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

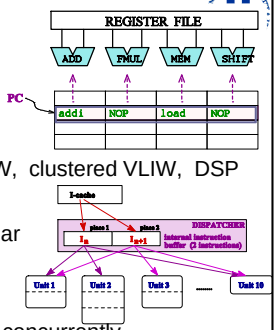
…

Technical Reasons for Multithreading

Single-thread performance is limited by

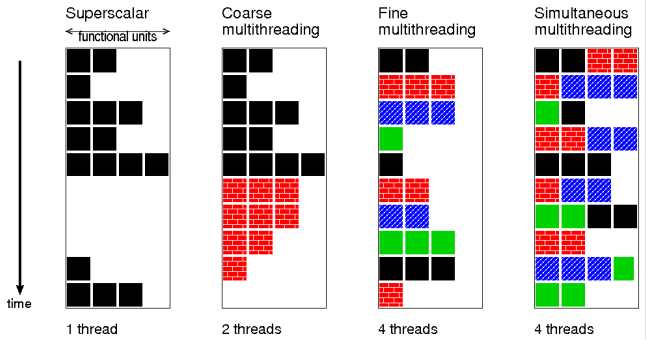
- **ILP Wall**
  - not enough instruction-level parallelism to keep the CPU busy
- **Memory Wall**
  - gap between CPU and memory speed
- **Power Wall**
  - higher clock rate → more power, heat problems

The ILP Wall

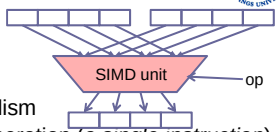


- **Instruction Level Parallelism**
  - Multiple functional units
  - Pipelining
  - progr./compiler-managed: VLIW, clustered VLIW, DSP
    - hard
  - hardware-managed: Superscalar
    - power + area overhead
- **ILP in applications is limited**
  - typ. <= 3...4 instructions issued concurrently
  - 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



SIMD



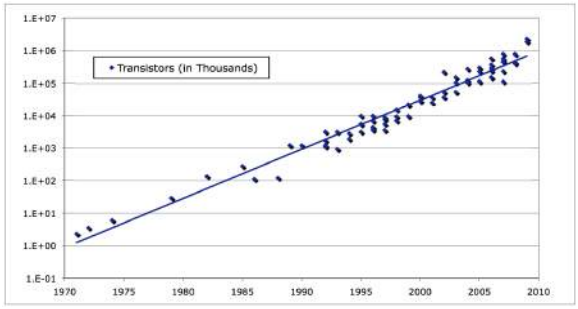
- **"Single Instruction stream, Multiple Data streams"**
  - single thread of control flow
  - restricted form of data parallelism
    - apply the same primitive operation (*a single instruction*) in parallel to multiple data elements stored contiguously
  - SIMD units use long "vector registers"
    - 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

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!

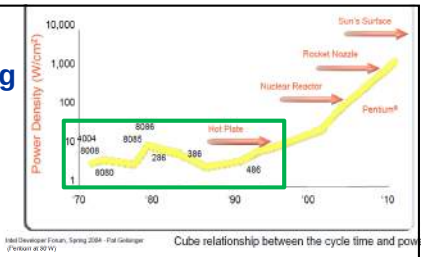
Towards the Power Wall …

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

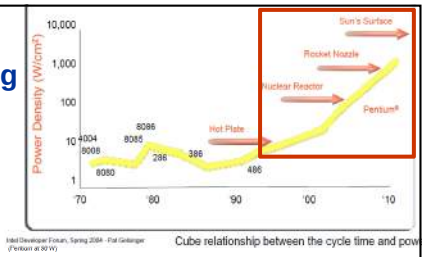


Cube relationship between the cycle time and pow

- **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…



Cube relationship between the cycle time and pow

- **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"

