

CS529 Lecture 01: Parallel Computing in the Many-core Processor Era

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

February 14, 2011

Course objectives

- ▶ Train students on **parallel programming** for modern multi-core system architectures
 - ▶ Using programming models that offer a relatively **high level of abstraction**
 - ▶ Using **a real general-purpose multi-core architecture during programming assignments** and exploring alternative architectures in class
- ▶ Train students on **performance-oriented parallel programming**
 - ▶ Explore techniques for **exposing more parallelism, exploiting locality, reducing synchronization and communication overhead**
 - ▶ Perform **performance analysis, explain experimental results**
- ▶ Train students on **reading, discussing, criticizing research papers**

Course logistics

- ▶ Graduate course with research training component
 - ▶ Discuss **research papers** in class on a weekly basis using a **hot seat model** (10% of class grade)
 - ▶ Perform **three programming assignments in groups** using **three programming models** to implement parallel applications of varying degrees of complexity (60% of class grade)
 - ▶ Write **two take-home exams** on class topics (30% of class grade)

Course logistics

- ▶ Course web page
 - ▶ `http://www.csd.uoc.gr/~hy529`
- ▶ Subscribe to the course mailing list by sending an e-mail to `majordomo@csd.uoc.gr` with body:
 - ▶ `subscribe hy529-list`
- ▶ Coordinate with instructor in first week of class to **get accounts at FORTH-ICS for accessing a multi-core system on which you will perform your class projects**

Parallel computing then and now

- ▶ Moore's law: the number of components in an integrated circuit doubles every X months $X = 12, \dots 18$
- ▶ New interpretation of Moore's Law
 - ▶ The number of **cores** doubles with every technology generation
- ▶ Shift to parallel computing necessary across market segments and not just in HPC
 - ▶ **Embedded systems**: Need programmable instead of custom platforms, diversity in applications, high demand for compute cycles and BW
 - ▶ **PCs, servers**: Unable to overcome the **power wall** and the **memory wall** with a single complex core. Attempt to do it with **many simple cores**.
- ▶ Shift to parallel computing changes fundamentally the way we program, debug, and analyze the performance of computers

Parallel computing then and now

- ▶ **Parallel computing** has been advocated in the past but failed to fulfill its promises (cost/performance of sequential processor vs. cost/performance of parallel machines)
- ▶ **Cost/performance ratio more favorable now:**
 - ▶ Can pack multiple cores on the same chip
 - ▶ Graceful technology scaling through replication
- ▶ Parallel computing may **fail again if we do not find a way to exploit parallelism in software**
 - ▶ **Challenges of parallelizing software:**
 - ▶ Understanding data dependencies
 - ▶ Synchronize accesses to shared data
 - ▶ Minimize communication, balance load

Parallelism in a new client/server paradigm

- ▶ Hundreds cores per chip available already on HW prototypes
 - ▶ NVIDIA GPUs build single-board systems with **up to 1600 cores**
- ▶ Vendors integrate easily up to 8 many-core CMPs on a single node
- ▶ Computing system architectures that benefit from parallelism:
 - ▶ **Datacenters** delivering virtualized software services
 - ▶ **Supercomputers** delivering high performance for computationally challenging, data-intensive problems
 - ▶ **Mobile computers** that replace PCs as the “personal computer”
 - ▶ Increased processing requirements for communication, video, audio

Application-driven parallel computing research

- ▶ Parallel computing driven by the needs of **scientific applications** in the past
 - ▶ Computations with "dense" arithmetic
- ▶ Dominant programming model: message-passing
 - ▶ Processor exchange data and synchronize via **explicit, user-defined messages, identifying source, destination, data type, and data size**
 - ▶ **Portable**, enables optimization by an expert
 - ▶ **Tedious** to use in practice
 - ▶ Overhead due to software layers between the program and the bare metal
- ▶ Many alternatives proposed in the past
 - ▶ Languages, libraries, auto-parallelization
 - ▶ All had difficulties in **abstracting away communication** while **remaining efficient**

