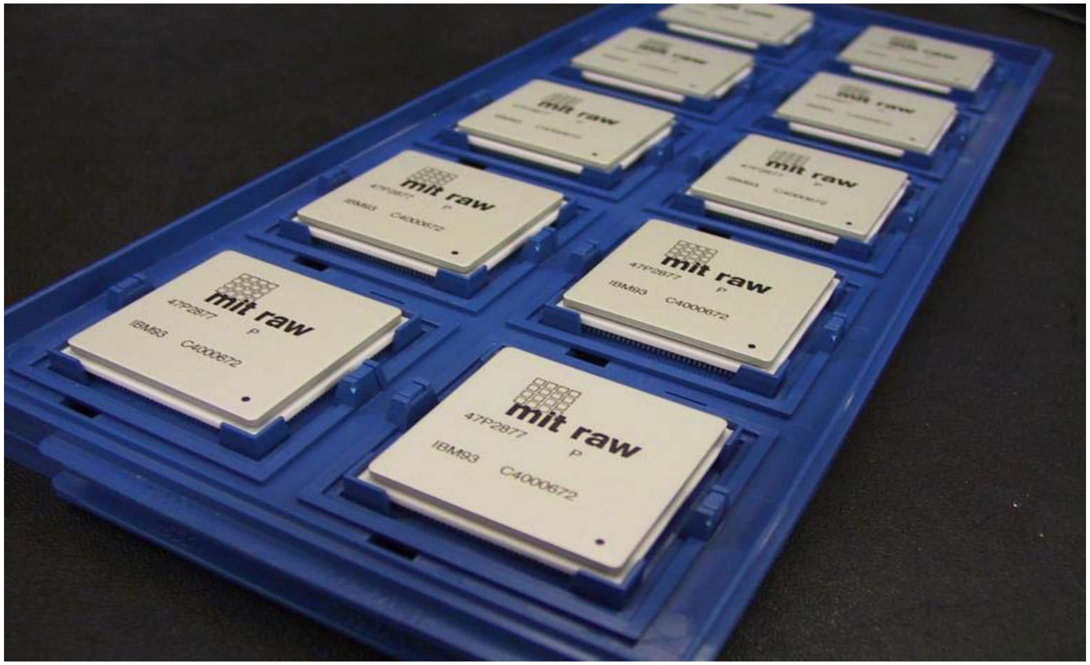
The Raw Tiled Processor Architecture

Is the future of architecture in tiles?

