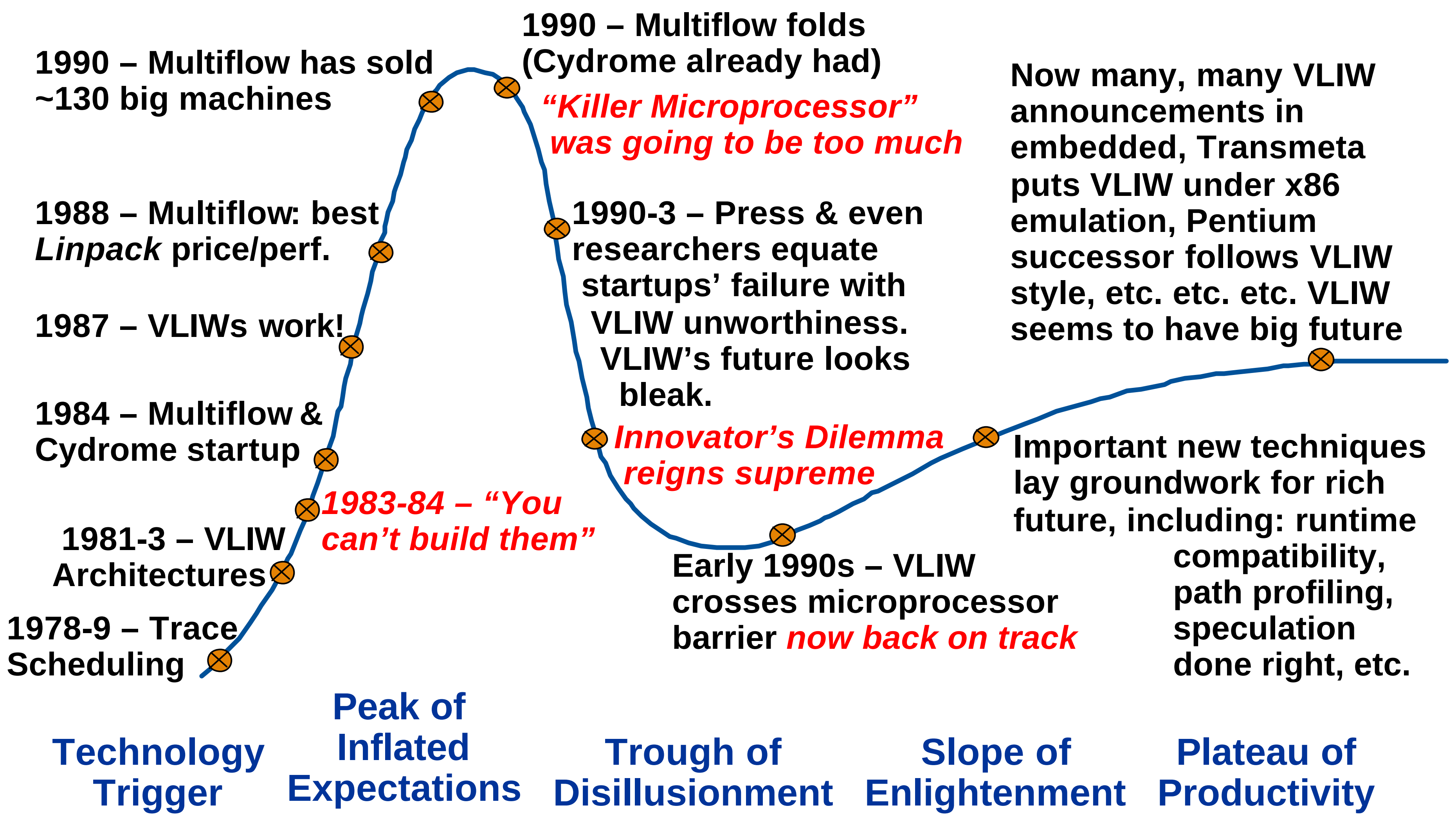
The History of VLIWs in One Slide
(A "Short Subject" Before the Main Feature)

Josh Fisher
Senior Fellow
Hewlett-Packard
josh.fisher@hp.com

Using the Gartner Group's Hype Cycle To Look At The History of VLIWs



1978-9 – Trace Scheduling

1981-3 – VLIW Architectures

1983-84 – "You can't build them"

1984 – Multiflow & Cydrome startup

1987 – VLIWs work!

