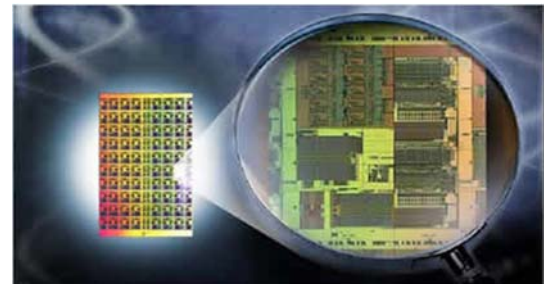
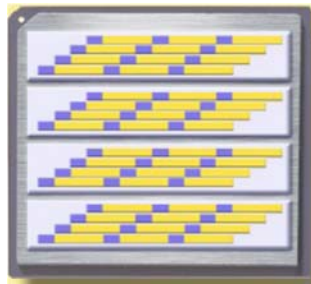
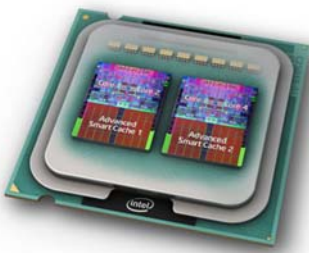


Overview of Multicore Architecture

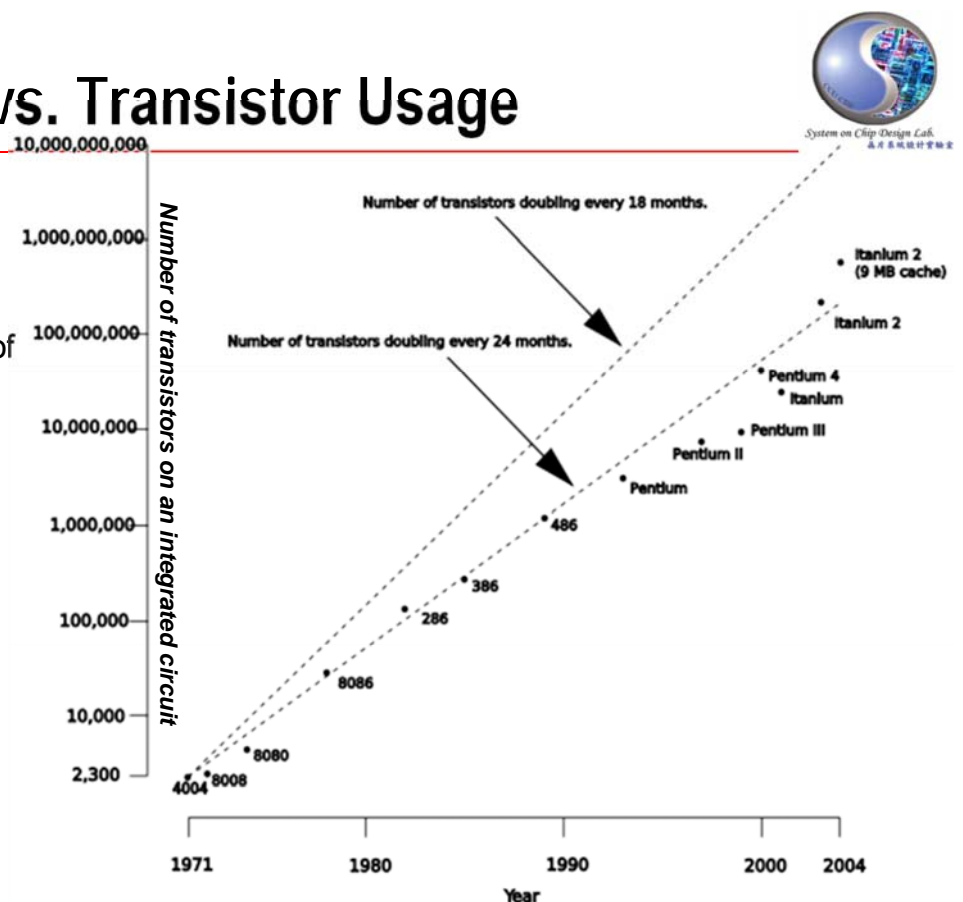
Tien-Fu Chen

Dept. of Computer Science
National Chiao Tung Univ.



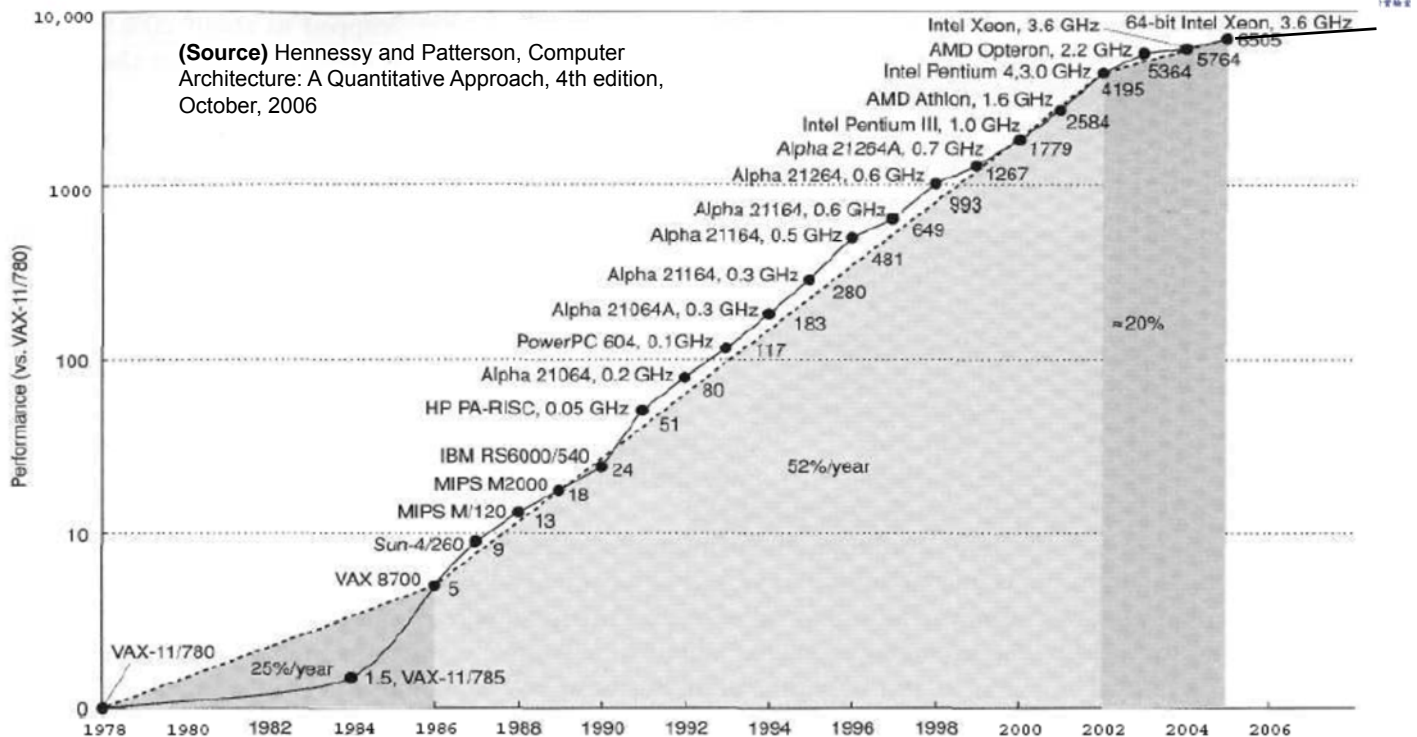
Moore's Law vs. Transistor Usage

- Moore's law: the number of transistors can be increasing exponentially, doubling approximately every 18 months
- Don't really know how to use them to gain more performance



(Source) http://en.wikipedia.org/wiki/Moore%27s_law

Challenges: Uniprocessor Performance (with SPECint benchmark)



• VAX : 25%/year 1978 to 1986

• RISC + x86: 52%/year 1986 to 2002

• RISC + x86: ???%/year 2002 to present -F. Chen@NCTU CSIE

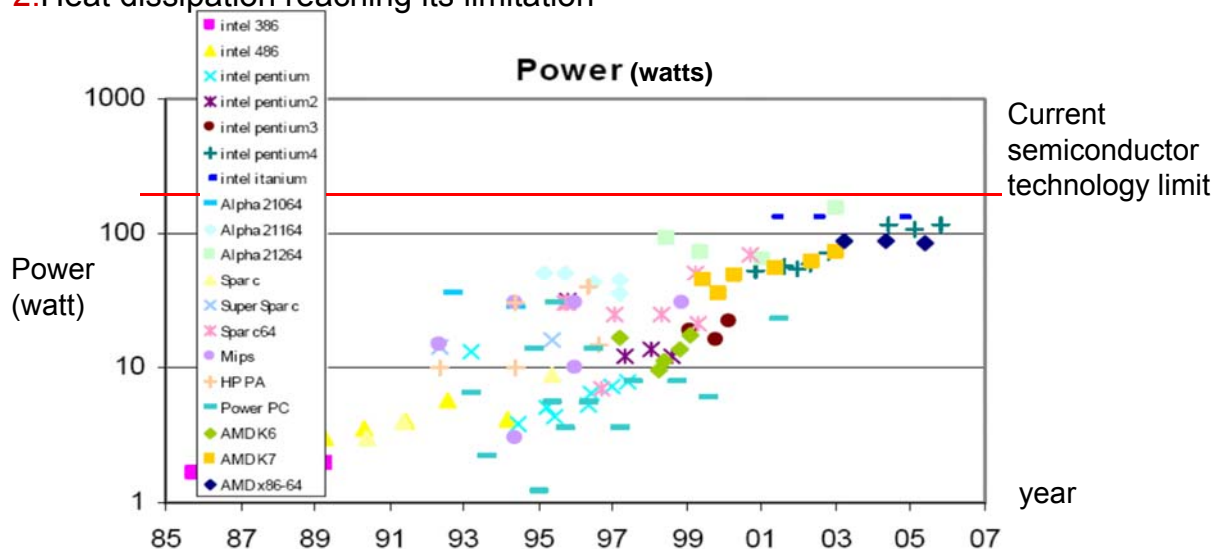
Server Architecture Overview

Challenges: Power Consumption



- For the uniprocessor performance, the power consumption becomes a problem due to:

1. Energy to operate the huge circuit
2. Heat dissipation reaching its limitation



(Source) MIT Course 6.189, "Multi-core Programming Primer: Learn and Compete in Programming the PLAYSTATION®3 Cell Processor."