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

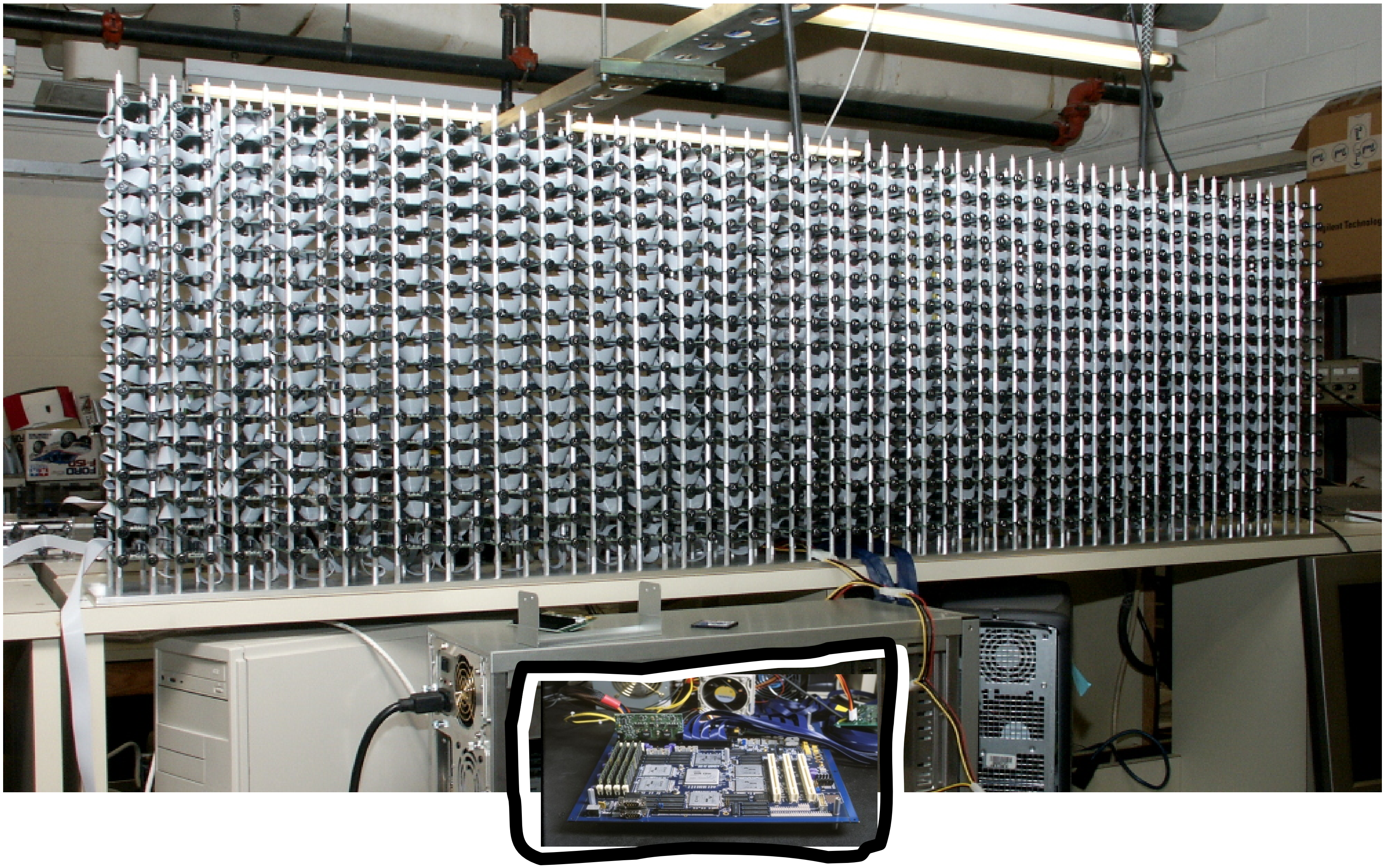
Anant Agarwal
MIT

A tiled processor architecture prototype: the Raw microprocessor



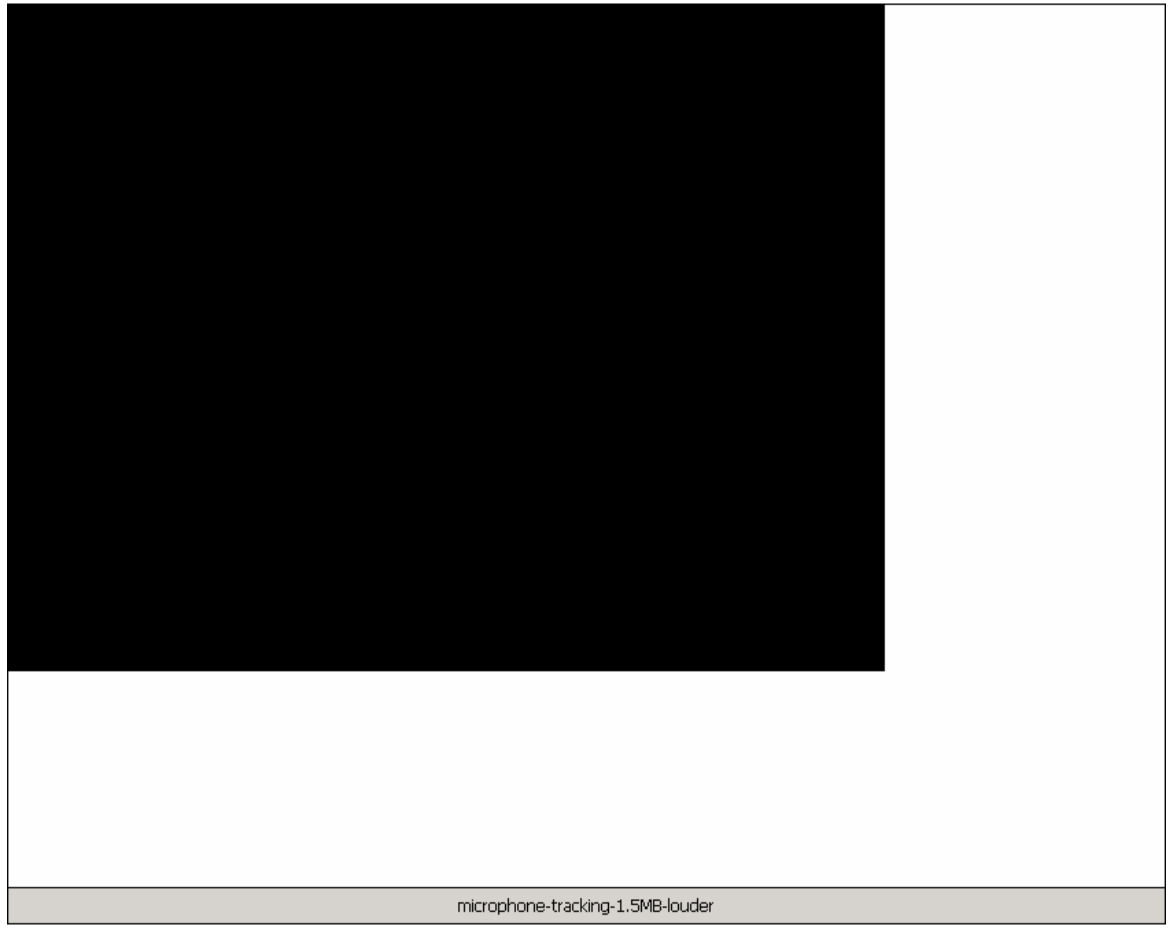
October 02

Embedded system:
1020 Element Microphone Array



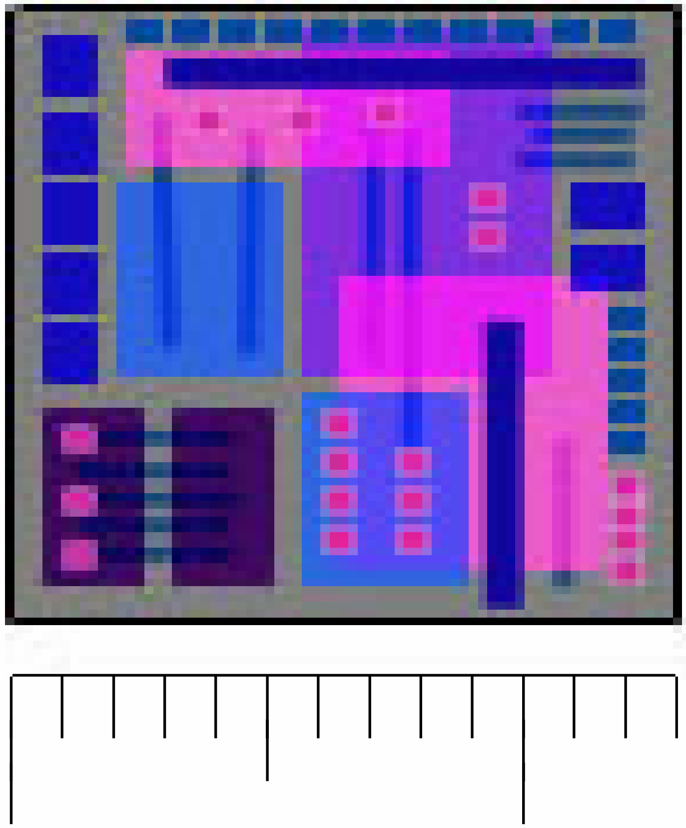
1020 Node Beamformer Demo

- 2 People moving about and talking
- Track movement of people using camera (vision group) and display on monitor
- Beamformer focuses on speech of one person
- Select another person using mouse
- Beamformer switches focus to the speech of the other person



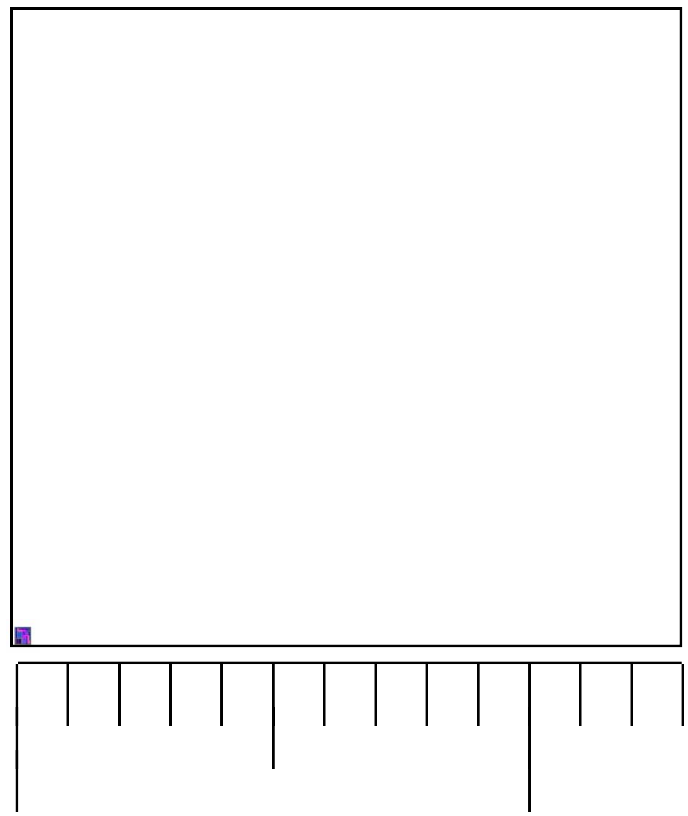
The opportunity

20MIPS cpu
1987



The opportunity

2007

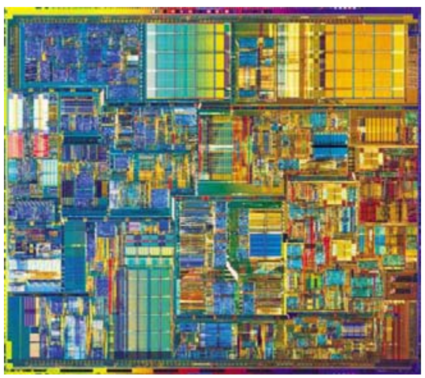


The billion transistor chip

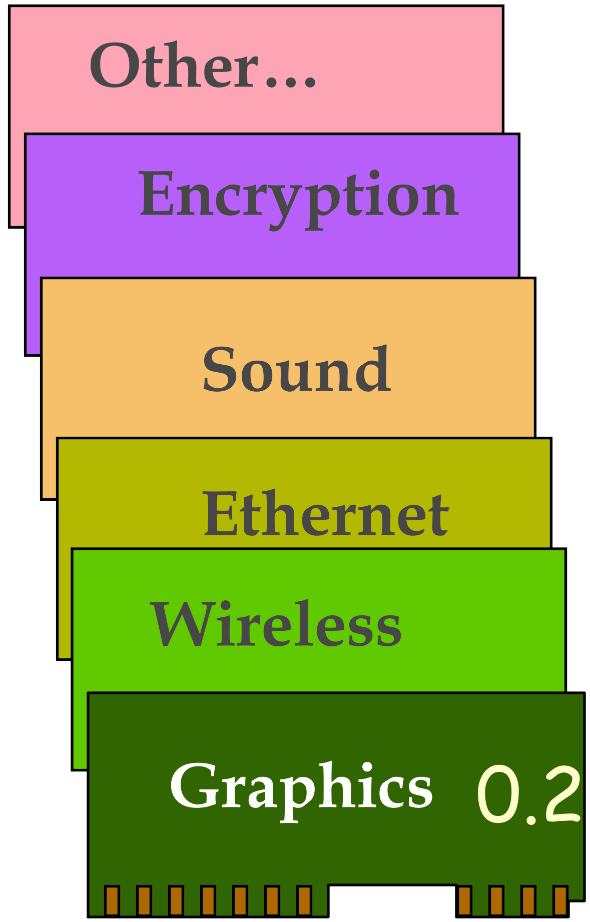
Enables seeking out new application domains

- 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?