The Power Issue

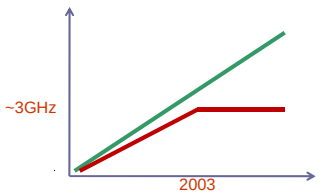
- Power = Static (leakage) power + Dynamic (switching) power
- Dynamic power ~ Voltage² * Clock frequency
  where Clock frequency approx. ~ voltage
  → Dynamic power ~ Frequency³
- Total power ~ #processors

Processor architecture	#cores	Volt-age	Fre-quency	Perfor-mance	Power	Power efficien-cy [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

→ Preferable to use multiple slower processors than one superfast processor

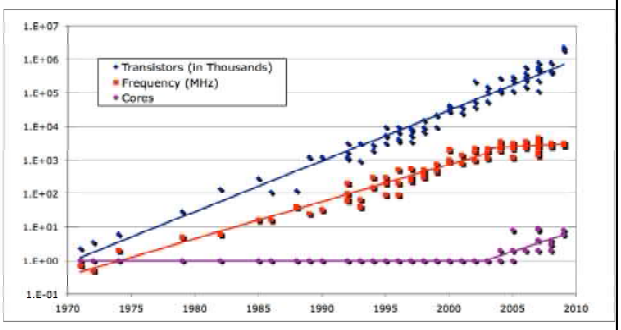
... PROVIDED THAT the application can be parallelized efficiently!

Moore's Law vs. Clock Frequency



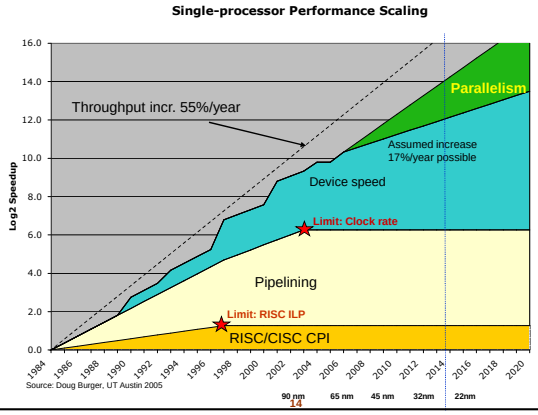
- **#Transistors / mm² still growing exponentially according to Moore's Law**
- **Clock speed flattening out**

More transistors + Limited frequency ⇒ more cores



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

Conclusion: Moore's Law Continues, But ...



Solution: Multicore + Multithreading

- Single-thread performance does not improve any more
  - ILP wall
  - Memory wall
  - Power wall
- but we can put more cores on a chip
  - And hardware-multithread the cores to hide memory latency
  - All major chip manufacturers produce multicore CPUs today

Multicores are everywhere

Status ~2012:

- **Dual-core** commonplace in laptops
- **Quad-core** in desktops
- **Dual quad-core** or more in servers

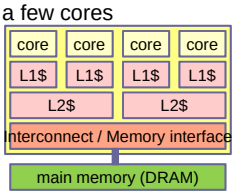
  …
- All major chip manufacturers produce multicore CPUs
- Dozens and hundreds of cores in current GPUs

Main features of a multicore system

- There are multiple computational cores on the same chip.
- The cores might have (small) private on-chip memory modules and/or access to on-chip memory shared by several cores.
- The cores have access to a common off-chip main memory
- There is a way by which these cores communicate with each other and/or with the environment.

Standard CPU Multicore Designs

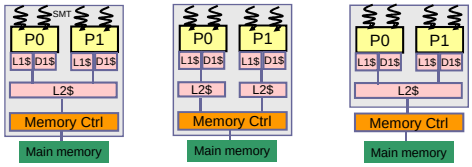
- Standard desktop/server CPUs have a few cores with shared off-chip main memory
  - On-chip cache (typ., 2 levels)
    - L1-cache mostly core-private
    - L2-cache often shared by groups of cores
  - Memory access interface shared by all or groups of cores



- Caching → multiple copies of the same data item
- Writing to one copy (only) causes inconsistency
- Shared memory coherence mechanism to enforce automatic updating or invalidation of all copies around

