



Multicore Architecture Concepts

TDDD56 Lecture 1

Christoph Kessler

PELAB / IDA
Linköping university
Sweden

2014



Outline

- Lecture 1: The Multicore Challenge**
 - Why Multicore? Why Now?
Architectural trends
 - Implications for programming
- Lecture 2: Parallel programming with threads and tasks**
- Lecture 3: Shared-memory parallel architecture concepts**
- Lecture 4: Non-blocking synchronization**
- Lecture 5: Design and analysis of parallel algorithms**
- ...

2



Technical Reasons for Multithreading

Single-thread performance is limited by

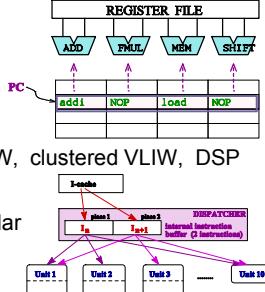
- **ILP Wall**
 - not enough instruction-level parallelism to keep the CPU busy
- **Memory Wall**
 - gap between CPU and memory speed
- **Power Wall**
 - higher clock rate → more power, heat problems

3



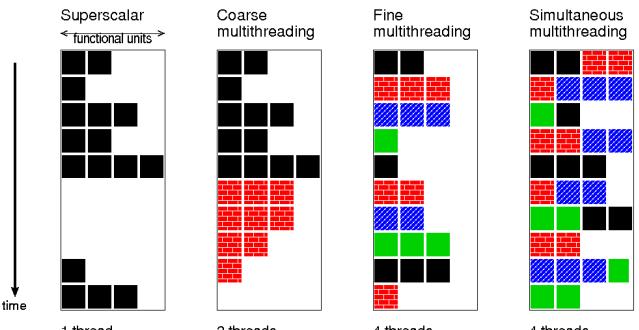
The ILP Wall

- **Instruction Level Parallelism**
 - Multiple functional units
 - Pipelining
 - progr./compiler-managed: VLIW, clustered VLIW, DSP
 - ▶ hard
 - hardware-managed: Superscalar
 - ▶ power + area overhead
- **ILP in applications is limited**
 - typ. <= 3...4 instructions issued concurrently
 - control and data dependences in applications
- **Solution: Multithread the application and the processor**
 - Hardware Multithreading, SMT / Hyperthreading
 - But increases pressure on memory bandwidth





Hardware Multithreading



Superscalar
Functional units
time
1 thread

Coarse multithreading
2 threads

Fine multithreading
4 threads

Simultaneous multithreading
4 threads

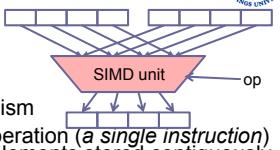
5



SIMD

- **“Single Instruction stream, Multiple Data streams”**
 - single thread of control flow
 - restricted form of data parallelism
 - ▶ apply the same primitive operation (*a single instruction*) in parallel to multiple data elements stored contiguously
 - SIMD units use long “vector registers”
 - ▶ each holding multiple data elements
- Common today
 - MMX, SSE, SSE2, SSE3,...
 - Altivec, VMX, SPU, ...
- Performance boost for operations on shorter data types
- Area- and energy-efficient
- Code to be rewritten (SIMDized) by programmer or compiler
- Does not help (much) for memory bandwidth

op



6

The Memory Wall

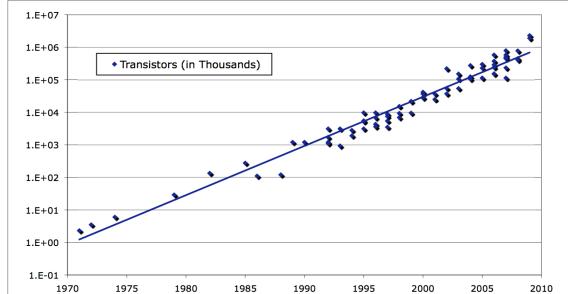
- Performance gap CPU – Memory
- Memory hierarchy
- Increasing cache sizes shows diminishing returns
 - Costs power and chip area
 - GPUs spend the area instead on many simple cores with little memory
 - Relies on good data locality in the application
- What if there is no / little data locality?
 - Irregular applications, e.g. sorting, searching, optimization...
- Solution: Spread out / overlap memory access delay
 - Programmer/Compiler: Prefetching, on-chip pipelining, SW-managed on-chip buffers
 - Generally: Hardware multithreading, again!