Server Architecture Overview

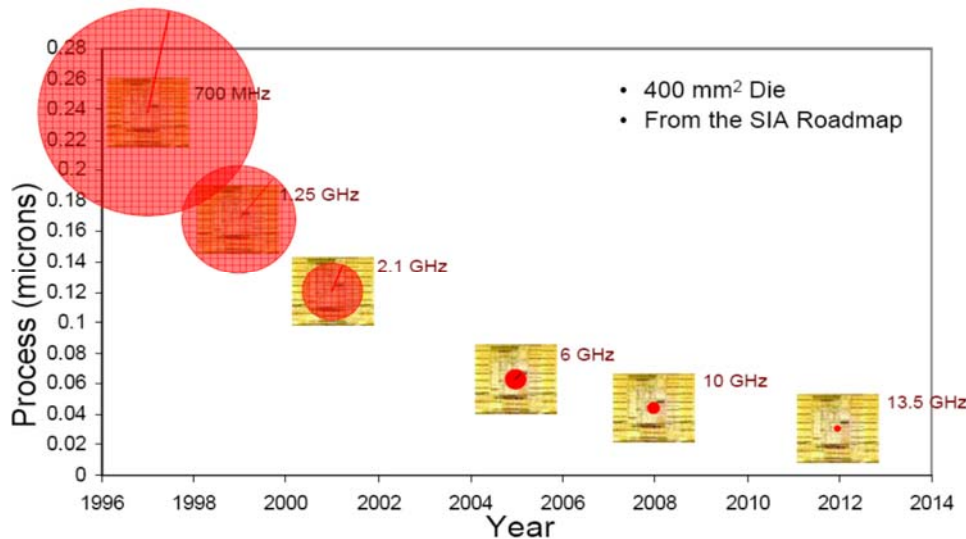
Lect01- 4

T.-F. Chen@NCTU CSIE

Challenges: Wire Delay becomes issue when clock rate raised



- Signal delay on a wire increases in proportion to the product of its resistance and capacitance
 - As feature size shrinks, wire gets shorter, but the resistance and capacitance per unit length get worse

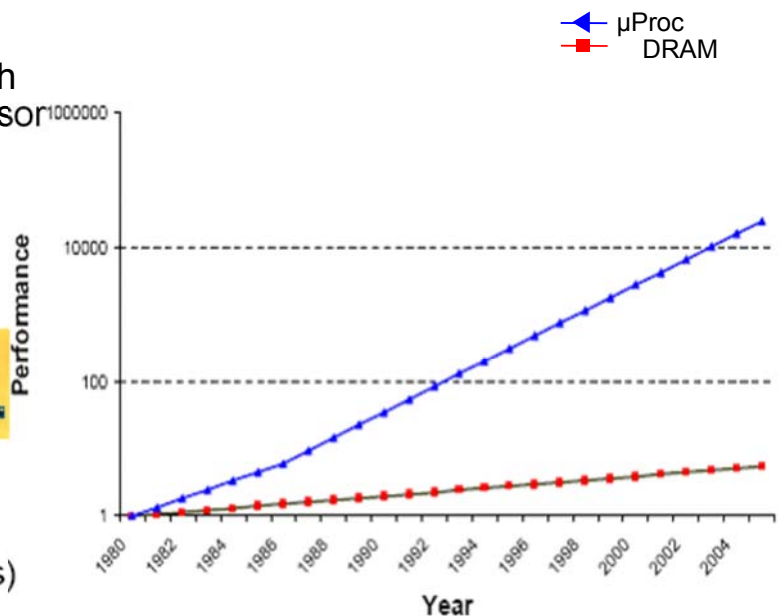
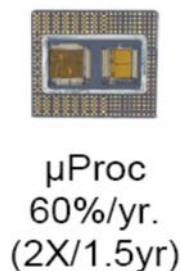


(Source) MIT Course 6.189, "Multi-core Programming Primer: Learn and Compete in Programming the PLAYSTATION®3 Cell Processor."

Challenges: Processor-Memory Gap



- The performance gap between processor and memory gets larger
- It takes more cycles to fetch data from DRAM to processor



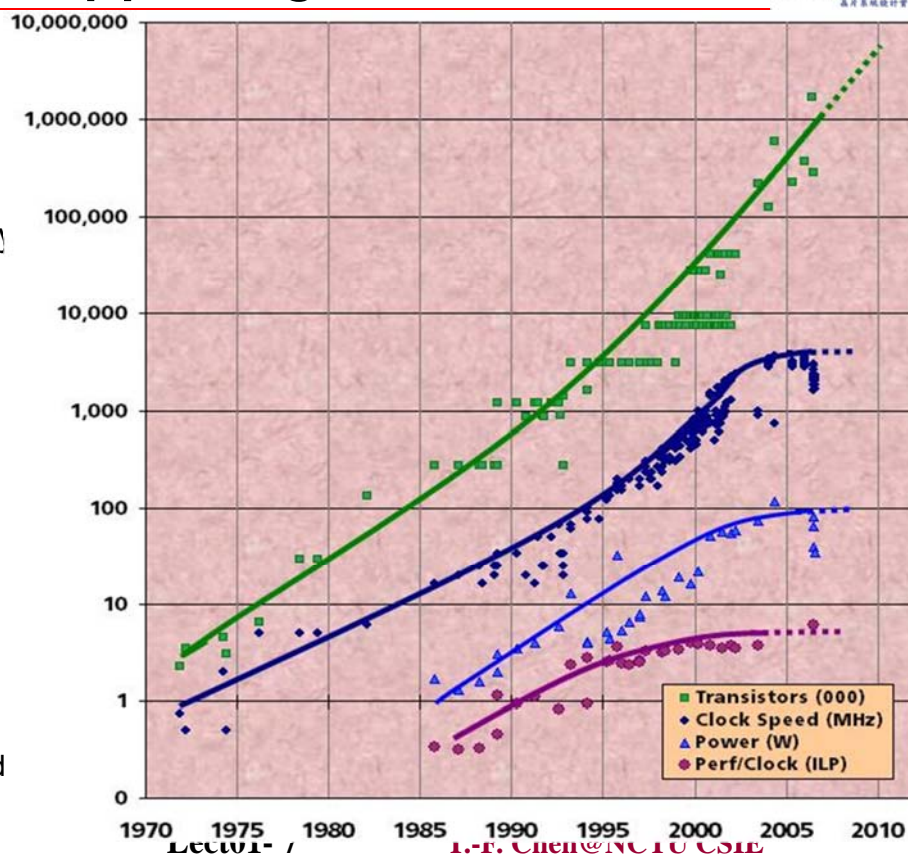
μProc v.s. DRAM performance trends
(Source) MIT Course 6.189, "Multi-core Programming Primer: Learn and Compete in Programming the PLAYSTATION®3 Cell Processor."



System on Chip Design Lab
晶片系統設計實驗室

Revolution is Happening Now

- ❑ Chip density is continuing increase
~2x every 2 years
 - Clock speed is not
 - Number of processor cores may double instead
- ❑ There is little or no hidden parallelism (ILP) to be found
- ❑ Parallelism must be exposed to and managed by software



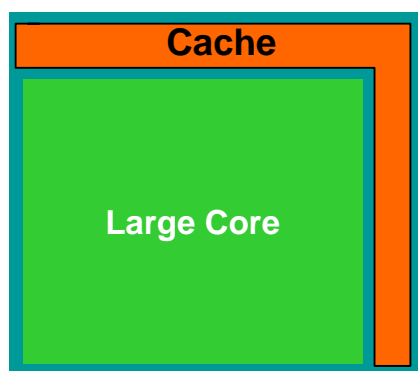
Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

Server Architecture Overview

Why multicore?



System on Chip Design Lab
晶片系統設計實驗室



Power

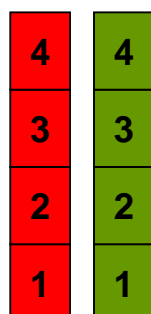
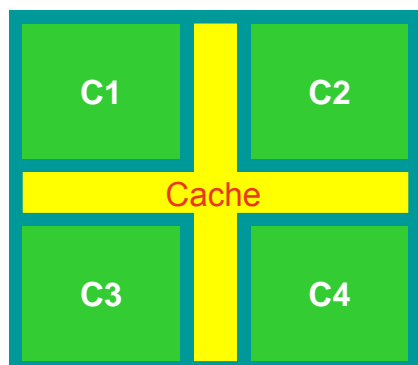
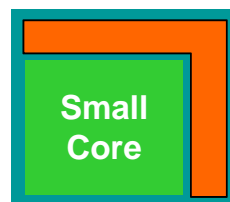


Performance



Power = 1/4

Performance = 1/2



Multi-Core:
Power efficient
Better power and thermal management

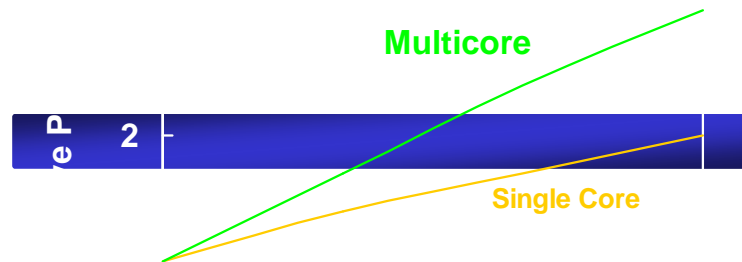
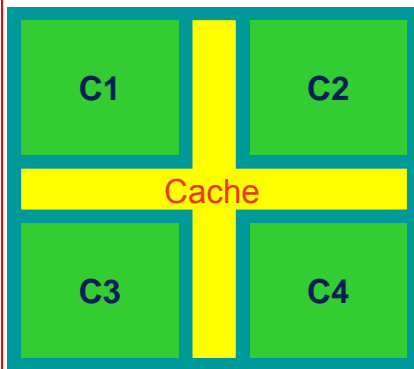
(Source) S. Borkar, "VLSI Design Challenges for Gigascale Integration," in International Conference on 18th VLSI Design, 2005

Server Architecture Overview

Lect01- 8

T.-F. Chen@NCTU CSIE

Intel's Perspective about Multicore



- Multicore is more power efficient
- In addition:
 - ❑ Each core can have its own power management
 - ❑ Cores can be multithreaded
 - ❑ Core hopping for better thermal management

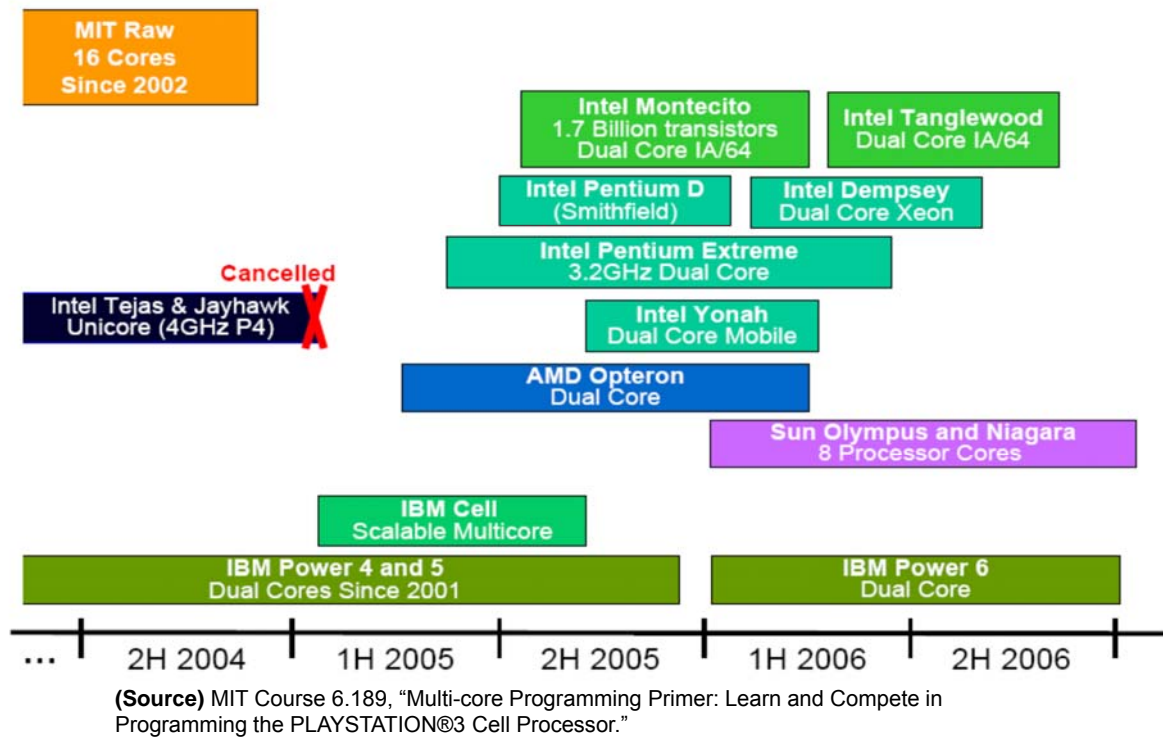
(Source) S. Borkar, "VLSI Design Challenges for Gigascale Integration,"
in *International Conference on 18th VLSI Design*, 2005

Given approximately same chip area, three different types of processors:

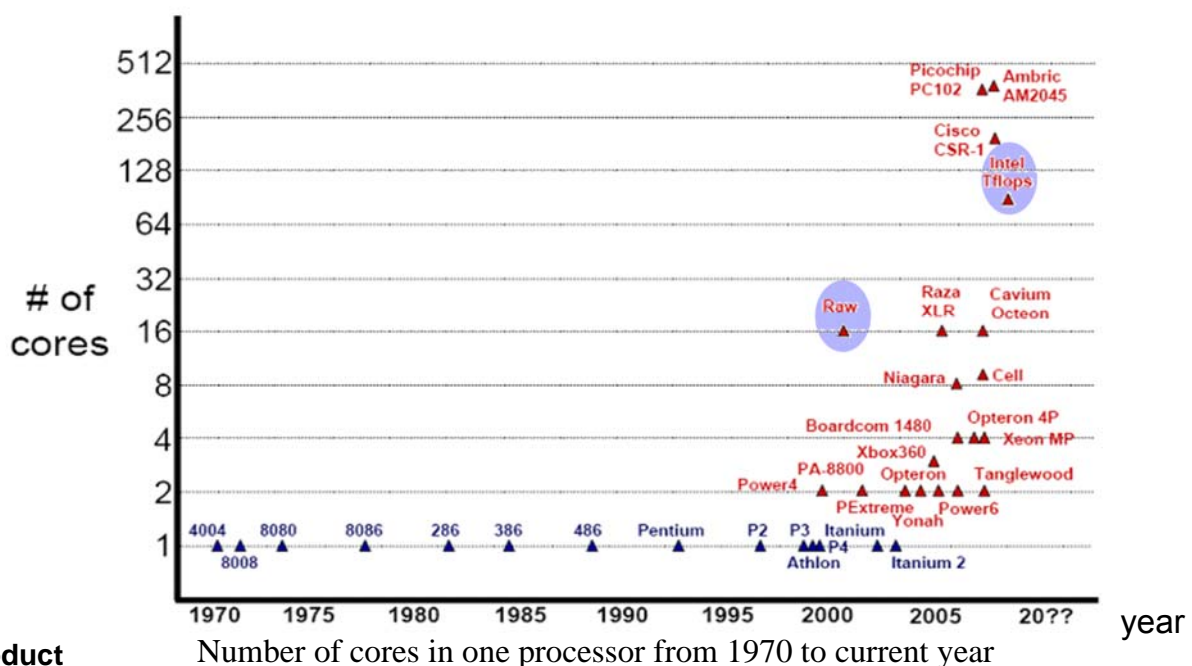
Characteristic	Superscalar	Simultaneous multithreading	Chip multiprocessor
Number of CPUs	1	1	8
CPU issue width	12	12	2 per CPU
Number of threads	1	8	1 per CPU
Architecture registers (for integer and floating point)	32	32 per thread	32 per CPU
Physical registers (for integer and floating point)	32 + 256	256 + 256	32 + 32 per CPU
Instruction window size	256	256	32 per CPU
Branch predictor table size (entries)	32,768	32,768	8 × 4,096
Return stack size	64 entries	64 entries	8 × 8 entries
Instruction (I) and data (D) cache organization	1 × 8 banks	1 × 8 banks	1 bank
I and D cache sizes	128 Kbytes	128 Kbytes	16 Kbytes per CPU
I and D cache associativities	4-way	4-way	4-way
I and D cache line sizes (bytes)	32	32	32
I and D cache access times (cycles)	2	2	1
Secondary cache organization (Mbytes)	1 × 8 banks	1 × 8 banks	1 × 8 banks
Secondary cache size (bytes)	8	8	8
Secondary cache associativity	4-way	4-way	4-way
Secondary cache line size (bytes)	32	32	32
Secondary cache access time (cycles)	5	5	7
Secondary cache occupancy per access (cycles)	1	1	1
Memory organization (no. of banks)	4	4	4
Memory access time (cycles)	50	50	50
Memory occupancy per access (cycles)	13	13	13

(Source) L. Hammond, et al, "A Single-Chip Multiprocessor," IEEE Micro, 1997

Unicores are on the verge of extinction

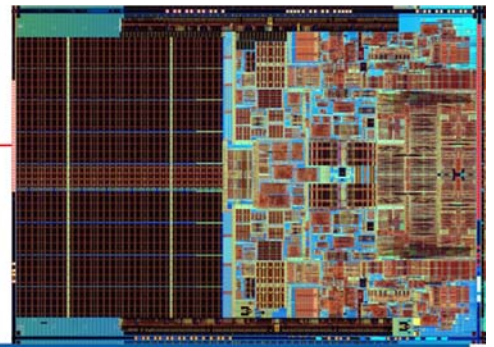


Multicores Are Future Trend

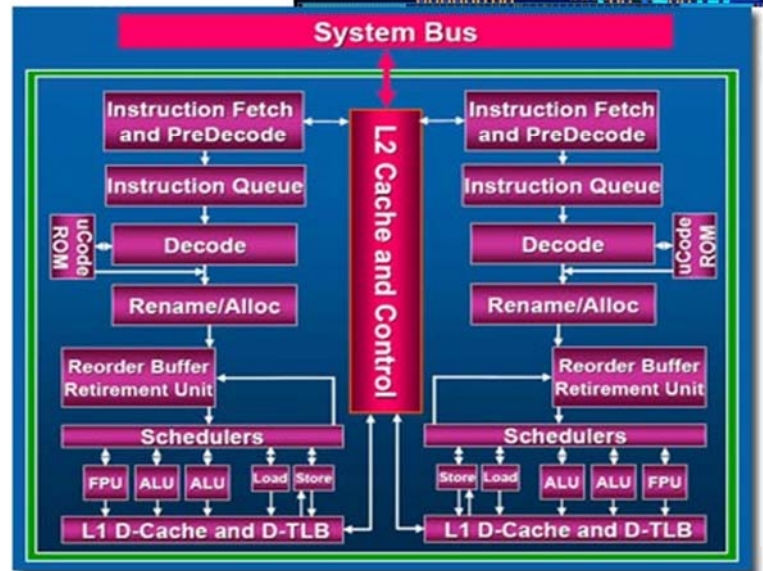


(Source) MIT Course 6.189, "Multi-core Programming Primer: Learn and Compete in Programming the PLAYSTATION®3 Cell Processor."

Intel Core2Duo: Overview

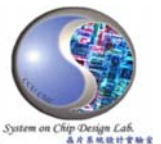


- ❑ Intel Core2Duo Processor
 - **Dual cores**
 - Out-of-Order Execution Engines
 - 32KB L1 instruction and 32KB L1 data caches, 2/3/4/6MB L2 cache
 - 667/800/1066/1333 MHz system bus
 - Intel® Smart Cache
 - Intel® Enhanced SpeedStep® Technology
 - Intel® Dynamic Power Coordination
 - Intel® Digital Media Boost

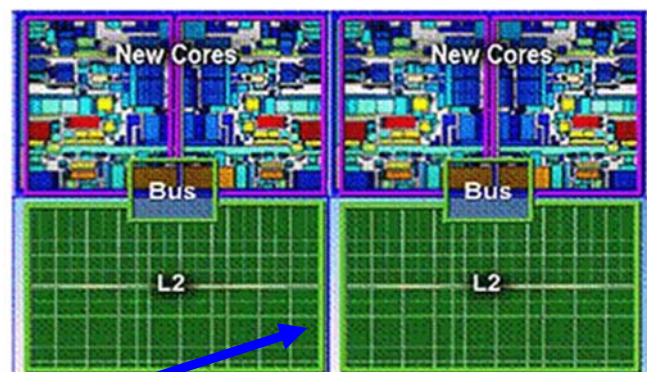
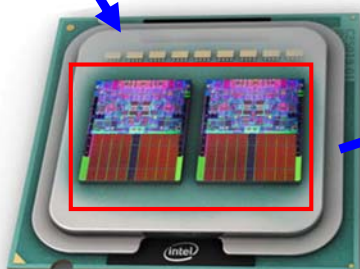
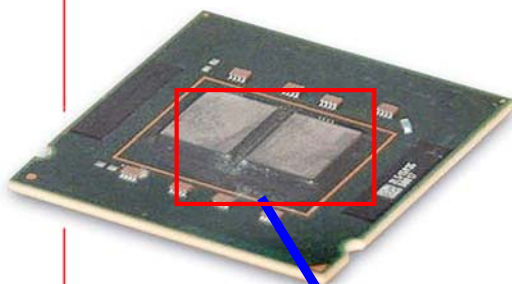


(Source) <http://www.intel.com/technology/architecture/coremicro/index.htm>

Intel Core2Duo: How about Core2Quad ?!



- ❑ Simply **pack two Core2Duos in one chip**














Intel Core2Quad architecture

(Source)

<http://www.intel.com/technology/architecture/coremicro/index.htm>

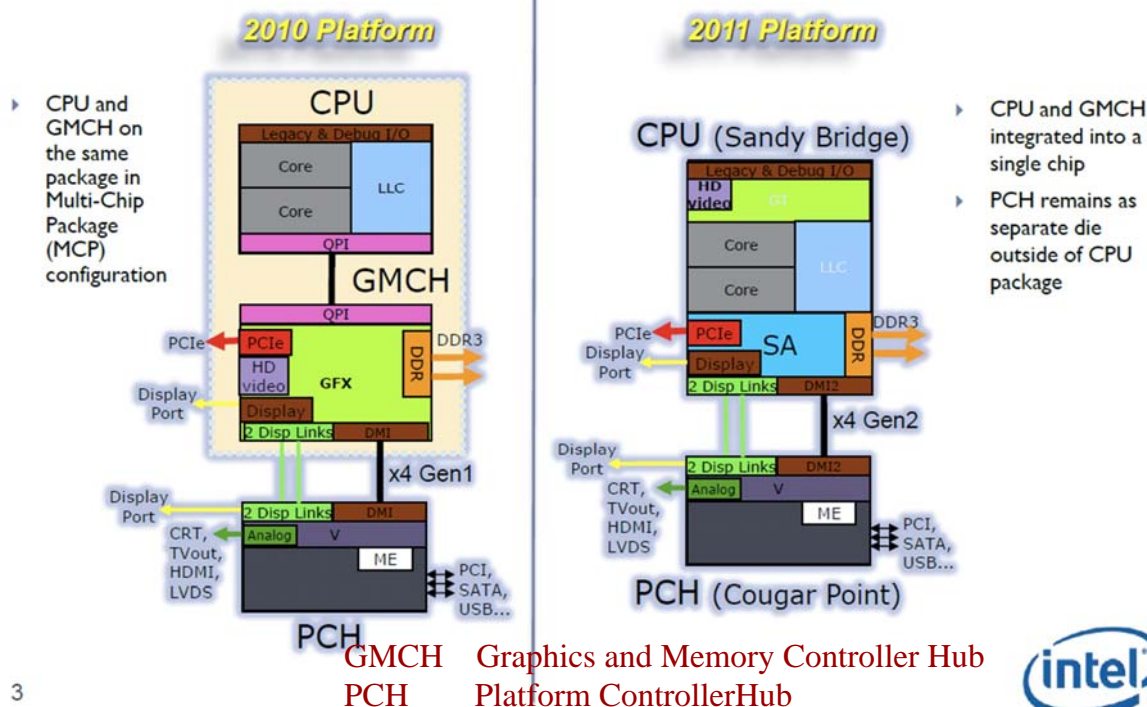
Do you understand the spec?

