



HPCSW Presentation
15 minutes on
Programming Heterogeneous Architectures

Ontologies and Realities

Douglas O'Flaherty

Programming Models for Heterogeneous Architectures

What are we talking about?

There is no Wikipedia entry for *Heterogeneous Architectures*

- *Flynn's Taxonomy isn't any help here...*

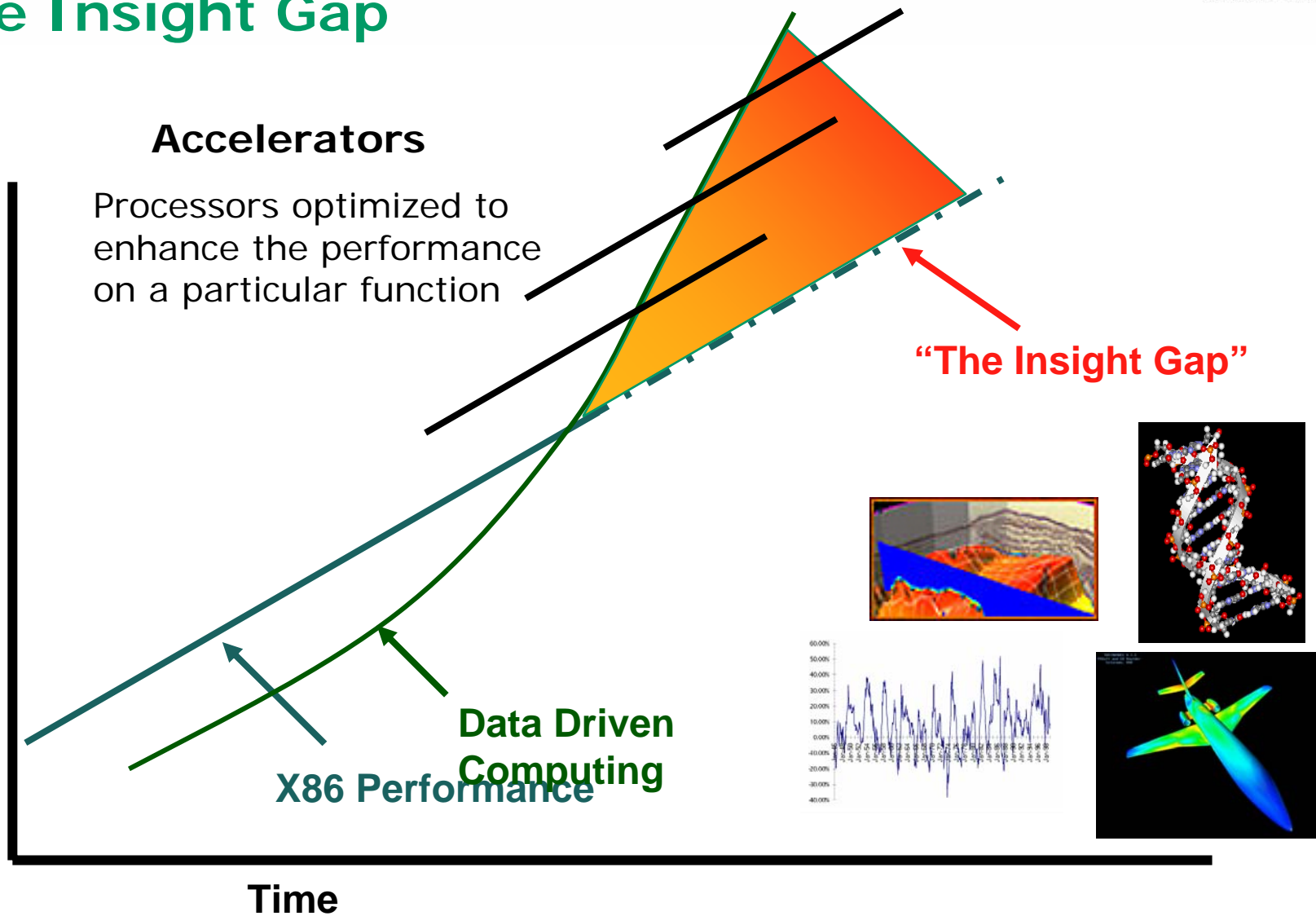
The Top Google Search Entry is

- *Optimizing Heterogeneous Architectures* in edn 3/2/2006
 - *A systems approach to multicore-DSP architectures extracts the high performance that today's applications require.*
- Not an HPC Application