7

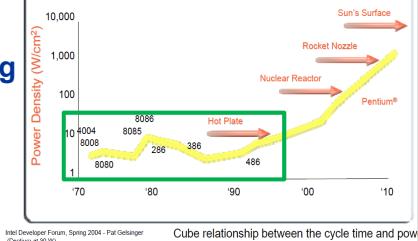
Towards the Power Wall ...

Moore's law: Exponential increase in transistor density



Data from Kunle Olukotun, Lance Hammond, Herb Sutter,
Burton Smith, Chris Batten, and Krste Asanovic

The Era of Dennard Scaling



Intel Developer Forum, Spring 2004 - Pat Gelsinger
(Power on 0.9V)

Moore's Law since 1965

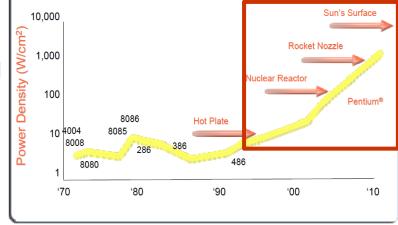
#transistors/mm² doubles every 18..24 months

Dennard 1974:

- With decreasing transistor size, **power density** (Watt / mm²) remains constant
- Due to linear reduction in both voltage and current
- Halving transistor size → power density decreased by 4x

The End of Dennard Scaling

Moore's Law will continue for the foreseeable future...



Intel Developer Forum, Spring 2004 - Pat Gelsinger
(Power on 0.9V)

But since late '90s:

- Static power losses (leakage currents, ...) are increasing
- Voltage cannot be decreased at same pace
- Clock frequency cannot be increased (stuck at ~3 GHz)

The end of Dennard scaling ~2000

- Power density now *increases* with Moore's Law
- "thermal runaway", "dark silicon problem"

10

The Power Issue



- Power = Static (leakage) power + Dynamic (switching) power
- Dynamic power ~ Voltage² * Clock frequency where Clock frequency approx. ~ voltage
→ Dynamic power ~ Frequency³
- Total power ~ #processors

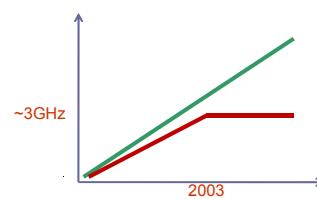
Processor architecture	#cores	Voltage	Frequency	Performance	Power	Power efficiency [Gflops/W]
Classical superscalar	1x	1x	1x	1x	1x	1x
"Faster" superscalar	1x	1.5x	1.5x	1.5x	3.3x	0.45x
Multi-core	2x	0.75x	0.75x	1.5x	0.8x	1.88x

Source: J. Dongara, 2009

→ Preferable to use multiple slower processors than one superfast processor
... PROVIDED THAT the application can be parallelized efficiently!