Intel 桌上型處理器規格表

處理器	時脈速度	核心/執行緒	快取記憶體	腳位	內建顯示晶片	TDP (W)
 i7 3960X	3.30 GHz	6 / 12	15.0 MB	2011	--	130 W
 i7 3930K	3.20 GHz	6 / 12	12.0 MB	2011	--	130 W
 i7 3820	3.60 GHz	4 / 8	10.0 MB	2011	--	130 W
 i7 2600K	3.40 GHz	4 / 8	8.0 MB	1155	HD3000	95 W
 i7 2600	3.40 GHz	4 / 8	8.0 MB	1155	HD2000	95 W
 i5 2500K	3.30 GHz	4 / 4	6.0 MB	1155	HD3000	95 W
 i5 2500	3.30 GHz	4 / 4	6.0 MB	1155	HD2000	95 W
 i5 2400	3.10 GHz	4 / 4	6.0 MB	1155	HD2000	95 W
 i5 2320	3.30 GHz	4 / 4	6.0 MB	1155	HD2000	95 W
 i3 2130	3.40 GHz	2 / 4	3.0 MB	1155	HD2000	65 W
 i3 2120	3.30 GHz	2 / 4	3.0 MB	1155	HD2000	65 W
 i3 2125	3.30 GHz	2 / 4	3.0 MB	1155	HD2000	65 W
 i3 2100	3.10 GHz	2 / 4	3.0 MB	1155	HD2000	65 W
 G860	3.00 GHz	2 / 2	3.0 MB	1155	HD	65 W
 G850	2.90 GHz	2 / 2	3.0 MB	1155	HD	65 W
 G840	2.80 GHz	2 / 2	3.0 MB	1155	HD	65 W
 G630	2.70 GHz	2 / 2	3.0 MB	1155	HD	65 W
 G620	2.60 GHz	2 / 2	3.0 MB	1155	HD	65 W
 G540	2.50 GHz	2 / 2	2.0 MB	1155	HD	65 W
 G530	2.40 GHz	2 / 2	2.0 MB	1155	HD	65 W
 G460	1.60 GHz	1 / 2	1.5 MB	1155	HD	65 W

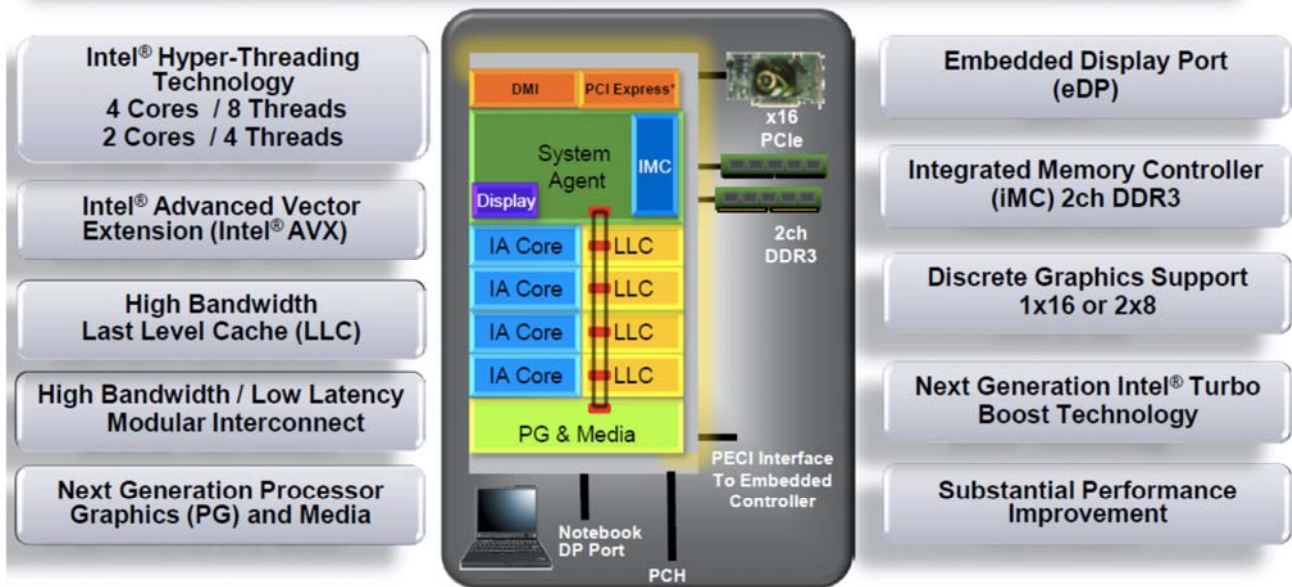
2nd Generation Intel®Core™ Processor Family

System On Chip Integration



Intel® Core™ Microarchitecture

Greater Performance/Lower Power Consumption



Integrates CPU, Graphics Core, Memory Controller and PCI Express on 32nm Process



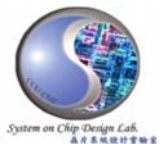
4

Server Architecture Overview

Lect01- 17

T.-F. Chen@NCTU CSIE

Key Architectural Trends



- ❑ Increase performance at 1.6x per year (2X/1.5yr)
 - True from 1985-present
- ❑ Combination of technology and architectural enhancements
 - Technology provides faster transistors ($\propto 1/\text{lithographic feature size}$) and more of them
 - Faster transistors leads to high clock rates
 - More transistors (“Moore’s Law”):
 - ❑ Architectural ideas turn transistors into performance
 - Responsible for about half the yearly performance growth
- ❑ Two key architectural directions
 - Sophisticated memory hierarchies
 - Exploiting instruction level parallelism

Server Architecture Overview

Lect01- 18

T.-F. Chen@NCTU CSIE

Exploiting Instruction Level Parallelism (ILP)

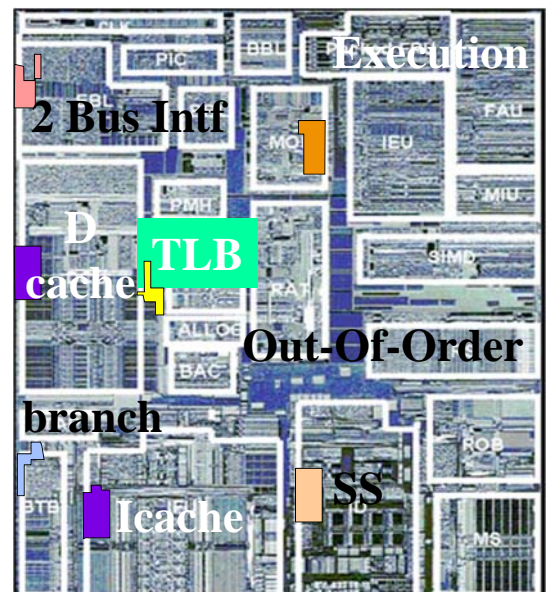


- ❑ ILP is the implicit parallelism among instructions (programmer not aware)
- ❑ Exploited by
 - Overlapping execution in a pipeline
 - Issuing multiple instruction per clock
 - ❑ superscalar: uses dynamic issue decision (HW driven)
 - ❑ VLIW: uses static issue decision (SW driven)
- ❑ 1985: simple microprocessor pipeline (1 instr/clock)
- ❑ 1990: first static multiple issue microprocessors
- ❑ 1995: sophisticated dynamic schemes
 - determine parallelism dynamically
 - execute instructions out-of-order
 - speculative execution depending on branch prediction
- ❑ “Off-the-shelf” ILP techniques yielded 15 year path of 2X performance every 1.5 years => 1000X faster!

Where have all the transistors gone?



- ❑ Superscalar (multiple instructions per clock cycle)
- 3 levels of cache
- Branch prediction (predict outcome of decisions)
- Out-of-order execution (executing instructions in different order than programmer wrote them)



Intel Pentium III
(10M transistors)

Why multicore ?

- ❑ Difficult to make single-core clock frequencies even higher
- ❑ Deeply pipelined circuits:
 - heat problems
 - speed of light problems
 - difficult design and verification
 - large design teams necessary
 - server farms need expensive air-conditioning
- ❑ Many new applications are multithreaded
- ❑ General trend in computer architecture (shift towards more parallelism)

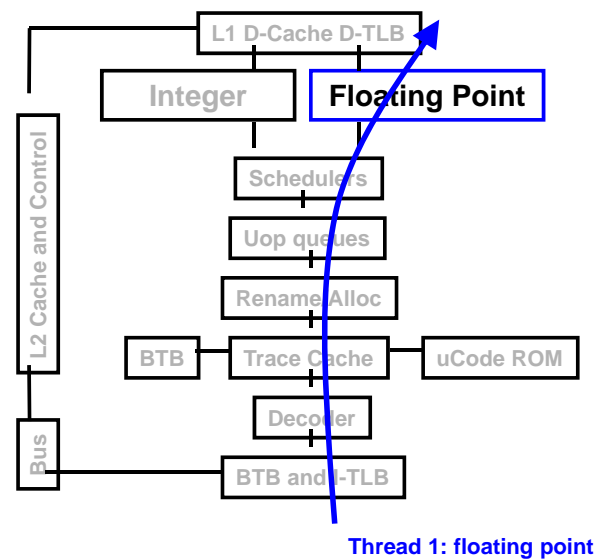


Thread-level parallelism (TLP)

- ❑ This is parallelism on a more coarser scale
- ❑ Server can serve each client in a separate thread (Web server, database server)
- ❑ A computer game can do AI, graphics, and physics in three separate threads
- ❑ Single-core superscalar processors cannot fully exploit TLP
- ❑ Multi-core architectures are the next step in processor evolution: explicitly exploiting TLP

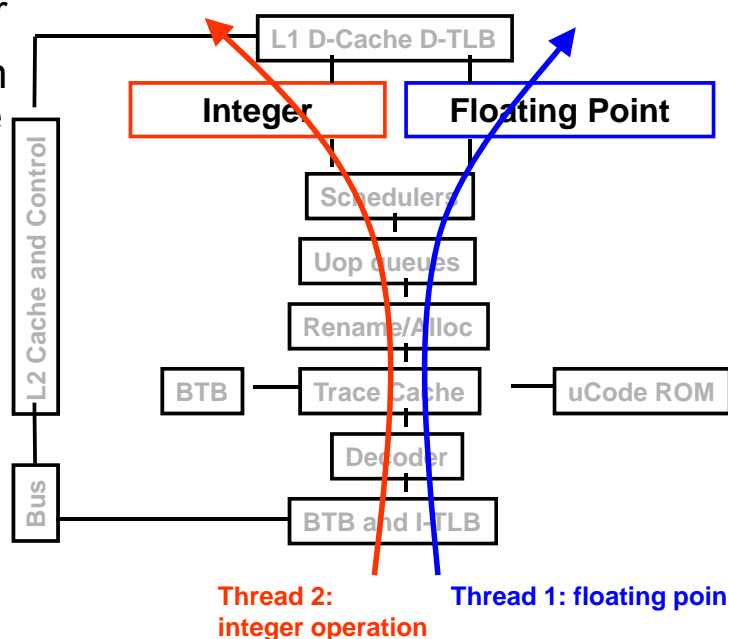
Simultaneous multithreading (SMT)

- ❑ Permits multiple independent threads to execute SIMULTANEOUSLY on the SAME core
- ❑ Weaving together multiple "threads" on the same core
- ❑ Example: if one thread is waiting for a floating point operation to complete, another thread can use the integer units
- ❑ Intel calls them "hyper-threads"

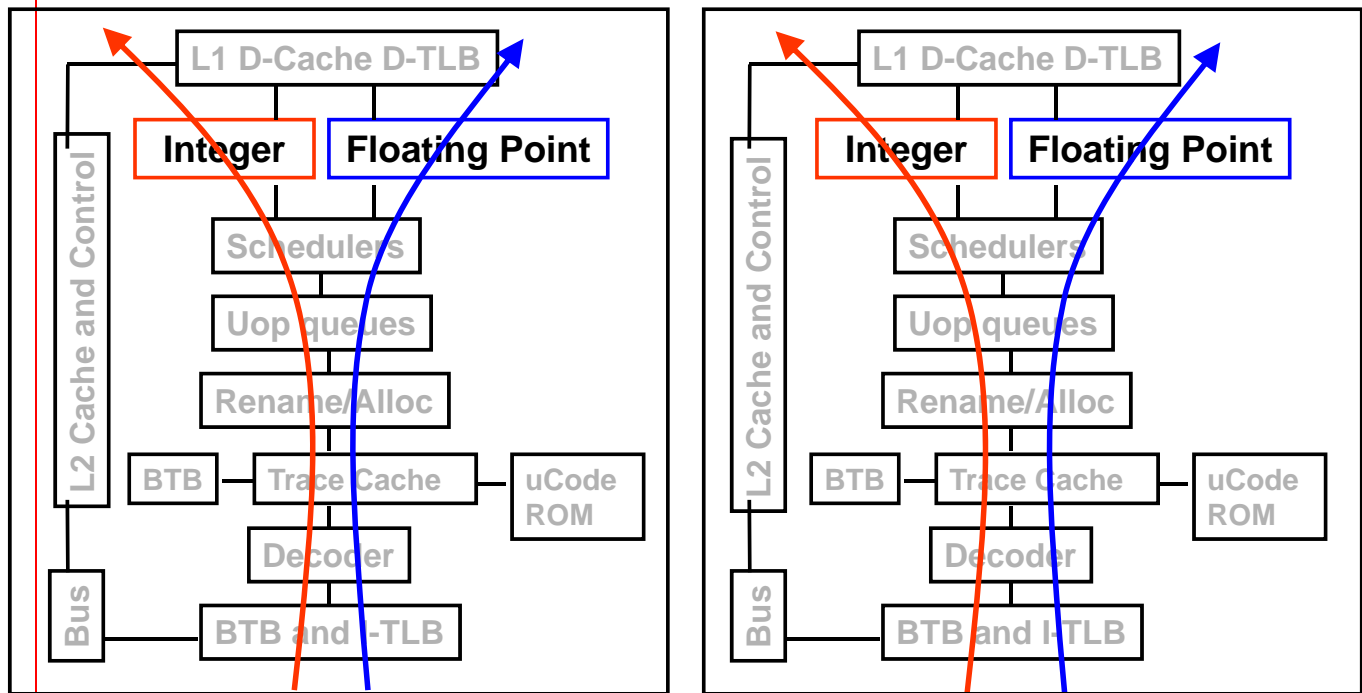


SMT processor: both threads can run concurrently

- ❑ SMT not a "true" parallel processor
- ❑ OS and applications perceive each simultaneous thread as a separate "virtual processor"
- ❑ The chip has only a single copy of each resource
- ❑ Compare to multi-core: each core has its own copy of resources