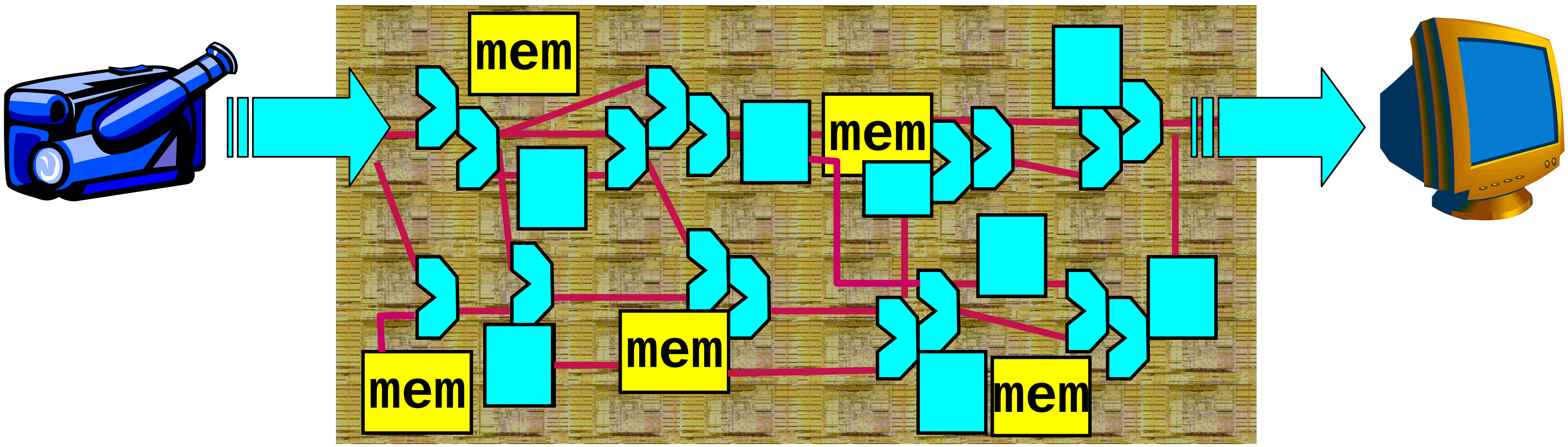
X86
Pentium IV



0.25TFLOPS

ASICs

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

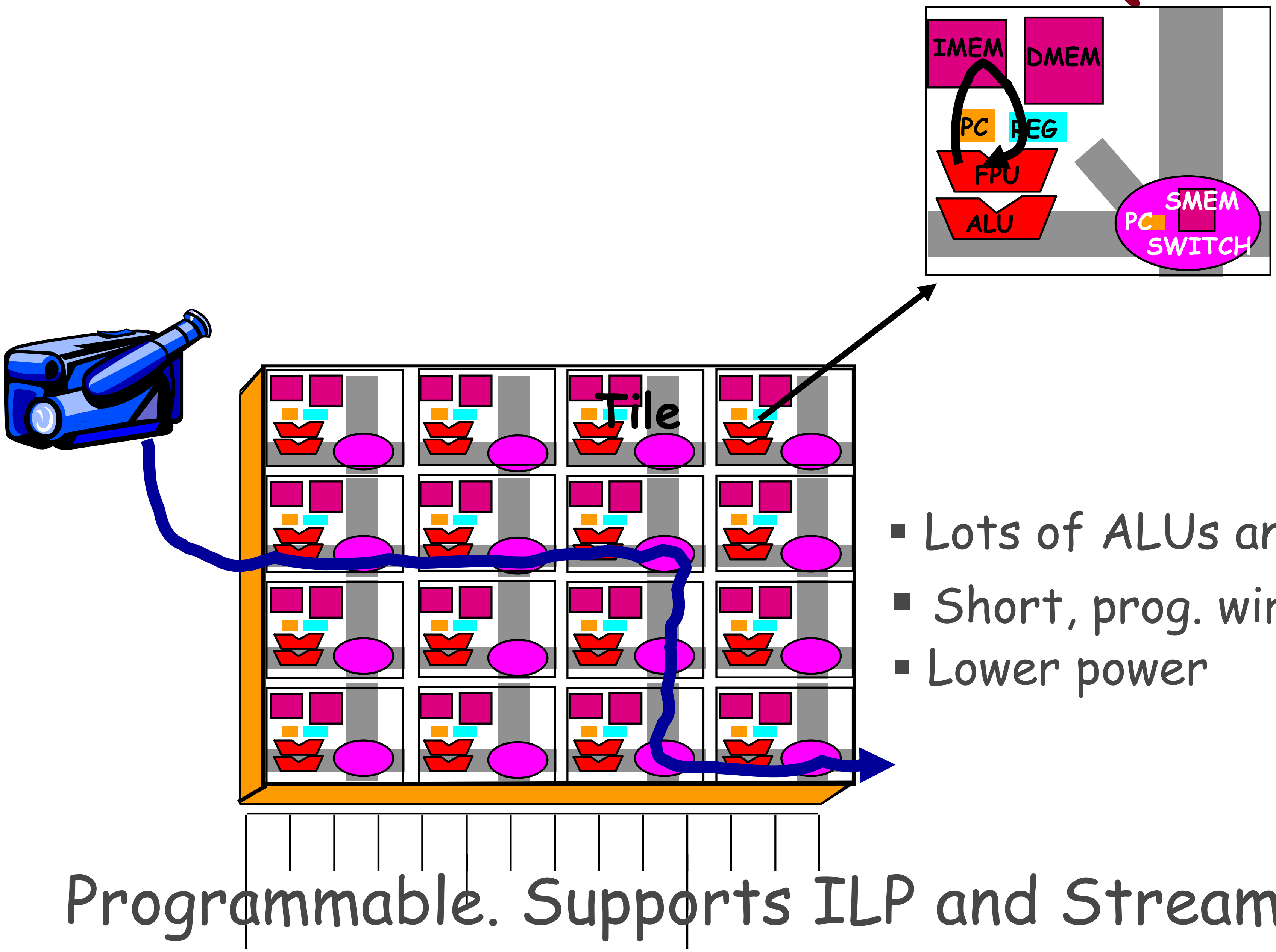
Our challenge

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)

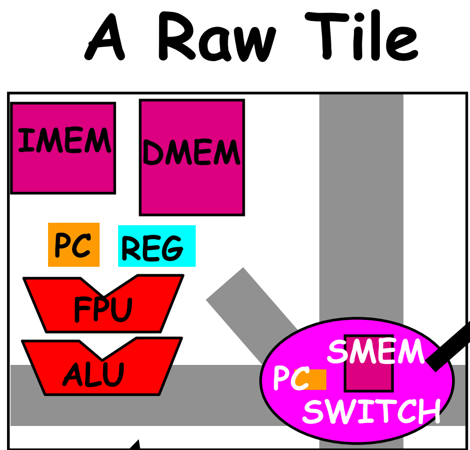


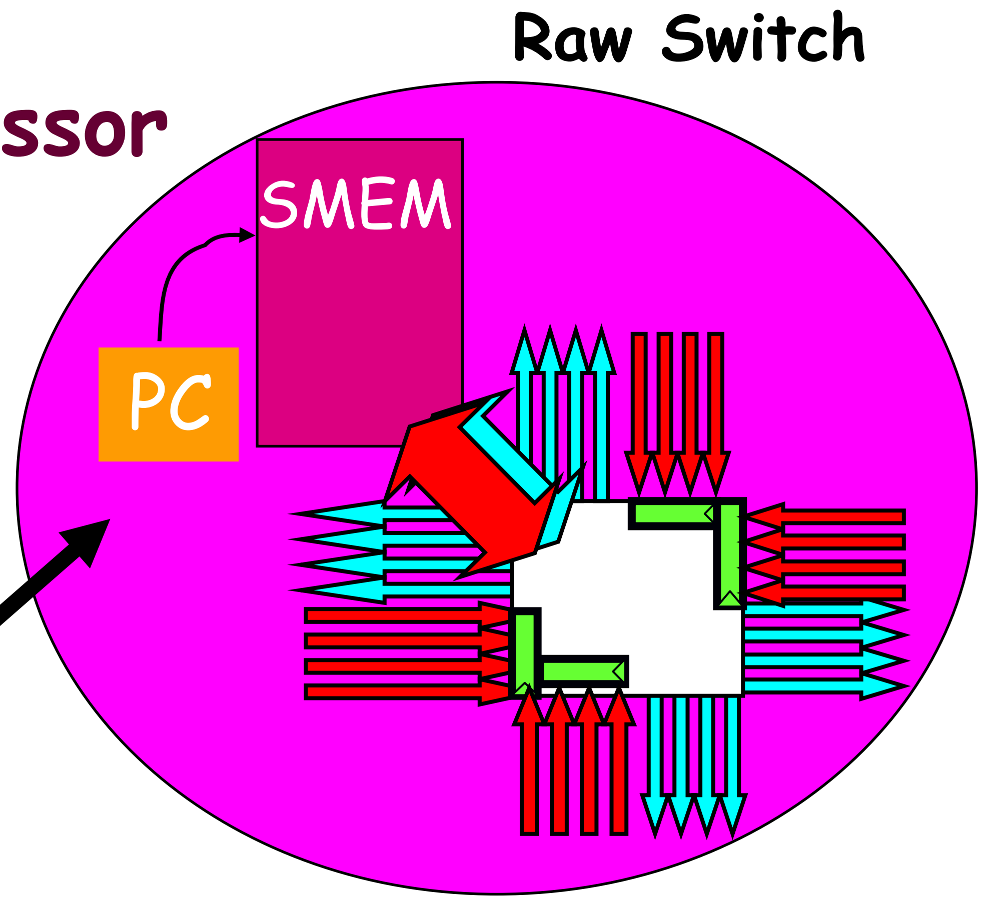
- Lots of ALUs and regs
- Short, prog. wires
- Lower power

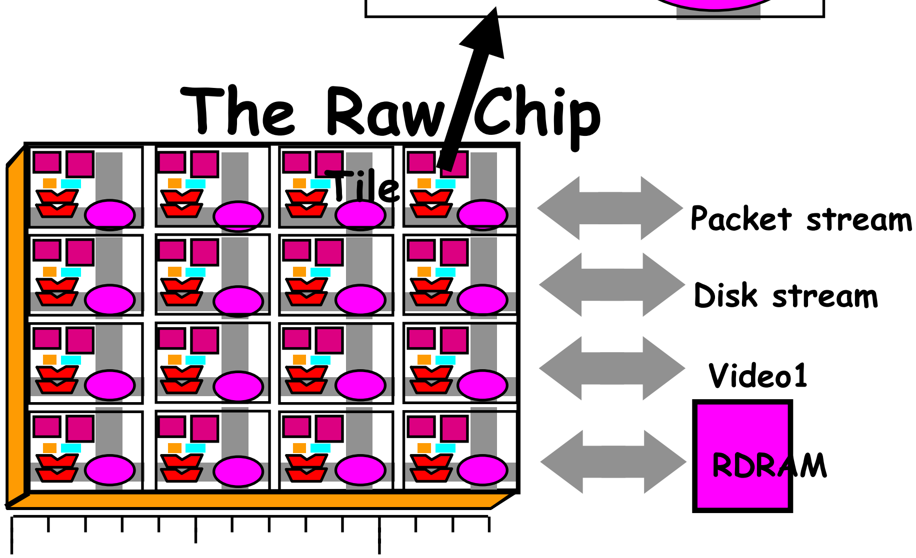
Programmable. Supports ILP and Streams

A Prototype TPA:
The Raw Microprocessor

[Billion transistor IEEE Computer Issue '97]