→ More about shared-memory architecture, caches, data locality, consistency issues and coherence protocols in Lecture 3

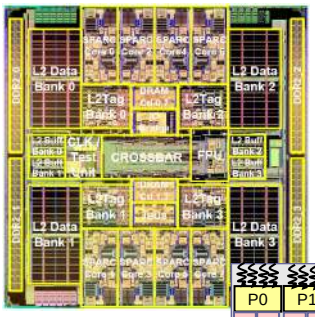
Some early dual-core CPUs (2004/2005)



IBM Power5 (2004) | AMD Opteron Dualcore (2005) | Intel Xeon Dualcore(2005)

$ = "cache"
L1$ = "level-1 instruction cache"
D1$ = "level-1 data cache"
L2$ = "level-2 cache" (uniform)

SUN/Oracle SPARC T Niagara (8 cores)



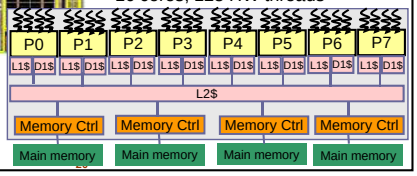
Sun UltraSPARC "Niagara"

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

Niagara T2 (2008):
8 cores, 64 HW threads

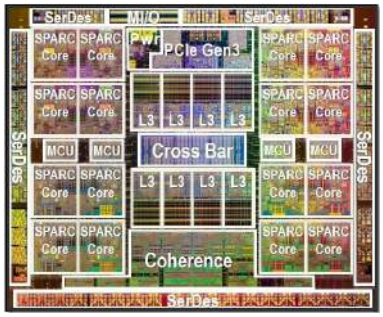
Niagara T3 (2010):
16 cores, 128 HW threads

T5 (2012):
16 cores, 128 HW threads



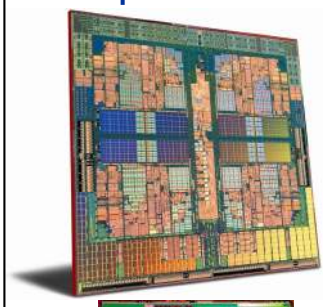
Niagara T1 (2005):
8 cores, 32 HW threads

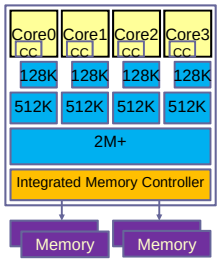
SUN / Oracle SPARC-T5 (2012)



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

AMD Opteron with 4 cores…



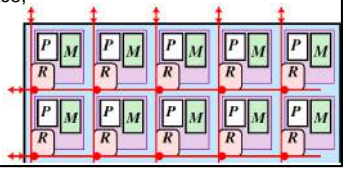


Source: AMD

… and 6 cores

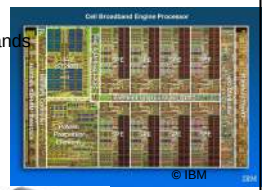
Scaling Up: Network-On-Chip

- 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



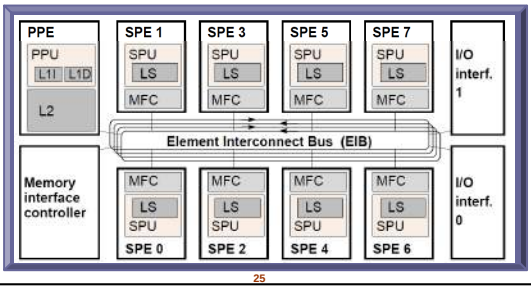
Heterogeneous Multicore Example: Cell/B.E.

- Sony/Toshiba/IBM **Cell/B.E.**™ (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 3™
  Software simulator



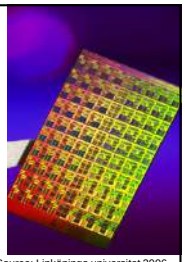
Cell/B.E.

