ATPESC
(Argonne Training Program on Extreme-Scale Computing)

Computer Architecture and Structured Parallel Programming

James Reinders, Intel
August 4, 2014, Pheasant Run, St Charles, IL
08:45 – 10:00

Computer Architecture & Structured Parallel Programming

- review aspects of computer architecture that are critical to high performance computing
- discuss how to think about best algorithm design using structured parallel programming techniques
- task vs. data parallelism and why data parallelism is key
- introduce TBB, OpenMP*
- introduce Intel® Xeon Phi™ architecture.

HARDWARE

SOFTWARE

SOFTWARE

SOFTWARE

HARDWARE

See the Forest

See the Forest

A cliché about someone missing the "big picture" because they focus too much on details:

They "cannot see the forest for the trees."

See the Forest

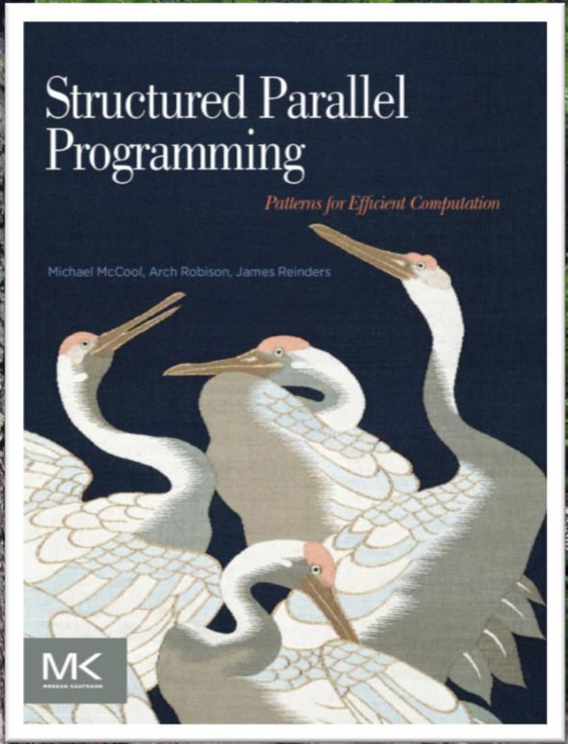
See the Forest

I ♥ architecture.

but…

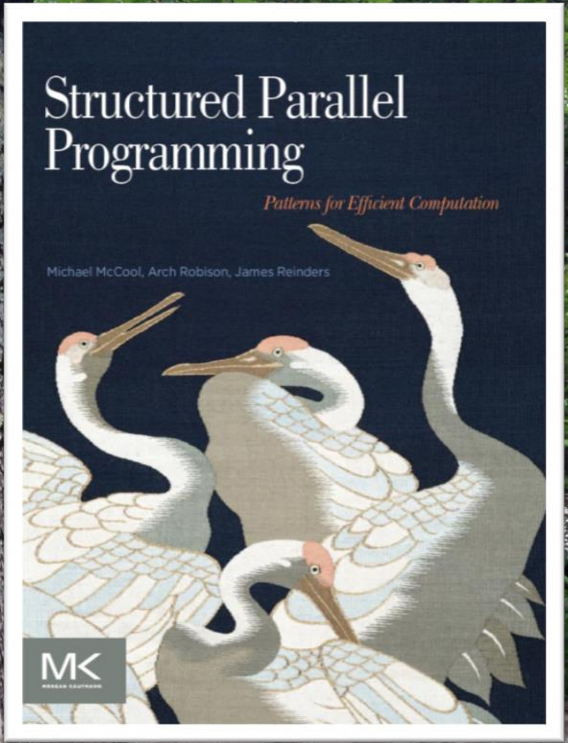
See the Forest

Can you teach parallel programming without first teaching computer architecture?



See the Forest

Can you teach parallel programming without first teaching computer architecture?
(Or without just teaching a single API?)



See the Forest

TREES
Cores
HW threads
Vectors
Offload
Heterogeneous
Cloud
Caches
NUMA

See the Forest

TREES	FOREST
Cores	Parallelism, Locality
HW threads	Parallelism, Locality
Vectors	Parallelism, Locality
Offload	Parallelism, Locality
Heterogeneous	Parallelism, Locality
Cloud	Parallelism, Locality
Caches	Parallelism, Locality
NUMA	Parallelism, Locality

See the Forest

TREES	Advice: proper abstractions
Cores	Use tasks
HW threads	Use tasks
Vectors	Use SIMD (10:30 talk)
Offload	Avoid, Use TARGET
Heterogeneous	Avoid via neo-hetero
Cloud	What's a cloud?
Caches	Use abstractions
NUMA	Use abstractions

See the Forest

TREES	FOREST
Cores	Parallelism, Locality
HW threads	Parallelism, Locality
Vectors	Parallelism, Locality
Offload	Parallelism, Locality
Heterogeneous	Parallelism, Locality
Cloud	Parallelism, Locality
Caches	Parallelism, Locality
NUMA	Parallelism, Locality

Teach the Forest

Increase exposing parallelism.

Increase locality of reference.

Teach the Forest

Increase exposing parallelism.

Increase locality of reference.

Why? Because it's programming that addresses the universal needs of computers today and in the future future.

Teach the Forest

Increase exposing parallelism.

Increase locality of reference.

THIS

IS

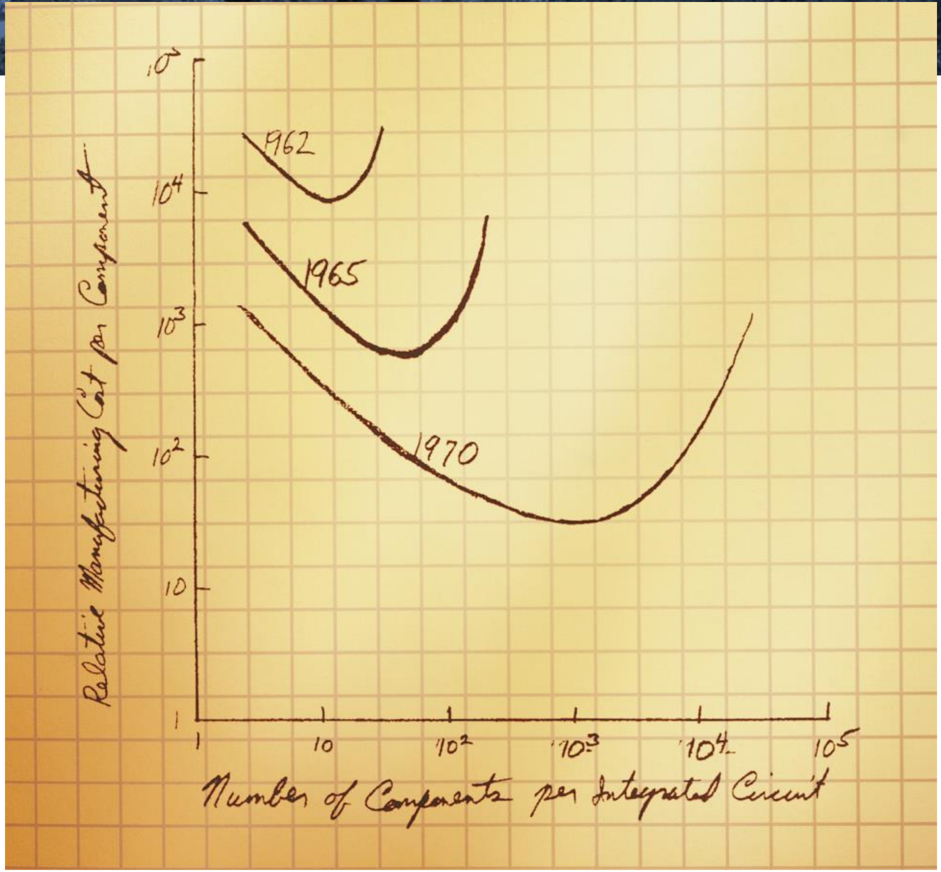
YOUR MISSION

Why so many cores?

Why Multicore?

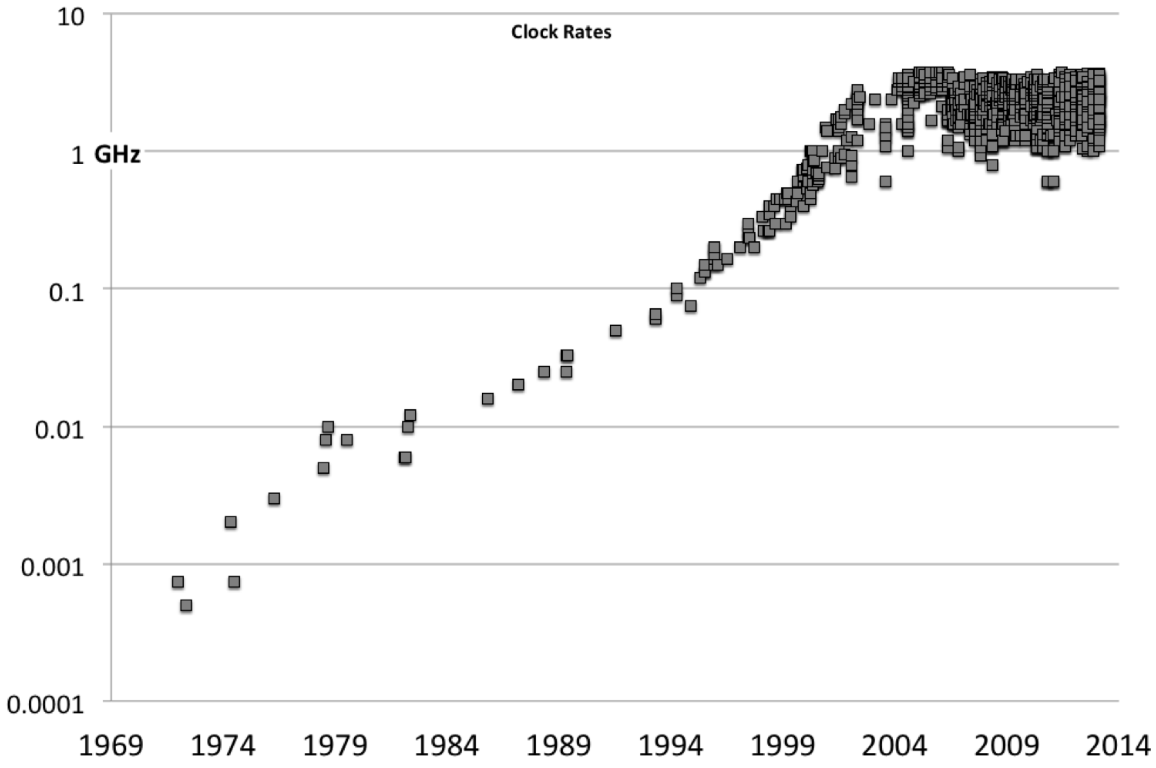
The "Free Lunch" is over, really.

But Moore's Law continues!