SMT Dual-core: all four threads can run concurrently



Thread 1 Thread 3

Thread 2 Thread 4

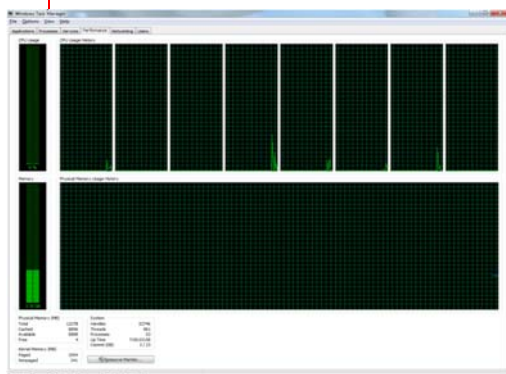
Server Architecture Overview

Lect01- 25

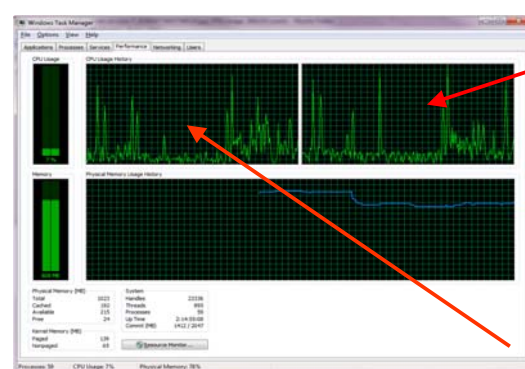
T.-F. Chen@NCTU CSIE

Individual Core Architecture

- ❑ Intel Core Duo uses superscalar cores
- ❑ Intel Core i7 uses simultaneous multi-threading (SMT)
 - Scales up number of threads supported
 - ❑ 4 SMT cores, each supporting 4 threads appears as 16 core (my corei7 has 2 threads per CPU)



Core i7



Core 2 duo

Server Architecture Overview

Lect01- 26

T.-F. Chen@NCTU CSIE

CPU mark Relative to Top 10 Common CPUs

16/March/2012 - Higher results represent better performance



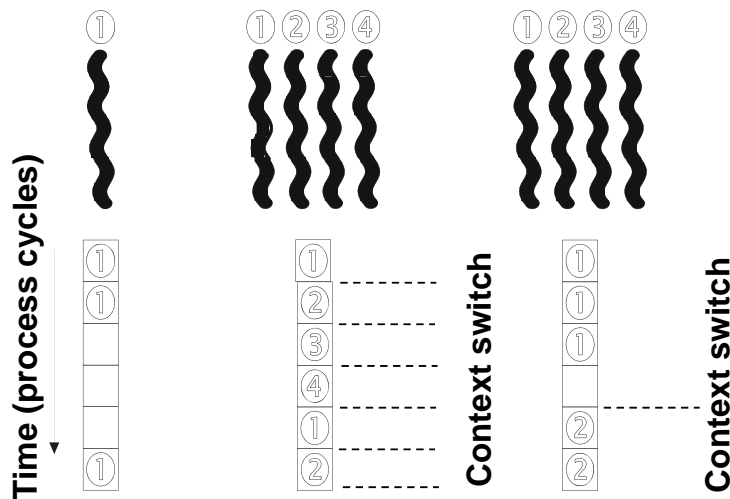
CPU Value (CPU Mark / \$Price)

16/March/2012 - Higher results represent better value



Server Architecture Overview

Multithreading vs Non-Multithreading

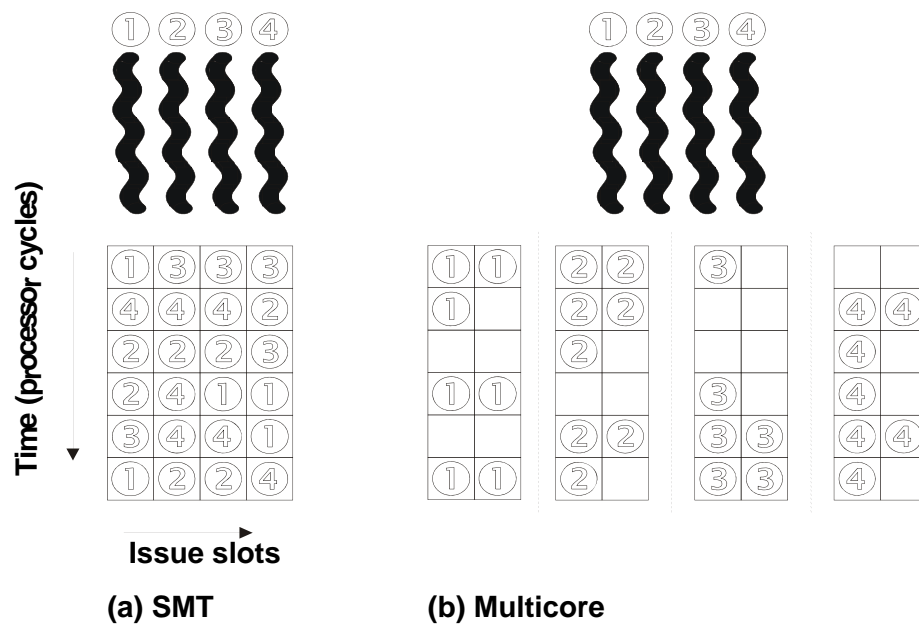
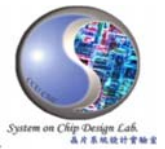


(a) single-threaded scalar

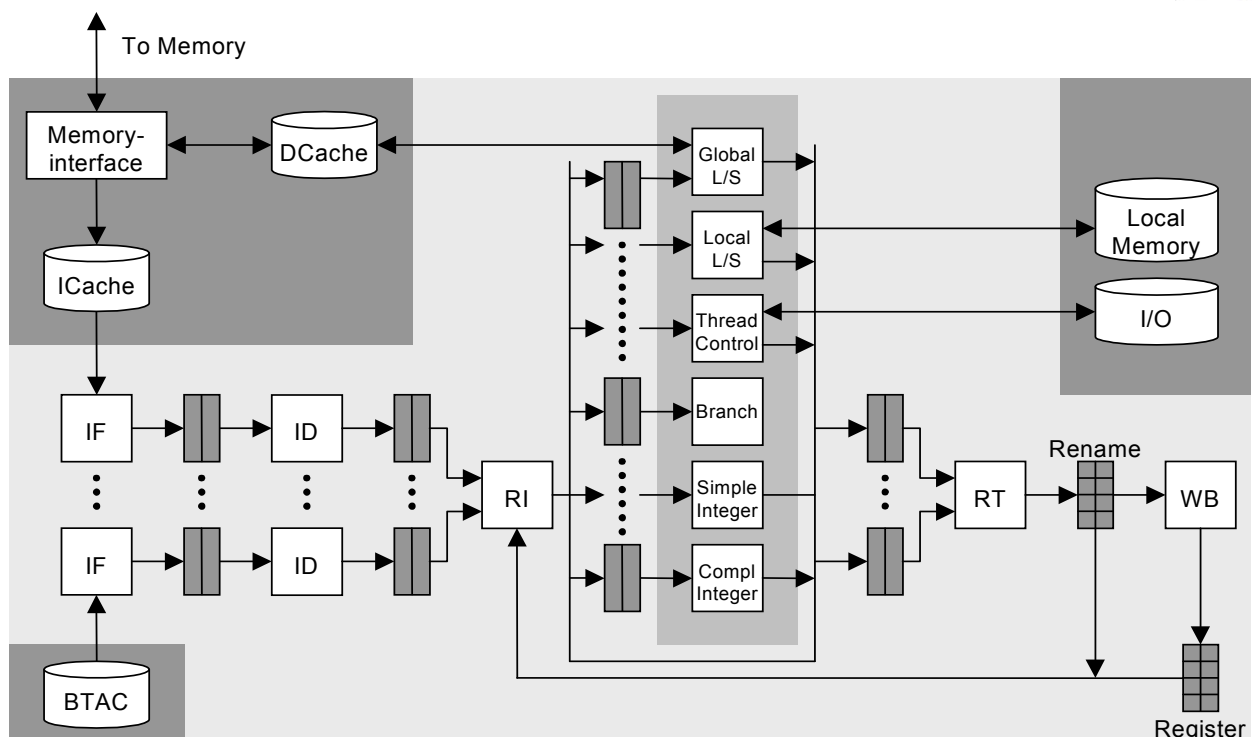
(b) cycle-by-cycle interleaving (fine-grained multithreading)

(c) block interleaving (coarse-grained multithreading)

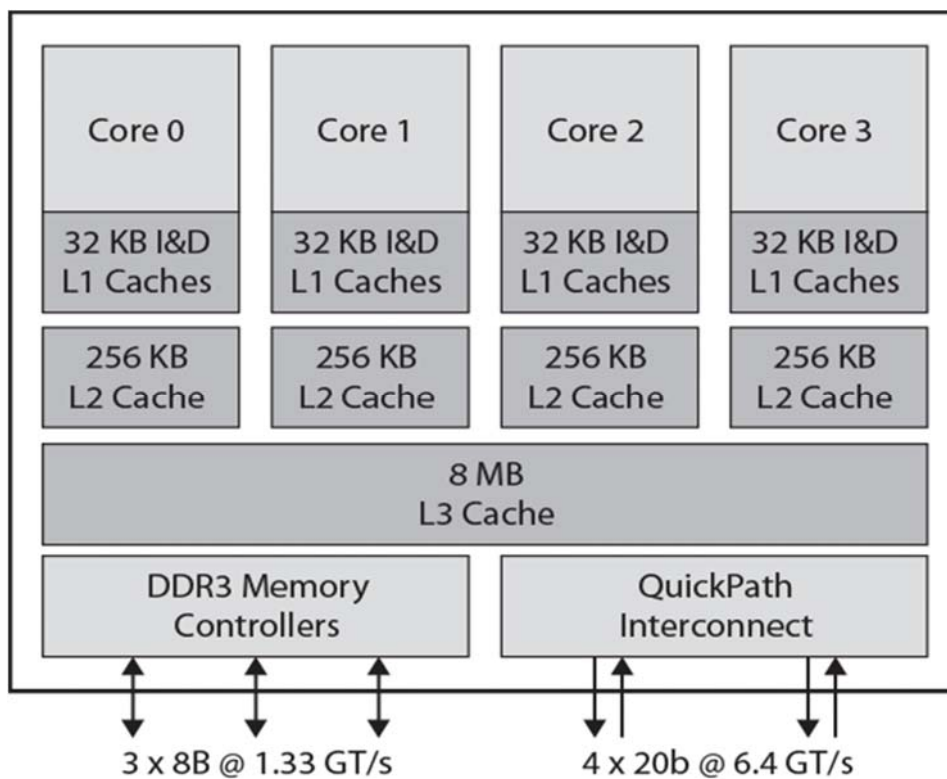
Simultaneous Multithreading (SMT) and Chip Multiprocessors (multicore)