1988 – Multiflow: best *Linpack* price/perf.

1990 – Multiflow has sold ~130 big machines

1990 – Multiflow folds (Cydrome already had)

"Killer Microprocessor" was going to be too much

1990-3 – Press & even researchers equate startups' failure with VLIW unworthiness. VLIW's future looks bleak.

Innovator's Dilemma reigns supreme

Early 1990s – VLIW crosses microprocessor barrier *now back on track*

Important new techniques lay groundwork for rich future, including: runtime compatibility, path profiling, speculation done right, etc.

Now many, many VLIW announcements in embedded, Transmeta puts VLIW under x86 emulation, Pentium successor follows VLIW style, etc. etc. etc. VLIW seems to have big future

Moving From Embedded Systems to Embedded Computing (Some Lessons in Embedded Processing)

Josh Fisher
Senior Fellow
Hewlett-Packard
josh.fisher@hp.com

Lessons in Embedded Processing

“This was one of the lessons from the Deep Blue project: only in exceptional cases does it make sense to design special purpose hardware for a particular application (e.g. to play chess). *Usually it is better to rely on general purpose processors.”*

http://www.research.ibm.com/resources/news/20021024_deepblue.shtml

This talk is about a few of the lessons I’ve learned myself in Embedded Processing, and some important changes that are occurring in that field.

Embedded & Me

In 1994, HP & Intel signed an R&D agreement that would result in the IA-64’s emergence from the HP Labs “Wide-Word” VLIW project.

My own work, and that of HPL Cambridge, which I had just started, was centered on that project. Clearly we’d be swamped by armies of engineers.

I fled; I moved my own work and that of my lab mostly into embedded computing: Trying to design a framework that would spit out customized cores.

Embedded Systems?

That project was commercially very successful (though I can’t go into details).

I’ve learned a lot doing this; some of it very unexpected. This talk is about some of what I’ve learned.

Lesson #1: was that there’s really big gulf, which my colleagues and I characterize as:

“Embedded Systems” vs. “Embedded Computing”

Because of Moore's Law, Embedded Systems are Moving to Embedded Computing

Embedded Systems

- Processor core is a commodity
- Many nonprogrammable computing elements
- Peripherals
- Critical buses
- Analog interfaces
- Heroic verification
- UML
- Etc.

Embedded Computing

- High-performance computing is the center of the system
- Most important functionality moved into processor core
- The important parts of the application are all written in software and compiled

A Huge Gulf Between Communities

Embedded Systems Developers

- … have tremendous skills in ASIC design &/or DSPs
- They use extensive libraries of "IP", and…
- …have partners of all varieties with IP blocks to sell for flat fees or royalties
- **There is far too little knowledge of what I would call advanced computer systems design**

General-Purpose Developers

… know perfectly well what embedded systems are:

really small computer systems, with limited capabilities

(maybe with a real-time or other similar twist you have to consider.)

They're usually clueless about what really is different in an embedded system.

Anecdotes

I was told this in all seriousness a few years ago by a very capable microprocessor designer:

“Oh, we have a lot competence in System-on-Chip in our lab. We integrated L1 cache on our last processor.”

Anecdotes

This happened to us in our lab in Cambridge:

We were working with a very capable group of engineers in a high-end printer division of HP. They had a severe performance problem with the embedded application in an important product, and were going through a lot of trouble to get a faster microprocessor. …

Anecdotes

… We were interested in the application itself (to design a custom chip for it), and profiled it. We were amazed to find a section of code that had no effect, but used a large chunk of the run time. It turned out they knew about it, but just figured it used a tiny amount of time. They had never profiled. Imagine if the equivalent had been true of an ASIC!!

I want to emphasize that this was a smart and capable design group and an important application.

Anecdotes

We recently had a similar situation, in another important application. The application was in Java, and the program contained a line like:

```
log.debug("xxx" + object)
```

Even with debugging off, this led to the evaluation of something like:

```
add_string("xxx", convert_to_string(object))
```

Adding *if (debug)* made the application 3.5x faster.