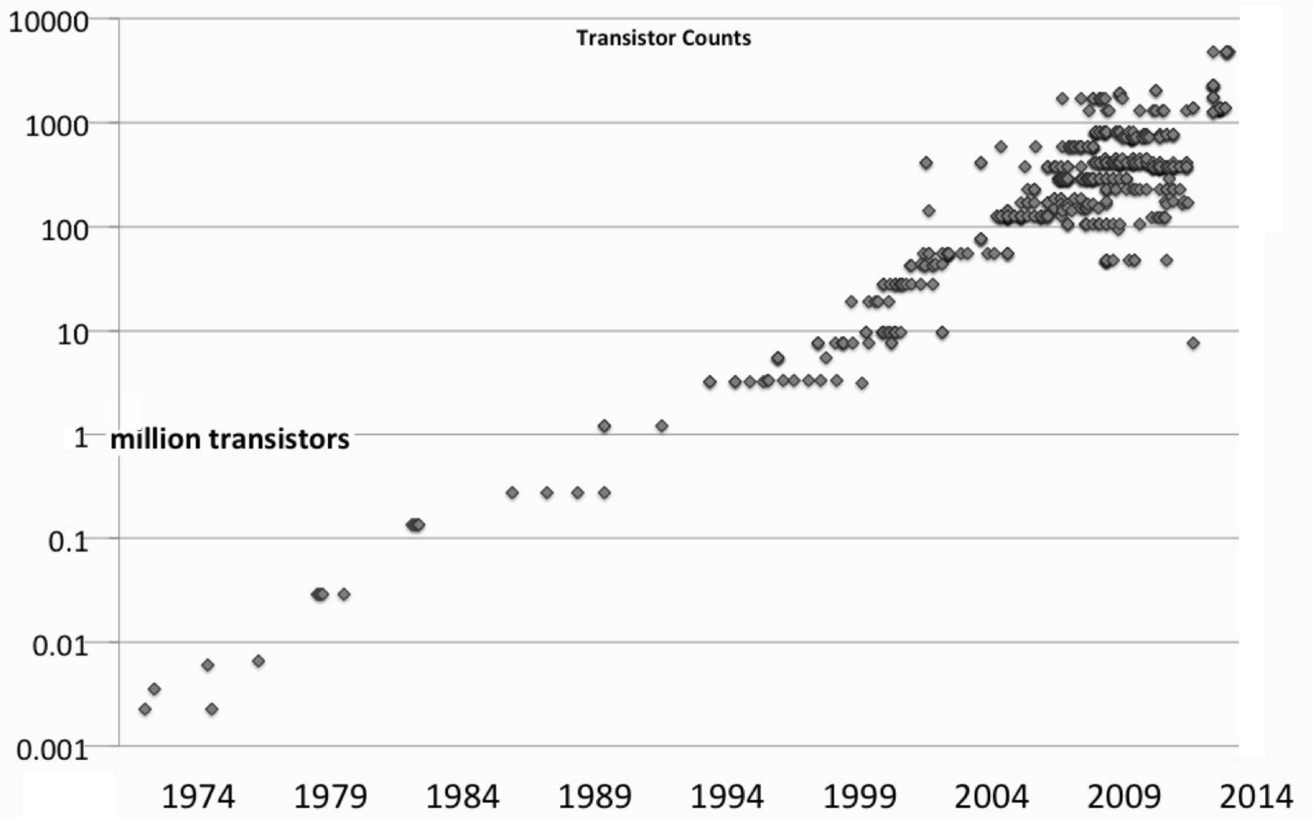
Processor Clock Rate over Time

Growth halted around 2005



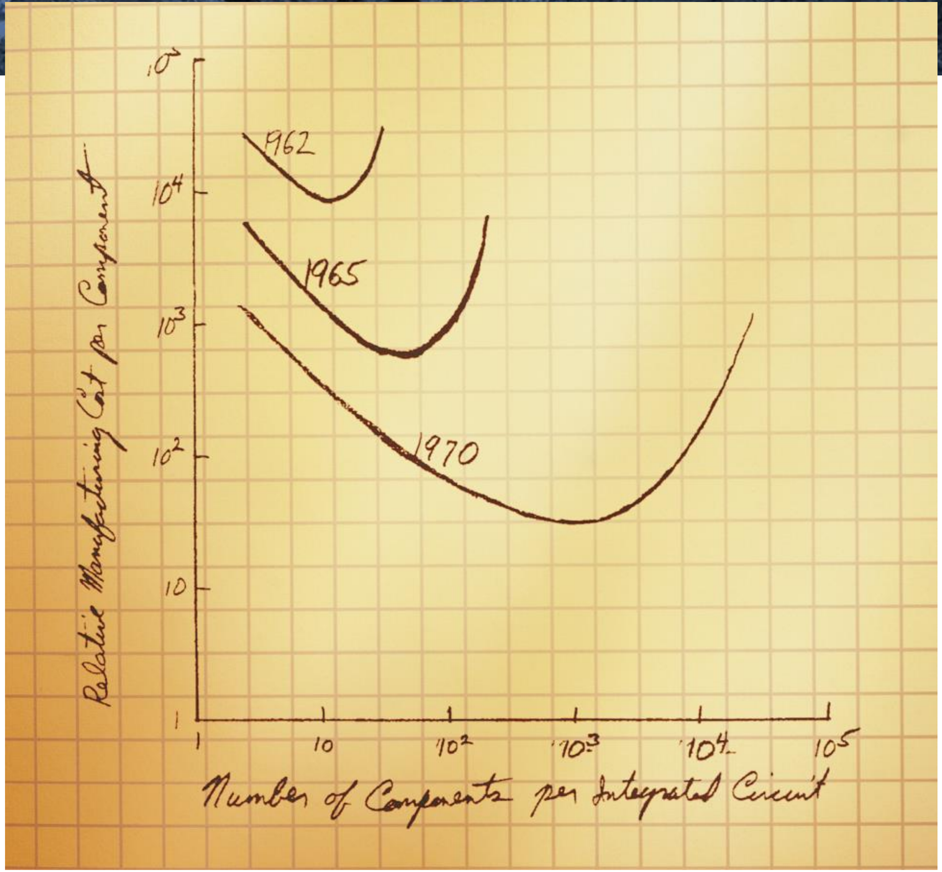
Transistors per Processor over Time

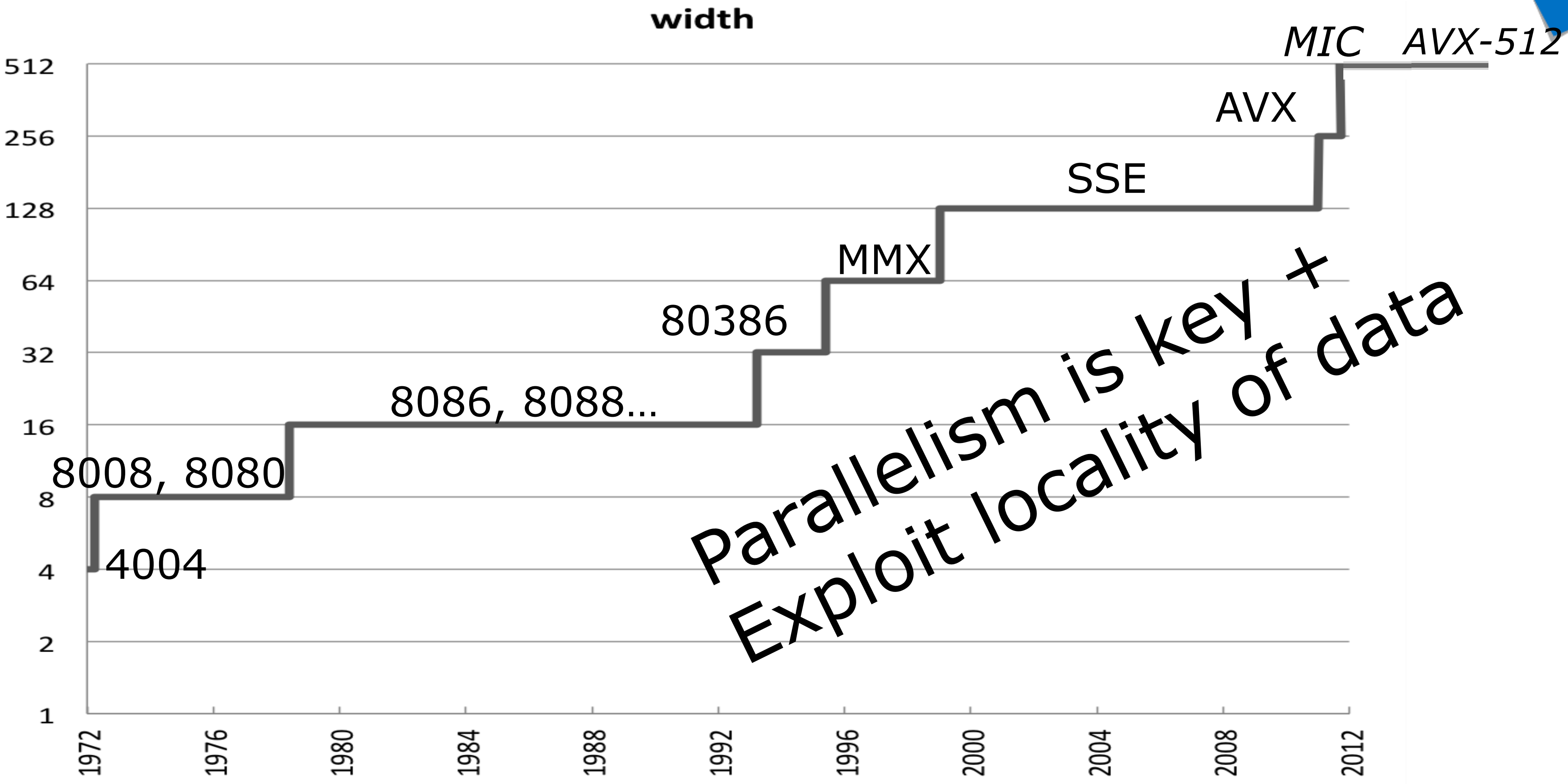
Continues to grow exponentially (Moore's Law)

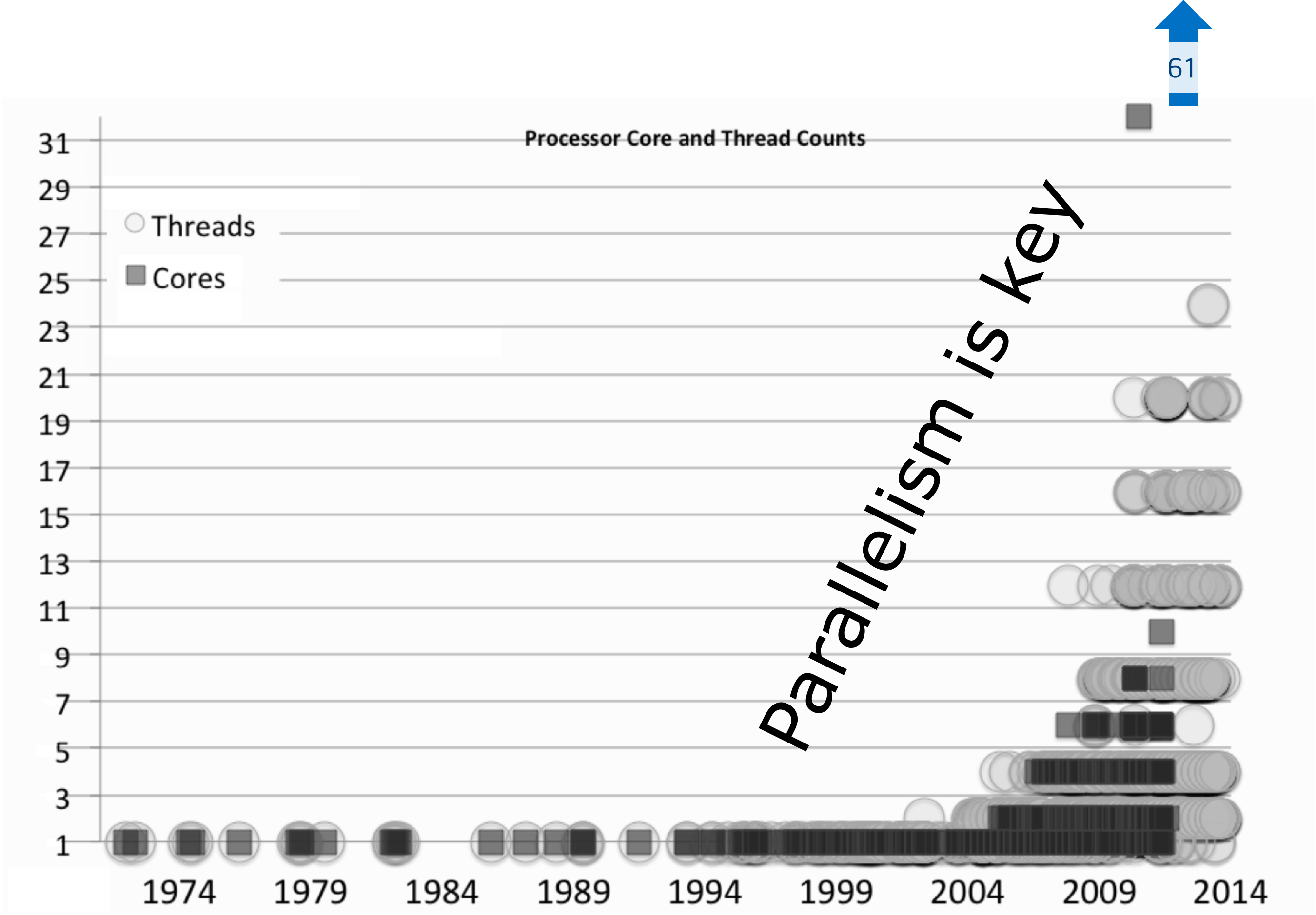


Moore’s Law

Number of components (transistors) doubles about every 18-24 months.

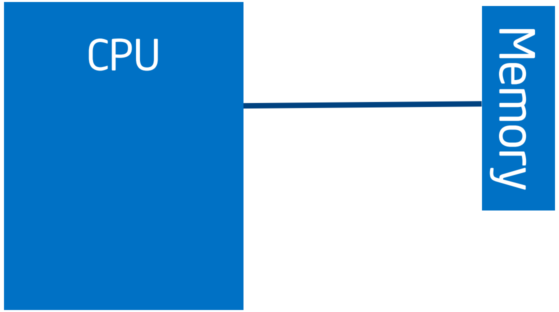



width
512
256
128
64
32
16
8
4
2
1
1972
1976
1980
1984
1988
1992
1996
2000
2004
2008
2012
4004
8008, 8080
8086, 8088…
80386
MMX
SSE
AVX
MIC
AVX-512
Parallelism is key +
Exploit locality of data


61
Processor Core and Thread Counts
Threads
Cores
Parallelism is key
31
29
27
25
23
21
19
17
15
13
11
9
7
5
3
1
1974
1979
1984
1989
1994
1999
2004
2009
2014

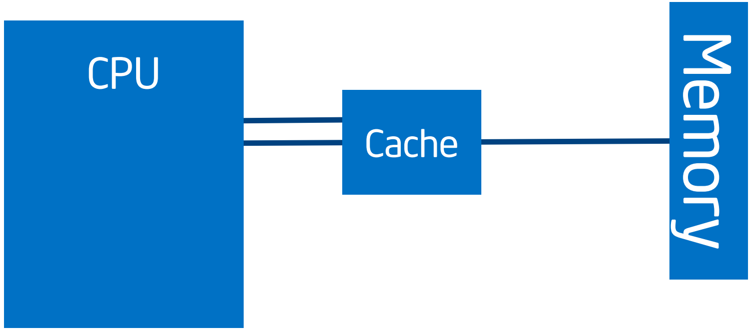
Is this the Architecture Track?

CPU



These were simpler times.

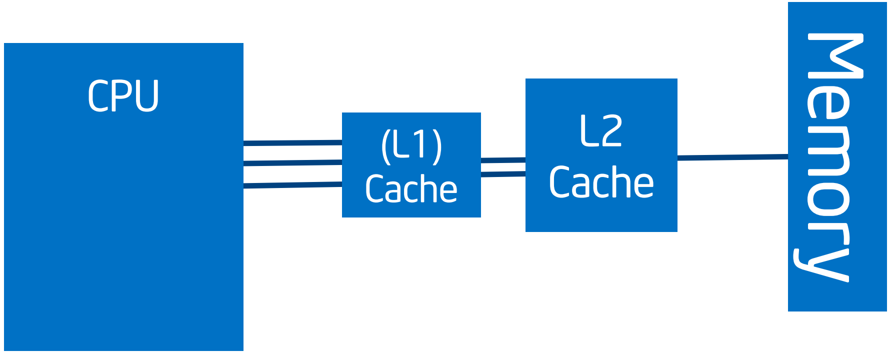
CPU + cache



Memories got "further away" (meaning: CPU speed increased faster than memory speeds)

A closer "cache" for frequently used data helps performance when memory is no longer a single clock cycle away.

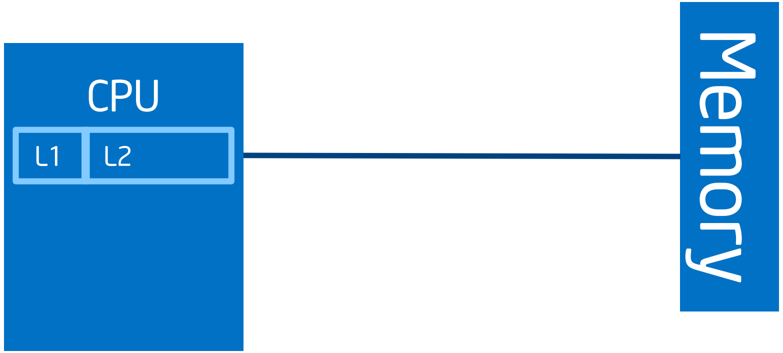
CPU + caches



Memories keep getting “further away” (this trend continues today).

More “caches” help even more (with temporal reuse of data).

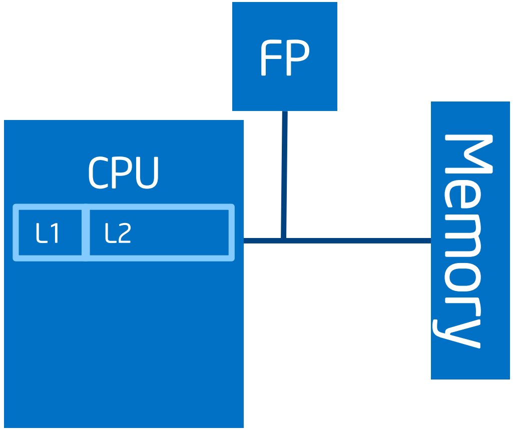
CPU with caches



As transistor density increased (Moore's Law), cache capabilities were integrated onto CPUs.

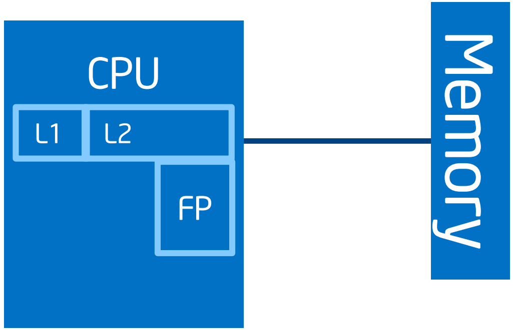
Higher performance external (discrete) caches persisted for some time while integrated cache capabilities increase.

CPU / Coprocessors



Coprocessors appearing first in 1970s were FP accelerators for CPUs without FP capabilities.

CPU / Coprocessors



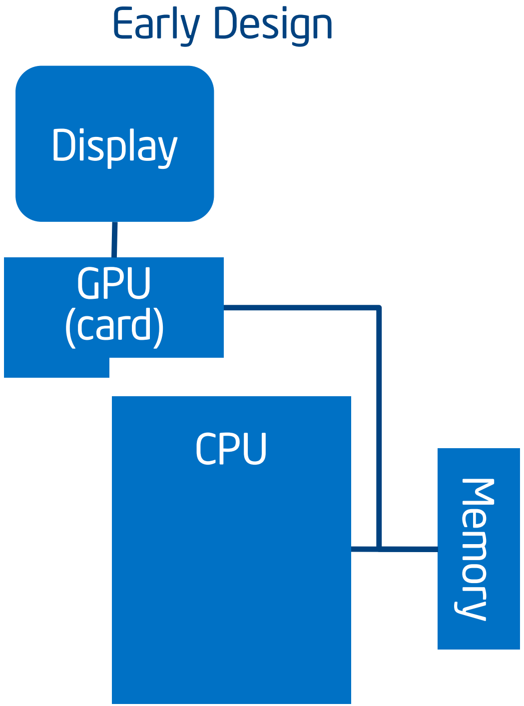
As transistor density increased (Moore's Law), FP capabilities were integrated onto CPUs.

Higher performance discrete FP "accelerators" persisted a little bit while integrated FP capabilities increase.

CPU / Coprocessors

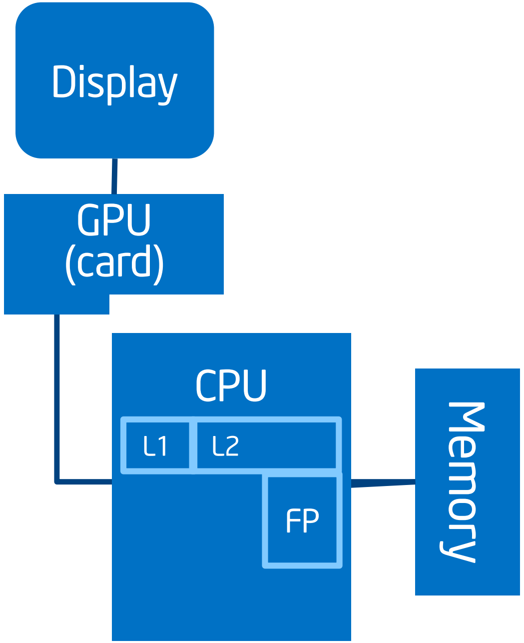
Interest to provide hardware support for displays increased as use of graphics grew (games being a key driver).

This led to graphics processing units (GPUs) attached to CPUs to create video displays.



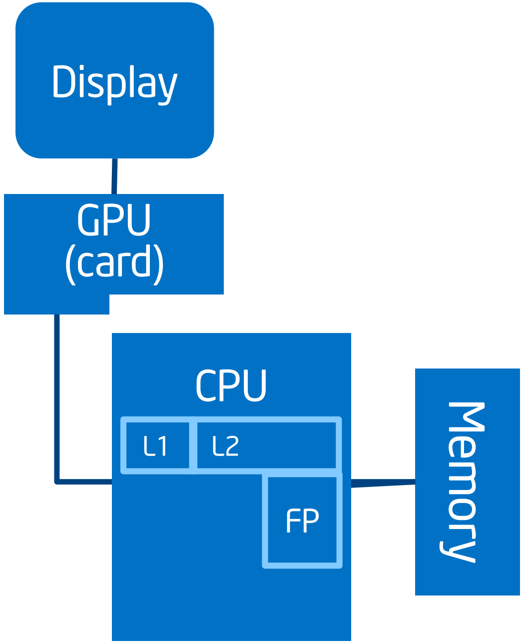
CPU / Coprocessors

GPU speeds and CPU speeds increase faster than memory speeds. Direct connection to memory best done via caches (on the CPU).



CPU / Coprocessors

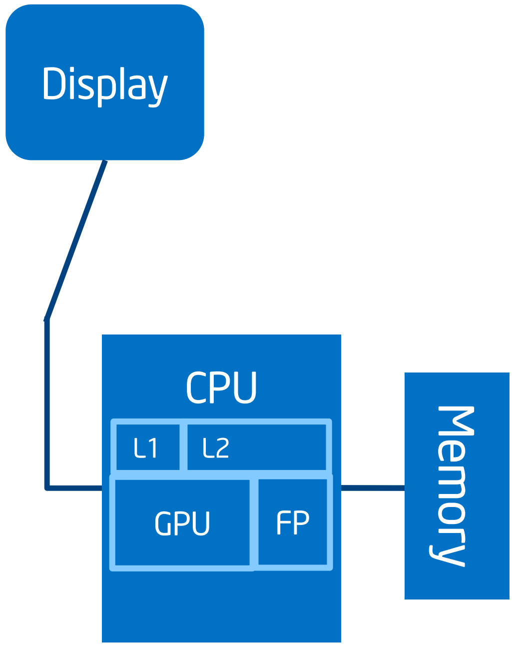
GPU speeds and CPU speeds increase faster than memory speeds. Direct connection to memory best done via caches (on the CPU).