SMT Processr Model



Intel Core i7 Block Diagram

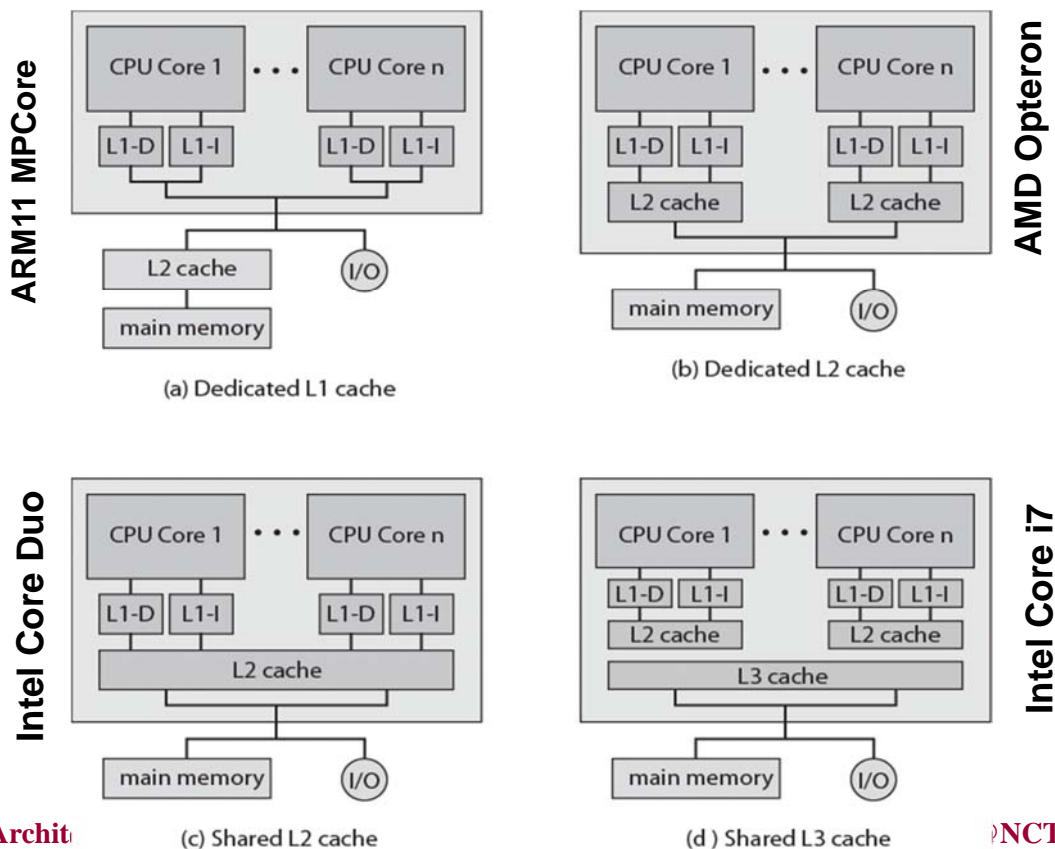


Server Architecture Overview

Lect01- 31

T.-F. Chen@NCTU CSIE

Multicore Organization Alternatives

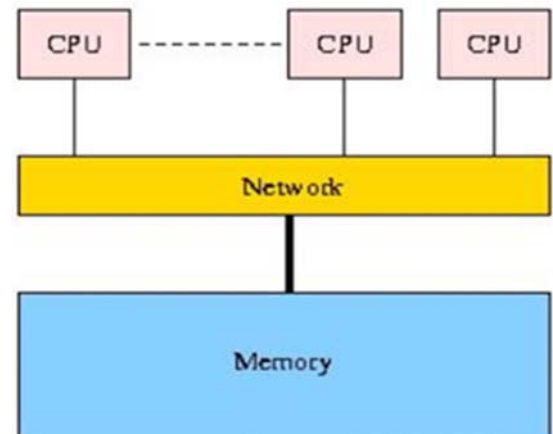


Server Archi

NCTU CSIE

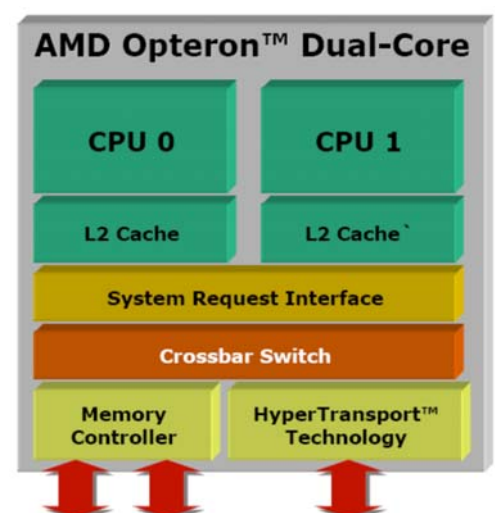
Shared-Memory Multicore Architecture

- ❑ Memory (centralized or distributed) can be directly accessed by different CPUs
- ❑ Communication between programs/threads occurs implicitly via memory instructions (e.g., loads and stores)
- ❑ A natural extension of uni-processor model
 - Shared data are location transparent



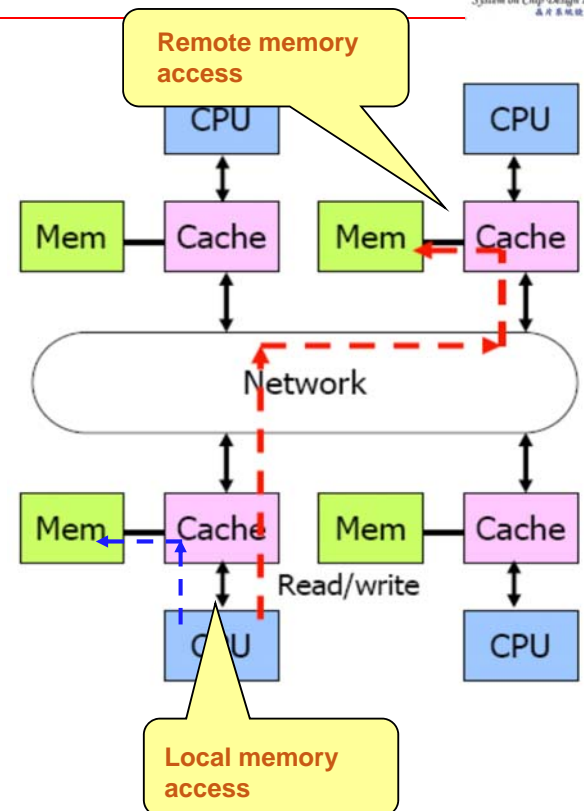
SMP: Symmetric (Shared-Memory) Multiprocessors

- ❑ UMA - Uniform Memory Access
 - Uniform memory access latency from any processor
- ❑ Symmetric (beyond UMA)
 - Equal access to computer's resources
 - Equal PE computing power
- ❑ **Connected by bus or crossbar switch**
- ❑ **Shared-everything architecture**
 - Centralized shared memory, shared I/O, same OSes
- ❑ **Cache coherence protocol is needed if caches exist**
 - Usually implemented by hardware

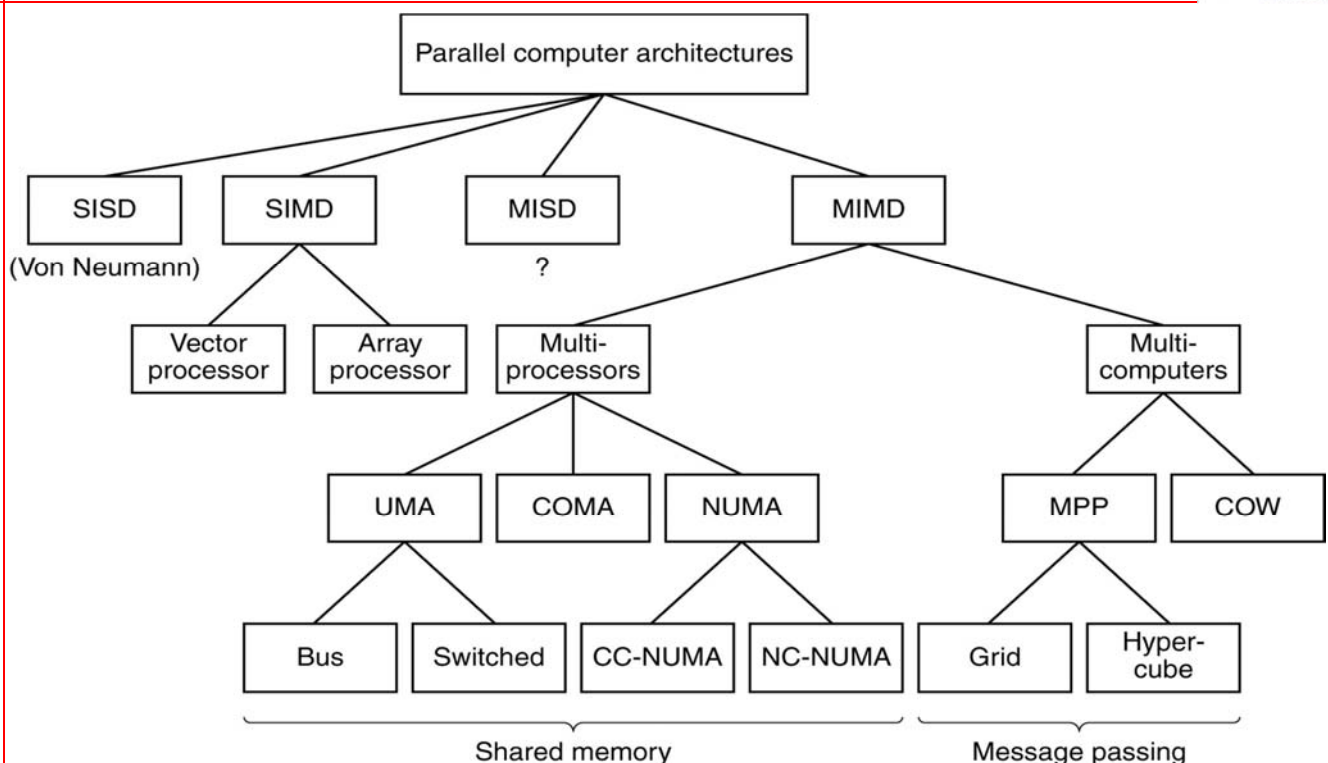


NUMA: non-uniform memory accesses

- ❑ All CPUs can read/write any part of the memory
- ❑ Access time depends on location of a data word in memory
- ❑ Local memory access
 - When local cache misses, and local memory hits
 - Shorter access latency than in UMA
- ❑ Remote memory access
 - Longer access latency than in UMA



Taxonomy of Parallel Computers



Parallel Programming Model

source
http://computing.llnl.gov/tutorials/parallel_comp

Parallel Programming Overview



❑ A parallel programming model is an abstraction above hardware and memory architectures

- It provides an abstracted view to the programmer
- Not necessarily tied to the actual architecture
 - ❑ E.g., message passing programs can run on a shared memory platform

❑ Common parallel programming models:

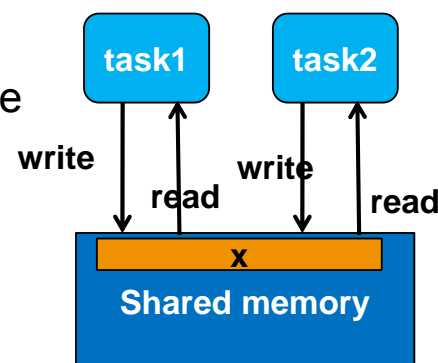
- Shared memory (e.g. pthread, OpenMP)
- Message passing (e.g. MPI)
- Data parallel (e.g. CUDA)

❑ There is no “best” model

- Which model to use depends on what is available and user’s preferences

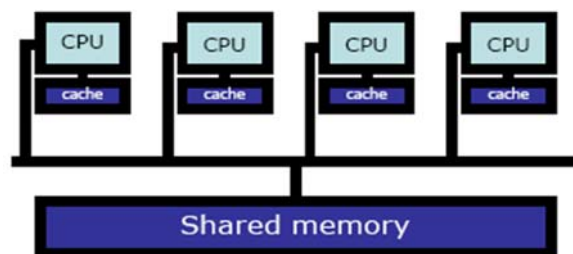
Shared Memory Model

- ❑ Tasks share the same address space
- ❑ Communications can be achieved by reading/writing data from/to the same memory location
- ❑ Advantage: need not to explicitly specify the communication of data between tasks
- ❑ Natural for tasks to share other system resources
 - OS image, kernel functions, and drivers
 - Storages, network, and I/O devices
- ❑ Example: OpenMP, Pthreads, Linux kernel threads