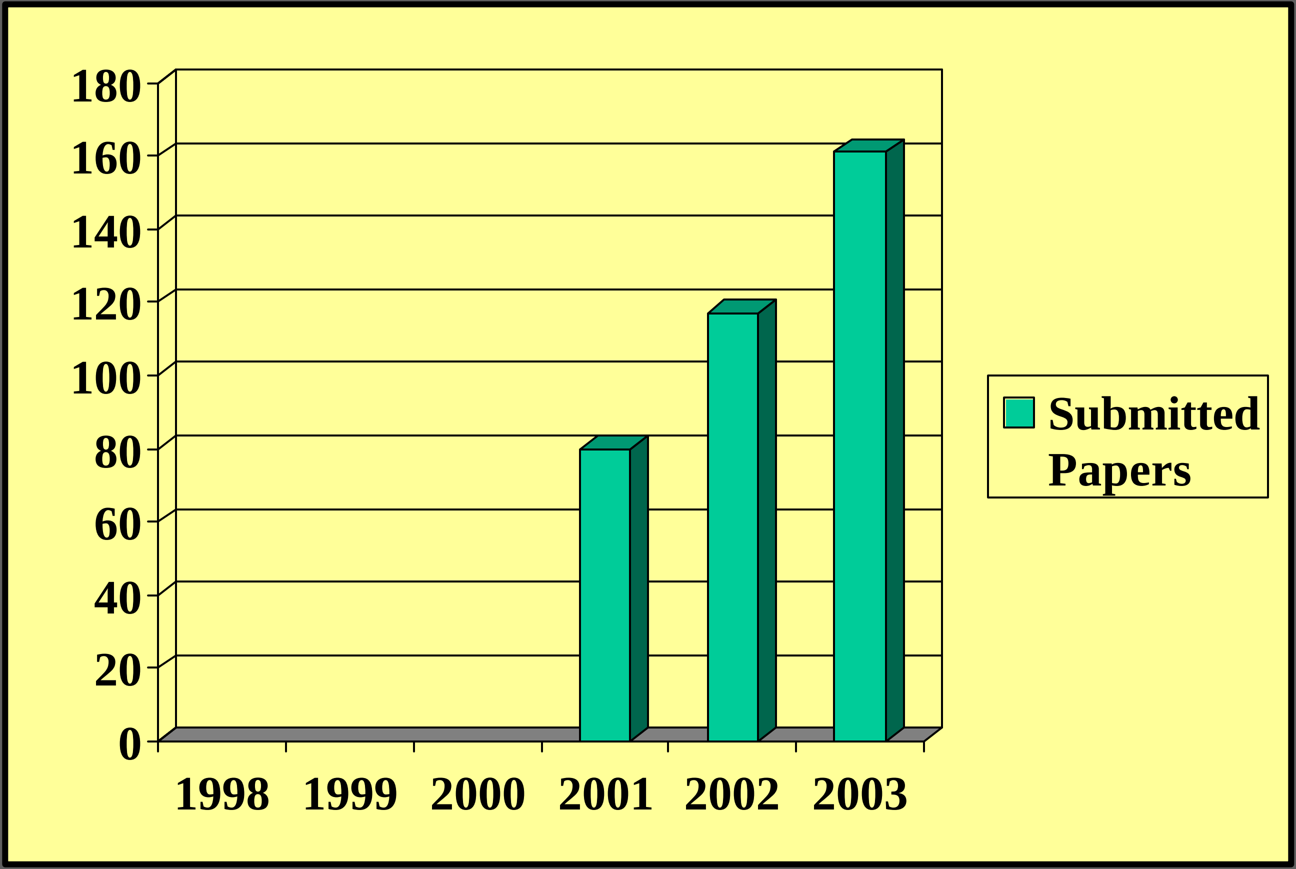
The Move to "Embedded Computing"

Because of Moore's Law, the main business of embedded is completely changing. Silicon companies find that they have to be the integrated solution provider, because with few exceptions, the customer cannot be, and they must:

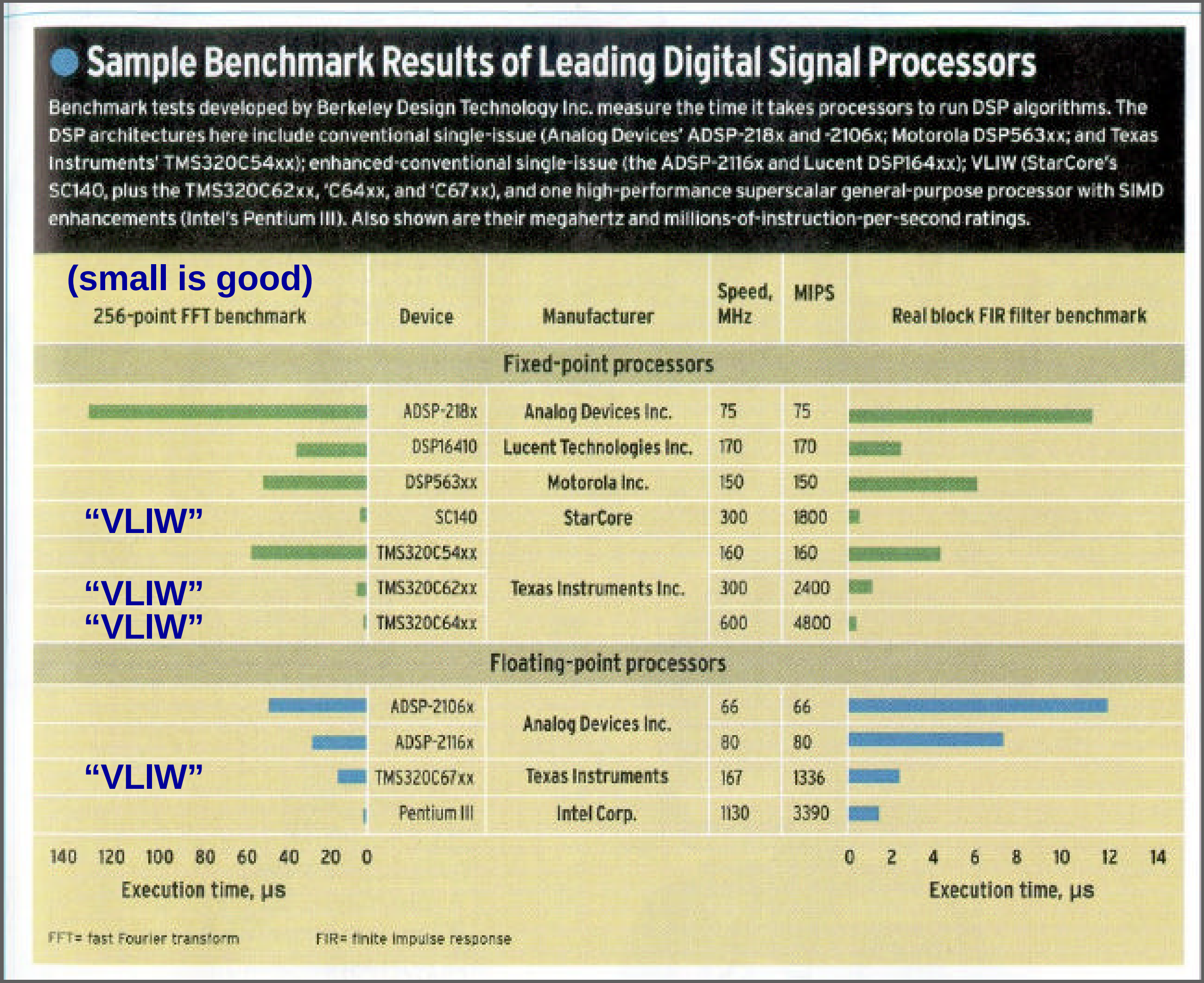
-Assemble applications themselves.
-Own the IP themselves, mostly in software, not silicon
-Therefore, become systems companies

They're trying: Major silicon companies are buying all the little compiler companies, etc. But it's very hard—the cultural and knowledge mismatches are pretty serious.

CASES Submissions are an Indicator



A Slippery Slope Towards General-Purpose



Sample Benchmark Results of Leading Digital Signal Processors

Benchmark tests developed by Berkeley Design Technology Inc. measure the time it takes processors to run DSP algorithms. The DSP architectures here include conventional single-issue (Analog Devices' ADSP-218x and -2106x; Motorola DSP563xx; and Texas Instruments' TMS320C54xx); enhanced-conventional single-issue (the ADSP-2116x and Lucent DSP164xx); VLIW (StarCore's SC140, plus the TMS320C62xx, 'C64xx, and 'C67xx), and one high-performance superscalar general-purpose processor with SIMD enhancements (Intel's Pentium III). Also shown are their megahertz and millions-of-instruction-per-second ratings.

