

TCSS 562:
SOFTWARE ENGINEERING
FOR CLOUD COMPUTING

Cloud Computing:
How did we get here?

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma



FEEDBACK FROM 9/26

- Material Rating:
- Mostly New to Me: 10- 6 respondents
9- 4 respondents
7- 1 respondent
6- 1 respondent
5- 3 respondents

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.2

FEEDBACK - 2

- I know absolutely nothing about cloud computing
- We need to go into basic/fundamental concepts of cloud computing
- I did not get the project description
- The team project(s)

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.3

OBJECTIVES

- Cloud Computing: How did we get here?
 - Parallel and distributed systems (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)
 - Data, thread-level, task-level parallelism
 - Parallel architectures
 - SIMD architectures, vector processing, multimedia extensions
 - Graphics processing units
 - Speed-up, Amdahl's Law, Scaled Speedup
 - Properties of distributed systems
 - Modularity


October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.4

CLOUD COMPUTING:
HOW DID WE GET HERE?

- General interest in parallel computing
 - Moore's Law - # of transistors doubles every 18 months
 - Post 2004: heat dissipation challenges: can no longer easily increase cloud speed
 - Overclocking to 7GHz takes more than just liquid nitrogen:
 - <https://tinyurl.com/y93s2yz2>
- Solutions:
 - Vary CPU clock speed
 - Add CPU cores
 - Multi-core technology

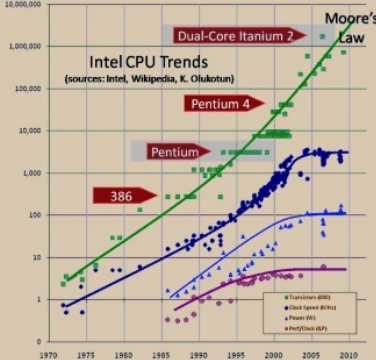


October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.5

Each Year We Get ~~More~~ More Processors



Historically:

Boost single-stream performance via more complex chips.

Now:

Deliver more cores per chip (+ GPU, NIC, SoC).

The free lunch is over for today's sequential apps and many concurrent apps. We need killer apps with lots of latent parallelism.

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.6

CLOUD COMPUTING: HOW DID WE GET HERE? - 2

- To make computing faster, we must go "parallel"
- Difficult to expose parallelism in scientific applications
- Not every problem solution has a parallel algorithm
 - Chicken and egg problem...
- Many commercial efforts in promoting pure parallel programming efforts have failed
- Enterprise computing world has been *skeptical* and less involved in parallel programming

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.7

CLOUD COMPUTING: HOW DID WE GET HERE? - 3

- **Cloud computing** provides access to "infinite" scalable compute infrastructure on demand
- Infrastructure availability is key to exploiting parallelism
- **Cloud applications**
 - Based on **client-server** paradigm
 - **Thin clients** leverage compute hosted on the cloud
 - Applications run many web service instances
 - Employ load balancing

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.8

CLOUD COMPUTING: HOW DID WE GET HERE? - 4

- **Big Data** requires massive amounts of compute resources
- MAP - REDUCE
 - Single instruction, multiple data (SIMD)
 - Exploit data level parallelism
- Bioinformatics example

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.9

SMITH WATERMAN USE CASE



- Applies dynamic programming to find best local alignment of two protein sequences
 - Embarrassingly parallel, each task can run in isolation
 - Use case for GPU acceleration
- **AWS Lambda Serverless Computing Use Case:**
 - **Goal:** Pair-wise comparison of all unique human protein sequences (20,336)
 - Python client as scheduler
 - C Striped Smith-Waterman (SSW) execution engine

From: Zhao M, Lee WP, Garrison EP, Marth GT: SSW library: an SIMD Smith-Waterman C/C++ library for use in genomic applications. PLoS One 2013, 8:e82138

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.10

SMITH WATERMAN RUNTIME

- Laptop server and client (2-core, 4-HT): 8.7 hours
- AWS Lambda server, laptop as client: 2.2 minutes
 - Partitions 20,336 sequences into 41 sets
 - Execution cost: ~ **82¢** (~**237x speed-up**)
- AWS Lambda server, EC2 instance as client: 1.28 minutes
 - Execution cost: ~ **87¢** (~**408x speed-up**)
- Hardware
 - Laptop client: Intel i5-7200U 2.5 GHz :4 HT, 2 CPU,
 - Cloud client: EC2 instance - m5.24xlarge: 96 vCPUs
 - Cloud server: Lambda ~1000 Intel E5-2666v3 2.9GHz CPUs

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.11

CLOUD COMPUTING: HOW DID WE GET HERE? - 3

- Compute clouds are large-scale distributed systems
 - Heterogeneous systems
 - Homogeneous systems
 - Autonomous
 - Self organizing

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.12

PARALLELISM

- Discovering parallelism and development of parallel algorithms requires considerable effort
- Example:** numerical analysis problems, such as solving large systems of linear equations or solving systems of Partial Differential Equations (PDEs), require algorithms based on domain decomposition methods.
- How can the problem be split into independent chunks?**
- Fine-grained parallelism**
 - Only small bits of code can run in parallel without coordination
 - Communication is required to synchronize state across nodes
- Coarse-grained parallelism**
 - Large blocks of code can run without coordination