- 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



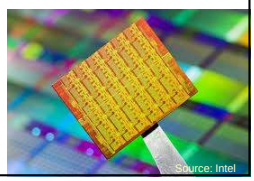
Towards Many-Core CPUs...

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



Source: Linköpings universitet 2006

- 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



Source: Intel

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

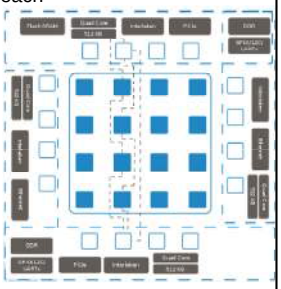
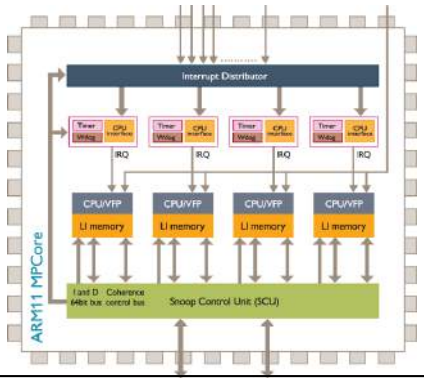


Image source: Kalray

Also Embedded Goes Multicore (1)

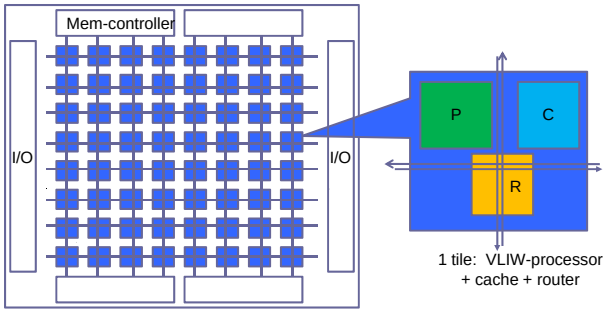
- Example: 4-core ARM-11 MPCore



Source: ARM

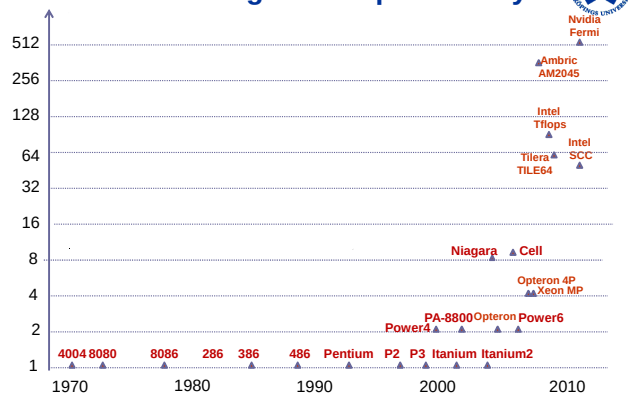
Also Embedded Goes Multicore (2)

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



(Image simplified)