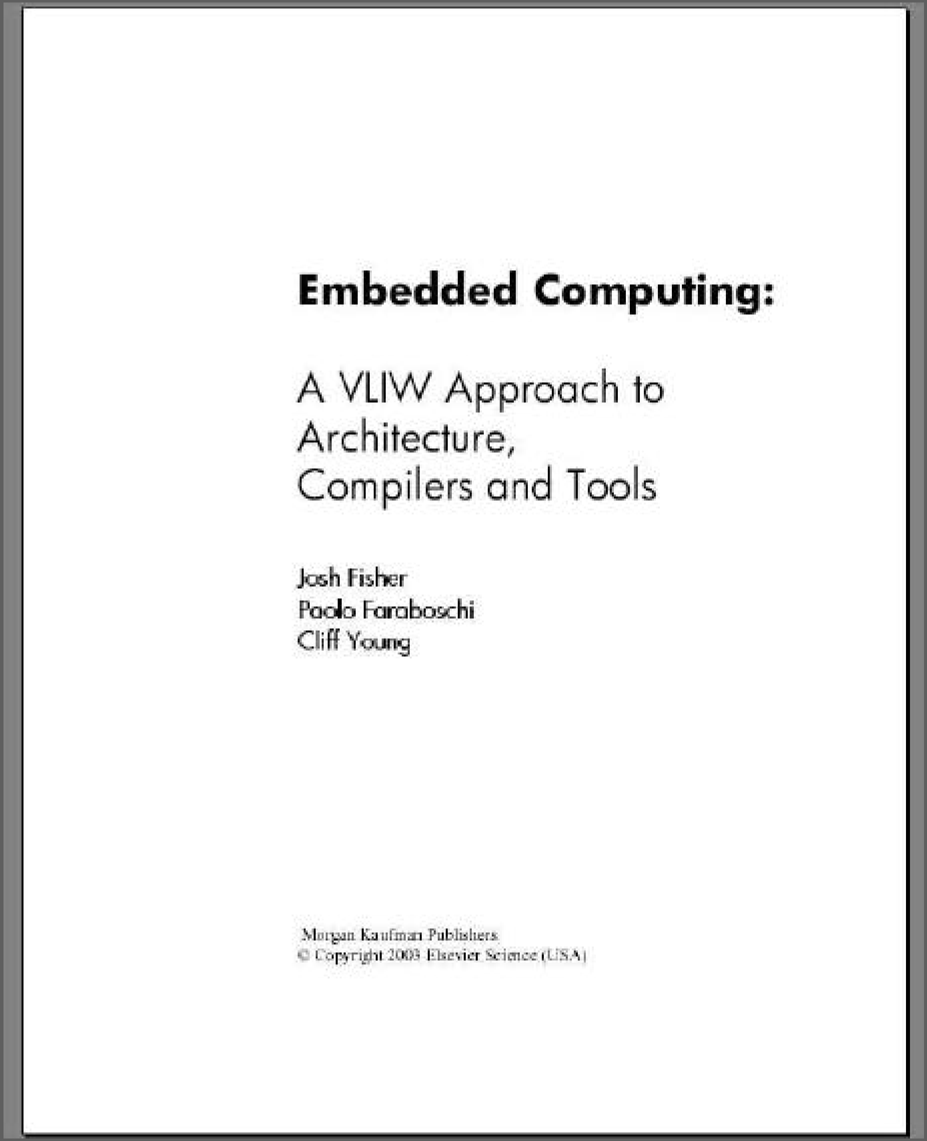
(small is good)

256-point FFT benchmark	Device	Manufacturer	Speed, MHz	MIPS	Real block FIR filter benchmark
		Fixed-point processors			
	ADSP-218x	Analog Devices Inc.	75	75	
	DSP16410	Lucent Technologies Inc.	170	170	
	DSP563xx	Motorola Inc.	150	150	
"VLIW"	SC140	StarCore	300	1800	
	TMS320C54xx	Texas Instruments Inc.	160	160	
"VLIW"	TMS320C62xx	Texas Instruments Inc.	300	2400	
"VLIW"	TMS320C64xx	Texas Instruments Inc.	600	4800	
		Floating-point processors			
	ADSP-2106x	Analog Devices Inc.	66	66	
	ADSP-2116x	Analog Devices Inc.	80	80	
"VLIW"	TMS320C67xx	Texas Instruments	167	1336	
	Pentium III	Intel Corp.	1130	3390	

FFT= fast Fourier transform FIR= finite impulse response

June, 2001. IEEE Spectrum. "Digital Signal Processors".