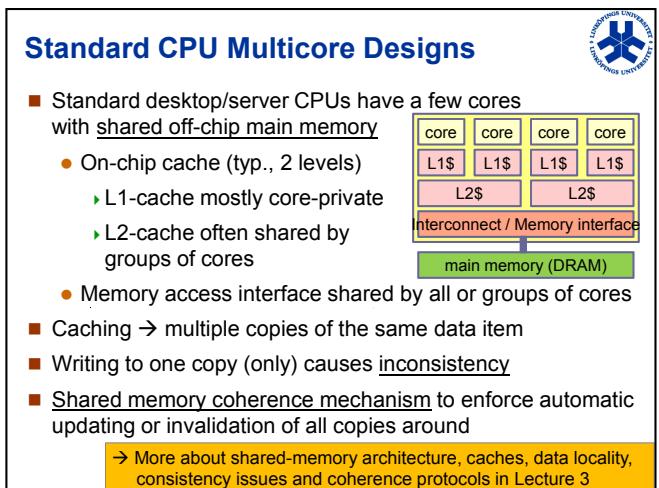
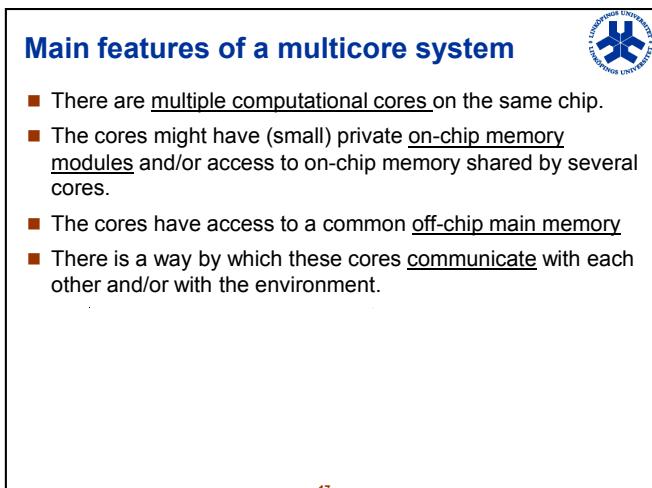
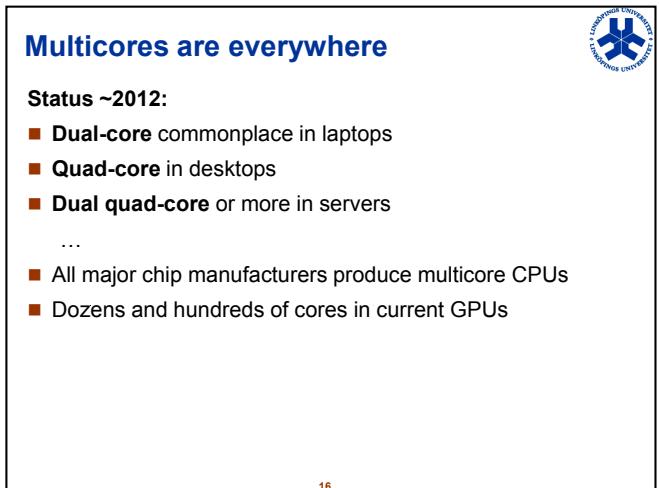
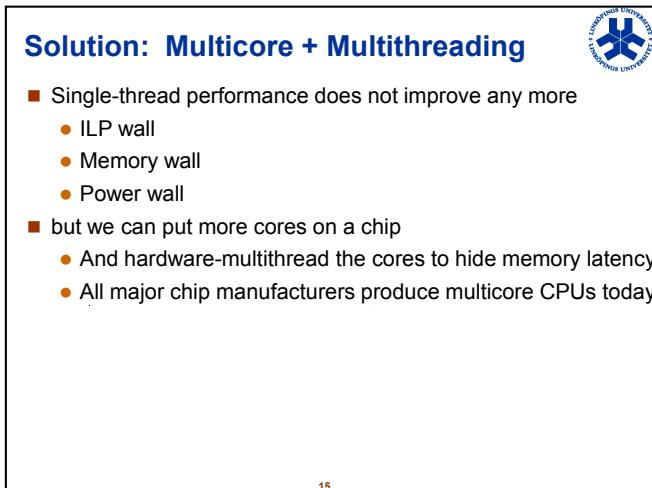
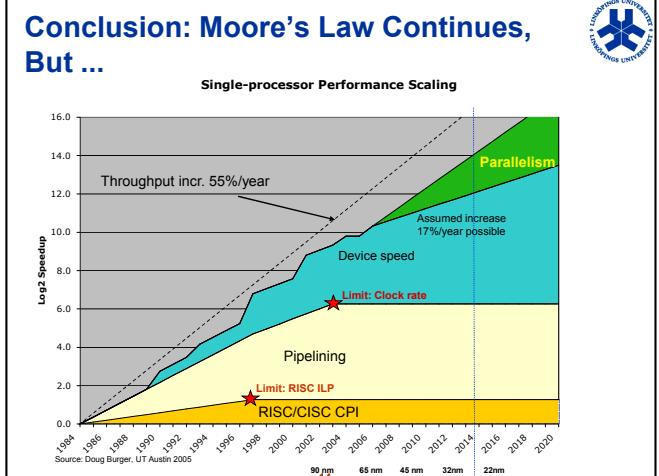
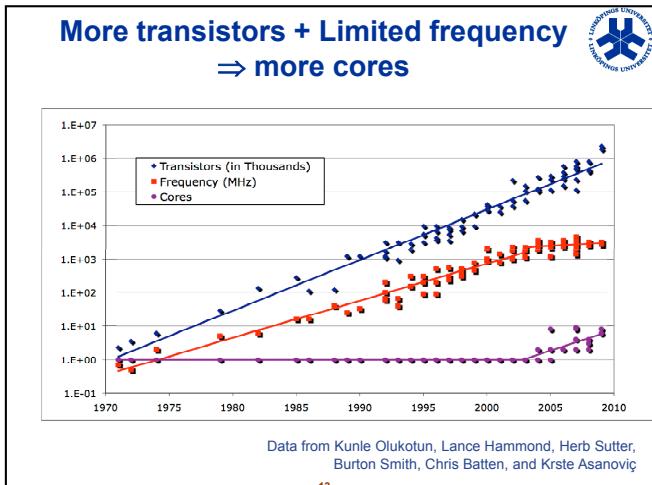
11