Shared Memory Model – *Threads* (1/2)

- ❑ *Threads* is an implementation of shared-memory programming model
- ❑ Key concepts in threads programming:
 - Creating and managing threads
 - ❑ Maybe explicit or implicit
 - Synchronization with shared memory
 - ❑ signaling (maybe events or condition variables)
 - Algorithm design on shared-memory platform
 - ❑ E.g., locality considerations



Shared Memory Model – *Threads* (2/2)

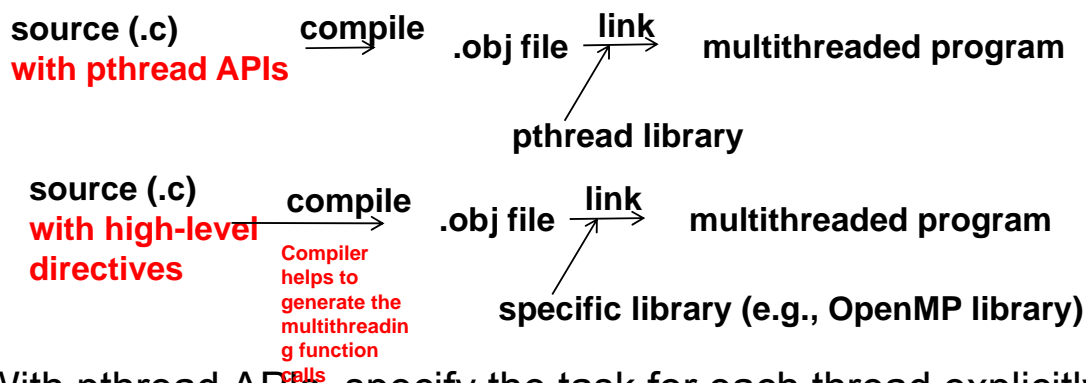
- ❑ A thread's work is described as a subtask
 - A thread executes its own function at the same time with other threads
- ❑ Each thread has local data, but also shares global variables with the other threads
 - So threads communicate with others through global memory
- ❑ Synchronization constructs are required to avoid race conditions
- ❑ Example: POSIX threads (Pthreads)

(source) https://computing.llnl.gov/tutorials/parallel_comp/

Shared Memory Model - Explicit v.s. Implicit Threads Programming

- ❑ Two threads programming styles:
 - **E: Explicit**
 - ❑ User creates threads using threads API
 - ❑ E.g., “Windows threads” and “POSIX Threads”
 - **I: Implicit**
 - ❑ User uses high-level directives to create threads with the help of tool chain
 - ❑ E.g., OpenMP, Intel Thread Building Block (TBB)
- ❑ Examples:
 - Thread creation
 - ❑ **E:** Programmers create threads and manage threads
 - ❑ **I:** Thread pools created and maintained by library
 - Assigning computation
 - ❑ **E:** Programmer inserts work division logic to assign tasks to threads
 - ❑ **I:** Work divided by library or additional *pragma* options
 - Wait for threads to complete
 - ❑ **E:** API call to pause waiting thread and detect thread termination
 - ❑ **I:** Implicit barrier at end of threading constructs

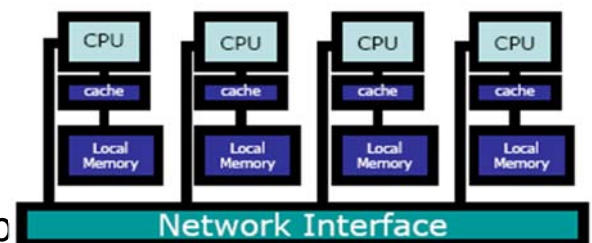
Explicit v.s. Implicit Threads Programming



- ❑ With pthread APIs, specify the task for each thread explicitly
 - Users have to decompose the computation in tasks, assign of tasks to processes (threads) and orchestrate data
- ❑ With OpenMP directives, users can simply use directives to offload the task partition job to tool chain
 - Only have to specify the parallelism in code (with high-level directives) and orchestrate data access, communication and synchronization

Message Passing Model (1/2)

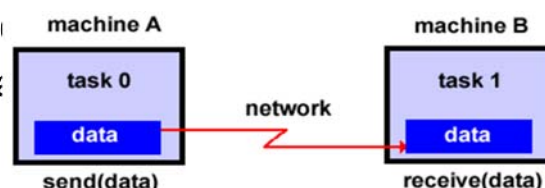
- ❑ In distributed memory platform, *message passing* model is used for communicating and coordinating work among concurrent processes



- ❑ Parallel programs consist of separate processes with their own address space
 - Programmer manages data movement
 - ❑ Programmer manages memory by placing data in a particular process
 - Data sent explicitly between processes

Message Passing Model (2/2)

- ❑ A set of tasks use their own *local memory* during computation
- ❑ Tasks communicate by sending and receiving messages
- ❑ Data transfer usually requires cooperative operations to be performed by each process
 - For example, a send operation must have a matching receive operation
- ❑ Examples:
 - Message Passing Interface (MPI)
 - `msgget()` and `msgsnd()` functions in linux
 - “DMA commands” are used to communicate
 - ❑ `mfc_get()` and `mfc_put()`



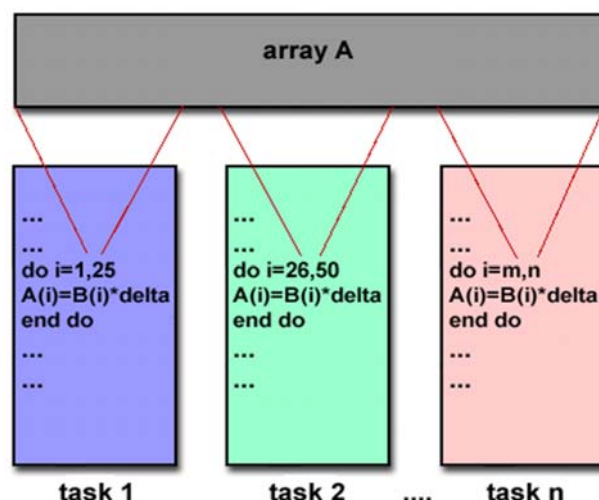
(source)
https://computing.llnl.gov/tutorials/parallel_comp/

Parallel Programming Overview

Data Parallel Model (Data Decomposition)

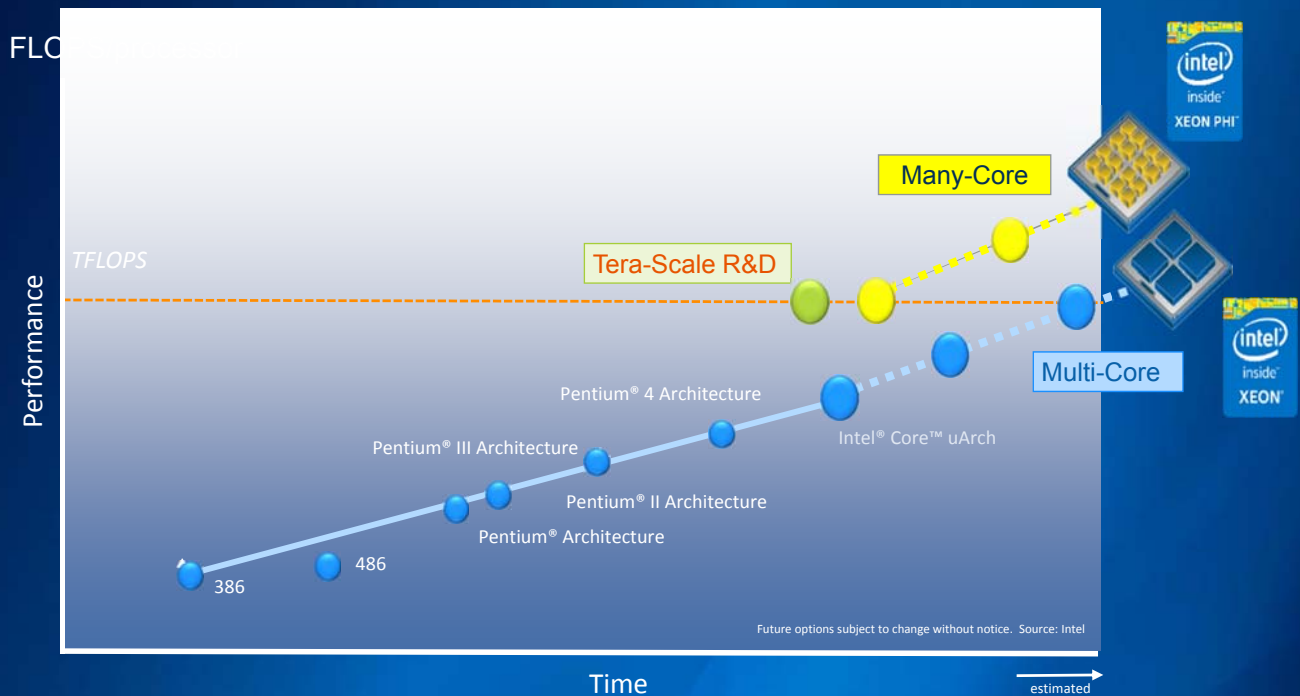
- ❑ Tasks perform the same function on their partition of work
 - Each task works on a different partition of the same data set

- ❑ Example:
 - Loop-level parallelism
 - Data streaming method
 - ❑ E.g., Brook, CUDA



(source) https://computing.llnl.gov/tutorials/parallel_comp/

Increasing Processor Performance Through Many-Core Technologies for Highly Parallel Workloads

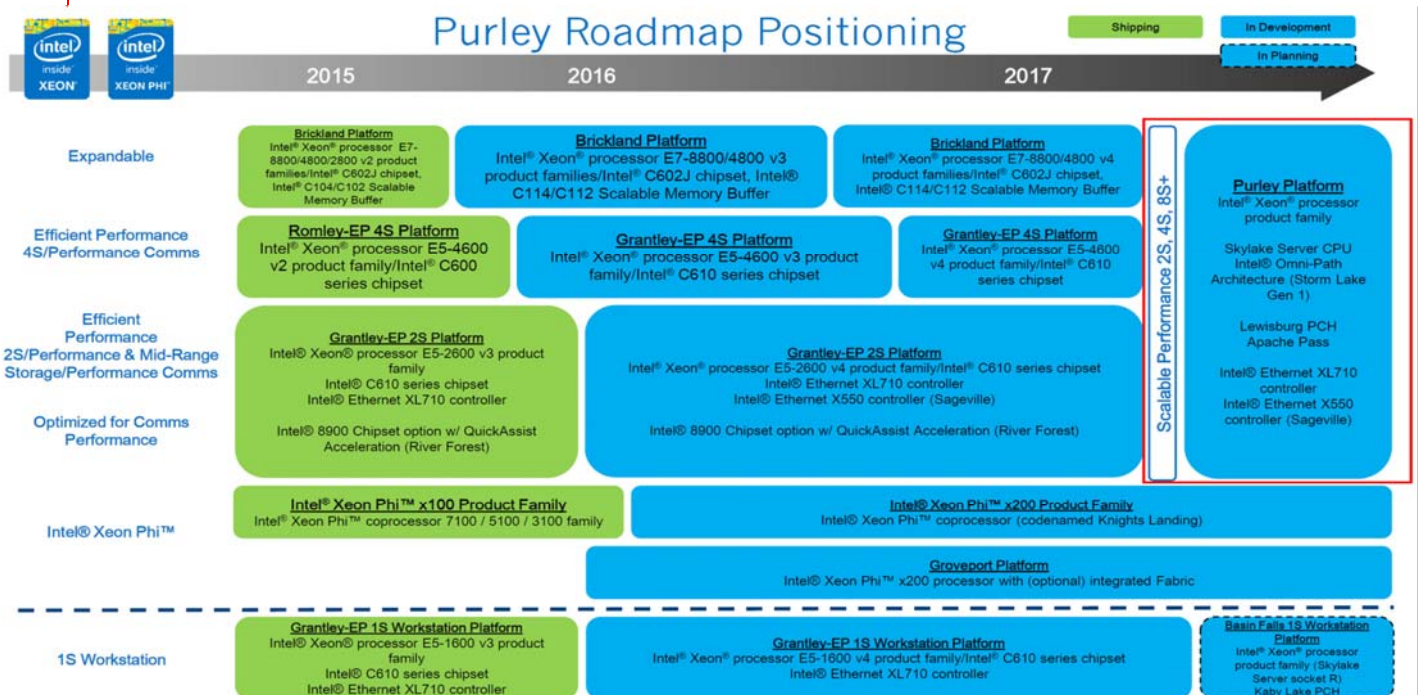
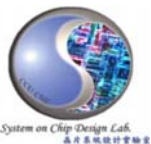


For illustration only, not drawn to scale. All dates, product descriptions, features, availability, and plans are forecasts and subject to change without notice.