We Hope This Captures the Needed Skills



Embedded Computing:

A VLIW Approach to
Architecture,
Compilers and Tools

Josh Fisher
Paolo Faraboschi
Cliff Young

Morgan Kaufman Publishers
© Copyright 2003 Elsevier Science (USA)

Being beta-tested in classes now.

To appear,
Fall 2004.

Morgan Kaufmann Publishers
Distinguished professional and educational publications in computer science, engineering, and information technology.

The High-Performance Center of the Application

Lesson #2: concerns what goes into software, running on the core. This is related to the 1st lesson.

ASICs always get there first. You get more performance out of fewer transistors by far.

These work when nothing else will. And they're invariably the fastest solution when practical (no 5-10x emulation cost).

The High-Performance Center of the Application

But eventually ASICs become less attractive:

- Incremental engineering is expensive, slow and filled with delays

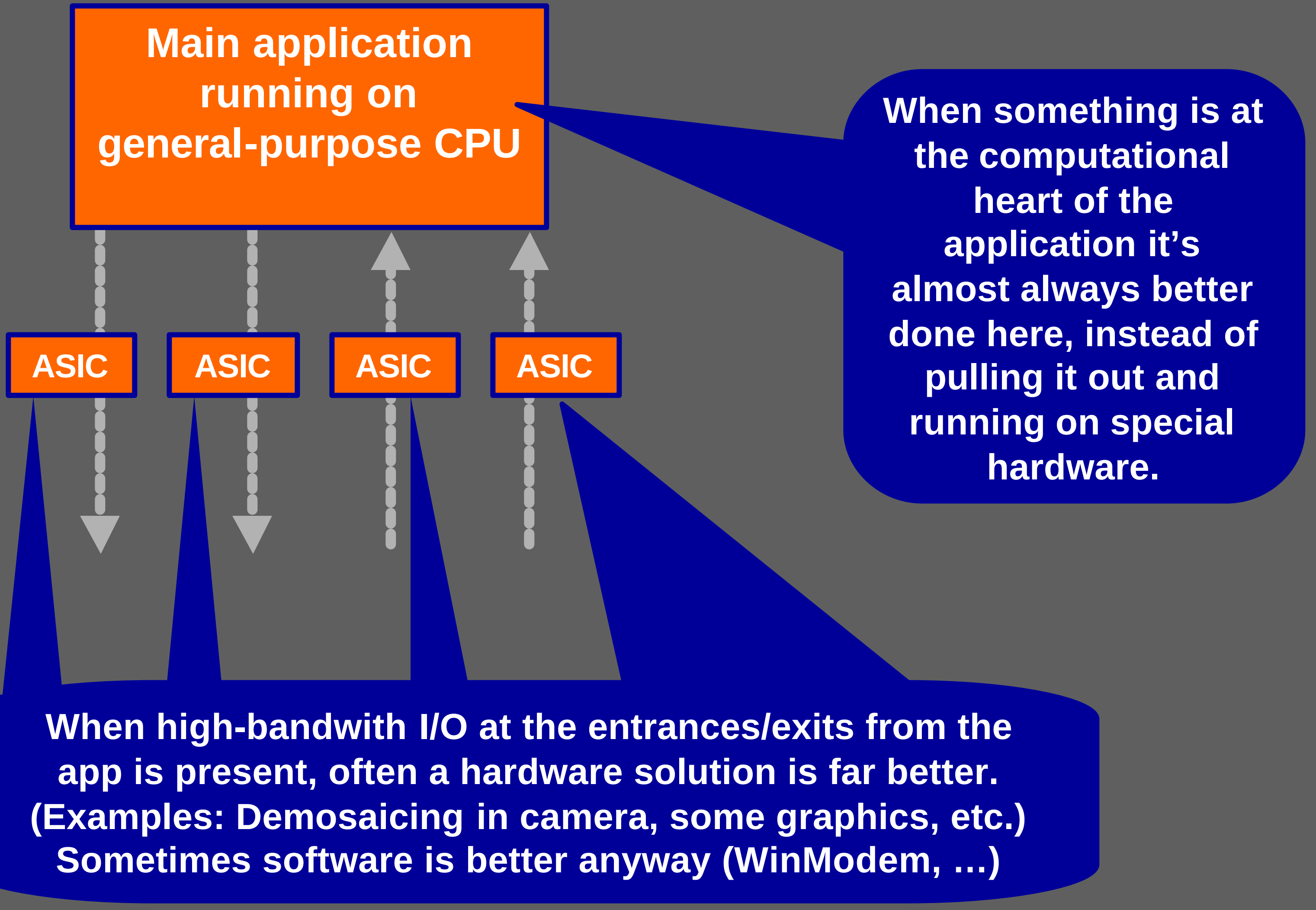
- It can’t keep up with changing market conditions

- It can’t adjust to new standards

- Just as Moore’s law makes the software solution better, it makes this solution worse! (sort of…)

In time, there’s enough silicon for software solution.