Moore's Law vs. Clock Frequency

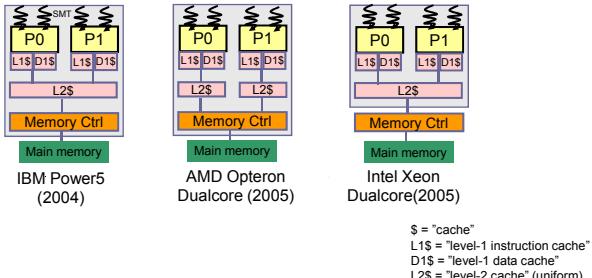


- #Transistors / mm² growing exponentially according to Moore's Law
- Clock speed flattening out

12

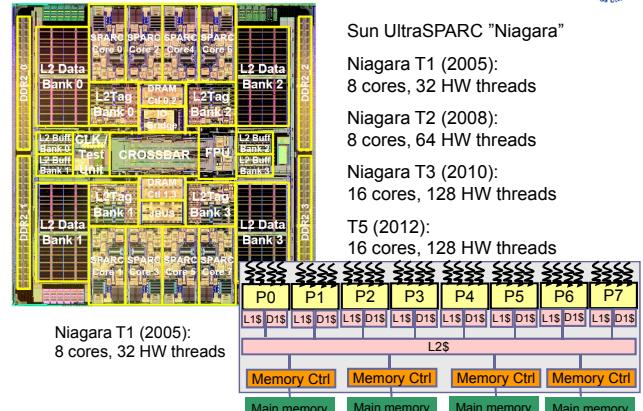


Some early dual-core CPUs (2004/2005)



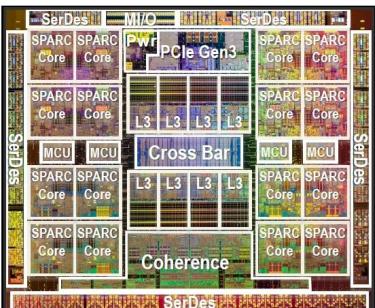
10

SUN/Oracle SPARC T Niagara (8 cores)



Niagara T1 (2005):
8 cores, 32 HW threads

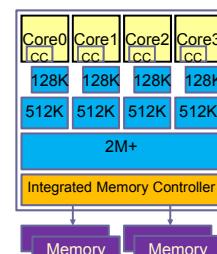
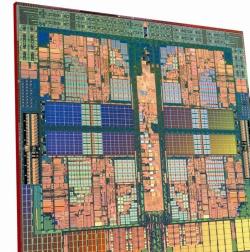
SUN / Oracle SPARC-T5 (2012)



28nm process, 16 cores x 8 HW threads, L3 cache on-chip,
On-die accelerators for common encryption algorithms

21

AMD Opteron with 4 cores...