CPU / Coprocessors

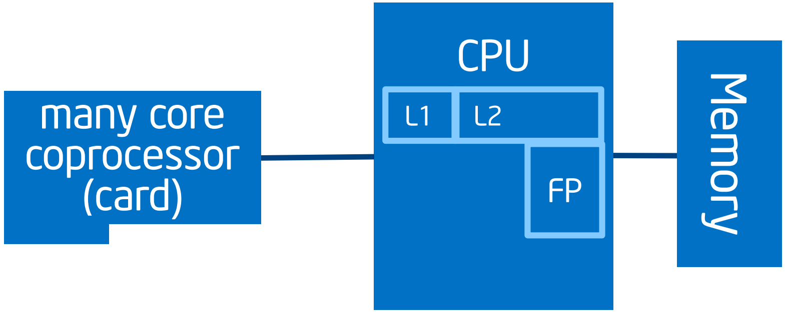
As transistor density increased (Moore’s Law), GPU capabilities were integrated onto CPUs.

Higher performance external (discrete) GPUs persist
while integrated GPU capabilities increase.



CPU / Coprocessors

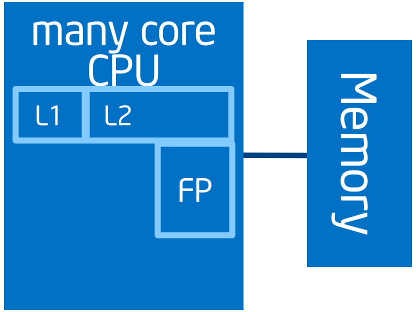
A *many core* coprocessor (Intel® Xeon Phi™) appears, purpose built for accelerating technical computing.



CPU / Coprocessors

As transistor density increased (Moore's Law), many core capabilities will be integrated to create a many core CPU.

("Knights Landing")

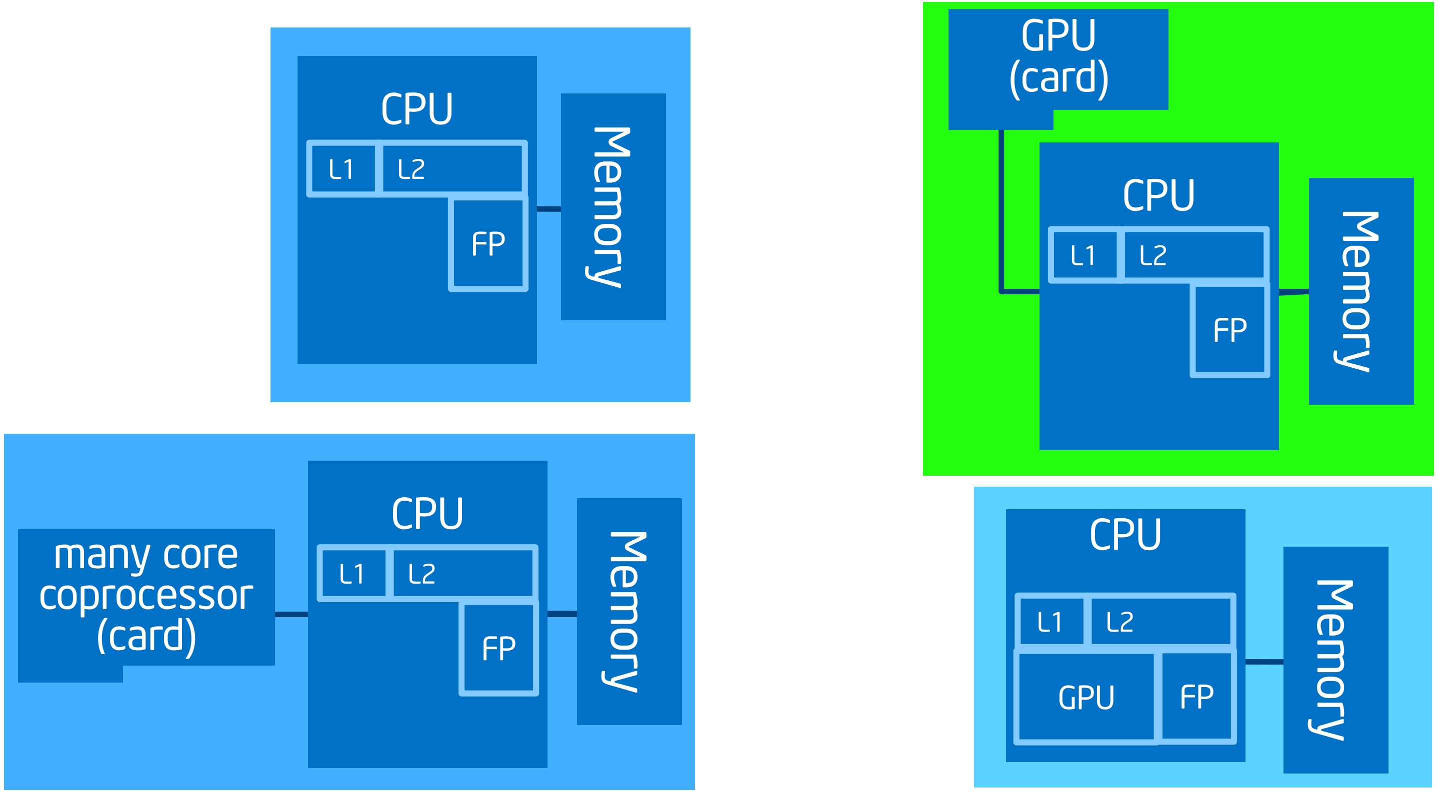


Nodes

"Nodes" are building blocks for clusters.

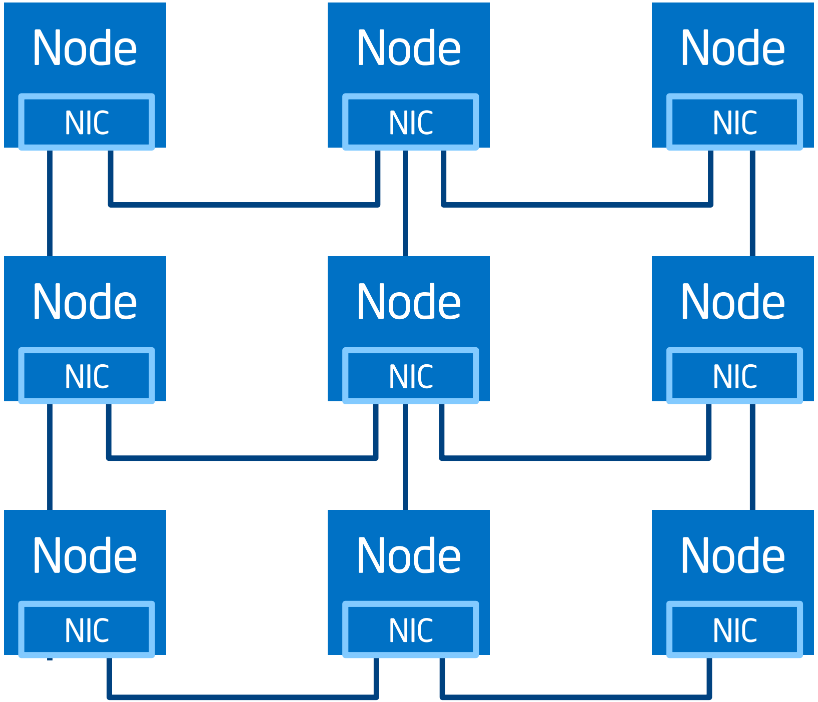
With or without GPUs.

Displays not needed.



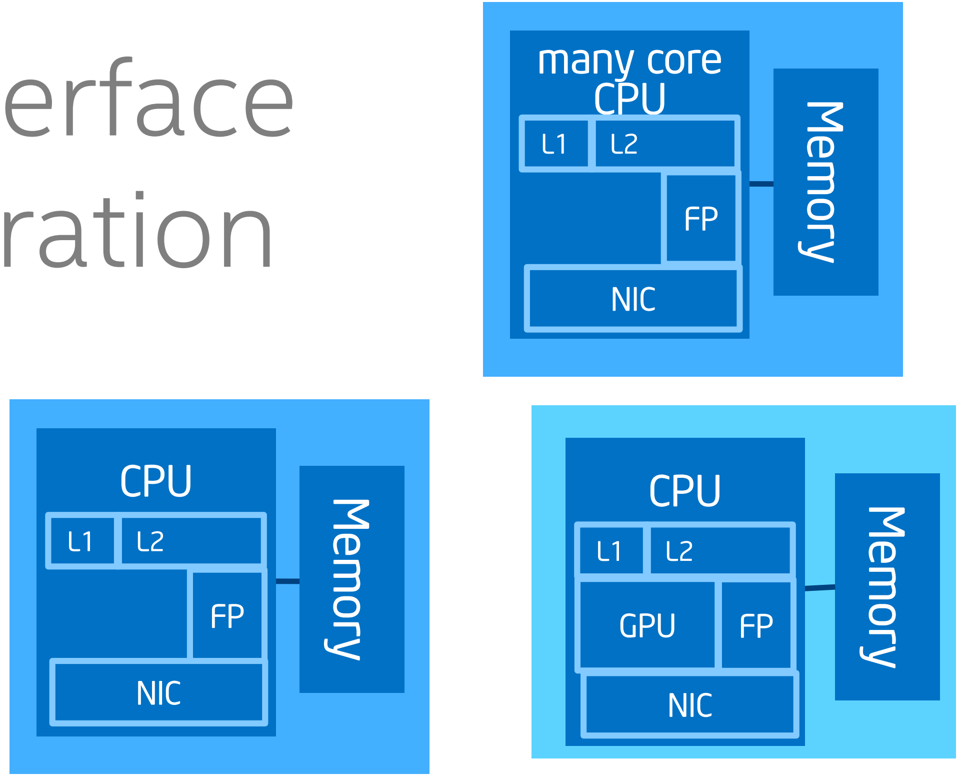
Clusters

Clusters are made by connecting nodes - regardless of "Nodes" type.



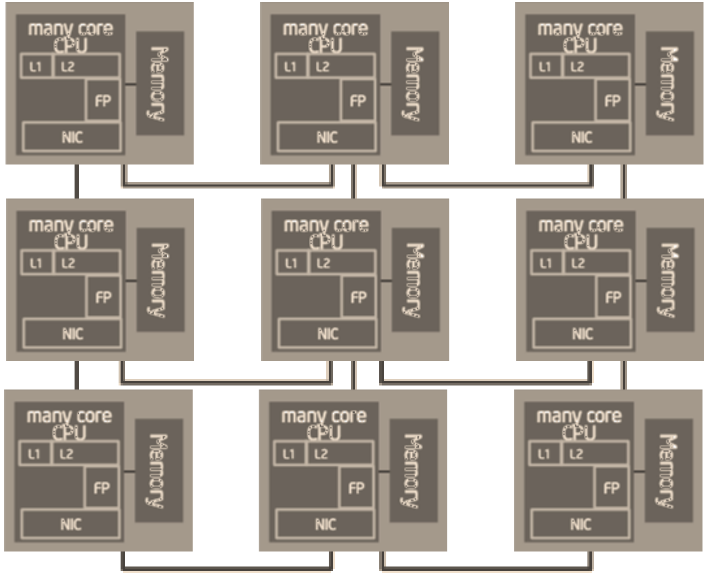
NIC (Network Interface Controller) integration

As transistor density increased (Moore's Law), NIC capabilities will be integrated onto CPUs.



What matters when programming?

- Parallelism
- Locality



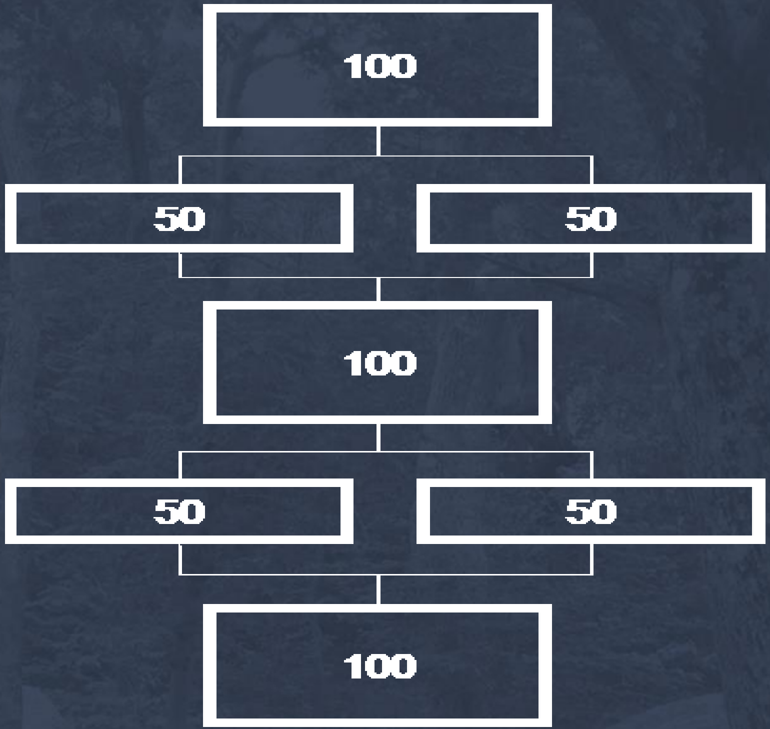
Amdahl who?

How much parallelism is there?

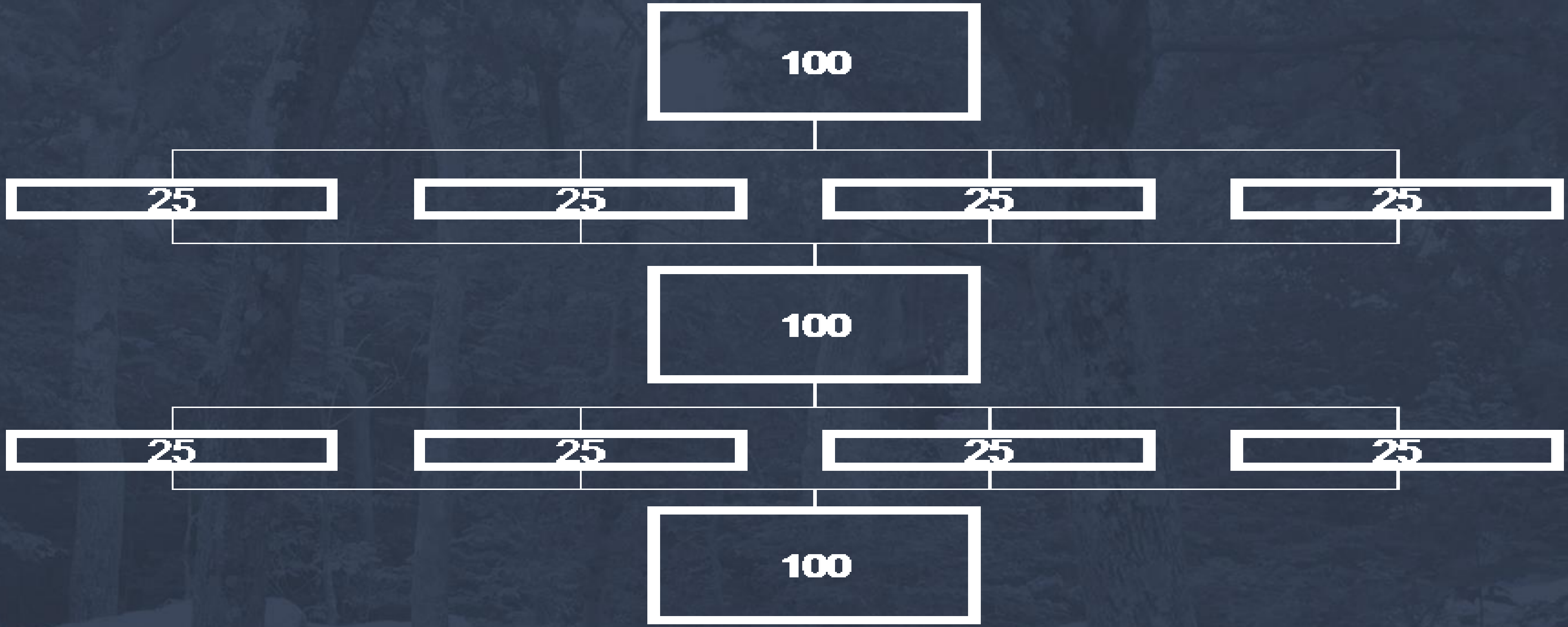
Amdahl's Law

Gustafson's observations on Amdahl's Law

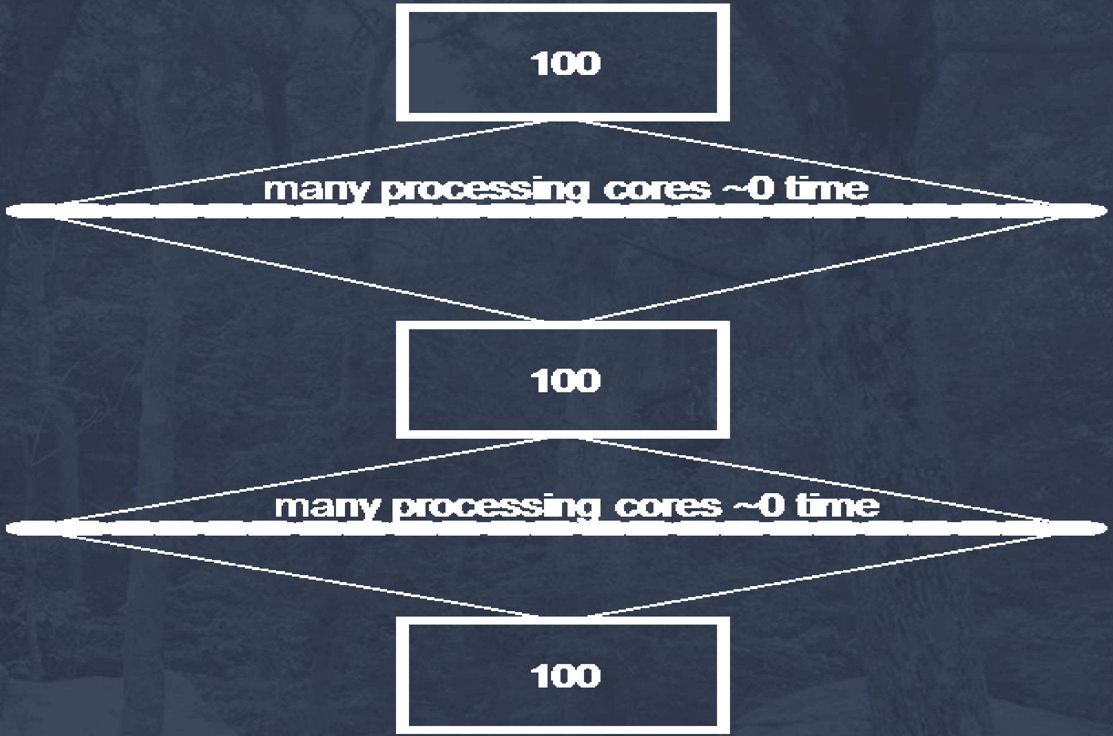
Work 500 Time 500
Speedup 1X



Work 500 Time 400
Speedup 1.25X



Work 500 Time 350
Speedup 1.4X



Work 500 Time 300

Speedup 1.7X

Amdahl's law

"…the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude."