Application-driven parallel computing research

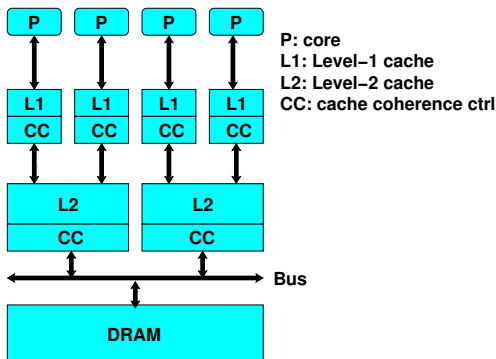
- ▶ Parallel computing is now driven by the needs of society at large
 - ▶ Parallel computing synonym to computing
 - ▶ Applications that need more processing power in less space and/or with a smaller power budget (batteries vs. outlet)
 - ▶ Real-time requirements (media, mobile communications, healthcare)
 - ▶ Massive data processing requirements (searching images, text, mining data)
 - ▶ More latency-critical applications (emergency response)
- ▶ Mobile computers replace PCs as **the “personal computer”**

Technology-driven parallel computing research

- ▶ Many cores per chip, per device
- ▶ High-speed I/O links and network links to each computer
 - ▶ 10 Gbit/s current technology, 100 Gbit/s near future technology
 - ▶ User-level communication technologies becoming less costly and commoditized
 - ▶ Infiniband at 20 Gb/s, 40 Gb/s comparable to 10G Ethernet
 - ▶ Such network speeds make network packet processing processor-bound
 - ▶ May need many cores to sustain throughput from many links
- ▶ Faster storage
 - ▶ Flash replacing or complementing disks (order of magnitude latency improvement)
 - ▶ Phase change RAM competitive as "main memory" or fast cache for disk

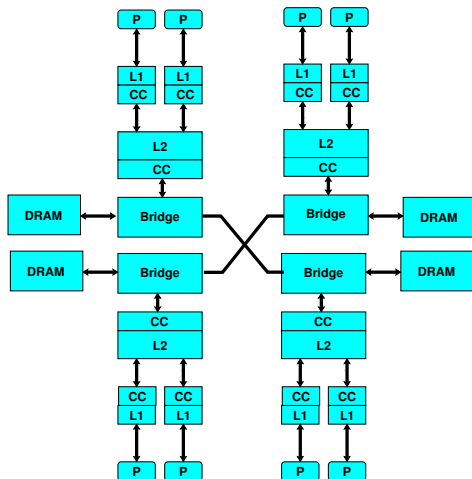
Multi-core processor architecture

- ▶ Typical general-purpose architecture (AMD, Intel variants)
- ▶ Private and shared levels of the cache (L2 or L3 shared)
- ▶ Varying degrees of sharing (including no sharing)
- ▶ Hardware-supported cache coherence



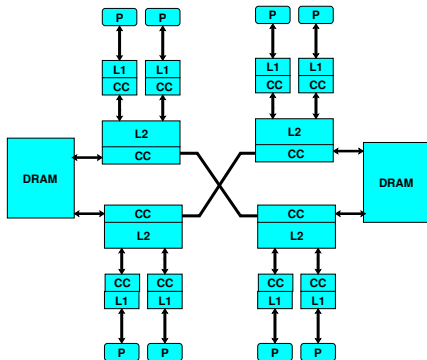
Multi-core processor architecture

- ▶ Non-uniform memory access architecture



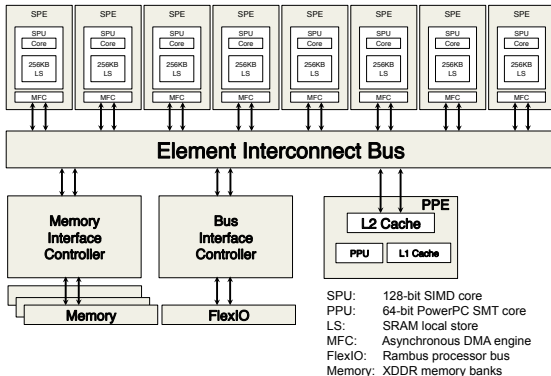
Multi-core processor architecture

- ▶ Non-uniform shared cache architecture



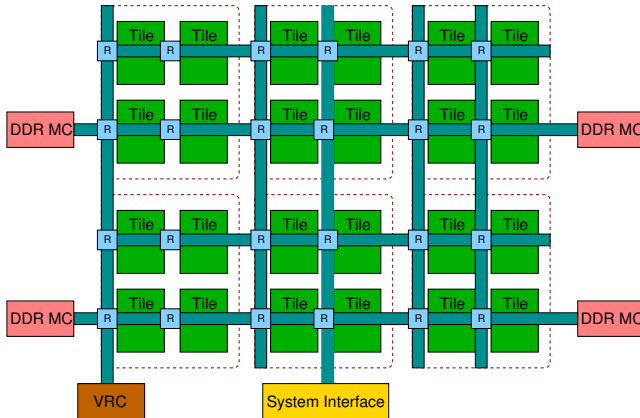
Multi-core processor architecture

- Private scratchpad memories, no cache coherence, explicitly managed memory hierarchy