47

Intel Server CPU Roadmap

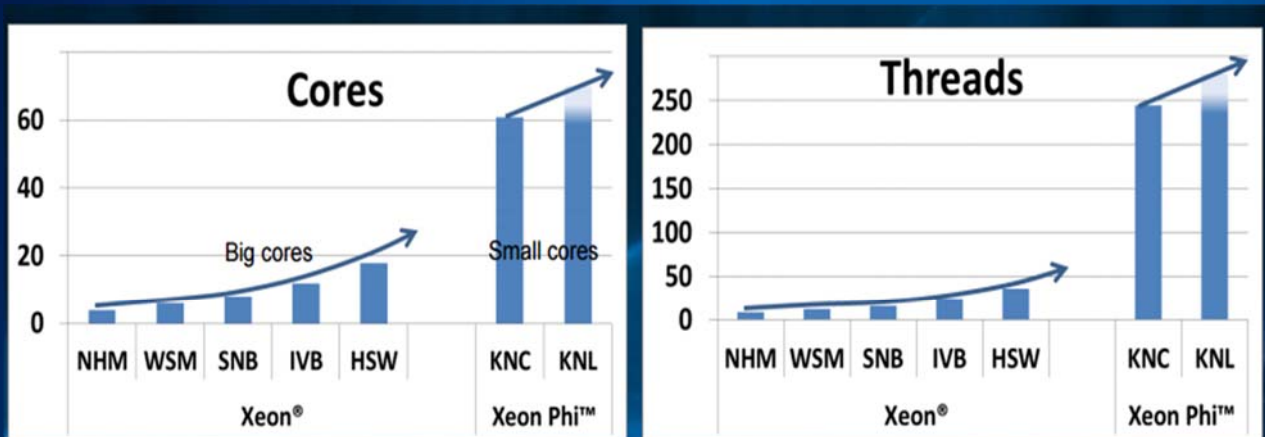


Product placement not representative of final launch date within the specified quarter. For more details refer to ILU and 5 Quarter Roadmap.



4

Trends: Cores and Threads per Chip



Continuous growth in Cores and Threads per socket both for Multi-core and Many-core

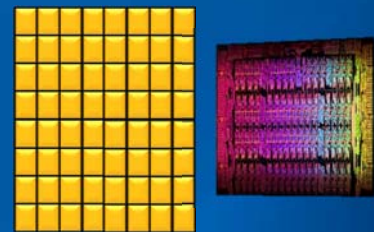


49

„Big Core “ – „Small Core “



*Different Optimization Points
Common Programming Models
and Architectural Elements*



Intel® Xeon® Processor Processor

Simply aggregating more cores generation after generation is not sufficient

Performance per core/thread must increase each generation, be as fast as possible

Power envelopes should stay flat or go down each generation

Balanced platform (Memory, I/O, Compute)

Cores, Threads, Caches, SIMD

Intel® Xeon Phi™ Coprocessor

Optimized for highest compute per watt

Willing to trade performance per core/thread for aggregate performance

Power envelopes should also stay flat or go down every generation

Optimized for highly parallel workloads

Cores, Threads, Caches, SIMD

For illustration only

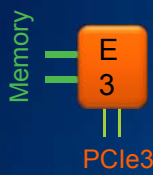


50

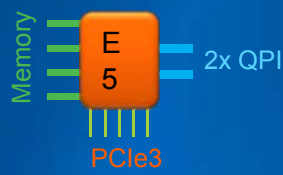


Intel® Xeon® Processors and Platforms

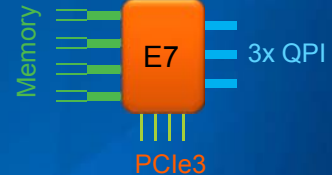
Intel® Xeon® E3



Intel® Xeon® E5



Intel® Xeon® E7



Intel® Xeon® E5-1xxx



CPU/Socket

Intel® Xeon® E3-1xxx

Intel® Xeon® E5-2xxx



QPI

Intel® Xeon® E5-4xxx



Intel® Xeon® E7-xxxx



Intel® Xeon® E7-xxxx



Future options are forecasts and subject to change without notice.



51

Portable & Scalable Parallel Programming

Data-Parallelism

Vectorization
Automatic
Directives/Pragmas
Libraries



Professional Edition

Thread/Task-Parallelism

Multi-Threading
OpenMP*
TBB, Cilk™ Plus
OpenCL
pthreads



Professional Edition

Process-Parallelism

Message Passing
MPI
IP-based



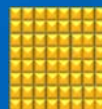
Cluster Edition



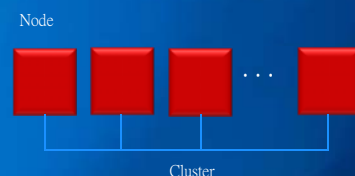
SIMD



Multicore

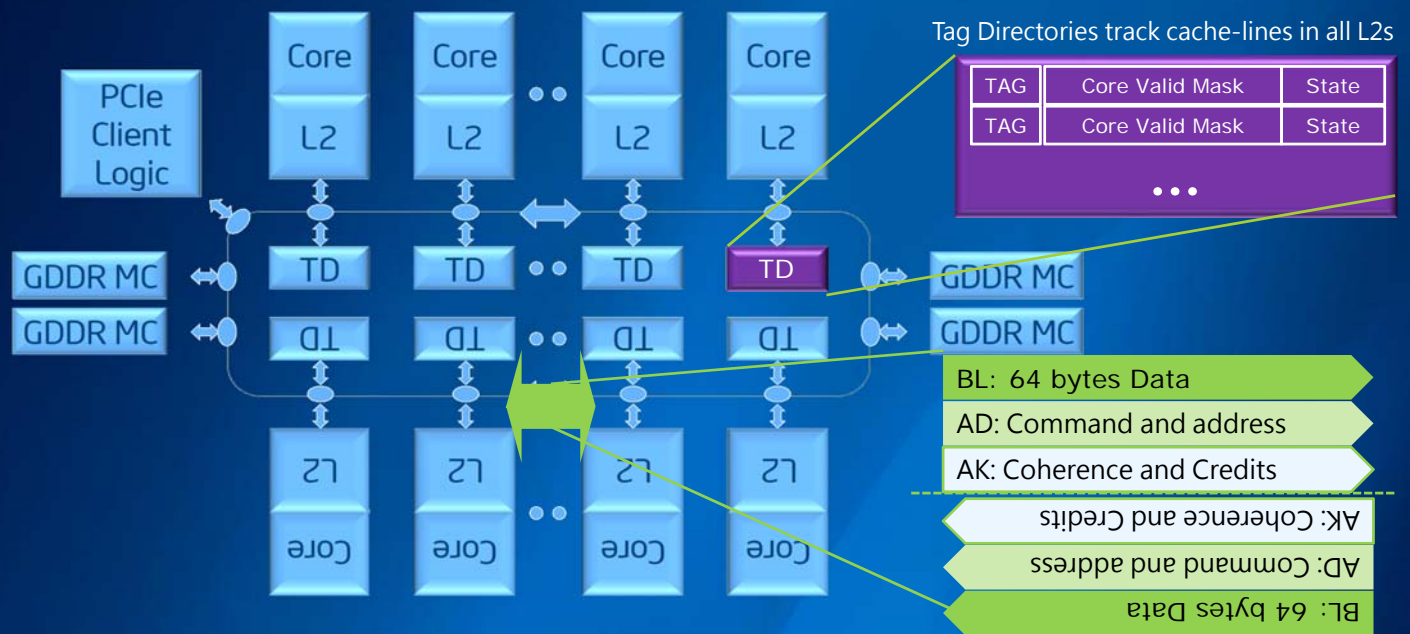


Many-Core



52

Knights Corner Microarchitecture

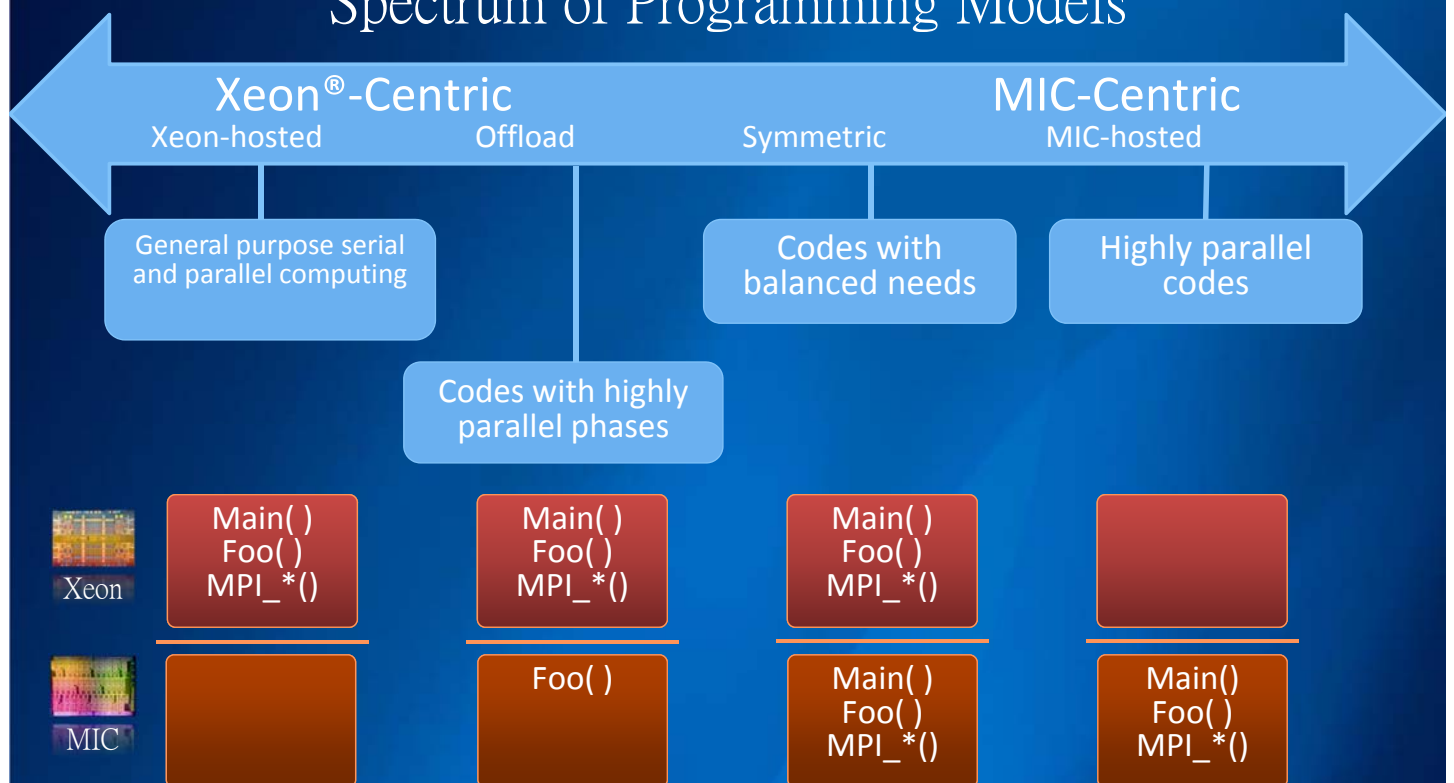


Standard IA Shared Memory Programming



55

Spectrum of Programming Models



All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.



56

New Intel Memory-Storage Hierarchy

Keeping data closer to compute → better data-intensive app performance and energy efficiency