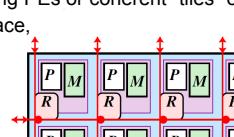


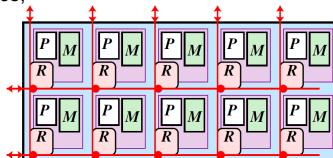
8

... and 6 cores

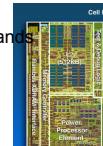
22

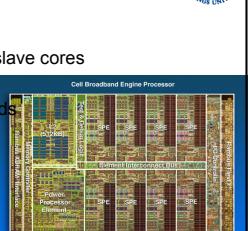
Scaling Up: Network-On-Chip

- Cache-coherent shared memory (hardware-controlled) – does not scale well to many cores
 - power- and area-hungry
 - signal latency across whole chip
 - not well predictable access times
 - Idea: NCC-NUMA – non-cache-coherent,non-uniform memory access
 - Physically distributed on-chip [cache] memory,
 - on-chip network, connecting PEs or coherent "tiles" of PEs
 - global shared address space,
 - but software responsible for maintaining coherence
 - Examples:
 - STI Cell/B.E.,
 - MIT Raw / Tilera TILE64,
 - Intel SCC, Kalray MPPA



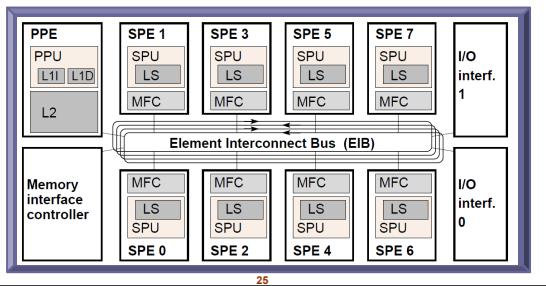
Heterogeneous Multicore

- Sony/Toshiba/IBM Cell/B.E.TM (2006)
 - 1 (SMT) PowerPC core + 8 SIMD-RISC slave cores
 - On-chip local memory (256 KB/slave) instead of cache – explicit DMA commands
 - Multi-level parallelism**
 - ILP, dual issue
 - SIMD 128bit
 - Thread-level parallelism over cores
 - Overlap DMA and computation
 - Hard to program efficiently!
 - Peak performance ca. 220 GFlop/s
 - IBM Blade Servers, LANL RoadRunner Sony PlayStation 3TM Software simulator




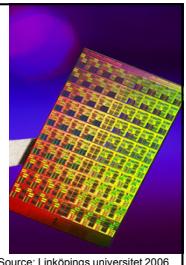
Cell/B.E.

- An on-chip network (four parallel unidirectional rings) interconnect the master core, the slave cores and the main memory interface
- LS = local on-chip memory, PPE = master, SPE = slave



Towards Many-Core CPUs...

- Intel, first-generation many-core research processor:
80-core processor Polaris, ~2006,
1 TFLOPs @ 62 watts,
but no software support



- Intel, second-generation many-core research processor:
48-core (in-order x86) SCC
"Single-chip cloud computer", 2010
 - No longer fully cache coherent over the entire chip
 - MPI-like message passing over 2D mesh network on chip



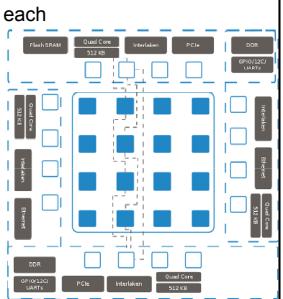
Intel Xeon PHI (since late 2012)

