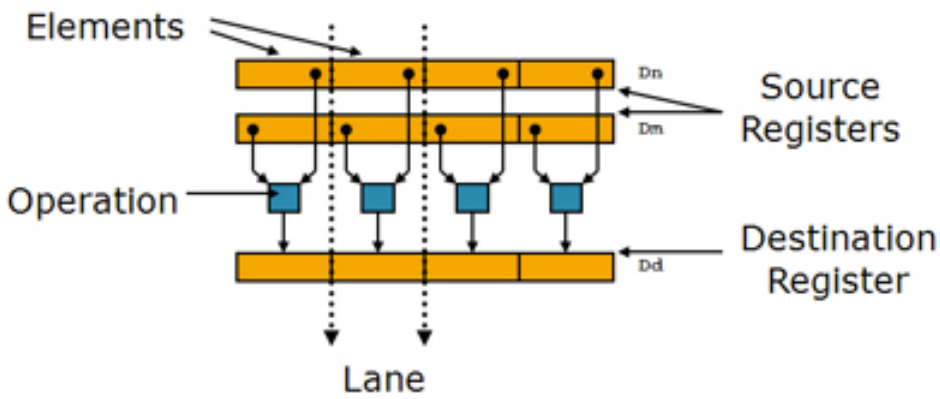

- Branch prediction using a Branch Target Address Cache (BTAC) and Global History Buffer (GHB)
  - Early in the pipeline to reduce the branch mispredict penalty.
  - Benchmarked to confirm final size
    - BTAC is 256 entries x 2ways to 1024 entries x 2ways
    - GHB between 1K to 4K entries
  - Return stack of 4 x 32bits
- Optimization of small loops
  - Removal of branch penalty
  - No RAM power consumption
- Branch folding is done in Integer Core

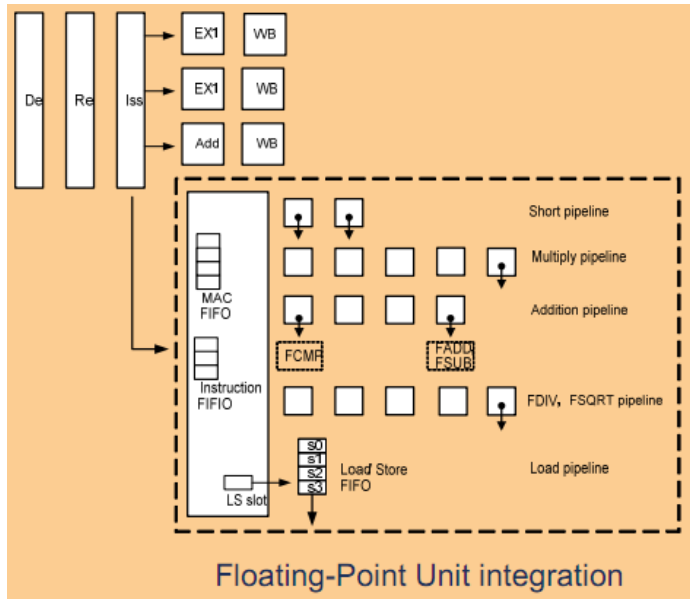
"Branch folding is a technique where, on the prediction of most branches, the branch instruction is completely removed from the instruction stream presented to the execution pipeline. Branch folding can significantly improve the performance of branches, taking the CPI for branches below 1."

# ARM NEON

NEON instructions perform "Packed SIMD" processing:

- Registers are considered as **vectors** of **elements** of the same **data type**
- Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single precision floating point
- Instructions perform the same **operation** in all **lanes**