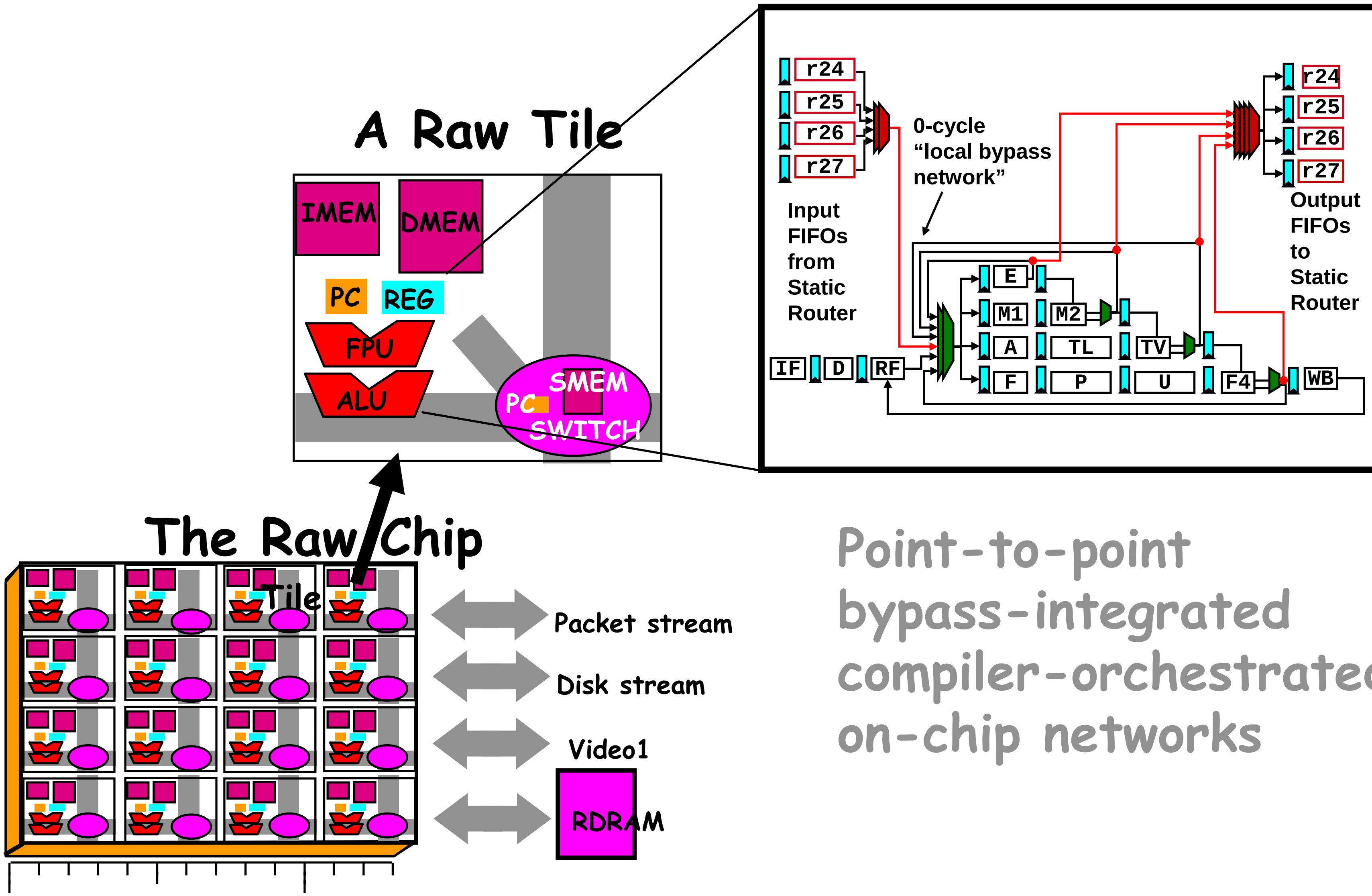




Software-scheduled interconnects

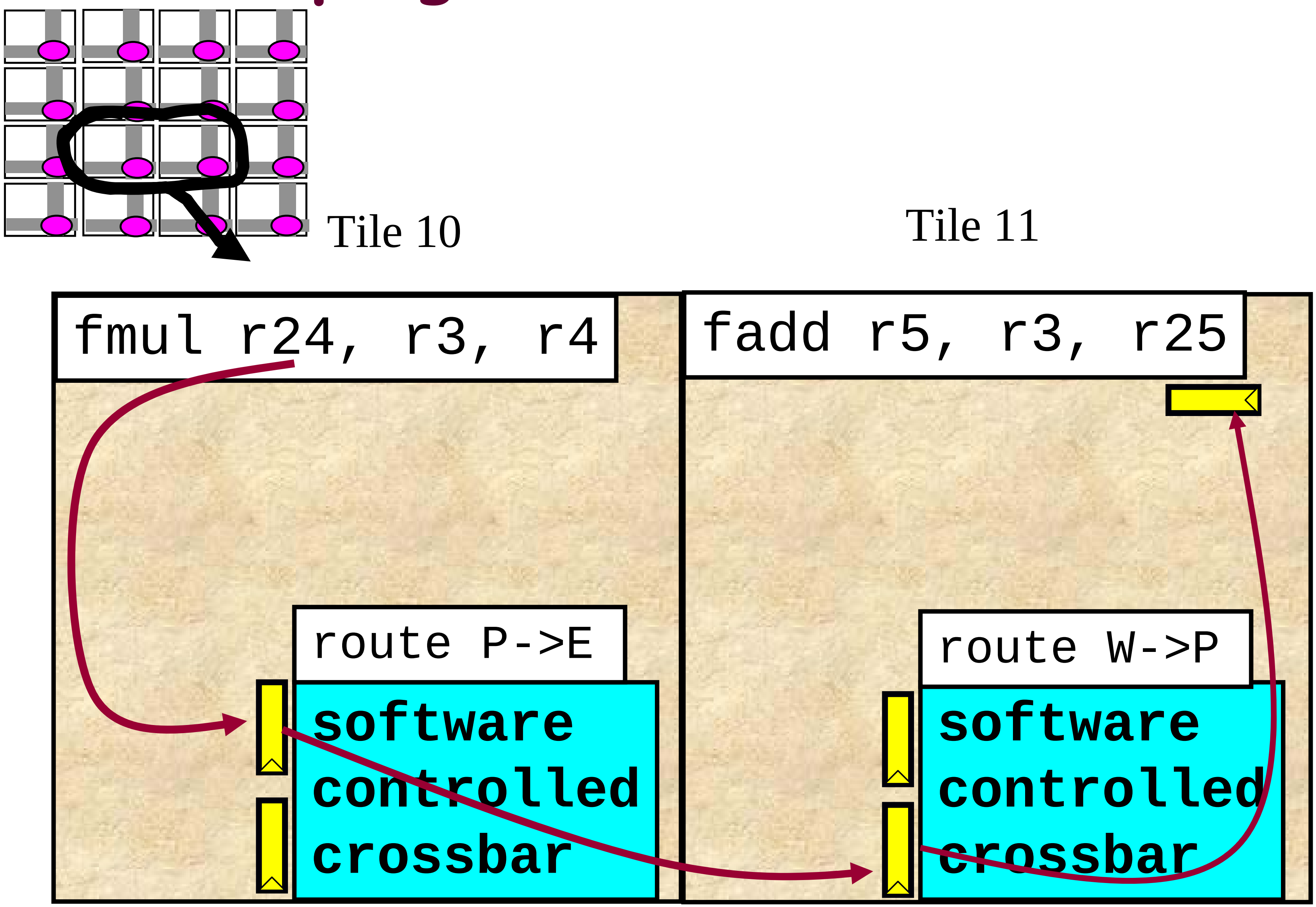
(can use static or dynamic routing – but compiler determines instruction placement and routes)

Tight integration of interconnect

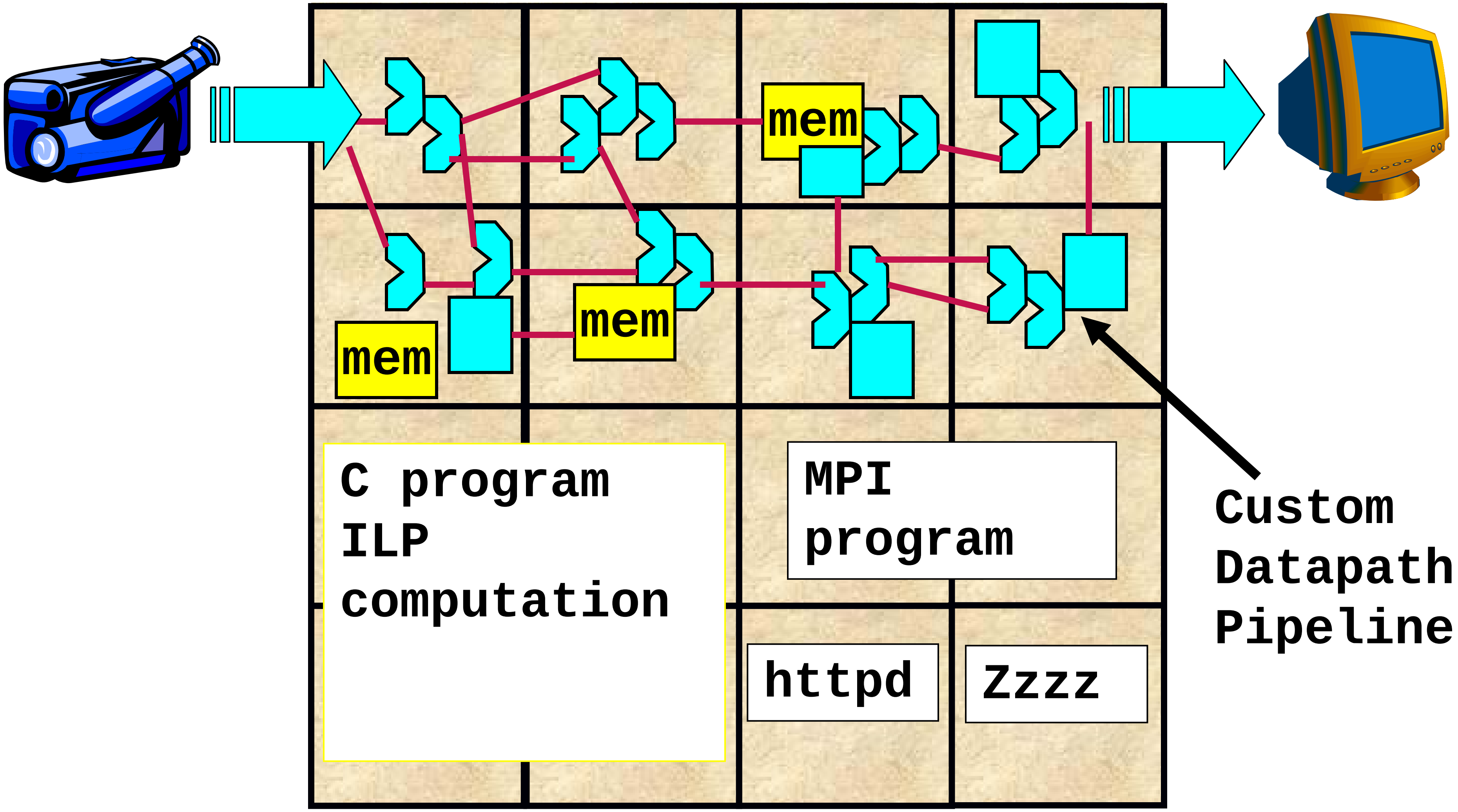


Point-to-point
bypass-integrated
compiler-orchestrated
on-chip networks

How to "program the wires"