A Working Definition:

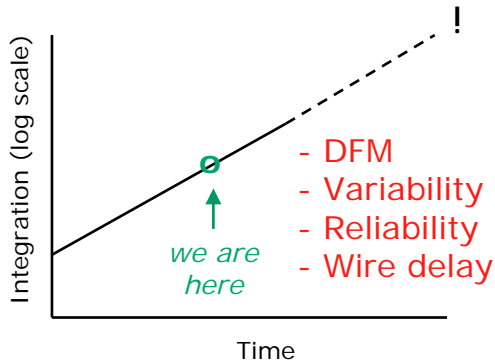
Programming a Heterogeneous Architecture is distributed computing across processors of different capabilities

Industry Landscape: The Insight Gap

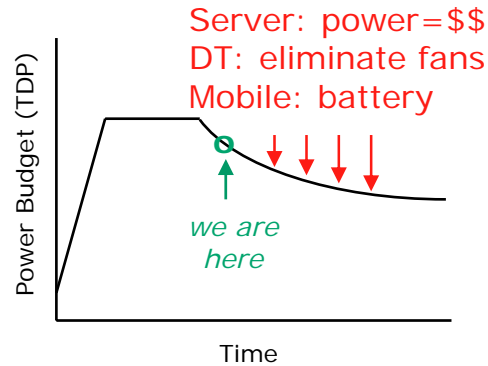


High-level Industry Trends

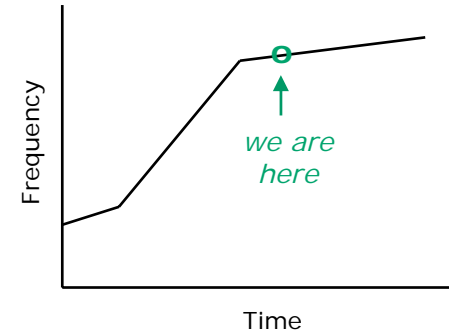
Moore's Law ☺



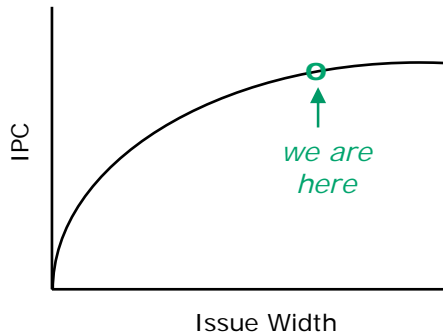
The Power Wall ☹



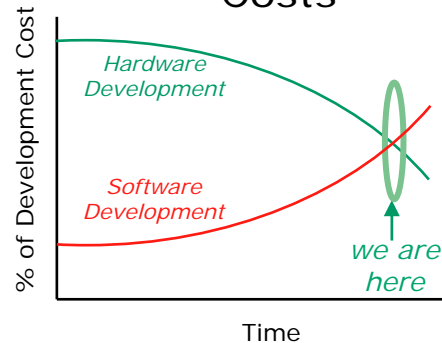
The Frequency Wall ☹



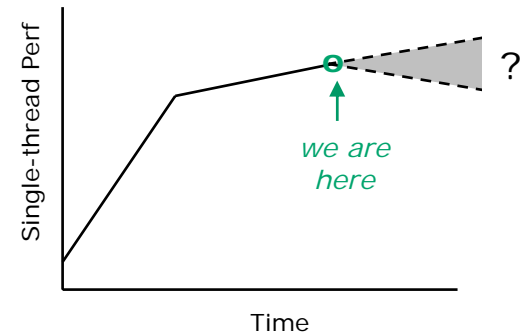
The Complexity Wall ☹



HW vs SW Costs



Single thread Perf (!)



Single Processor Scaling Alone Provides Limited Incremental Value!