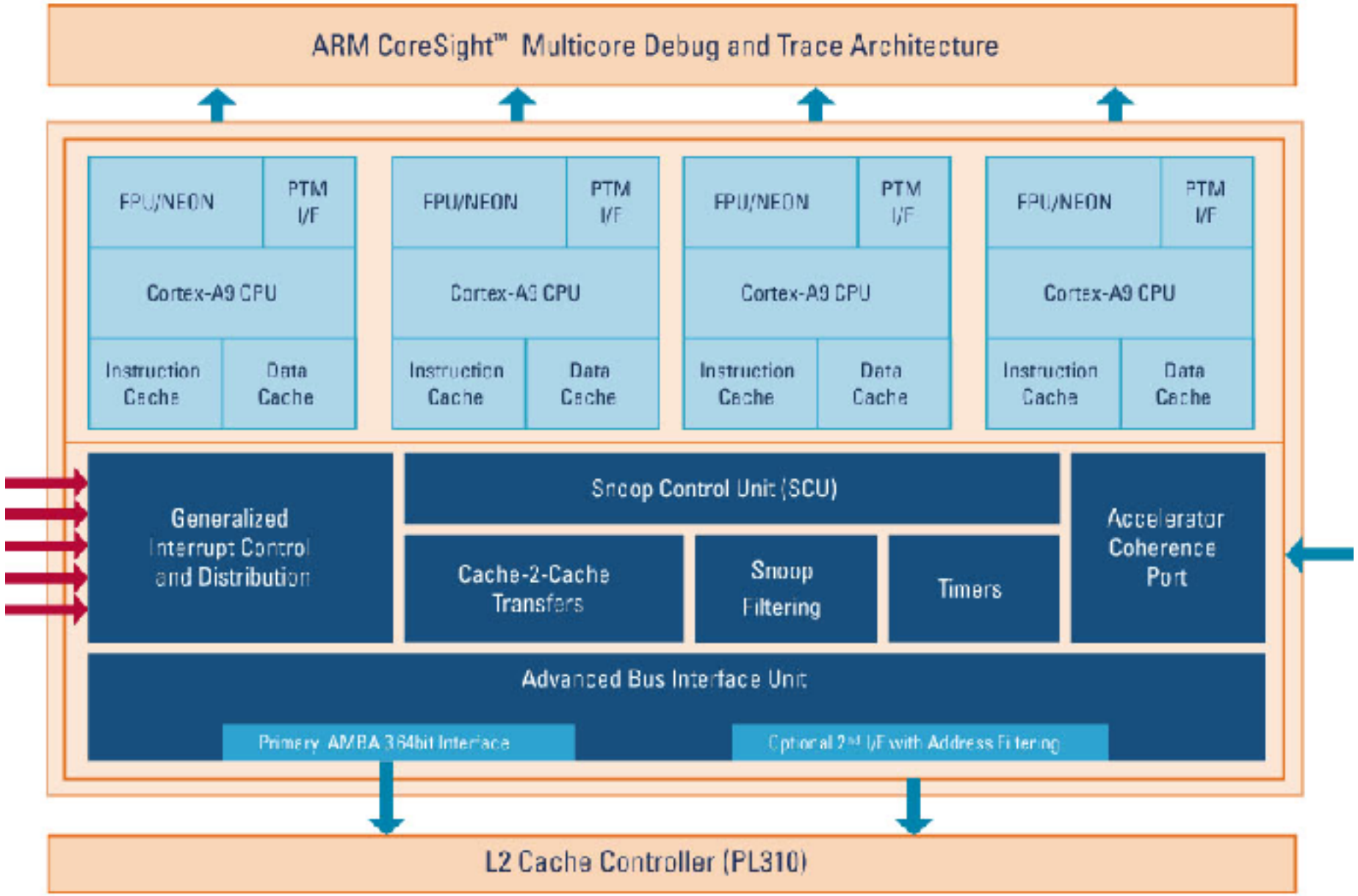



Floating-Point Unit integration

**NEON technology features a number of elements to increase performance and simplify software development, such as:**

- Aligned and unaligned data access allows for efficient vectorization of SIMD operations.
- Clean instruction set architecture designed for autovectorizing compilers and hand coding.
- Efficient access to packed arrays such as ARGB or xyz coordinates
- Support for both integer and floating point operations ensures adaptability to a broad range of applications, from codecs to High Performance Computing to 3D graphics.
- Tight coupling to the ARM processor provides a single instruction stream and a unified view of memory, presenting a single development platform target with a simpler tool flow.
- The large NEON register file with its dual 128-bit/64-bit views enables efficient handling of data and minimizes access to memory, enhancing data throughput.

# Multi-core ARM A9




http://www.linuxfordevices.com/files/misc/arm_cortex_a9_mpcore_diag.gif

# ARM A9 Coherency

## About Cortex-A9 MPCore coherency

Memory coherency in a Cortex-A9 MPCore is maintained following a weakly ordered memory consistency model.

Cache coherency among multiple L1 caches of the Cortex-A9 processors is enforced when the Cortex-A9 processors are operating in *Symmetric Multi-Processing* (SMP) mode. This mode is controlled by the SMP bit of the Auxiliary Control Register.

To be kept coherent, the memory must be marked as Normal memory, with the following attributes:

- write-back
- Write Allocate
- Shareable.

**Note**

When the Shareable attribute is applied to a memory region that is not write-back Normal memory, data held in this region is treated as non-cacheable.

The SCU connects one to four Cortex-A9 processors to the memory system through the AXI interfaces.

The SCU functions are to:

- maintain data cache coherency between the Cortex-A9 processors
- initiate L2 AXI memory accesses
- arbitrate between Cortex-A9 processors requesting L2 accesses
- manage ACP accesses.

**Note**

The Cortex-A9 SCU does not support hardware management of coherency of the instruction cache.