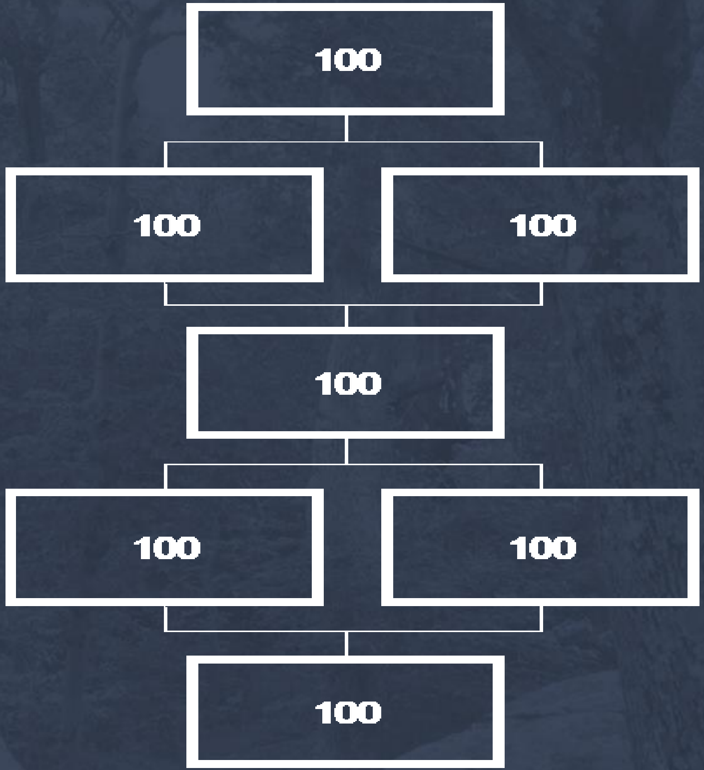
– Amdahl, 1967

Amdahl's law – an observation

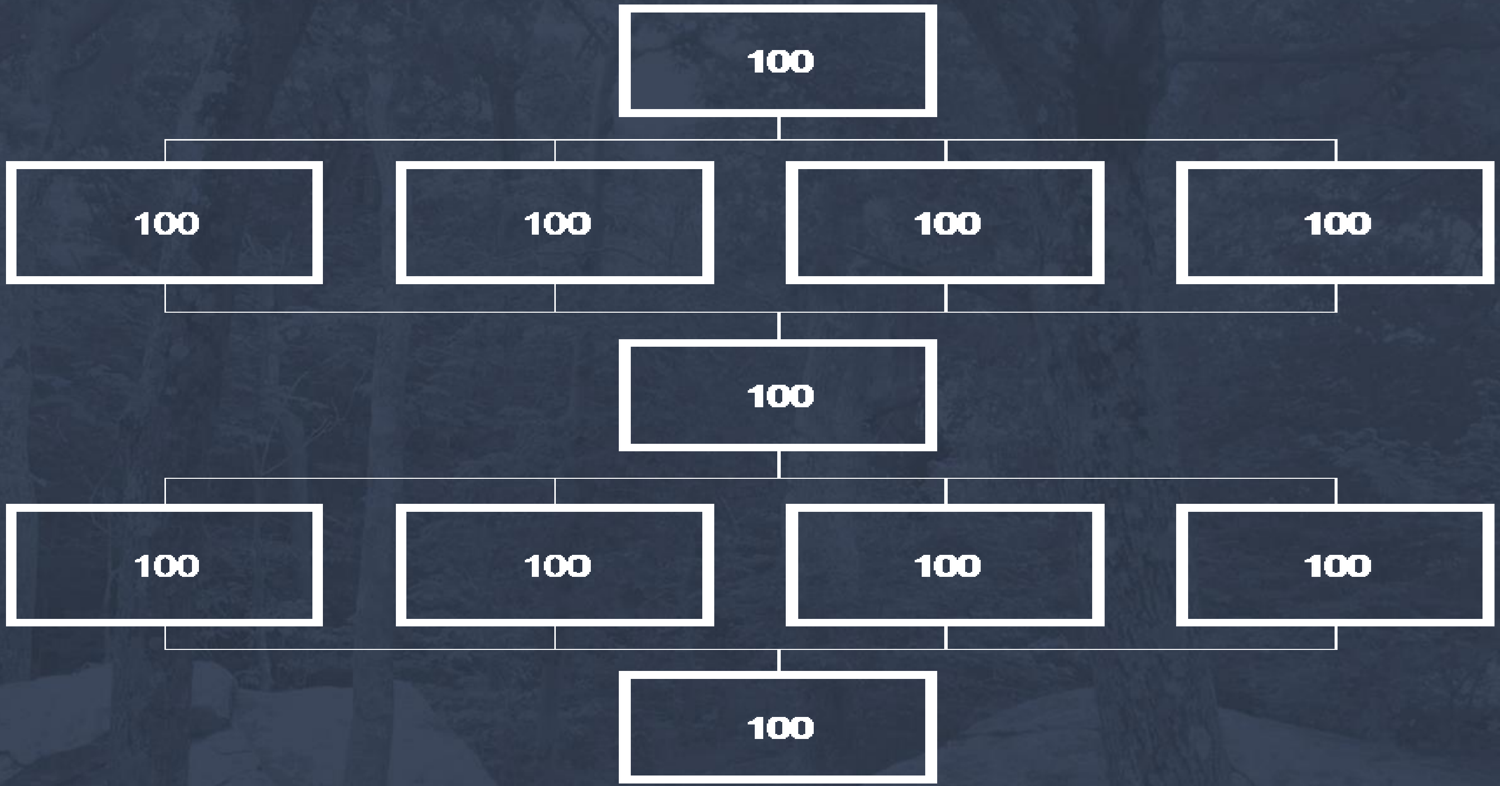
"…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size."

– Gustafson, 1988

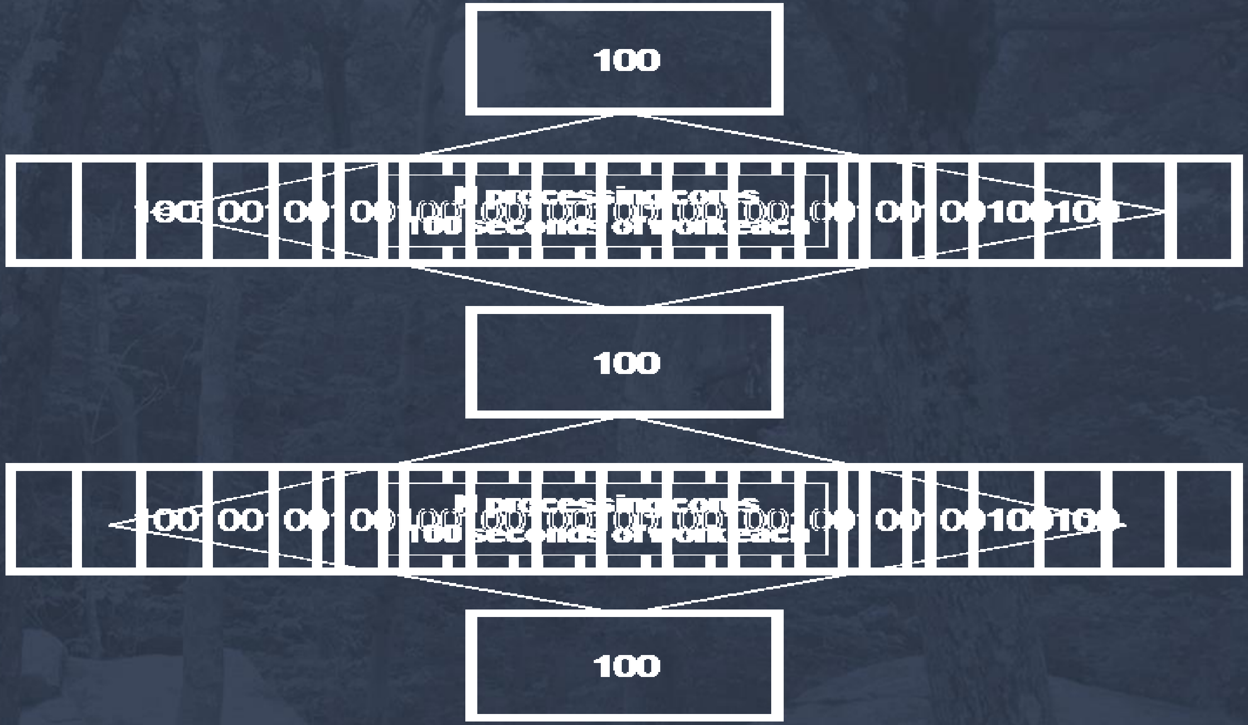
Work 500 Time 500
Speedup 1X



Work 700 Time 500
Speedup 1.4X



Work 1100 Time 500
Speedup 2.2X



Work 2*N*100+300 Time 500

Speedup O(N)

How much parallelism is there?

Amdahl’s Law

Gustafson’s observations on Amdahl’s Law

Plenty –

but the workloads need to continue to grow !

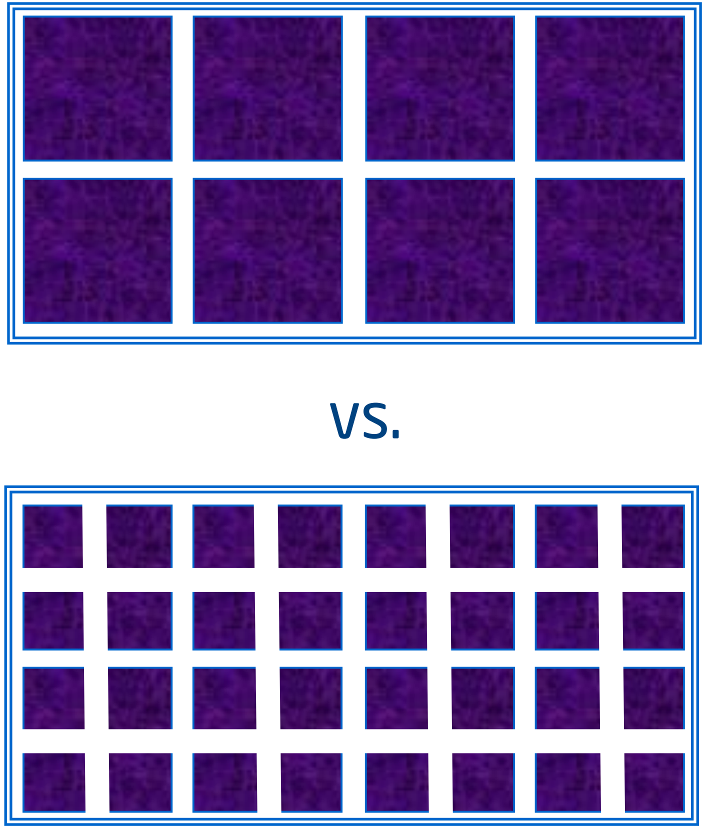
Why Intel® Xeon Phi™ ?

Intel® Xeon Phi™ Coprocessor

It’s just a different design point.

Not a different programming paradigm.

Little cores vs. big cores. All x86.



Performance

$$\frac{\text{Work}}{\text{Time}} = \underbrace{\frac{\text{Work}}{\text{Instructions}}}_{\boxed{\text{Path Length}}} \times \underbrace{\frac{\text{Instruction}}{\text{Cycle}}}_{\boxed{\text{IPC}}} \times \underbrace{\frac{\text{Cycle}}{\text{Time}}}_{\boxed{\text{Frequency}}}$$

Better algorithm → same work with fewer instructions

The compiler can optimize for fewer instructions, choose instructions with better IPC

<u>Cache efficient algorithms</u>: higher IPC

<u>Vectorization</u>: same work with fewer instructions

<u>Parallelization</u>: more instructions per cycle

Remember Pollack's rule: Performance ~

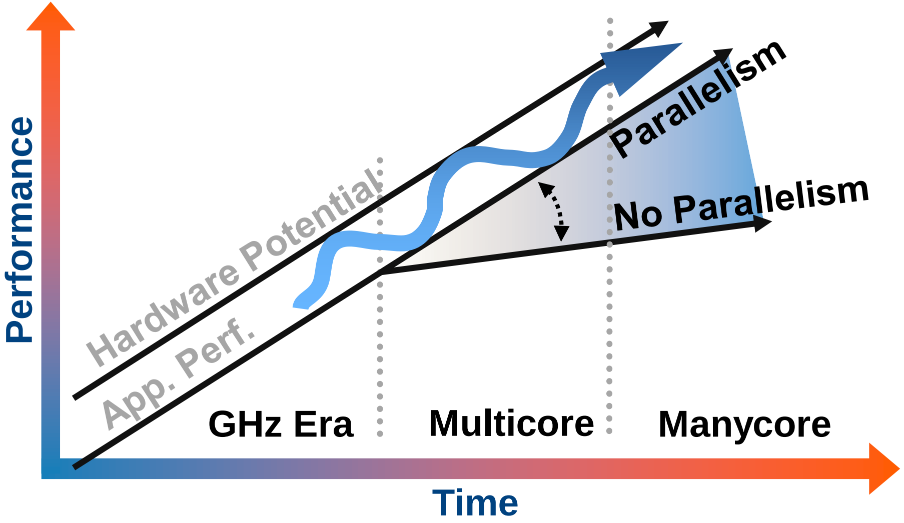
4x the die area gives 2x the performance in one core, but
4x the performance when dedicated to 4 cores

Conclusions (with respect to Pollack's rule)

A powerful handle to adjust "Performance/Watt"

Weaker cores can be beneficial (but many of them)

→ Parallel hardware
→ Parallel algorithms
→ Appropriate tools



Speedup?

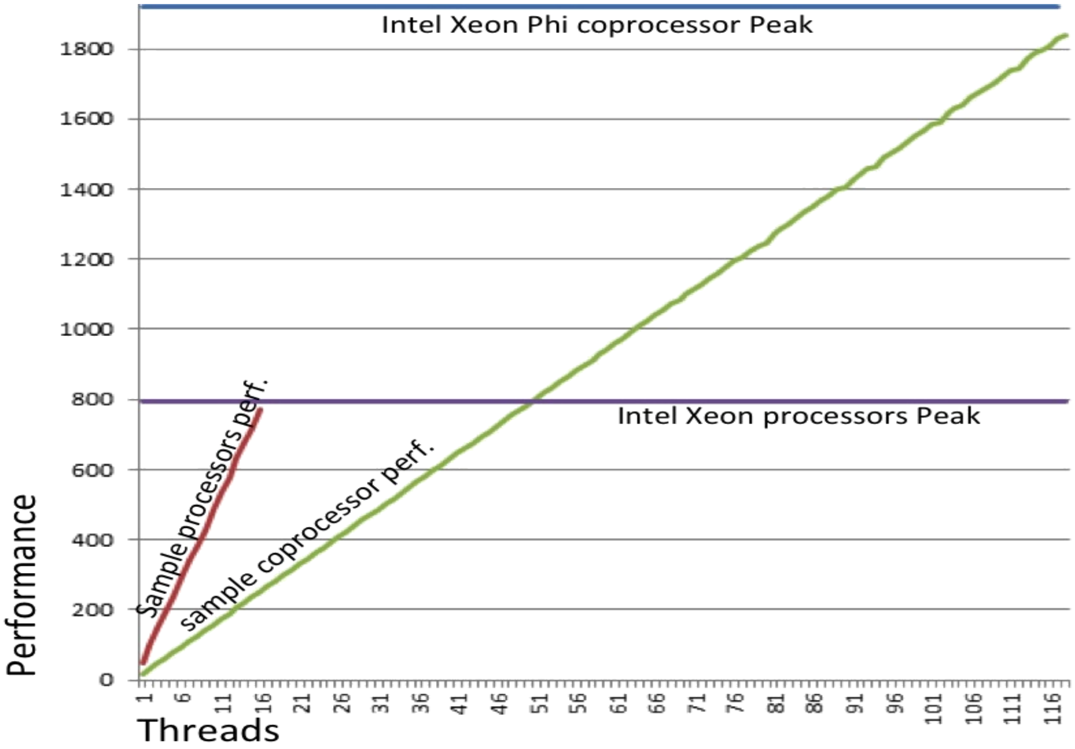
Peak perf. by example (http://ark.intel.com/)

- Intel Xeon E5-2680 (not the top-bin)
  2S x 8C x 2.7 GHz x 4F^{DP} x 2 ops* → ~345 GF/s

- Intel Xeon Phi 3120A (lowest bin)
  57C x 1.1 GHz x 8F^{DP} x 2 ops* → ~1 TF/s

Amdahl's Law determines the total speedup S* with S* = 1 / [(1-P) + P/S] of a mixture of serial and parallel code sections with the parallel speedup S and an amount of parallel code P (strong scaling).