- Earlier called Many-Integrated-Core architecture (MIC)
- 22nm process
- Up to 61 cores, 244 HW threads, 1.2 Tflops peak performance
- Simpler x86 (Pentium) cores (x 4 HW threads), with 512 bit wide SIMD vector registers
- Can also be used as a coprocessor, instead of a GPU
- <http://intel.com/software/mic>



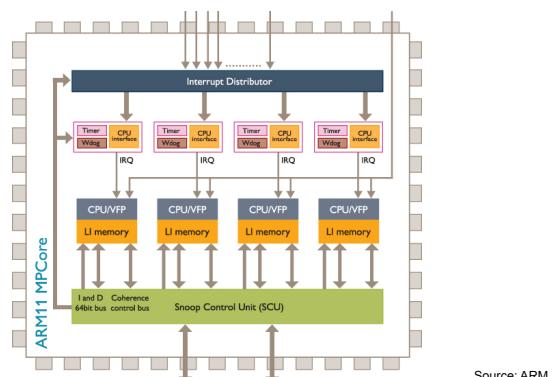
Kalray MPPA-256

- 16 tiles with 16 VLIW compute cores each plus 1 control core per tile
- Message passing network on chip
- Programming models:
 - POSIX processes, threads, sockets etc.
 - Cyclostatic dataflow
- FPU 32 bits / 64 bits IEEE 754
- Virtually unlimited array extension by clustering several chips
- Standard I/Os and interfaces
- Hi-throughput memory controllers Flash / DDR, for external storage of up to 128 GB
- 28 nm CMOS technology
- Low power dissipation, typ. 5 W



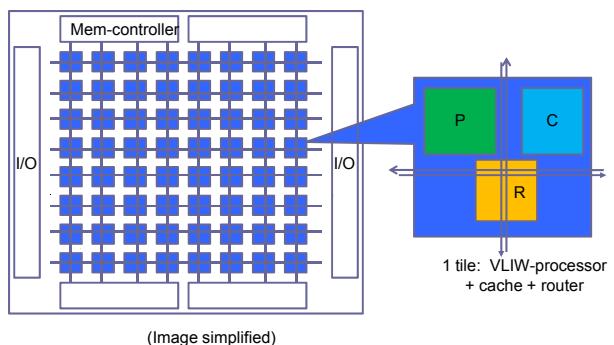
Also Embedded Goes Multicore (1)

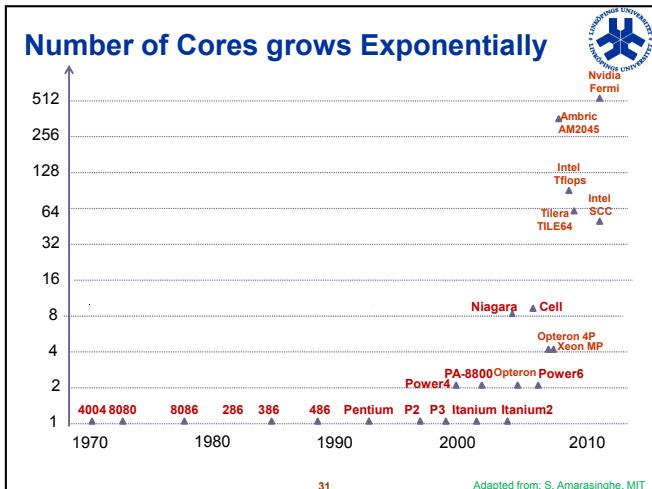
- Example: 4-core ARM-11 MPCore



Also Embedded Goes Multicore (2)