Migrating Compute-Intensive Apps to Software, A Constant Industry Trend



When something is at the computational heart of the application it's almost always better done here, instead of pulling it out and running on special hardware.

When high-bandwith I/O at the entrances/exits from the app is present, often a hardware solution is far better. (Examples: Demosaicing in camera, some graphics, etc.) Sometimes software is better anyway (WinModem, …)

Before Stacker and built-in compression

DATRAN
FOR IBM PC, XT, AT AND COMPATIBLES
diskdoubler dd2000
Datran diskdoubler short card
doubles the storage capacity of any hard disk

A board to speed up Photoshop

From: Seybold Seminars, Spring 93. About *Adaptive Solutions' PowerShop* board.

Last year Adaptive Solutions announced PowerShop, an image-processing acceleration board featuring its new DSP chip technology.

This year, it demonstrated two released versions, one for the Nubus and another for the PCI bus. PowerShop lists for $2,000 and consists of a board with four 16-bit processor chips and 4 MB of memory bundled with a copy of ScanPrepPro software.

(From ebay June 7, 2001: "Very rare to still find these cards available.")

But It’s a Pretty Good Beachhead!

Sadly, the process of moving the computational heart out of special circuits into software suffers from the “Innovator’s Dilemma” very badly.

ASICs start out entrenched! Going from Generation (N) to Generation (N+1) of any product, the best thing to do is redo the ASIC (or whatever).

Making the jump to the superior software solution requires investment, loss, use of best engineers, building a less performant product than you could have built, and at higher cost.

Trading Speed for Flexibility

speed

technology	performance/ cost	time to market	time to change functionality
CPU + ASIC	very high	very long	impossible: redesign
CPU + DSP	high	long	long
High-ILP CPU	high	short	short
Sequential RISC	low-medium	very short	very short

flexibility

Adding Customization?

speed

technology	performance/ cost	time to market	time to change functionality
CPU + ASIC	very high	very long	impossible: redesign
CPU + DSP	high	long	long
Custom VLIW	even higher than DSP	Still pretty short	short
Sequential RISC	low-medium	very short	very short