Three Generations of x86 Processing

	<i>Frequency</i>	<i>Cores</i>	<i>Accelerators</i>
Emphasis:			
Characteristics:	<ul style="list-style-type: none"> • Single Core • Increased frequency • Single-threaded performance • Single-threaded applications 	<ul style="list-style-type: none"> • Homogeneous multi-core • Course-grain parallelism • Multi-tasked and limited multi-threaded applications 	<ul style="list-style-type: none"> • Heterogeneous multi-core • Fine-grain parallelism • Parallel applications • "The right cores for the problem"
Environment:	<ul style="list-style-type: none"> • Smaller geometries • Exponential growth in design size • Limited design impact on frequency 	<ul style="list-style-type: none"> • Threaded programming hard • Increasing role of media-rich apps 	<ul style="list-style-type: none"> • Performance AND energy efficiency AND cost • Programmer productivity a primary goal
Limitations:	<ul style="list-style-type: none"> • Frequency plateau • Power increase • Complexity • Single-threaded performance 	<ul style="list-style-type: none"> • Balanced MP scaling • Total performance • Power, cost limits • "The Looming Crisis" 	



Time

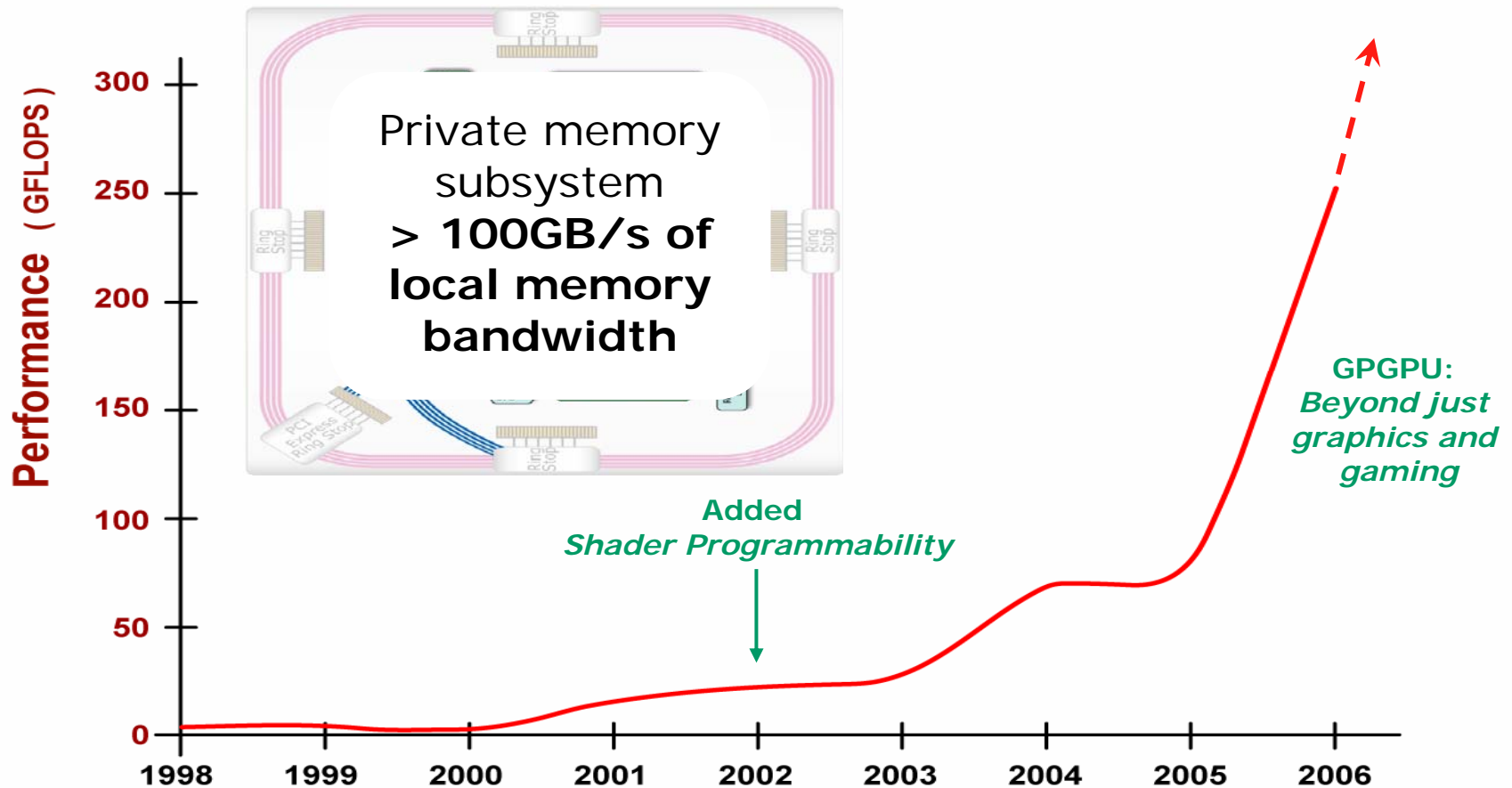
Definitions

- Processors
 - We all think we know what this is.
 - Common ISA, programmable, debug
 - System view of memory
 - OS scheduled
- Co-processors
 - Shared instructions from main thread
 - Substantively similar to processors
 - Programmable, interruptible
 - Same view of memory as the Processor
 - Sequential codelet, Run-to-completion?
- Devices
 - Instructions are dispatched and not under OS control
 - Could be the same or different ISA;
 - Private memory spaces, though it may have a shared memory section with processor
 - Maintains its own state.
 - OS treats it like run-to-completion or run-to-checkpoint

Distinguishing Characteristics

- Data Movement
- Synchronization
- ISAs
- Debug Domains
- Virtual v. Physical Memory

Compute Density Trend: *Graphics Processor Performance*



Scaling Inter-process Communications

Hardware maintained coherency *was* minimal relative to everything else and have no impact on performance

- This is less true as we get to lots o' cores