October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.13

PARALLELISM - 2

- Coordination of nodes**
- Requires **message passing** or **shared memory**
- Debugging parallel **message passing** code is easier than parallel **shared memory** code
- Message passing:** all of the interactions are clear
- Shared memory:** interactions can be implicit – *must read the code!!*
- Processing speed is orders of magnitude faster than communication speed (CPU > memory bus speed)
- Avoiding coordination achieves the best speed-up

October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.14

TYPES OF PARALLELISM

- Thread-level parallelism (TLP)**
 - Control flow architecture
- Data-level parallelism**
 - Data flow architecture
- Bit-level parallelism**
- Instruction-level parallelism (ILP)**

October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.15

DATA-LEVEL PARALLELISM

- Partition data into big chunks, run separate copies of the program on them with little or no communication
- Problems are considered to be **embarrassingly parallel**
- Also perfectly parallel or pleasingly parallel...
- Little or no effort needed to separate problem into a number of parallel tasks
- MapReduce programming model is an example

October 1, 2018

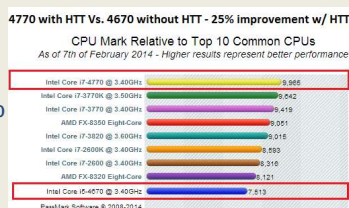
TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.16

HYPER THREADING

- Modern CPUs provide multiple instruction pipelines, supporting multiple execution threads, usually 2 to feed instructions to a single CPU core...

- Two hyper-threads are not equivalent to (2) CPU cores
- i7-4770 and i5-4760 same CPU, with and without HTT



October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.17

THREAD LEVEL PARALLELISM (TLP)

- Number of threads an application runs at any one time
- Varies throughout program execution
- As a metric:
- Minimum:** 1 thread
- Can measure **average**, **maximum (peak)**
- QUESTION:** What are the consequences of **average (TLP)** for scheduling an application to run on a computer with a fixed number of CPU cores and hyperthreads?
- Let's say there are 4 cores, or 8 hyper-threads...
- Key to avoiding waste of computing resources is knowing your application's TLP...**

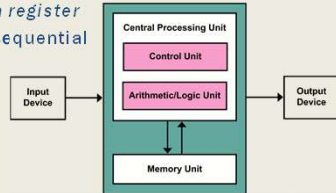
October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.18

CONTROL-FLOW ARCHITECTURE

- By John von Neumann (1945)
- Also called the Von Neumann architecture
- Dominant computer system architecture
- Program counter (PC) determines next instruction to load into *instruction register*
- Program execution is sequential



October 1, 2018

TCCS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.19

DATA FLOW ARCHITECTURE

- Alternate architecture** used by network routers, digital signal processors, special purpose systems
- Operations performed when input (data) becomes available
- Envisioned to provide much higher parallelism
- Multiple problems has prevented wide-scale adoption
 - Efficiently broadcasting data tokens in a massively parallel system
 - Efficiently dispatching instruction tokens in a massively parallel system
 - Building content addressable memory large enough to hold all of the dependencies of a real program

October 1, 2018

TCCS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.20

DATA FLOW ARCHITECTURE - 2

- Architecture not as popular as control-flow
- Modern CPUs emulate data flow architecture for dynamic instruction scheduling since the 1990s
 - Out-of-order execution - reduces CPU idle time by not blocking for instructions requiring data by defining execution windows
 - Execution windows identify instructions that can be run by data dependency
 - Instructions are completed in data dependency order within execution window
 - Execution window size typically 32 to 200 instructions

Utility of data flow architectures has been much less than envisioned

October 1, 2018

TCCS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.21

BIT-LEVEL PARALLELISM

- Computations on large words (e.g. 64-bit integer) are performed as a single instruction
 - Fewer instructions are required on 64-bit CPUs to process larger operands (A+B) providing dramatic performance improvements
 - Processors have evolved: 4-bit, 8-bit, 16-bit, 32-bit, 64-bit
- QUESTION: How many instructions are required to add two 64-bit numbers on a 16-bit CPU? (Intel 8088)**
- 64-bit MAX int = 9,223,372,036,854,775,807 (signed)
 - 16-bit MAX int = 32,767 (signed)
 - Intel 8088 - limited to 16-bit registers

October 1, 2018

TCCS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.22

INSTRUCTION-LEVEL PARALLELISM (ILP)

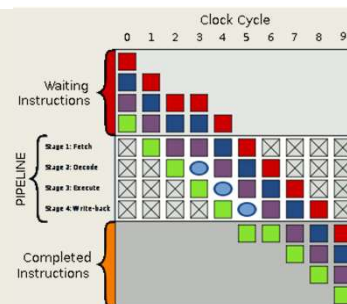
- CPU pipelining architectures enable ILP
- CPUs have multi-stage processing pipelines
- Pipelining: split instructions into sequence of steps that can execute concurrently on different CPU circuitry
- Basic RISC CPU - Each instruction has 5 pipeline stages:
 - IF** - instruction fetch
 - ID** - instruction decode
 - EX** - instruction execution
 - MEM** - memory access
 - WB** - write back

October 1, 2018

TCCS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.23

CPU PIPELINING



October 1, 2018

TCCS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.24

INSTRUCTION LEVEL PARALLELISM - 2

- After 5 clock cycles, all 5 stages of an instruction are loaded
- Starting with 6th clock cycle, one full instruction completes each cycle
- The CPU performs 5 tasks per clock cycle!
Fetch, decode, execute, memory read, memory write back
- Pentium 4 (CISC) – 35 stages!