flexibility

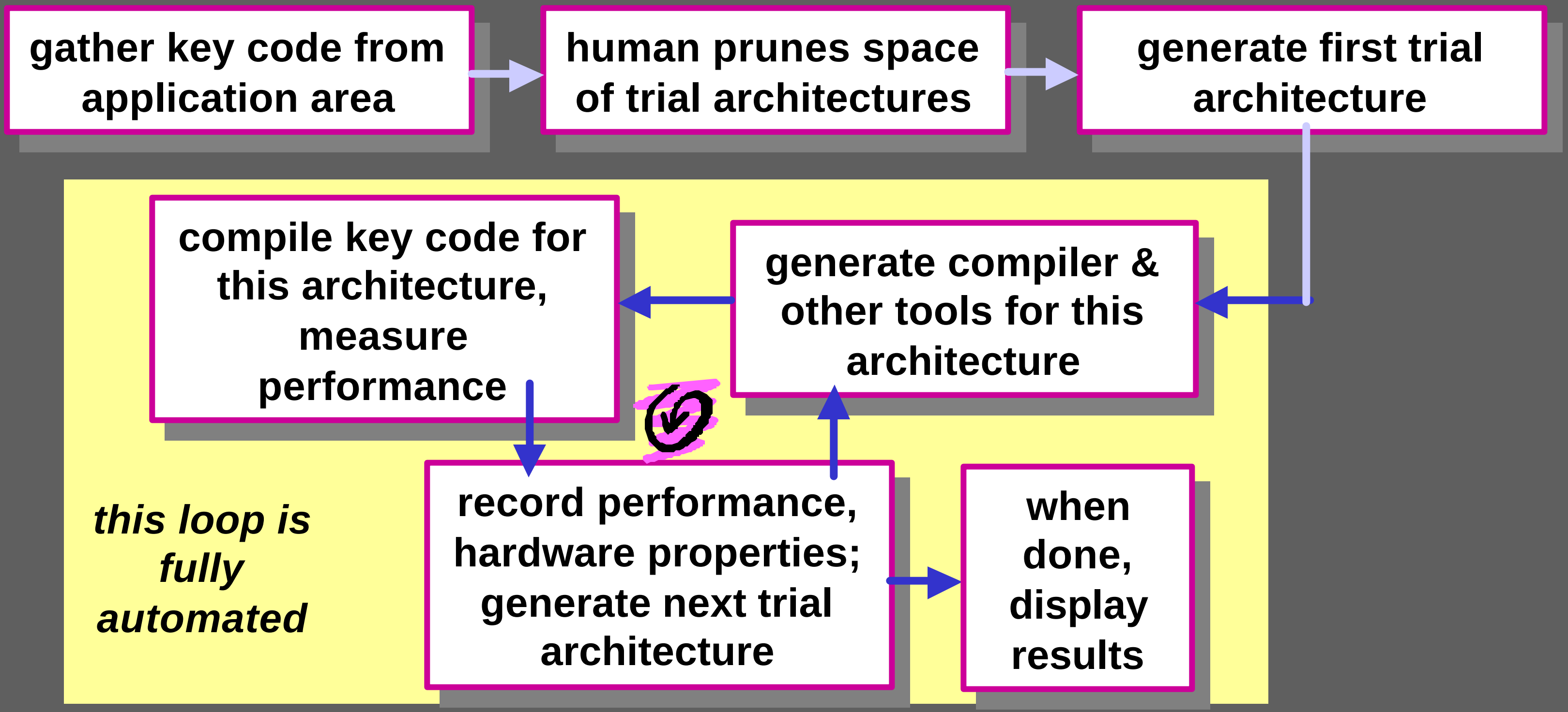
Custom Embedded CPUs

Since embedded CPUs suffer far less from object-code incompatibility, and run one application repeatedly, there is a tremendous temptation to design and build customized CPUs.

We set out to build a framework to do this in 1994.

My own expectation was that I'd be told that performance wasn't really important (I'd heard this all my professional life). But I heard the opposite.

A Framework for Automatic Processor Generation



gather key code from application area
human prunes space of trial architectures
generate first trial architecture
compile key code for this architecture, measure performance
generate compiler & other tools for this architecture
this loop is fully automated
record performance, hardware properties; generate next trial architecture
when done, display results

- ***requirements:*** software tools driven by architecture models (compiler, toolchain, libraries, simulators, verification, ...)
- ***challenge:*** a truly retargetable compiler that can find large amounts of ILP in the presence of customization

Customize to a Single Application?

Lesson #3: It’s just not practical to optimize to a single application.

1. Having in hand, well in advance, the application that will run on the device is only a dream.

   For too many reasons to enumerate, the code will change until the product ships, and long after.

2. The whole point of a software solution is that you want to be able to change it. Design too narrowly, and you’ll lose.

Fortunately, one can still customize to a “domain”, and find enough commonality to make it well worthwhile.

Do Fully-Automatic Customization?

Lesson #4: The picture of automatic customization is very appealing. But keep the application programming expert around anyway for a while. Or maybe forever.

“What?! You’re willing to give me *TWO* integer multipliers?? If I’d known that, I would have inverted those two loops and rewritten the second computation. By the way, if I do that, 16-bit multiplies will do it just as fast, I won’t need 32-bit units.”

This is hard enough to do when the architecture is fixed, to do it as part of an exploration process is way beyond us.

No automatic process will capture this process anytime soon.

Lessons in Embedded Systems

Some things we learned:

1. Circuit density will move embedded systems as we know it today to something that resembles high performance computing, but competence and cultural issues make this a battle.

2. There is a constant movement of the computational heart of applications into the main processor, but there are serious “innovator’s dilemma” problems in doing this.

Lessons in Embedded Systems

Some things we learned:

3. Customization is an advantage available in embedded, but it’s hard to customize to an application—customize to a domain instead.

4. However automated customization gets, it will be a long time before the application-expert programmer ever gets out of the loop, if ever.

Moving From Embedded Systems to Embedded Computing (Some Lessons in Embedded Processing)

Josh Fisher
Senior Fellow
Hewlett-Packard
josh.fisher@hp.com