- Tilera TILE64 (2007): 64 cores, 8x8 2D-mesh on-chip network

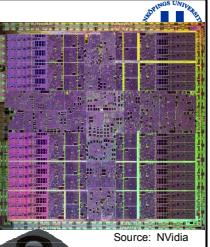




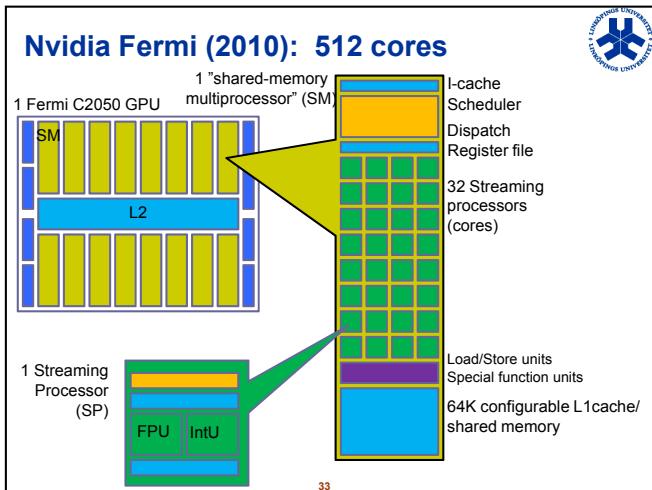
"General-purpose" GPUs

- Main GPU providers for laptop/desktop Nvidia, AMD(ATI), Intel

- Example:**
NVIDIA's 10-series GPU (Tesla, 2008) has 240 cores

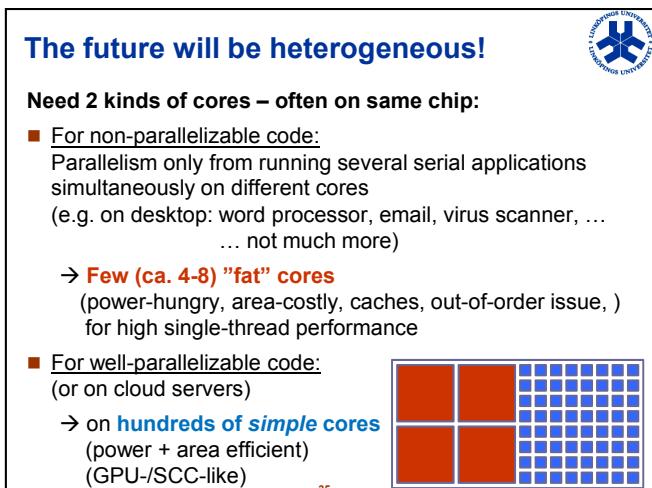


- Each core has a
 - Floating point / integer unit
 - Logic unit
 - Move, compare unit
 - Branch unit
- Cores managed by thread manager
 - Thread manager can spawn and manage 30,000+ threads
 - Zero overhead thread switching



GPU Architecture Paradigm

- Optimized for high throughput
 - In theory, ~10x to ~100x higher throughput than CPU is possible
- Massive hardware-multithreading hides memory access latency
- Massive parallelism
- GPUs are good at data-parallel computations
 - multiple threads executing the same instruction on different data, preferably located adjacently in memory



Heterogeneous / Hybrid Multi-/Manycore

Key concept: Master-slave parallelism, offloading

- General-purpose CPU (master) processor controls execution of slave processors by submitting tasks to them and transferring operand data to the slaves' local memory
 - Master offloads computation to the slaves
- Slaves often optimized for heavy throughput computing
 - Master could do something else while waiting for the result, or switch to a power-saving mode
- Master and slave cores might reside on the same chip (e.g., Cell/B.E.) or on different chips (e.g., most GPU-based systems today)
- Slaves might have access to off-chip main memory (e.g., Cell) or not (e.g., today's GPUs)

36

Heterogeneous / Hybrid Multi-/Manycore System

- Cell/B.E.
- GPU-based system:

Offload heavy computation →
Data transfer ←

Multi-GPU Systems

- Connect one or few general-purpose (CPU) multicore processors with shared off-chip memory to several GPUs
- Increasingly popular in high-performance computing
 - Cost and (quite) energy effective if offloaded computation fits GPU architecture well

38

The Multicore Challenge

- Multi-/Many-Core is upon us
 - Single-thread performance stagnating
 - Dozens, hundreds of cores
 - Heterogeneous
- Utilizing more than one CPU core requires thread-level parallelism
- One of the biggest future software challenges: **exploiting parallelism**
 - Every programmer will eventually have to deal with it
 - All application areas, not only traditional HPC
 - ▶ General-purpose, graphics, games, embedded, DSP
 - Affects HW/SW system architecture, programming languages, algorithms, data structures ...
 - Parallel programming is more error-prone (deadlocks, races, further sources of inefficiencies)
 - ▶ And thus more expensive and time-consuming

39

Can't the compiler fix it for us?