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

Programming models and software for tiled processor architectures

- o Conventional scalar programs (C, C++, Java)
  Or, how to do ILP

- o Stream programs

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
```

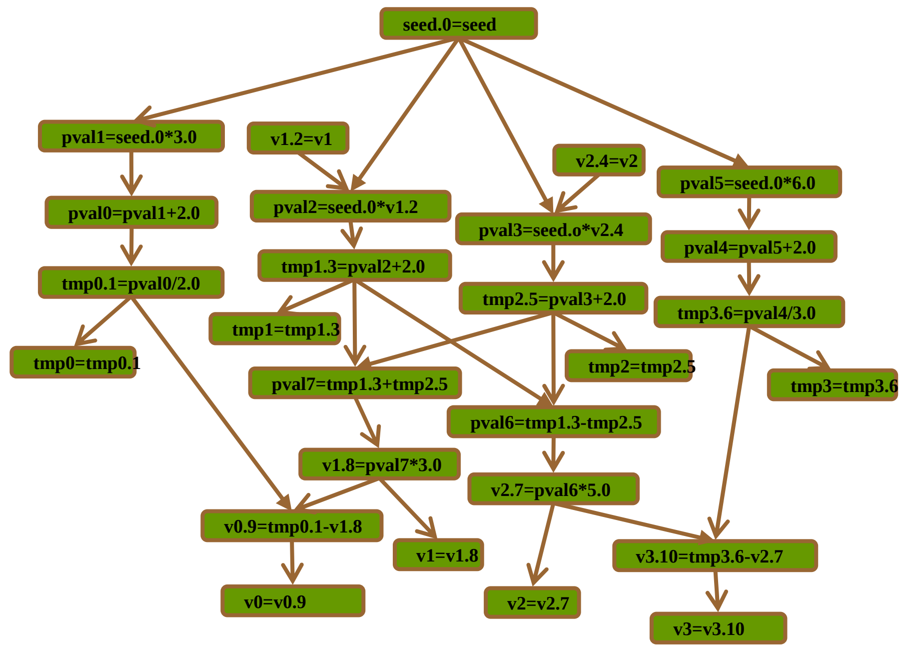
Existing languages will work

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

Scalar program mapping

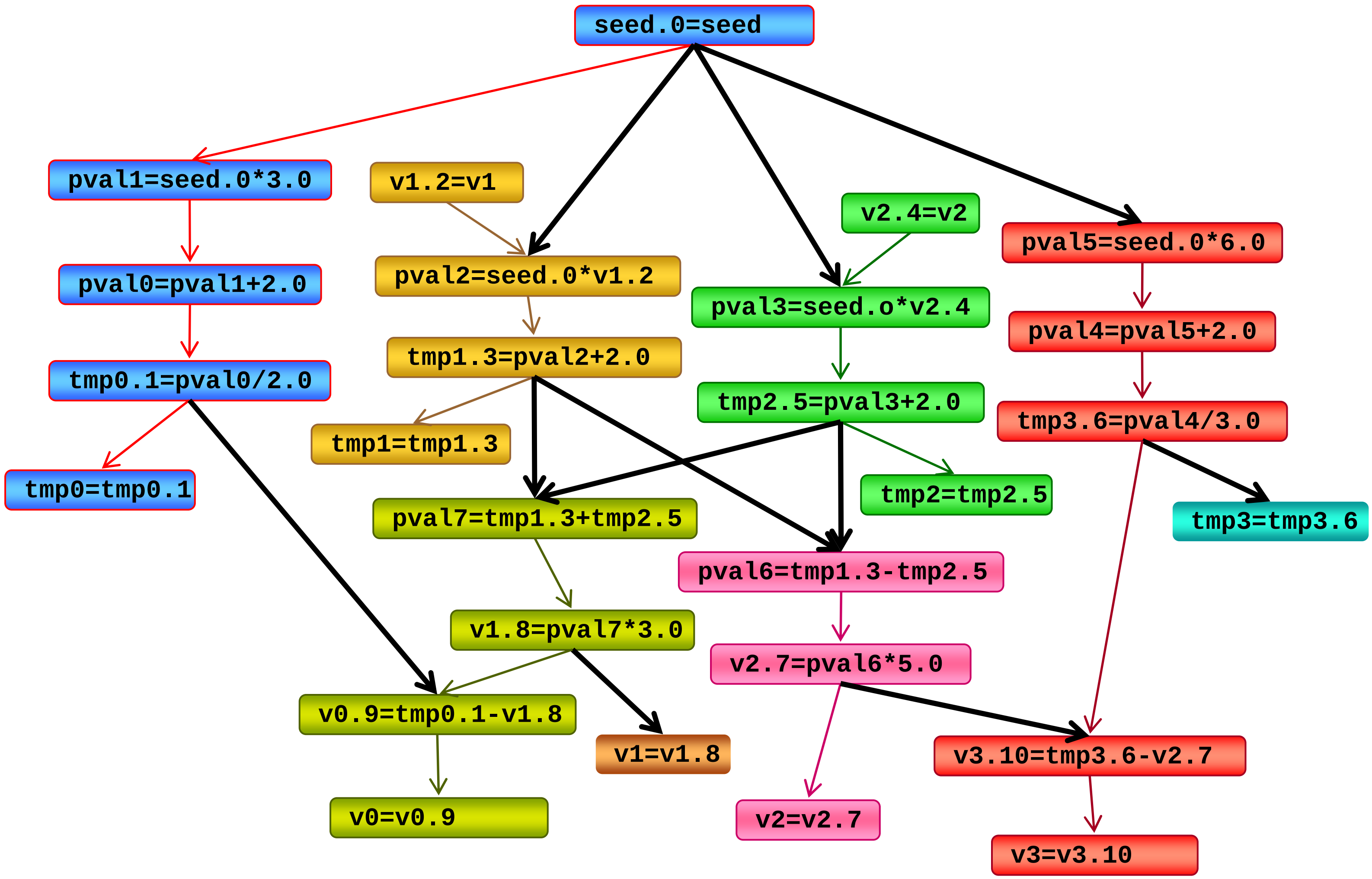
```
v2.4 = v2
seed.0 = seed
v1.2 = v1
pval1 = seed.0 * 3.0
pval0 = pval1 + 2.0
tmp0.1 = pval0 / 2.0
pval2 = seed.0 * v1.2
tmp1.3 = pval2 + 2.0
pval3 = seed.0 * v2.4
tmp2.5 = pval3 + 2.0
pval5 = seed.0 * 6.0
pval4 = pval5 + 2.0
tmp3.6 = pval4 / 3.0
pval6 = tmp1.3 - tmp2.5
v2.7 = pval6 * 5.0
pval7 = tmp1.3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