Number of Cores grows Exponentially

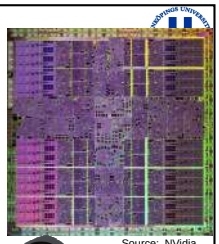


Adapted from: S. Amarasinghe, MIT

31

”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

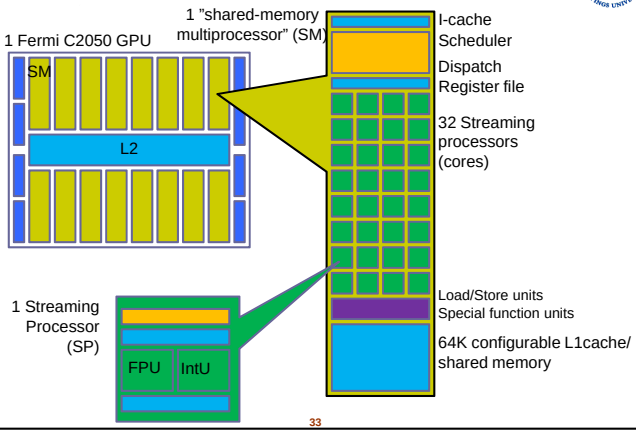


Source: NVidia

Nvidia Tesla C1060:
933 GFlops

32

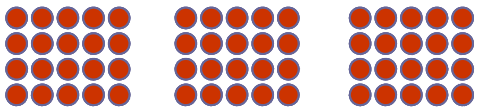
Nvidia Fermi (2010): 512 cores



33

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

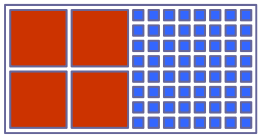


34

The future will be heterogeneous!

Need 2 kinds of cores – often on same chip:

- For non-parallelizable code:
  Parallelism only from running several serial applications simultaneously on different cores
  (e.g. on desktop: word processor, email, virus scanner, …
  … not much more)
  → **Few (ca. 4-8) ”fat” cores**
  (power-hungry, area-costly, caches, out-of-order issue,)
  for high single-thread performance
- For well-parallelizable code:
  (or on cloud servers)
  → on **hundreds of *simple* cores**
  (power + area efficient)
  (GPU-/SCC-like)



35

Heterogeneous / Hybrid Multi-/Manycore

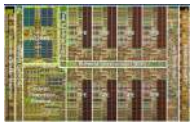
Key concept: Master-slave parallelism, offloading

- General-purpose CPU (master) processor controls execution of slave processors by submitting tasks to them and transfering 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)

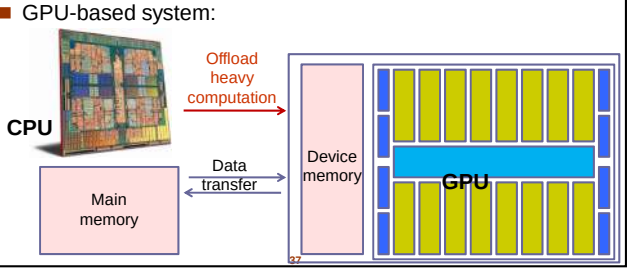
36

Heterogeneous / Hybrid Multi-/Manycore System

- Cell/B.E.

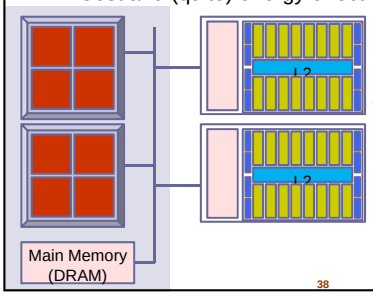


- GPU-based system:



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



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

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

And worse yet,

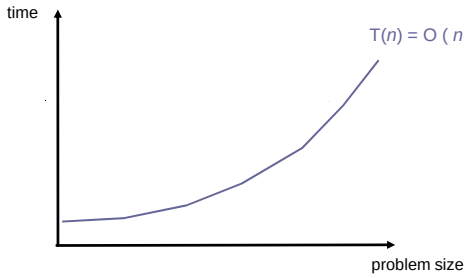
- A lot of variations/choices in hardware
  - Many will have performance implications
  - No standard parallel programming model
    - portability issue
- Understanding the hardware will make it easier to make programs get high performance
  - Performance-aware programming gets more important also for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

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

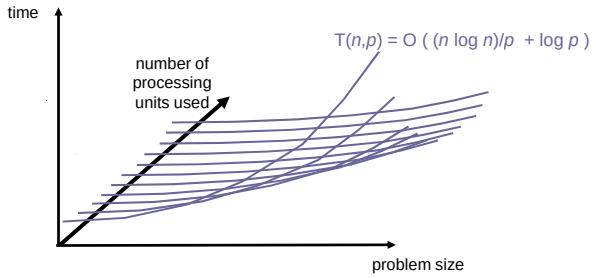
What we learned in the past…

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



… 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



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.

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

Questions?