October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.25

MICHAEL FLYNN'S COMPUTER ARCHITECTURE TAXONOMY

- Michael Flynn's proposed taxonomy of computer architectures based on concurrent instructions and number of data streams (1966)
- **SISD (Single Instruction Single Data)**
- **SIMD (Single Instruction, Multiple Data)**
- **MIMD (Multiple Instructions, Multiple Data)**
- **LESS COMMON:** MISD (Multiple Instructions, Single Data)
- Pipeline architectures: functional units perform different operations on the same data
- For fault tolerance, may want to execute same instructions redundantly to detect and mask errors – for task replication

October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.26

FLYNN'S TAXONOMY

- **SISD (Single Instruction Single Data)**
 Scalar architecture with one processor/core.
 - Individual cores of modern multicore processors are "SISD"
- **SIMD (Single Instruction, Multiple Data)**
 Supports vector processing
 - When SIMD instruction is issued, operations on individual vector components are carried out concurrently
 - Two 64-element vectors can be added in parallel
 - Vector processing instructions added to modern CPUs
 - Example: Intel MMX (multimedia) instructions

October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.27

(SIMD): VECTOR PROCESSING ADVANTAGES

- Exploit data-parallelism: vector operations enable speedups
- Vectors architecture provide vector registers that can store entire matrices into a CPU register
- SIMD CPU extension (e.g. MMX) add support for vector operations on traditional CPUs
- Vector operations reduces total number of instructions for large vector operations
- Provides higher potential speedup vs. MIMD architecture
- Developers think sequentially; not worry about parallelism

October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.28

FLYNN'S TAXONOMY - 2

- **MIMD (Multiple Instructions, Multiple Data)** - system with several processors and/or cores that function asynchronously and independently
- At any time, different processors/cores may execute different instructions on different data
- Multi-core CPUs are MIMD
- Processors share memory via interconnection networks
 - Hypercube, 2D torus, 3D torus, omega network, other topologies
- MIMD systems have different methods of sharing memory
 - Uniform Memory Access (UMA)
 - Cache Only Memory Access (COMA)
 - Non-Uniform Memory Access (NUMA)

October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.29

ARITHMETIC INTENSITY

- **Arithmetic Intensity** – number of floating point operations per byte of data read
- SIMD provides fast matrix operations
- Characterizes application scalability based on SIMD support
- **High arithmetic Intensity:**
 Programs with dense matrix operations scale up nicely
- **Low arithmetic Intensity:**
 Programs with sparse matrix operations do not scale well with problem size

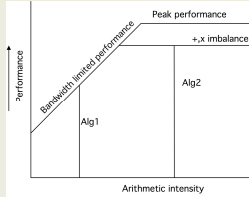
October 1, 2018

TCS5562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.30

ROOFLINE MODEL

- When program reaches a given arithmetic intensity performance of code running on CPU hits a "roof"
- CPU performance bottleneck changes from: memory bandwidth (left) → floating point performance (right)



October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.31

GRAPHICAL PROCESSING UNITS (GPUS)

- GPU provides multiple SIMD processors
- Typically 7 to 15 SIMD processors each
- 32,768 total registers, divided into 16 lanes (2048 registers each)
- GPU programming model: single instruction, multiple thread
- Programmed using CUDA- C like programming language by NVIDIA for GPUs
- CUDA threads – single thread associated with each data element (e.g. vector or matrix)
- Thousands of threads run concurrently

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.32

PARALLEL COMPUTING

- Parallel hardware and software systems allow:
 - Solve problems demanding resources not available on single system.
 - Reduce time required to obtain solution
- The *speed-up* (S) measures effectiveness of parallelization:

$$S(N) = T(1) / T(N)$$

T(1) → execution time of total sequential computation
 T(N) → execution time for performing N parallel computations in parallel

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.33

SPEED-UP EXAMPLE

- Consider embarrassingly parallel image processing
- Eight images (multiple data)
- Apply image transformation (greyscale) in parallel
- 8-core CPU, 16 hyperthreads
- Sequential processing: perform transformations one at a time using a single program thread
 - 8 images, 3 seconds each: T (1) = 24 seconds
- Parallel processing
 - 8 images, 3 seconds each: T (N) = 3 seconds
- Speedup: S (N) = 24 / 3 = 8x speedup
- Called "**perfect scaling**"
- Must consider data transfer and computation setup time

October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.34

QUESTIONS



October 1, 2018

TCSS562: Software Engineering for Cloud Computing [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma

L2.83