Picture worth many words



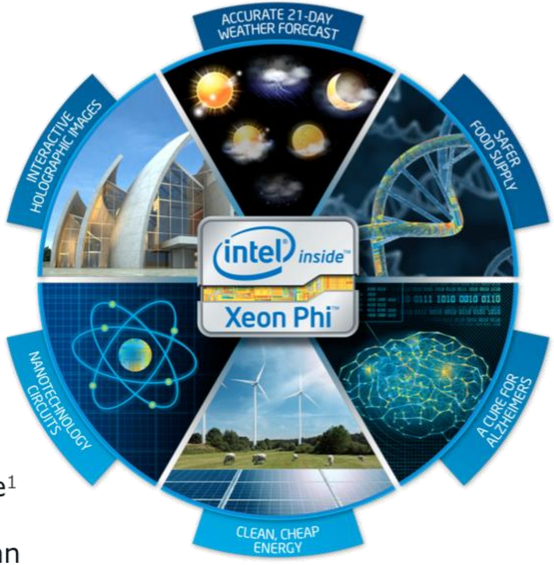
© 2013, James Reinders & Jim Jeffers, diagram used with permission

Intel® Xeon Phi™ Coprocessors

Highly-parallel Processing for Unparalleled Discovery

Groundbreaking: differences

- Up to 61 IA cores/1.1 GHz/ 244 Threads
- Up to 8GB memory with up to 352 GB/s bandwidth
- 512-bit SIMD instructions
- Linux operating system, IP addressable
- Standard programming languages and tools

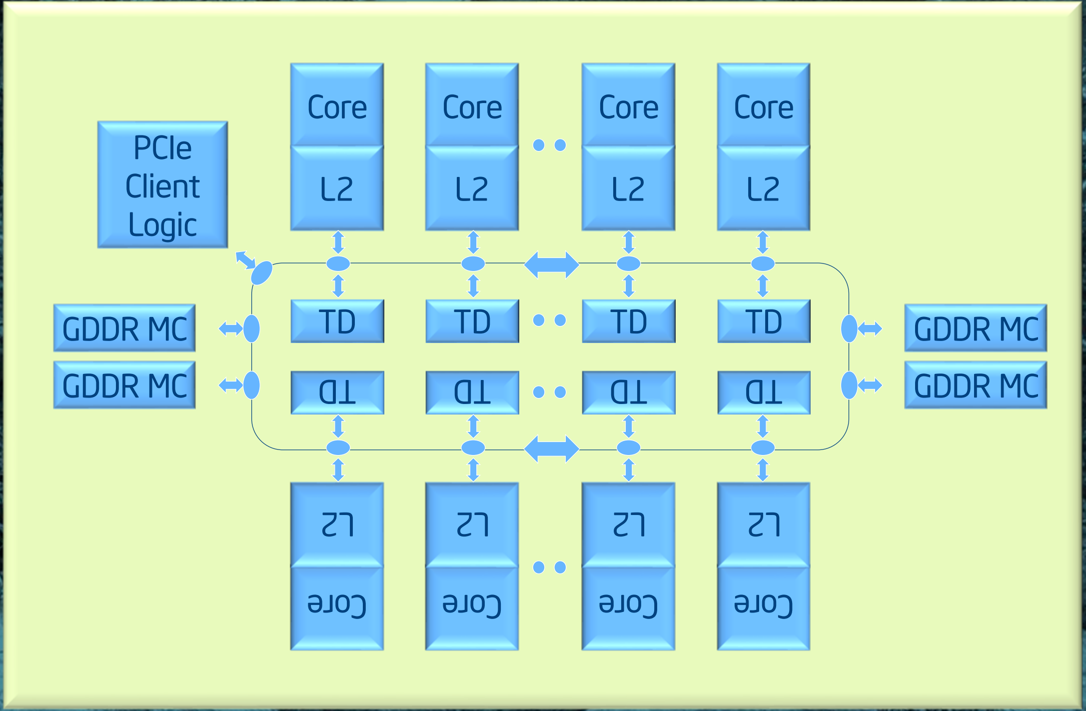


Leading to Groundbreaking results

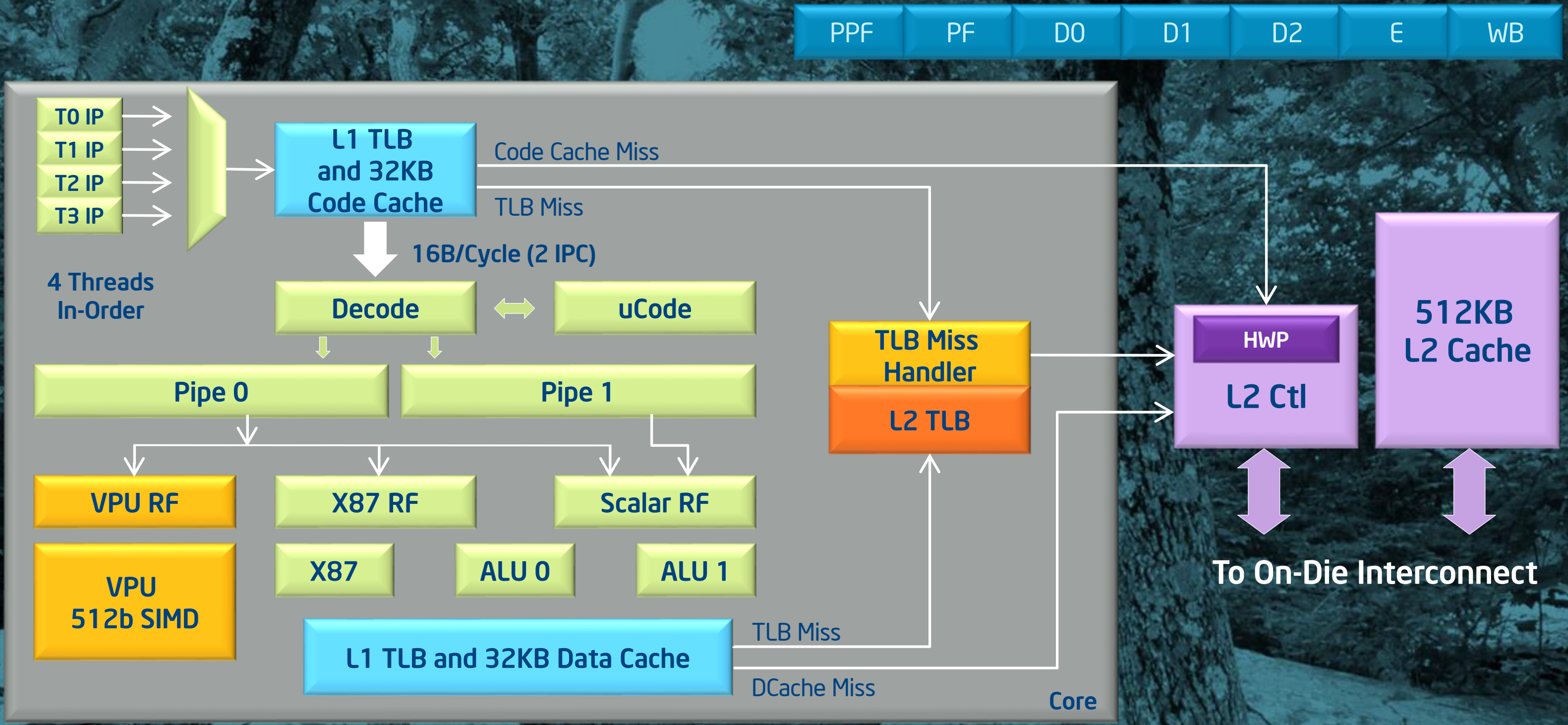
- Up to 1 TeraFlop/s double precision peak performance[1]
- Enjoy up to 2.2x higher memory bandwidth than on an Intel® Xeon® processor E5 family-based server.[2]
- Up to 4x more performance per watt than with an Intel® Xeon® processor E5 family-based server. [3]

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.
For more information go to http://www.intel.com/performance Notes 1, 2 & 3, see backup for system configuration details.