- No one thought data movement was insignificant in a 32 node SMP system
- Why do you think it remains insignificant in a 32 core 1, 2 or 4 socket system?
- Current parallel programming for multi-core is uses explicit communications
- Drivers in commercial workloads also align with the need for communications as a 'first class citizen'

Producer Consumer Latency

Do we understand the underlying protocols?

- hand-off of a value from a "Producer" (P) to a "Consumer" (C) in a weakly consistent shared-memory system, with the transaction involving separate cache lines for the data (D) and flag (F). This simple case of a single P and a single C does not require locks.

<u>Producer</u>	<u>Consumer</u>
Store Data (the real stuff)	until (Flag) (spin on Flag)
SFENCE (force stores to become globally visible in program order)	LFENCE (prevent speculative load of Data)
Store Flag (tell the consumer that the Data is ready)	Load Data (Finally!)

Code Sequence is more expensive than might be assumed...

How many transactions does it take...

Producer Operation	Producer Cache	Coherence transaction	Consumer Cache	Consumer Operation
Store D				
	D hit "O" (shared)			
		invalidate D		
			D hit "S"	
				invalidate D
			D goes "I"	
		invalidate D ack		
	D upgrade to "M"			
(store D completes)				
SFENCE				
Store F				
	F hits "O"			
		invalidate F		
			F hit "S"	
			F goes "I"	
		invalidate F ack		
	F upgrade to "M"			
(store F completes)				
				Load F
			miss (F is "I")	
		read request F		
	F hits "M"			
intervene F				
	F hits "M"			

			F hit "S"	
			F goes "I"	
		invalidate F ack		
	F upgrade to "M"			
(store F completes)				
				Load F
			miss (F is "I")	
		read request F		
	F hits "M"			
intervene F				
	F downgrade to "O"			
		intervention response F		
			F goes "S"	
				(load F completes)
				Compare F
				Branch out of spinloop
				LFENCE
				Load D
			miss (D is "I")	
		read request D		
	D hits "M"			
intervene D				
	D downgrade to "O"			
		intervention response D		
			D goes "S"	
				(load D completes)

There are 8 coherence transactions

The Metric that Matters

Moving Data Around

The LogP model describes the parameters of communications based upon the passing of messages between nodes:

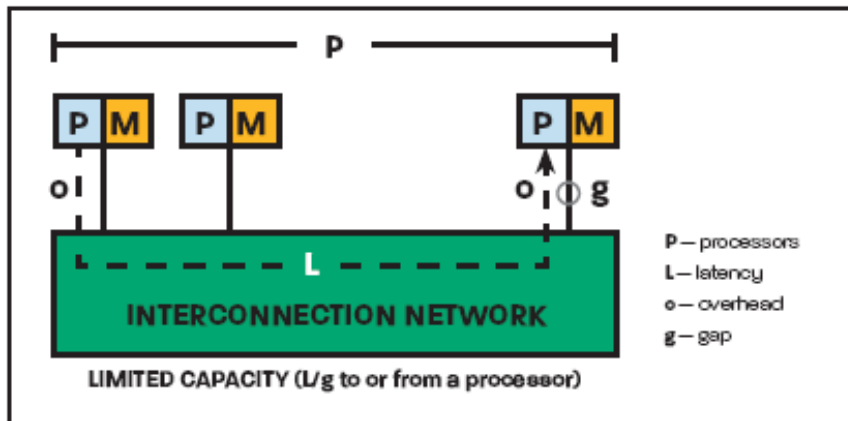


Figure 2: Illustration of LogP Model

Latency – the delay in the network for a single message.

Overhead – the amount of time a processor is engaged in the act of sending or receiving a message.

Gap – The interval between messages. It is also the reciprocal of the network bandwidth.

Parallelism – the concurrency of messages on the network, typically measured in nodes

http://www.hp.com/techservers/hpcn/hpccollaboration/ADCatalyst/downloads/AMD_HPTCAWP.pdf

Things that Matter

The Simple List of Requirements for Programming Heterogeneous Environment

Mark the code to be executed *over there*

- Synchronous *or* Asynchronous

A compiler, or Run Time, that knows how that code executes on the *other processor*

A method to pass instructions to the *other processor*

Reliable access to the data by the *other processor*

Low-overhead messaging for start, end, input, output and error

The Evolving Mainstream

Virtualization

- Consolidated IT Infrastructure
 - Improves utilization
 - Promises new levels of manageability

10Gb (and beyond) Ethernet

- Converge multiple networks to a single network platform
 - Storage on the wire is required for virtualization
 - Data growth is driving enterprises and others to centralized stores

Programming Multi-core

- Threads are getting really cheap, but...
- IO and memory consumption scale with cores

NUMA Architectures

- Performance is dependent upon locality

Embedded (and quasi-embedded) Markets

- Growth in consumer, home and appliances based upon standard platforms
- History of Heterogeneous Architectures

Power Consumption

- Green is equal to money
- Many segments care more about idle power, rather than peak

Security

- Extending protection models

How could this relate to Heterogeneous Architectures (p1)

Topology recognition for Run Time support

- An emerging Processor Abstraction Layer
- Run time support for functionality, allocation & affinity

The need for IO as a 'first class citizen'

- Minimize memory copies
- Moving data efficiently between virtual guests on the same system
- Devices working in virtual, not physical memory
- Packet steering within the CPU complex
- RDMA and exploration of direct cache injection

Non-sequential programming models

- Asynchronous User Level Threading
 - Future<T>
- Low overhead synchronization and asynchronous primitives

How could this relate to Heterogeneous Architectures (p2)

Increased dependency on communication “within” the CPU

- Structures for message passing or queues
- Transactional Memory

Deterministic low-latency execution

- RTOS in enterprise applications
- Allowing the user to manage what was previously only the kernel

Mutability of state and memory

- Tiered virtual memory hierarchies
- Coherency domains (why am I asking core 0 if he has data)
- Standards to manage state and context in devices

The Simple List of Requirements for Programming Heterogeneous Environment

Mark the code to be executed *over there*

- Synchronous *or* Asynchronous

A compiler, or Run Time, that knows how that code executes on the *other processor*

A method to pass instructions to the *other processor*

Reliable access to the data by the *other processor*

Low-overhead messaging for start, end, input, output and error

Never Before have so many smart people sought other smart people to solve their problems

- Emergence of compelling proprietary solutions (back to the future?)
 - Cray, *Thinking Machines*, KSR
 - A change in the economic model for HPC?
- The Benevolent Dictator
 - DirectX, OpenGL, MSFT, Java?
- Cooperative Competition
 - Co-optition (AMD, Intel, Nvidia, others)
- Group action
 - New programming languages
 - ‘Emerging Standards’

Thank You!

*(Yes, this opinions are mostly my own.
Any forward-looking statements are strictly
commentary on industry trends and should not
be construed as anything else.)*