Program code

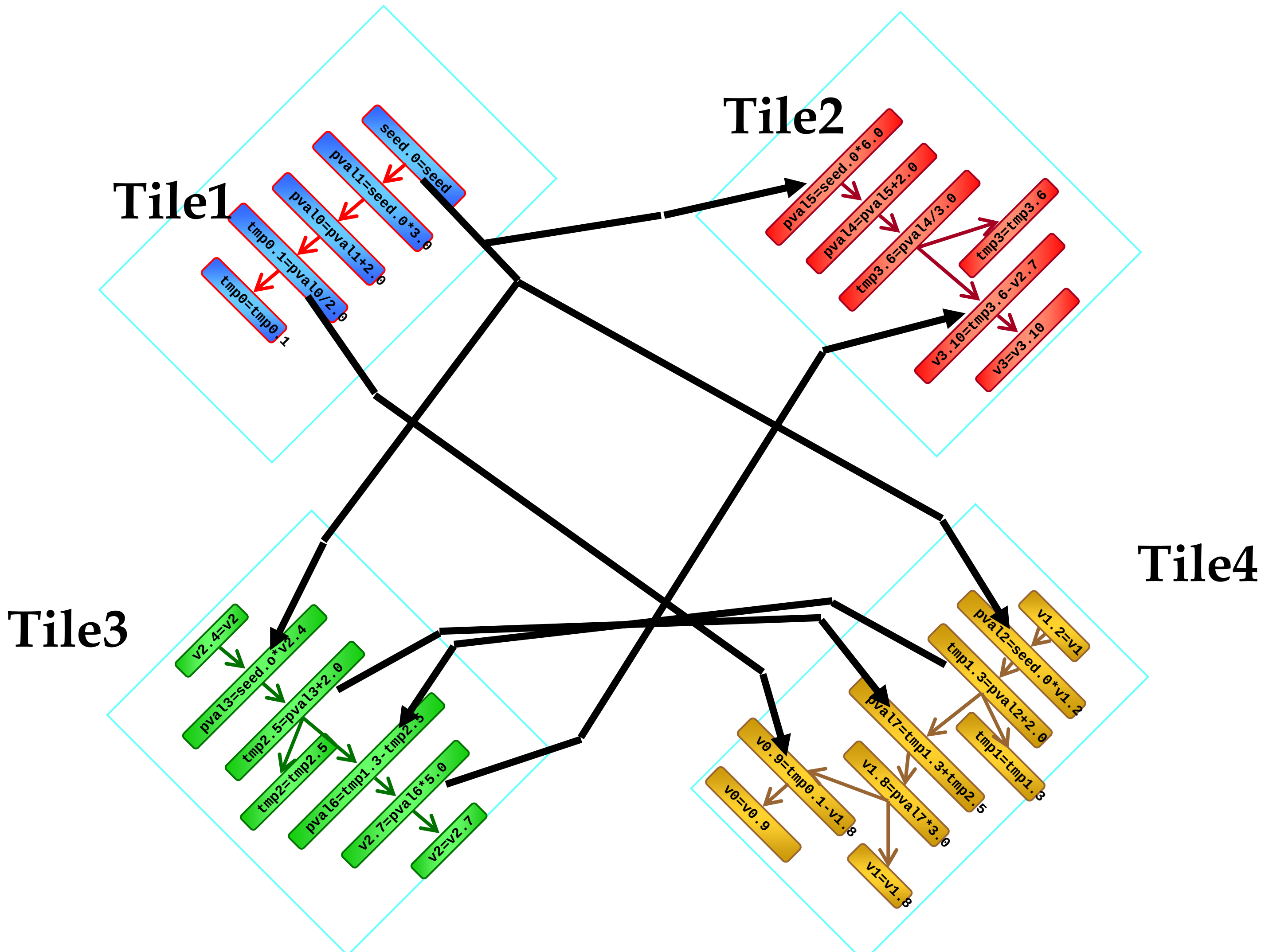


Graph

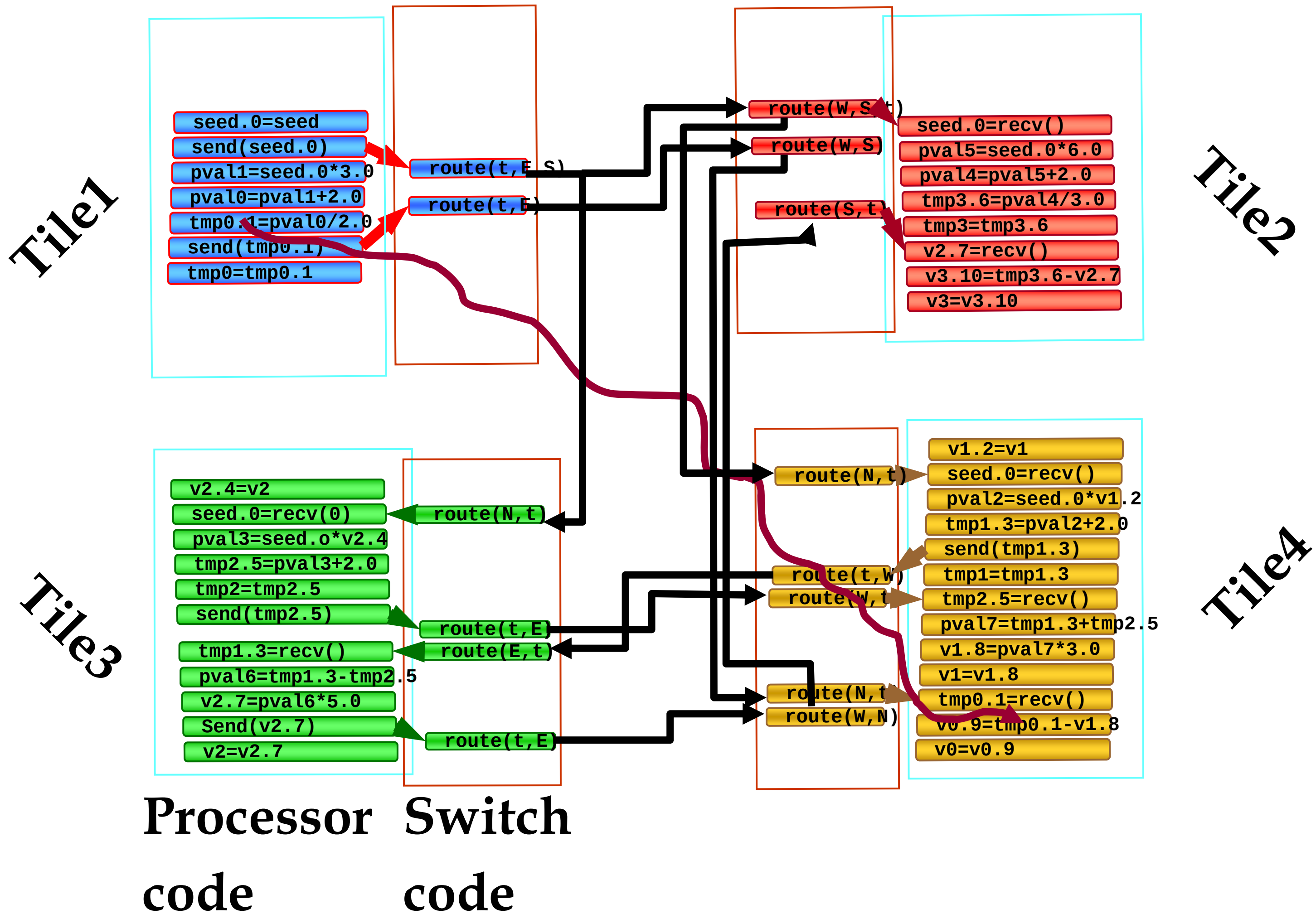
Program graph clustering



Placement

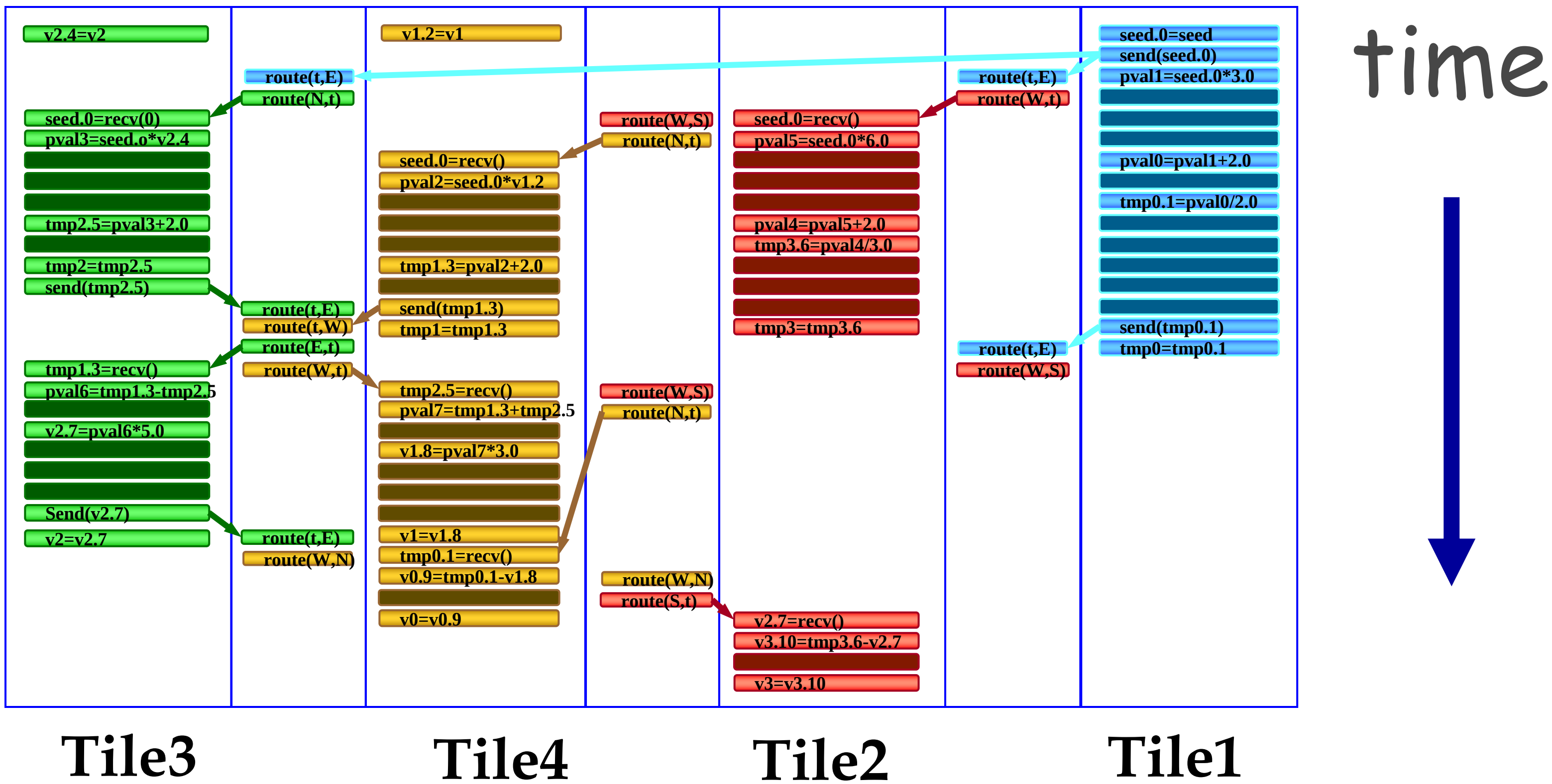

Tile1
seed.0=seed
pval1=seed.0*3.0
pval0=pval1+2.0
tmp0.1=pval0/2.0
tmp0=tmp0.1
Tile2
pval5=seed.0*6.0
pval4=pval5+2.0
tmp3.6=pval4/3.0
tmp3=tmp3.6
v3.10=tmp3.6-v2.7
v3=v3.10
Tile3
v2.4=v2
pval3=seed.o*v2.4
tmp2.5=pval3+2.0
tmp2=tmp2.5
pval6=tmp1.3-tmp2.5
v2.7=pval6*5.0
v2=v2.7
Tile4
v1.2=v1
pval2=seed.0*v1.2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

Routing



Tile1
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)
Tile2
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
Tile3
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)
Tile4
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