Knights Corner Micro-architecture

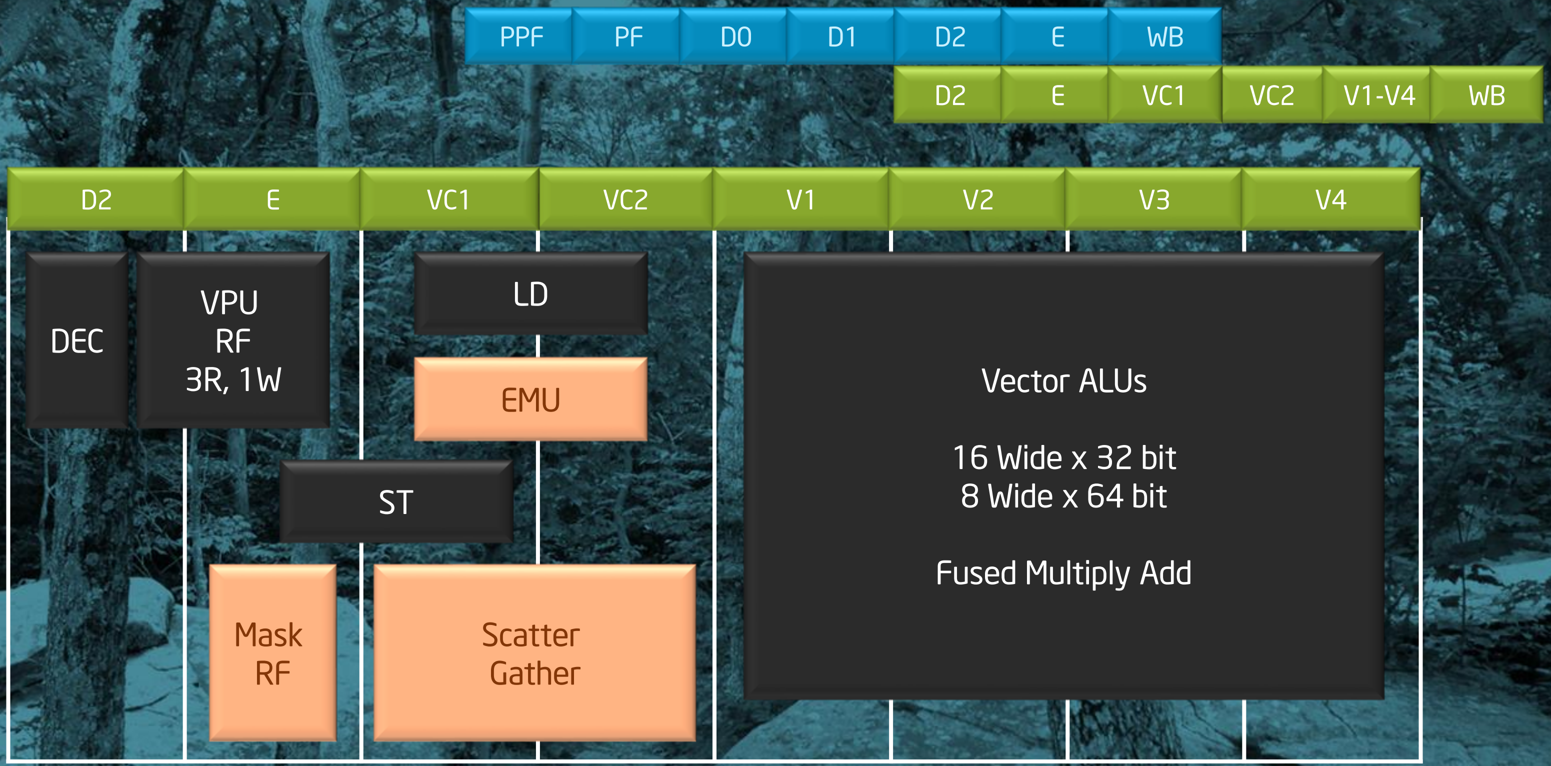


Knights Corner Core

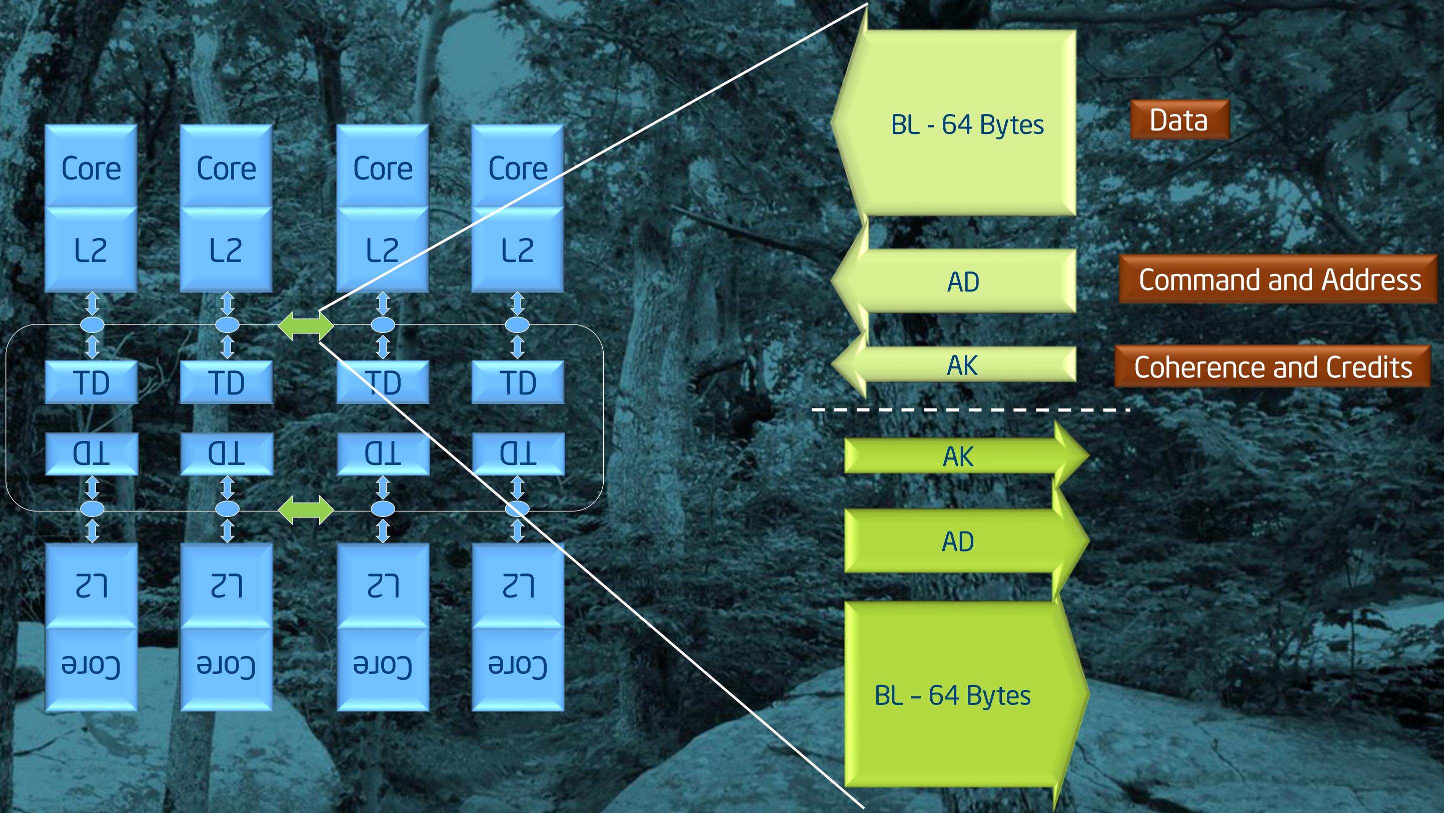


x86 specific logic < 2% of core + L2 area

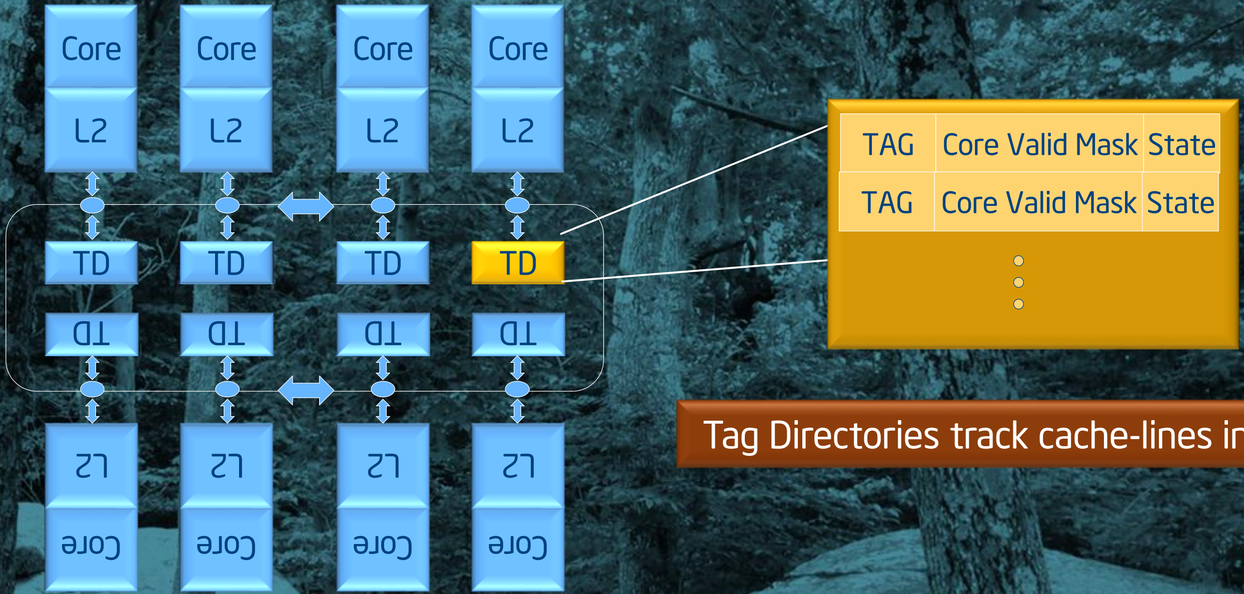
Vector Processing Unit



Interconnect

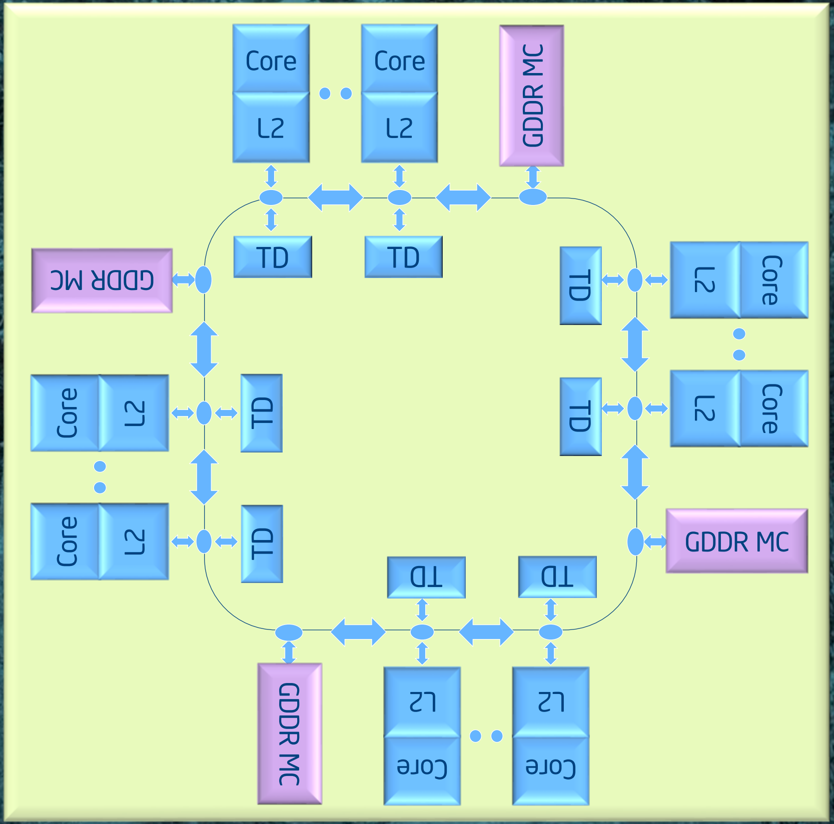


Distributed Tag Directories

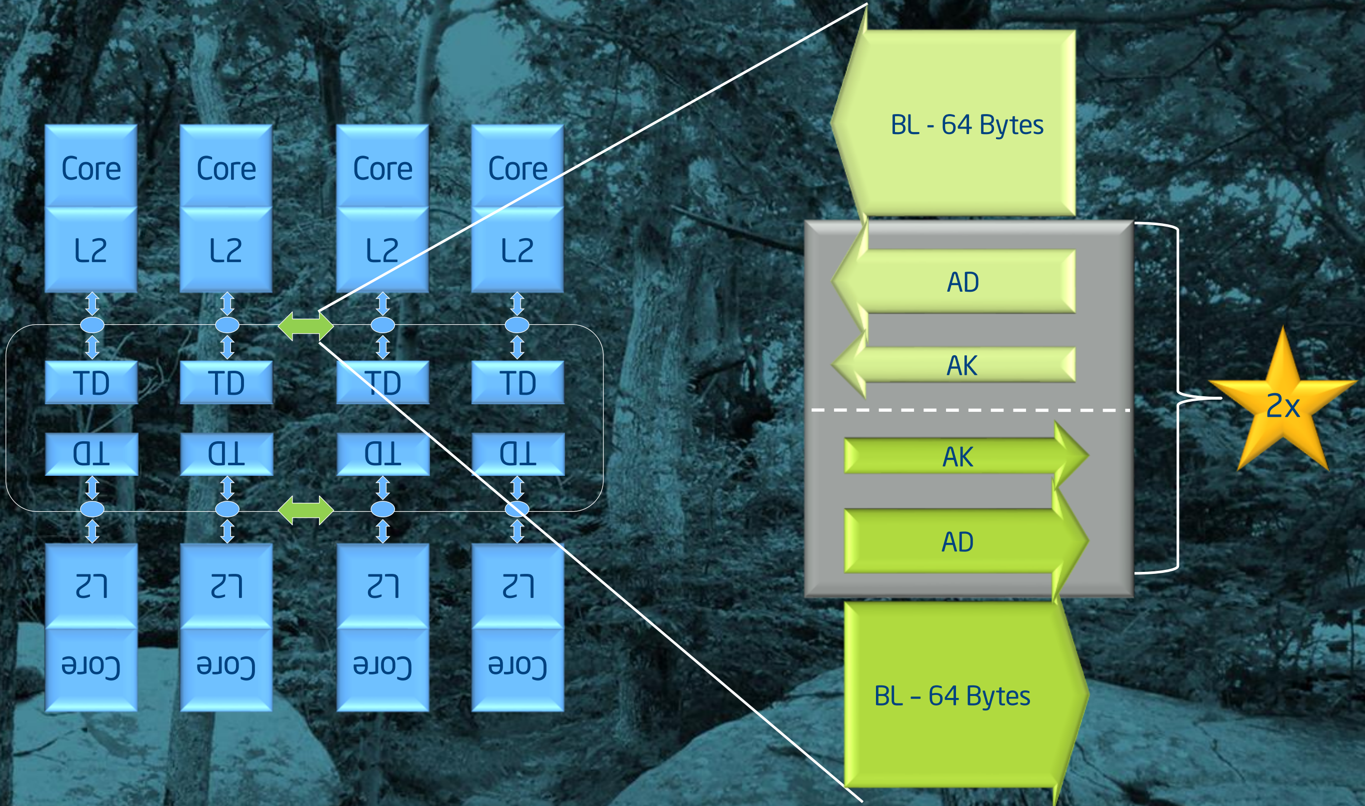


Tag Directories track cache-lines in all L2s

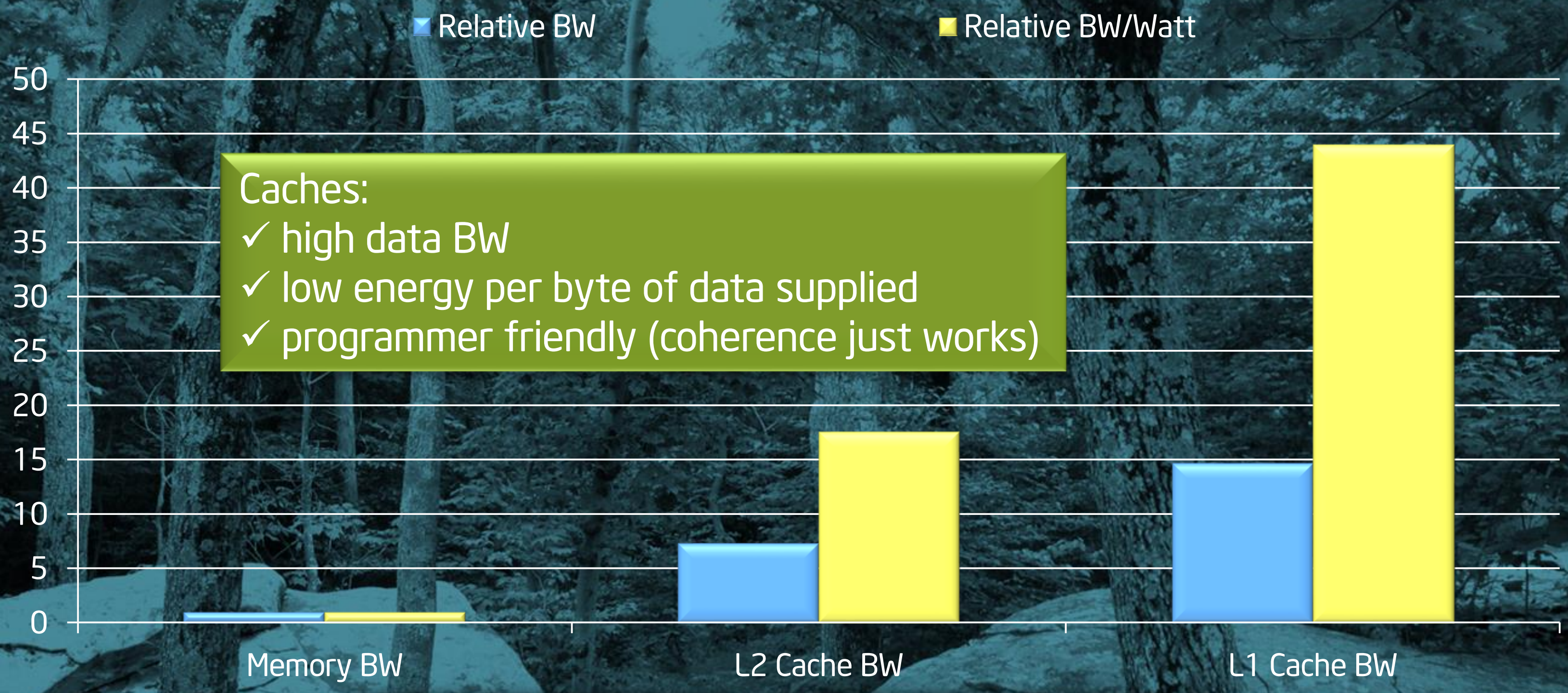
Interleaved Memory Access



Interconnect: 2X AD/AK



Caches – For or Against?

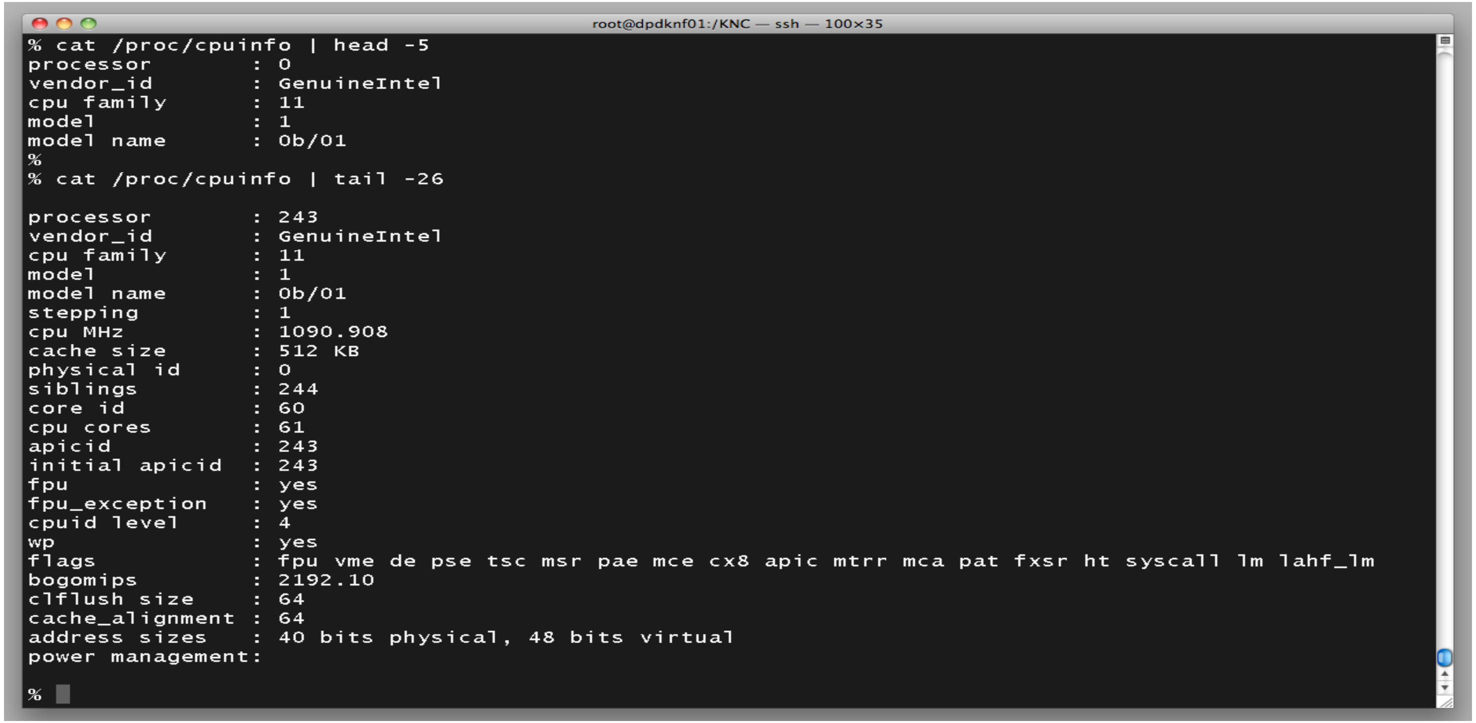


Caches:
- ✓ high data BW
- ✓ low energy per byte of data supplied
- ✓ programmer friendly (coherence just works)

Coherent Caches are a key MIC Architecture Advantage

Results have been simulated and are provided for informational purposes only. Results were derived using simulations run on an architecture simulator or model. Any difference in system hardware or software design or configuration may affect actual performance.

it is an SMP-on-a-chip running Linux



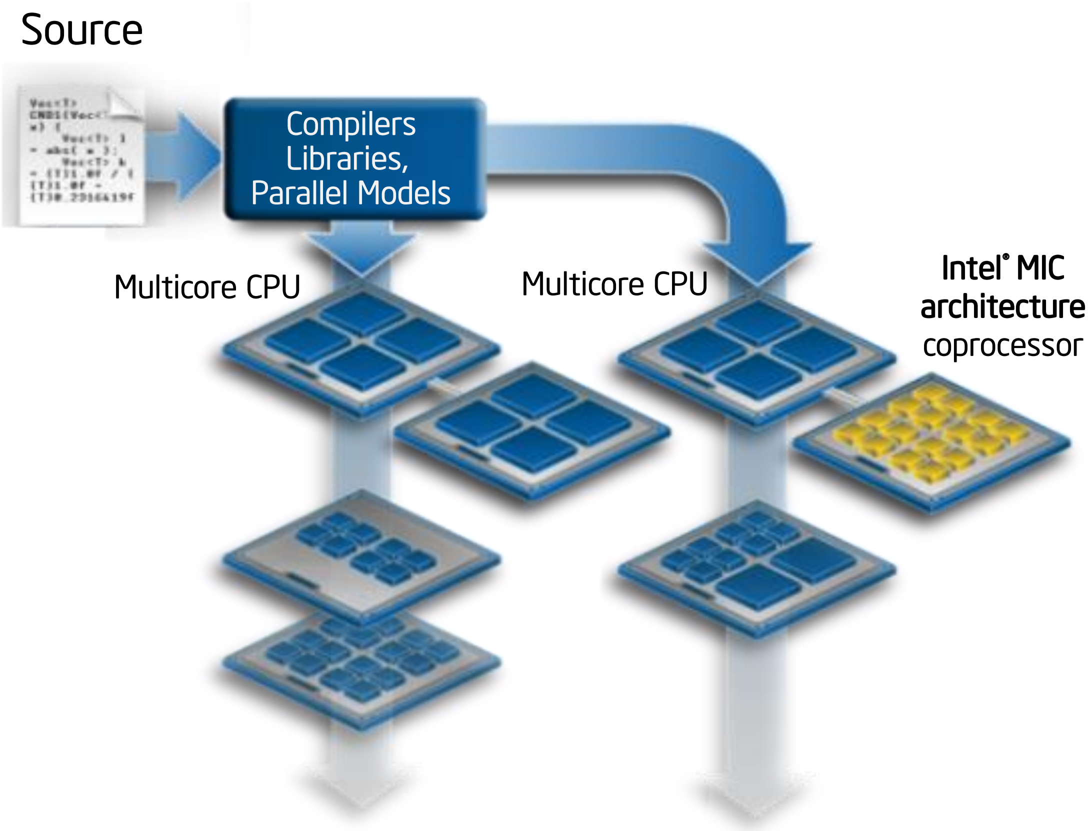

```
% cat /proc/cpuinfo | head -5
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 11
model           : 1
model name      : 0b/01
%
% cat /proc/cpuinfo | tail -26

processor       : 243
vendor_id       : GenuineIntel
cpu family      : 11
model           : 1
model name      : 0b/01
stepping        : 1
cpu MHz         : 1090.908
cache size      : 512 KB
physical id     : 0
siblings        : 244
core id         : 60
cpu cores       : 61
apicid          : 243
initial apicid  : 243
fpu             : yes
fpu_exception   : yes
cpuid level     : 4
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr mca pat fxsr ht syscall lm lahf_lm
bogomips        : 2192.10
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:

%
```