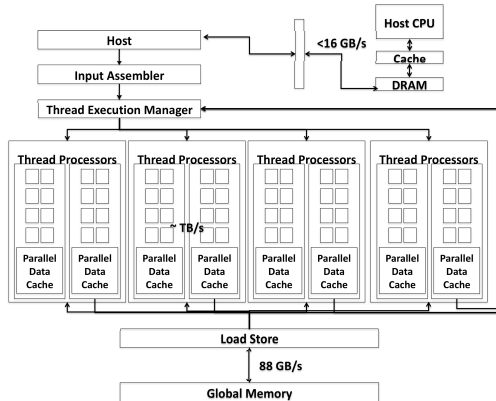
Multi-core processor architecture

- ▶ Private cache memories, no cache coherence, **explicitly managed coherence**, automatic caching



Multi-core processor architecture

- ▶ Massively parallel processor with **simple scalar cores** and **software-managed shared memory**



Key components of parallel software

- ▶ Well-known **parallel kernels** or **algorithms** or **algorithmic patterns**
 - ▶ Dense/sparse linear algebra, FFT, finite-state machines, graph algorithms, . . .
 - ▶ Sometimes packaged in **libraries** tuned by vendors for specific platforms
 - ▶ Typically form building blocks of real applications
 - ▶ Optimization targets for compilers, runtime systems
 - ▶ Common benchmarking tools

Key components of parallel software

- ▶ **Control structures** for parallelism
 - ▶ Expressing **partitioning** of computation into tasks and **assignment** of data to tasks
 - ▶ Often defining entire **frameworks** (languages or libraries) for parallel programming
 - ▶ **Low-level language control structures**: loops, tasks, vector processing, reductions, splitters-joiners, event-driven execution . . .
 - ▶ **High-level pattern control structures**: Owner-computes, MapReduce, searching optimization spaces (branch&bound, simulated annealing), graph traversals, N-body problems, dynamic programming, . . .)

Productivity vs. efficiency

- ▶ Efficiency
 - ▶ Approximate performance of **bare metal** hardware with **minimum** software overheads (assembly-level performance)
 - ▶ Peak FLOPS for **processor-bound** peak bytes/second for **memory-bound**
 - ▶ Optimization target for compilers, runtime systems, expert programmers
- ▶ Productivity
 - ▶ Develop parallel applications in **short time** by **reusing software modules**
 - ▶ Needs **reusable control patterns**
 - ▶ Needs **reusable composition frameworks**
 - ▶ Need **highly tuned libraries** of common functions

Correctness vs. productivity and efficiency

- ▶ Productivity
 - ▶ Correctness **simplified** if programmer **guarantees independence** of tasks through **data decomposition**
 - ▶ This helps remove **races** and reduce **communication, synchronization** due to dependences
 - ▶ It can be hard to **guarantee in all cases**
 - ▶ Example: graph algorithms, mesh generation
- ▶ Efficiency
 - ▶ Detect **races** and remove them using various tools (static analysis, model checking, deterministic replay, deterministic execution . . .)
 - ▶ Minimize **dependences** in cases where programmers **impose more constraints than needed**
 - ▶ Remove barriers or locks or other primitives that impose waiting

New directions in parallel software

- ▶ Programming models
 - ▶ Support for **multiple parallel control structures**
 - ▶ Flexibility in **scheduling computation and data transfers**
 - ▶ Runtime systems become critical for performance
 - ▶ **Simplification of synchronization** (through transactions, static analysis, dynamic analysis, ...)
- ▶ Tools
 - ▶ Auto-tuning of program, compiler, runtime **tunable parameters**
 - ▶ Using **sophisticated search algorithms, machine learning**
 - ▶ Usable in common important kernels (e.g. FFT, sorting, matrix-vector multiple, ...)
 - ▶ Promising direction with questions on whether **blind search** algorithms are preferable to **analytical models**