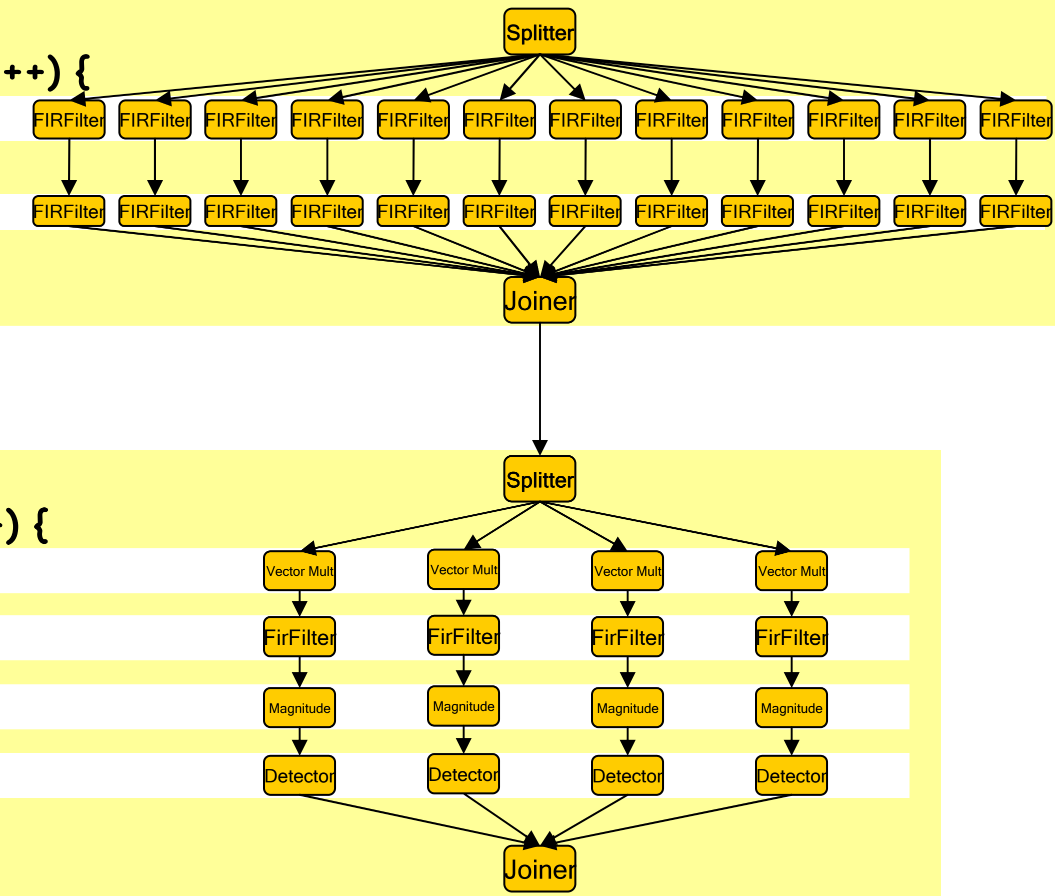


StreamIt: Stream Language and Compiler

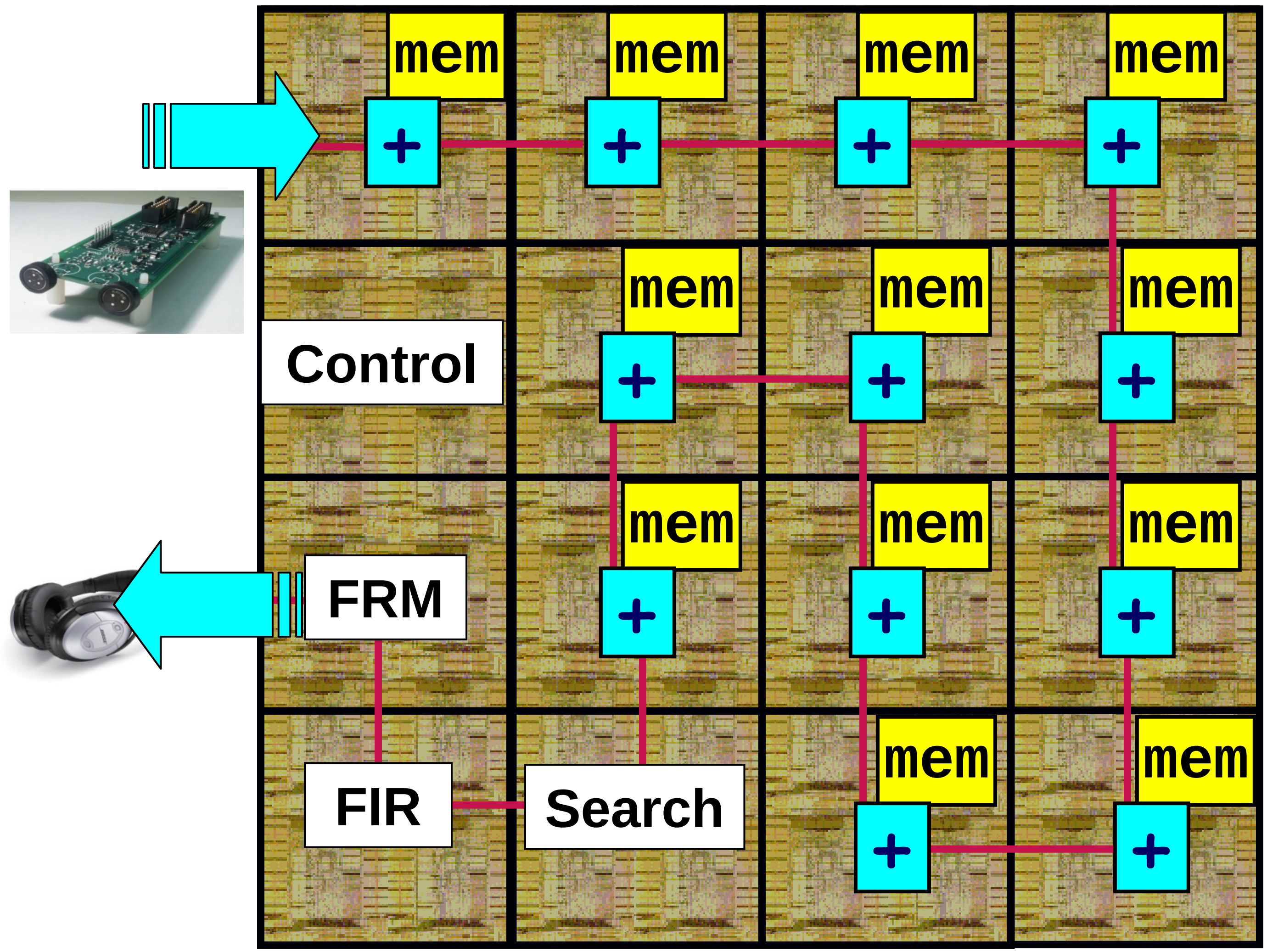
Amarasinghe et al.