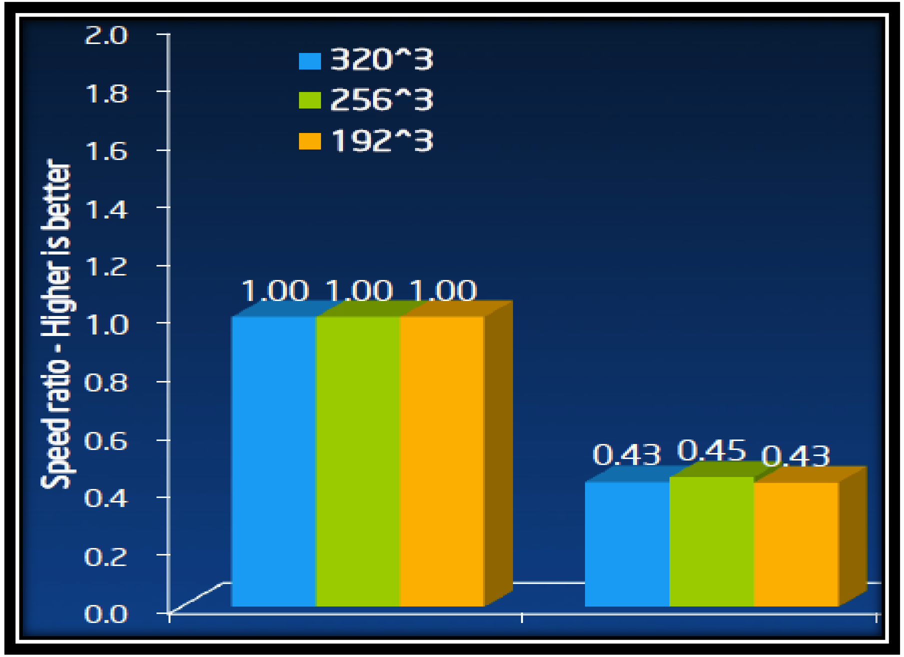
vision

span from few cores to many cores with consistent models, languages, tools, and techniques


Source
Compilers
Libraries,
Parallel Models
Multicore CPU
Multicore CPU
Intel® MIC
architecture
coprocessor

Illustrative example

Fortran code using MPI, single threaded originally.
Run on Intel® Xeon Phi™ coprocessor natively (no offload).



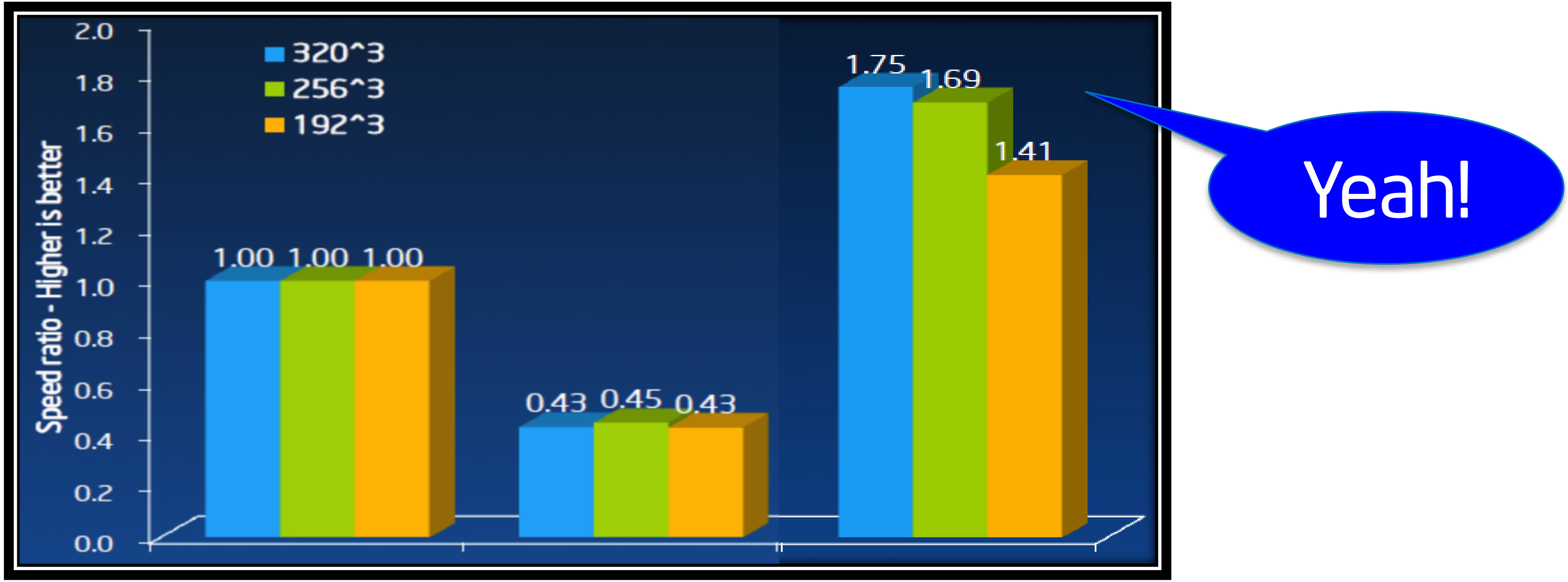
Untuned Performance on Intel® Xeon® processor

Untuned Performance on Intel® Xeon Phi™ coprocessor

Based on an actual customer example.
Shown to illustrate a point about common techniques.
Your results may vary!

Illustrative example

Fortran code using MPI, single threaded originally.
Run on Intel® Xeon Phi™ coprocessor natively (no offload).



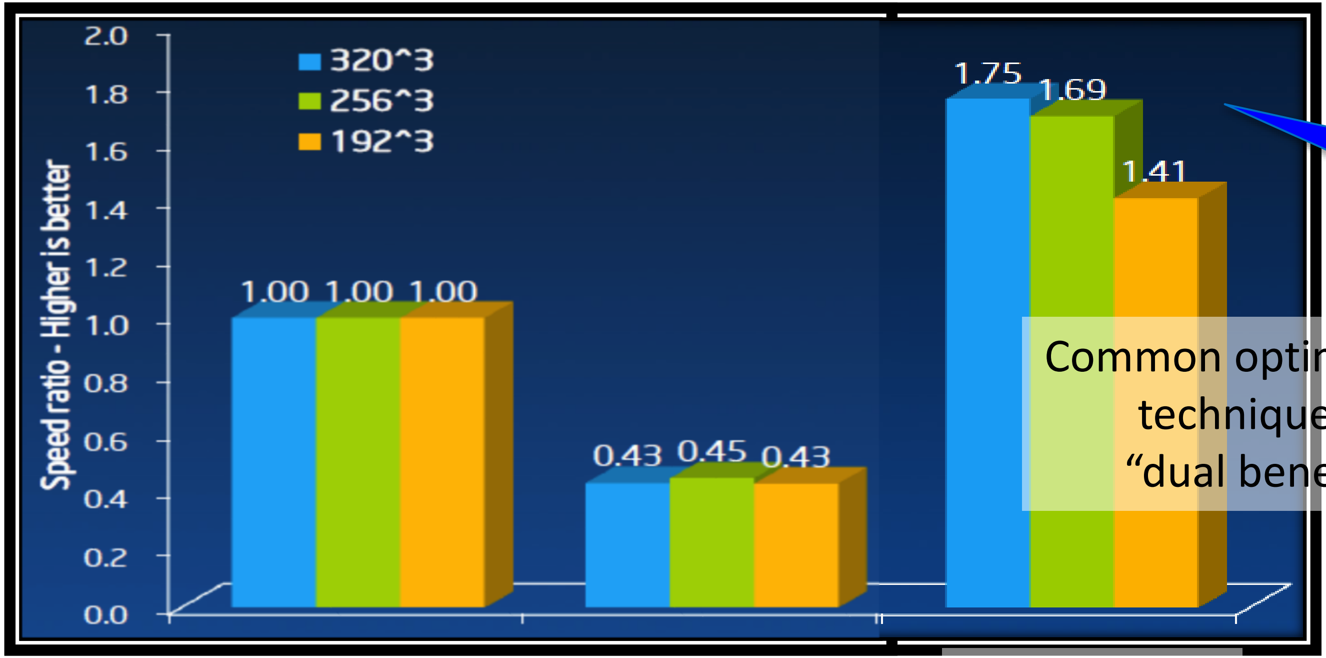
Untuned Performance on Intel® Xeon® processor

Untuned Performance on Intel® Xeon Phi™ coprocessor

TUNED Performance on Intel® Xeon Phi™ coprocessor

Illustrative example

Fortran code using MPI, single threaded originally.
Run on Intel® Xeon Phi™ coprocessor natively (no offload).



Yeah!

Common optimization techniques…
“dual benefit”

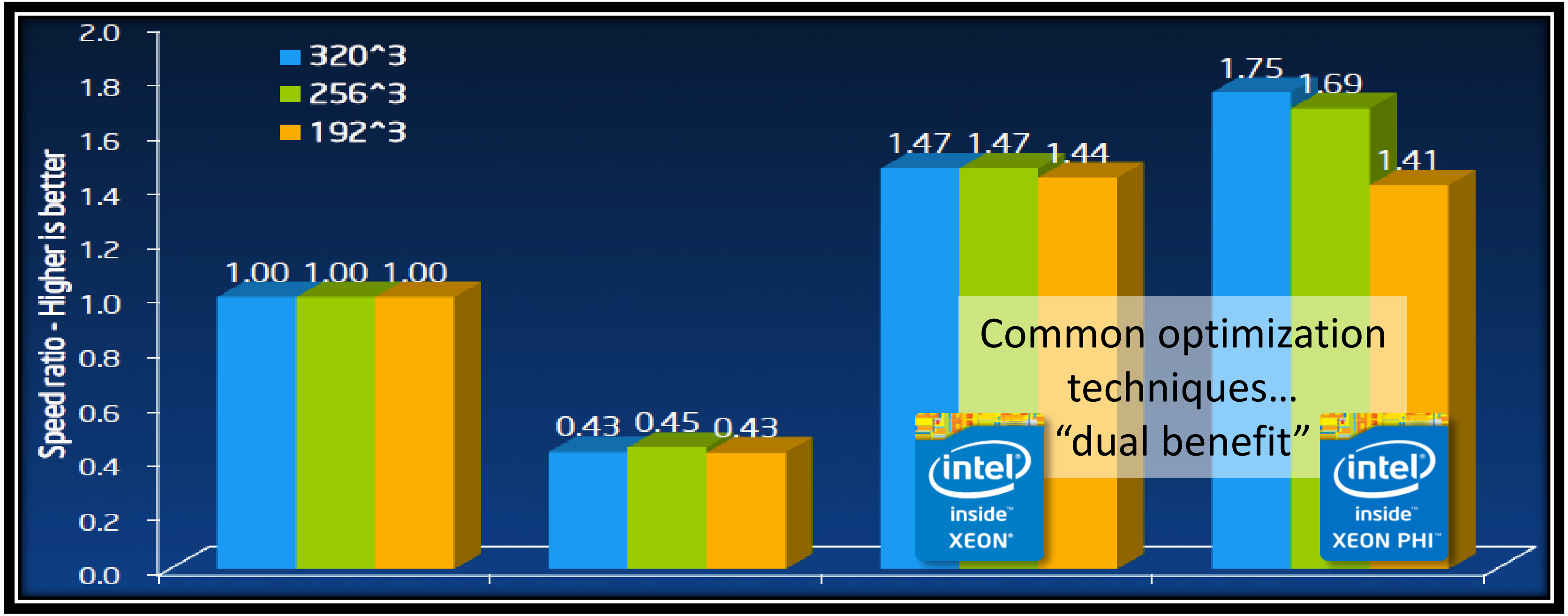
Untuned Performance on Intel® Xeon® processor

Untuned Performance on Intel® Xeon Phi™ coprocessor

TUNED Performance on Intel® Xeon Phi™ coprocessor

Illustrative example

Fortran code using MPI, single threaded originally.
Run on Intel® Xeon Phi™ coprocessor natively (no offload).

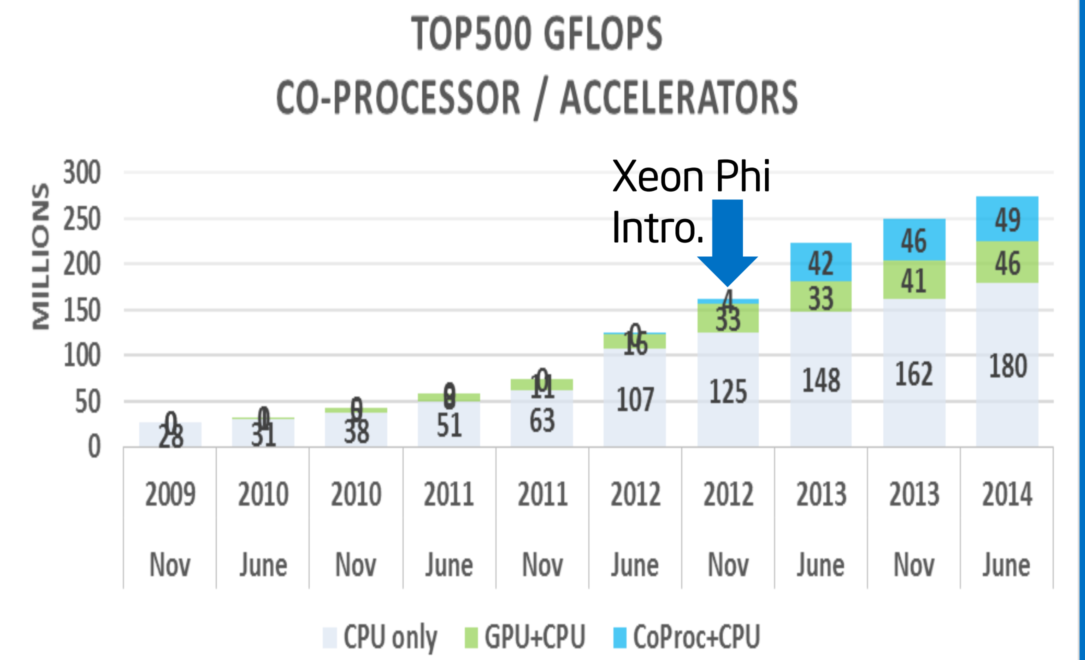


Untuned Performance on Intel® Xeon® processor

Untuned Performance on Intel® Xeon Phi™ coprocessor

TUNED Performance on Intel® Xeon® processor

TUNED Performance on Intel® Xeon Phi™ coprocessor



Source: June 2014 "Top 500" - www.top500.org

Top 500 (June 2014):
Again… the
#1 system
(third time)

is a
Neo-heterogeneous
system
(Common Programming Model)

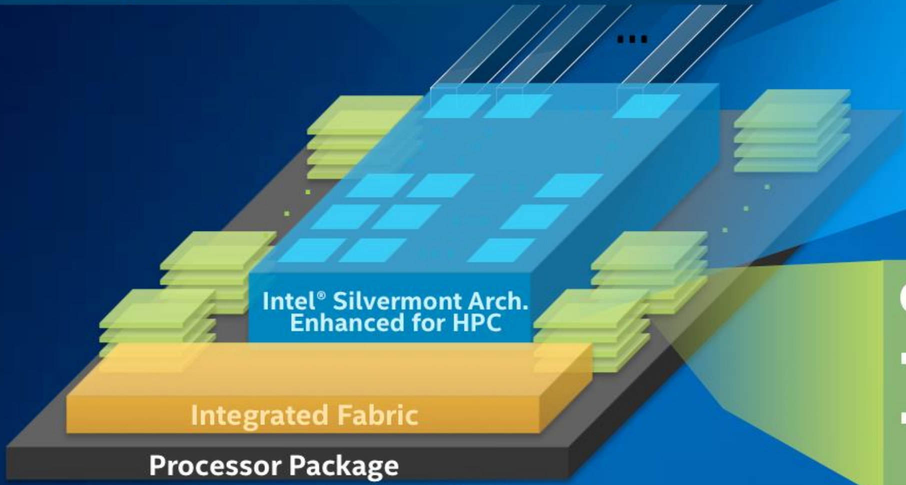
(Intel® Xeon® Processors + Intel® Xeon Phi™ Coprocessor)

Knights Landing

(Next Generation Intel® Xeon Phi™ Products)

Continued programming model advantage
Add Intel® AVX-512 instructions
gcc work well underway

Platform Memory: DDR4 Bandwidth and Capacity Comparable to Intel® Xeon® Processors