- Automatic parallelization?
 - at compile time:
 - ▶ Requires static analysis – not effective for pointer-based languages
 - ▶ needs programmer hints / rewriting ...
 - ▶ ok for few benign special cases:
 - (Fortran) loop SIMDization,
 - extraction of instruction-level parallelism, ...
 - at run time (e.g. speculative multithreading)
 - ▶ not scalable
- More about parallelizing compilers in Lecture 8

40

And worse yet,

- A lot of variations/choices in hardware
 - Many will have performance implications
 - No standard parallel programming model
 - ▶ portability issue
- Understanding the hardware will make it easier to make programs get high performance
 - Performance-aware programming gets more important also for single-threaded code
 - Adaptation leads to portability issue again
- How to write future-proof parallel programs?

41

The Multicore Challenge (cont.)

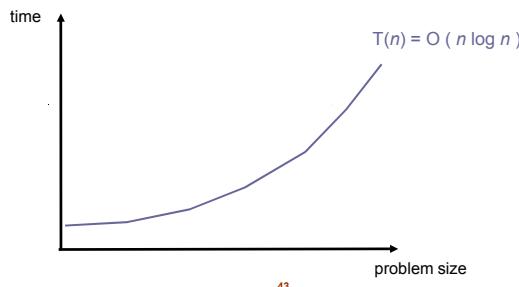
- **Bad news 1:**
Many programmers (also less skilled ones) need to use parallel programming in the future
- **Bad news 2:**
There will be no single uniform parallel programming model as we were used to in the old sequential times
→ Several competing general-purpose and domain-specific languages and their concepts will co-exist

42

What we learned in the past...

■ Sequential von-Neumann model

- programming, algorithms, data structures, complexity
- Sequential / few-threaded languages: C/C++, Java, ... not designed for exploiting massive parallelism



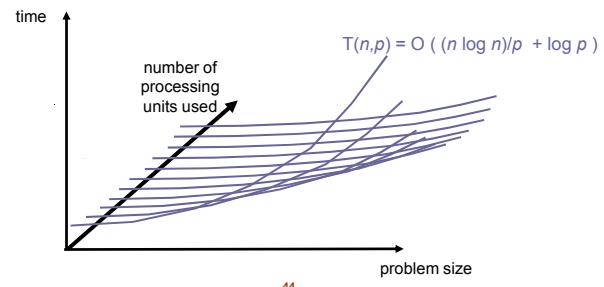
43



... and what we need in the future

■ Parallel programming!

- Parallel algorithms and data structures
- Analysis / cost model: parallel time, work, cost; scalability;
- Performance-awareness: data locality, load balancing, communication



44



Which Programmers?

■ Foreseen: 2 types of programmers

- "Joe" programmers (Matlab, Java, Script, ...): domain experts, goal is programmer productivity
- "Hero" programmers (C/C++): platform experts, goal is performance.



45

Some Statements about Multicore

- "We are dedicating all of our future product development to multicore designs. ... This is a sea change in computing"*
- Paul Otellini, President, Intel (2005)
- "Multicore: This is the one which will have the biggest impact on us. We have never had a problem to solve like this. A breakthrough is needed in how applications are done on multicore devices."*
- Bill Gates, Microsoft
- "When we start talking about parallelism and ease of use of truly parallel computers, we're talking about a problem that's as hard as any that computer science has faced. ... I would be panicked if I were in industry."*
- John Hennessy, President of Stanford University

46



Questions?