e.g., BeamFormer

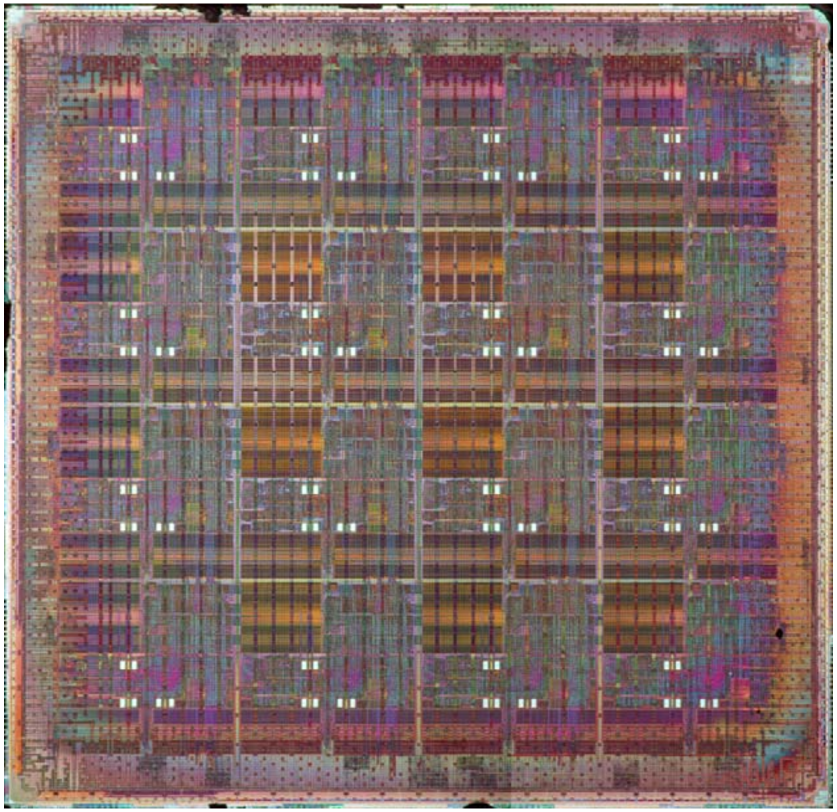
```
class BeamFormer extends Pipeline {
    void init(numChannels, numBeams) {
      add(new SplitJoin() {
        void init() {
          setSplitter(Duplicate());
          for (int i=0; i<numChannels; i++) {
            add(new FIR1(N1));
            add(new FIR2(N2));
          }
          setJoiner(RoundRobin());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



Raw Beamformer Layout (by hand)

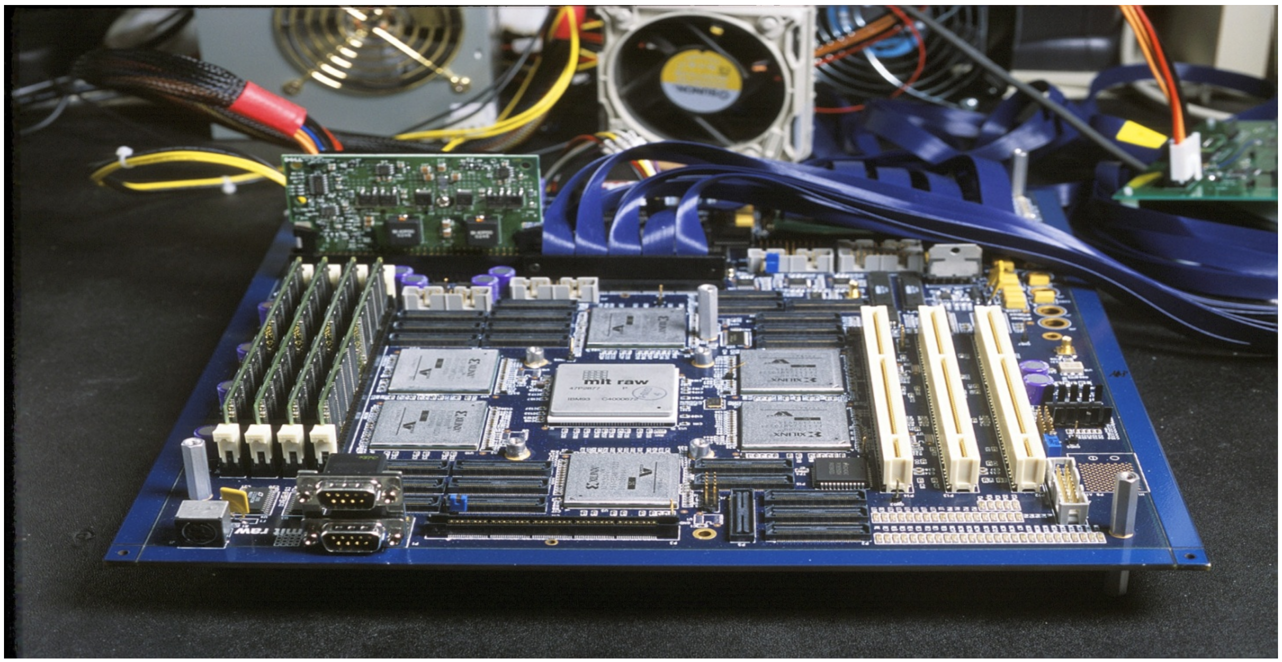


Raw die photo



.18 micron process, 16 tiles, 425MHz, 18 Watts (vpenta)
Of course, custom IC designed by industrial design team could do much better

Raw motherboard



More Experimental Systems

Systems Online or in Pipeline

- Raw Workstation ✓
- Raw-based 1020 Microphone Array ✓
- Raw 802.11a/g wireless system
  (collab with Engim)
- Raw Gigabit IP router
- Raw graphics system
- Raw supercomputing fabric

Empirical Evaluation

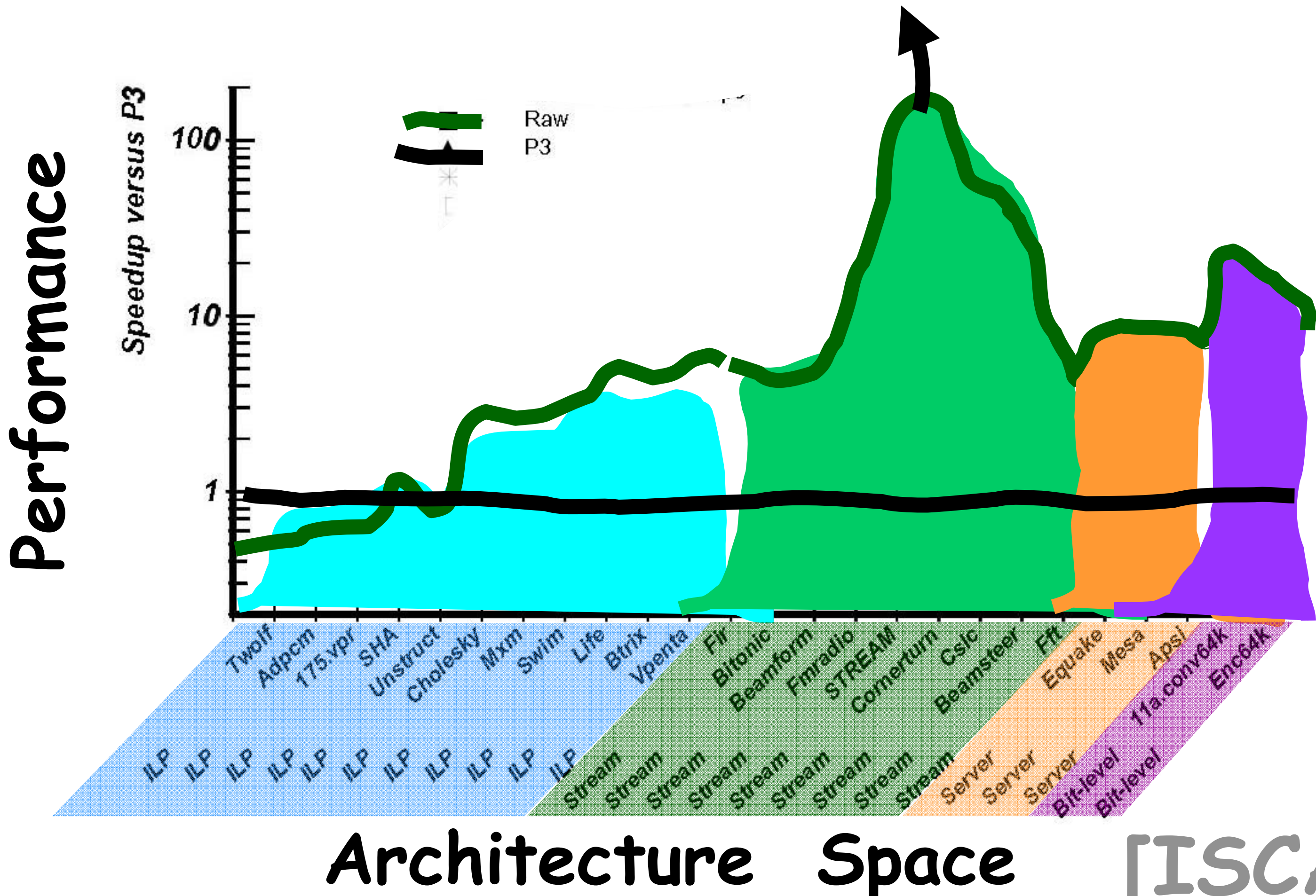
Compare to P3 implemented in similar technology

Parameter	Raw (IBM ASIC)	P3 (Intel)
Litho	180 nm	180 nm
Process	CMOS 7SF	P858
Metal Layers	Cu 6	Al 6
FO1 Delay	23 ps	11 ps
Dielectric *k*	4.1	3.55
Design Style	Standard Cell SA27E ASIC	Full custom
Initial Freq	425 MHz	500-733 MHz (use 600)
Die Area	331 mm^2	106 mm^2

Raw #s from cycle-accurate simulator validated against real chip
-- FPGA mem controller in Raw -- Raw SW i-caching adds 0-30% ovhd

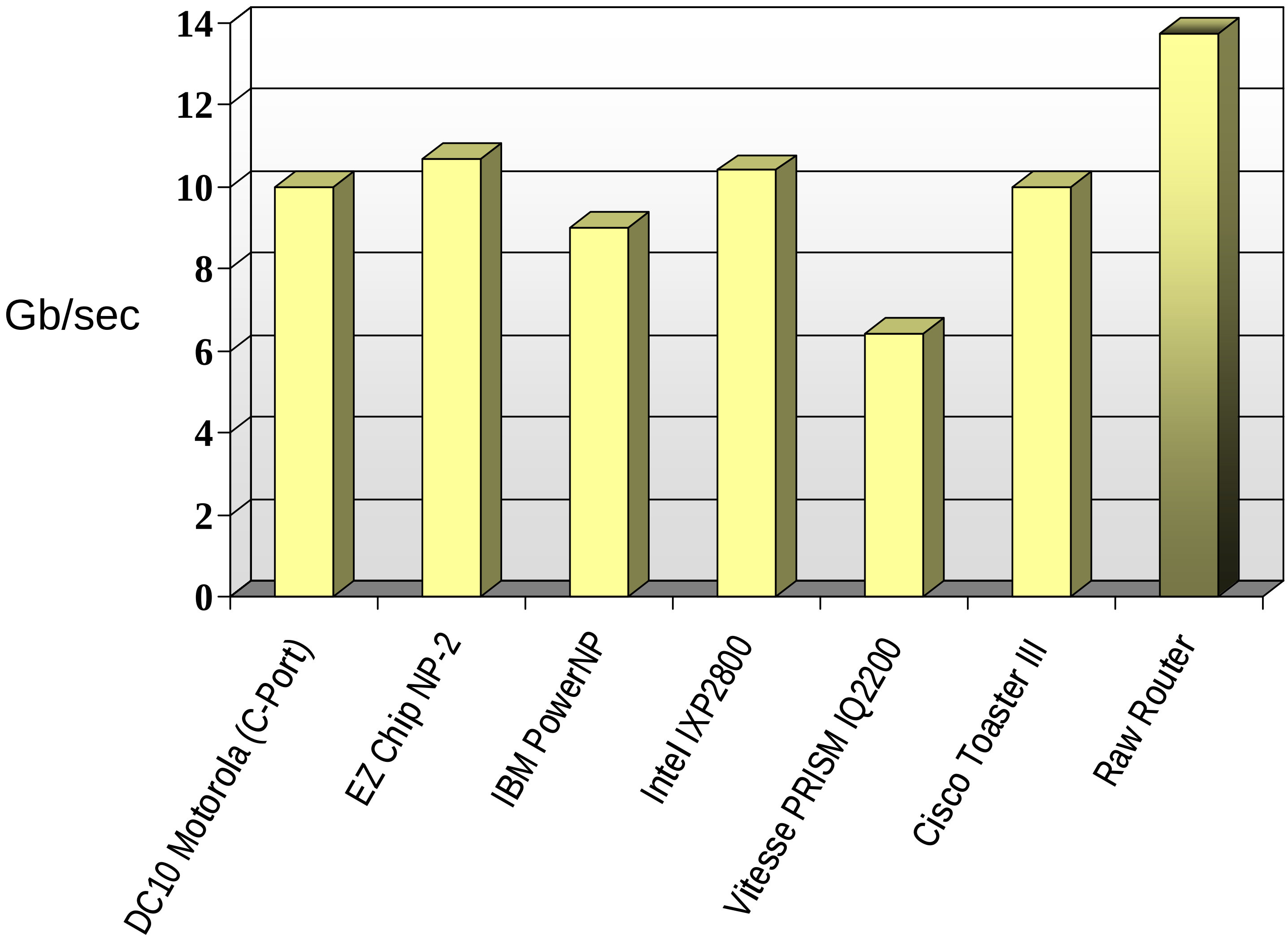
Performance Results

~10x parallelism
~ 4x ld/store elim
~ 4x stream mem bw



[ISCA'04]

Raw IP Router



VersaBench

www.cag.csail.mit.edu/versabench

Sharing a benchmark set to stress
versatility of processors

Categories of programs:
ILP - Desktop and Scientific
Streams
Throughput oriented servers
Bit-level embedded

Summary

Raw: single chip for ILP and streaming

Scalar operand network is key to
ILP and streams

Tiled processor chip demonstrated in 2002

www.cag.csail.mit.edu/raw