Conceptual—Not Actual Package Layout

Compute: Energy-efficient IA cores

- *Microarchitecture enhanced for HPC*
- ***3X** Single Thread Performance vs Knights Corner*
- *Intel Xeon Processor Binary Compatible*

On-Package Memory:

- *up to **16GB** at launch*
- ***1/3X** the Space*
- ***5X** Bandwidth vs DDR4*
- ***5X** Power Efficiency*

Jointly Developed with Micron Technology

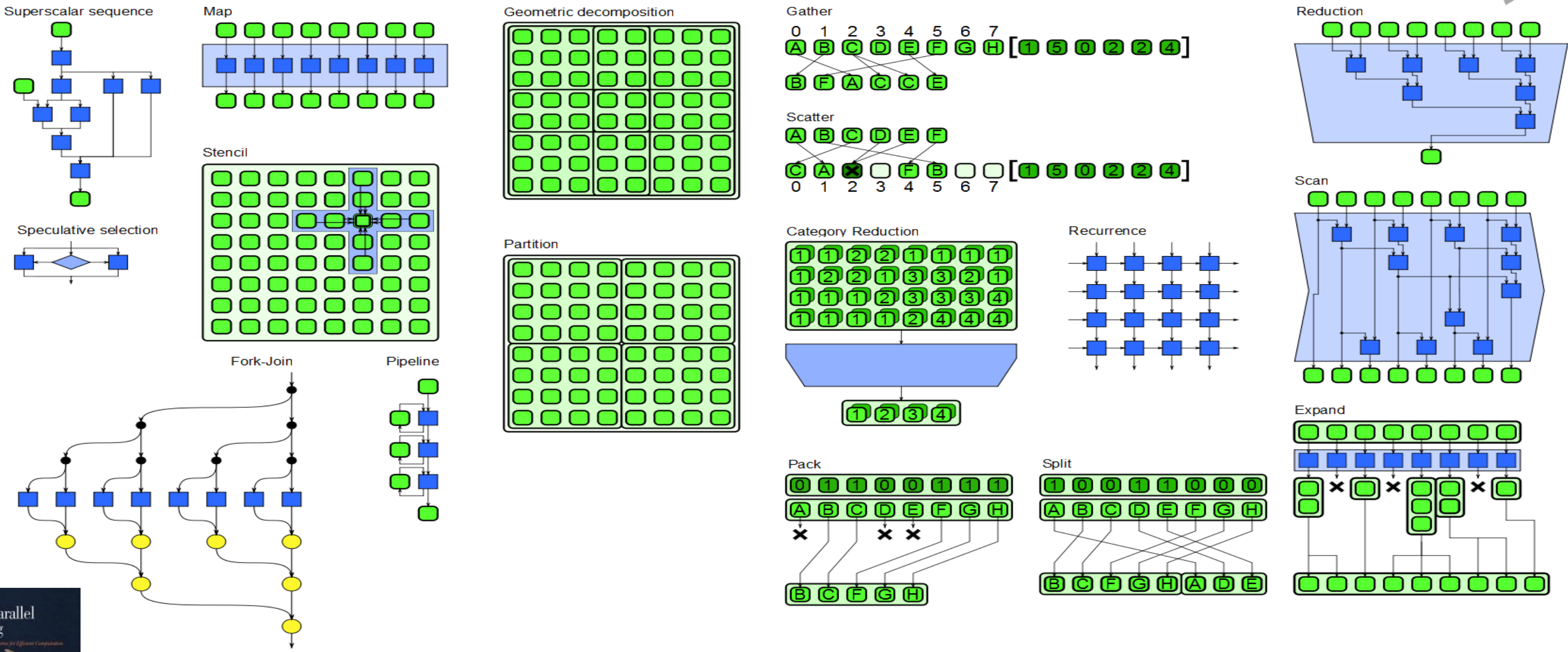
3+ TFLOPS
In One Package
Parallel Performance & Density

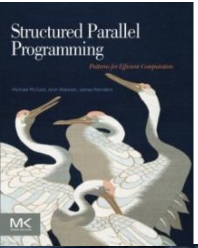
2nd half '15
1st commercial systems

Source: June 2014 Intel @ ISC'14

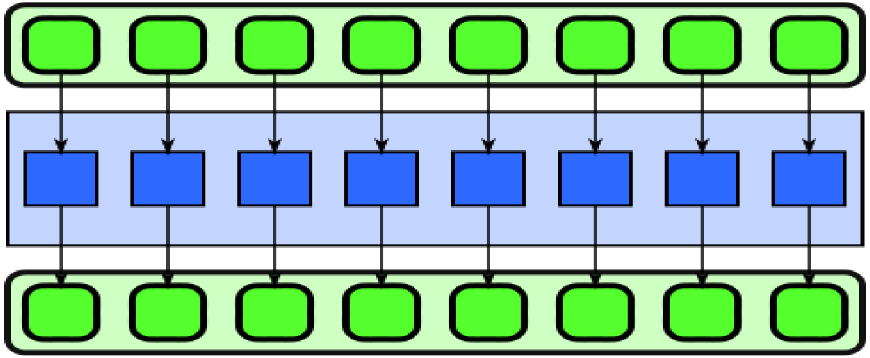
How do I “think parallel” ?

Parallel Patterns: Overview





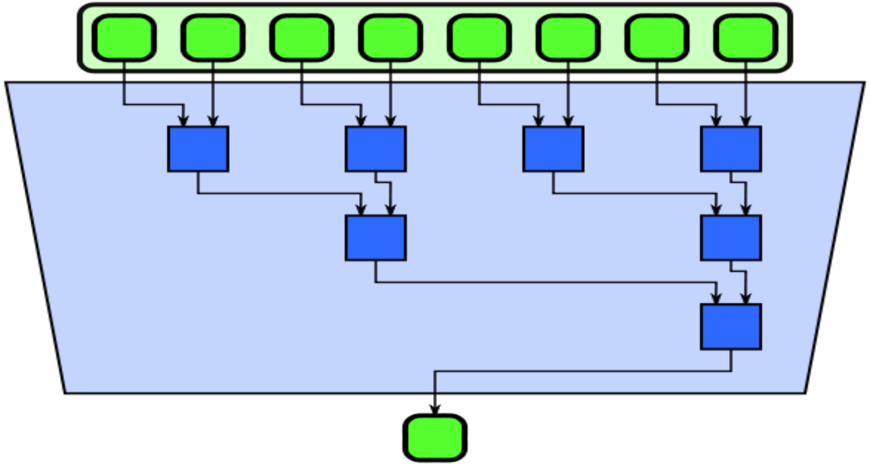
Map



Examples: gamma correction and thresholding in images; color space conversions; Monte Carlo sampling; ray tracing.

- *Map* invokes a function on every element of an index set.
- The index set may be abstract or associated with the elements of an array.
- Corresponds to “parallel loop” where iterations are independent.

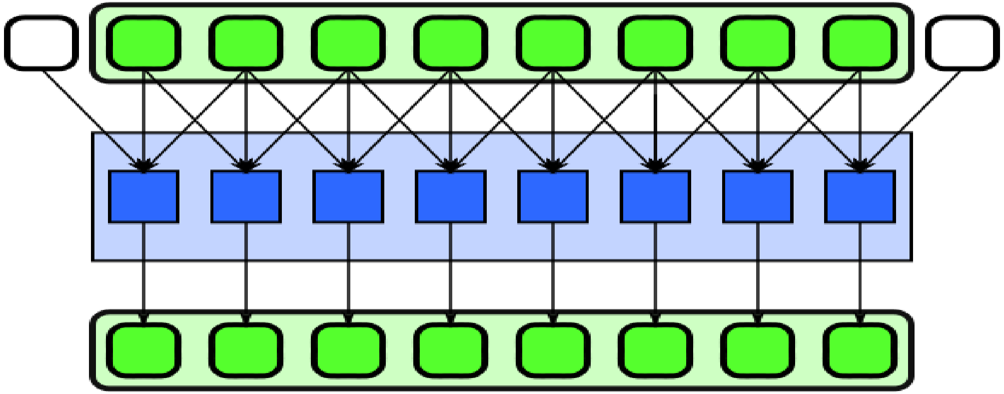
Reduce



Examples: averaging of Monte Carlo samples; convergence testing; image comparison metrics; matrix operations.

- *Reduce* combines every element in a collection into one using an *associative* operator:
  $x+(y+z) = (x+y)+z$
- For example: *reduce* can be used to find the sum or maximum of an array.
- Vectorization may require that the operator *also* be *commutative*:
  $x+y = y+x$

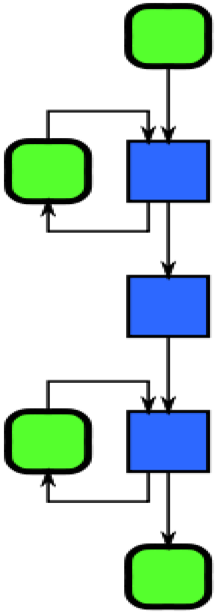
Stencil



Examples: image filtering including convolution, median, anisotropic diffusion

- *Stencil* applies a function to neighbourhoods of an array.
- Neighbourhoods are given by set of relative offsets.
- Boundary conditions need to be considered.

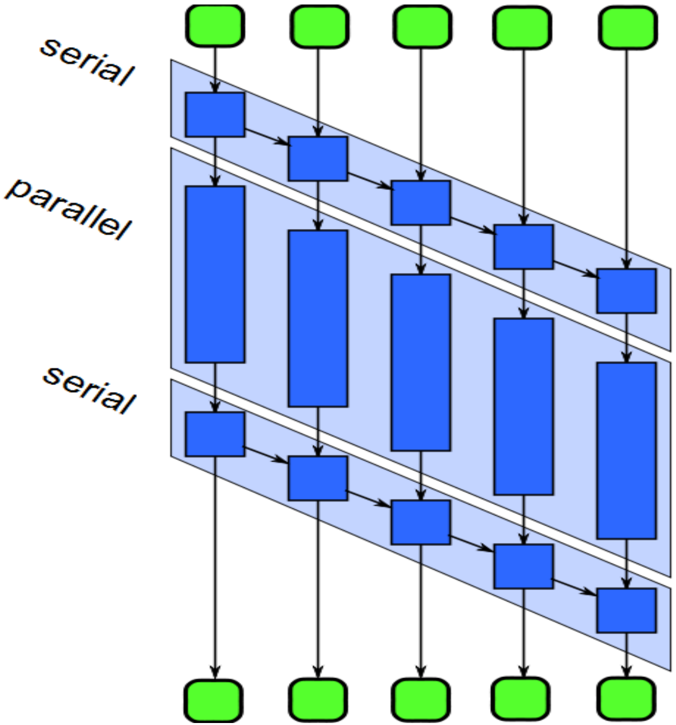
Pipeline



- *Pipeline* uses a sequence of stages that transform a flow of data
- Some stages may retain state
- Data can be consumed and produced incrementally: "online"

Examples: image filtering, data compression and decompression, signal processing

Pipeline



- Parallelize pipeline by
  - Running different stages in parallel
  - Running *multiple copies* of stateless stages in parallel

- Running multiple copies of stateless stages in parallel requires reordering of outputs

- Need to manage buffering between stages

For More Information

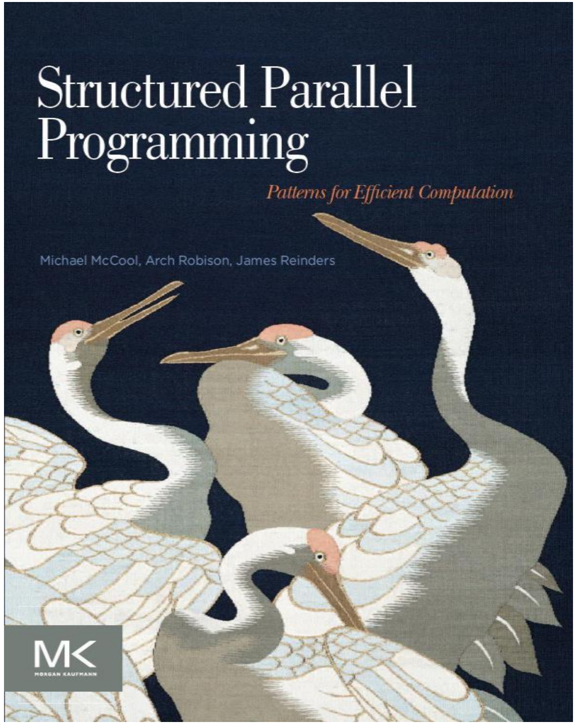
Structured Parallel Programming

- Michael McCool
- Arch Robison
- James Reinders

Uses Cilk Plus and TBB as primary frameworks for examples.

Appendices concisely summarize Cilk Plus and TBB.

www.parallelbook.com



Use abstractions !!!

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++

Use abstractions !!!

Avoid direct programming to the low level interfaces (like pthreads).

PROGRAM IN TASKS, NOT THREADS

Is OpenCL* low level? For HPC – YES.

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++

Choose First
(limited functions)

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++

Choose First
(limited functions)

Cluster
(distributed memory)

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++

Choose First
(limited functions)

Cluster
(distributed
memory)

Node
(shared
memory)

Intel Threading Building Blocks

We asked ourselves:

- How should C++ be extended?
  - "templates / generic programming"
- What do we want to solve?
  - Abstraction with good performance (scalability)
  - Abstraction that steers toward easier (less) debugging
  - Abstraction that is readable

Generic Parallel Algorithms
Efficient scalable way to exploit the power of multi-core without having to start from scratch

Concurrent Containers
Concurrent access, and a scalable alternative to containers that are externally locked for thread-safety

Flow Graph
A set of classes to express parallelism via a dependency graph or a data flow graph

Thread Local Storage
Supports infinite number of thread local data

Task Scheduler
Sophisticated engine with a variety of work scheduling techniques that empowers parallel algorithms & the flow graph

Synchronization Primitives
Atomic operations, several flavors of mutexes, condition variables

Thread-safe timers

Threads
OS API wrappers

Memory Allocation
Per-thread scalable memory manager and false-sharing free allocators

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++

Choose First
(limited functions)

Cluster
(distributed
memory)

Node
(shared
memory)

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++

Choose First
(limited functions)

Cluster
(distributed
memory)

Node
(shared
memory)

Up and coming
for C++
(keywords,
compilers)

Because… you
just have to
expect "more"

Affect future
C++ standards?
(2021?)

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++
implemented	vendor libraries	many	in compiler	portable	in compiler
standard	open interfaces	open interfaces	OpenMP standard (1997-)	open source (2007, Intel)	open interfaces (MIT, Intel)
supported by	most vendors	open src & vendors	most compilers	ported most everywhere	gcc and Intel (llvm future)

Compare...

proprietary	NVidia* CUDA	NVidia OpenACC	Intel LEO
purpose	data parallel	offload	offload
target (perf.)	NVidia GPUs	NVidia GPUs	portable
alternative	OpenCL*	OpenMP 4.0	OpenMP 4.0

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++
implemented	vendor libraries	many	in compiler	portable	in compiler
standard	open interfaces	open interfaces	OpenMP standard (1997-)	open source (2007, Intel)	open interfaces (MIT, Intel)
supported by	most vendors	open src & vendors	most compilers	ported most everywhere	gcc and Intel (llvm future)

Compare...

proprietary	NVidia CUDA	NVidia OpenACC	Intel LEO
purpose	data parallel	offload	offload
target (perf.)	NVidia GPUs	NVidia GPUs	portable
alternative	OpenCL	OpenMP 4.0	OpenMP 4.0

Choosing a non-proprietary *parallel abstraction*

non-proprietary	BLAS, FFTW	MPI	OpenMP*	TBB	Cilk™ Plus
prog. lang.	Fortran, C, C++	Fortran, C, C++	Fortran or C	C++	C++
implemented	vendor libraries	many	in compiler	portable	in compiler
standard	open interfaces	open interfaces	OpenMP standard (1997-)	open source (2007, Intel)	open interfaces (MIT, Intel)
supported by	most vendors	open src & vendors	most compilers	ported most everywhere	gcc and Intel (llvm future)
composable?	usually	YES	NO	YES	YES
memory	shared/distributed	distributed	shared (in implementations)	shared memory	shared memory
tasks	yes	n/a	YES	YES	limited keywords, TBB
explicit SIMD	internal	n/a	YES (OpenMP 4.0: SIMD)	use compiler options, OpenMP directives, or Cilk Plus keywords	keywords
offload	some	n/a	YES (OpenMP 4.0: SIMD)	use Cilk Plus or OpenMP	keywords

It's your Forest

Questions?

james.r.reinders@intel.com

Break Now

We resume @ 10:30am

(to talk about SIMD/vectors)

intel®

james.r.reinders@intel.com

James Reinders. Parallel Programming Evangelist. **Intel.**

James is involved in multiple engineering, research and educational efforts to increase use of parallel programming throughout the industry. He joined Intel Corporation in 1989, and has contributed to numerous projects including the world's first TeraFLOP/s supercomputer (ASCI Red) and the world's first TeraFLOP/s microprocessor (Intel® Xeon Phi™ coprocessor). James been an author on numerous technical books, including VTune™ Performance Analyzer Essentials (Intel Press, 2005), Intel® Threading Building Blocks (O'Reilly Media, 2007), Structured Parallel Programming (Morgan Kaufmann, 2012), Intel® Xeon Phi™ Coprocessor High Performance Programming (Morgan Kaufmann, 2013), and Multithreading for Visual Effects (A K Peters/CRC Press, 2014). James is working on a project to publish a book of programming examples featuring Intel Xeon Phi programming scheduled to be published in late 2014.

Legal Disclaimer & Optimization Notice

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804