# TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

## Introduction

**Wes J. Lloyd**

**School of Engineering and Technology**

**University of Washington - Tacoma**

1

# OBJECTIVES

- Syllabus and course introduction

- Cloud Computing – How did we get here?
  Introduction to parallel and distributed systems
  (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.2 |
|---|---|---|

2

# DEMOGRAPHICS SURVEY

- Please complete the ONLINE demographics survey:

- https://forms.gle/dE4Q7Lt13rAXtahJ9

- Linked from course webpage in Canvas:
- http://faculty.washington.edu/wlloyd/courses/tcss562/
  announcements.html

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.3 |
|---|---|---|

3

# TCSS562 – SOFTWARE ENGINEERING FOR CLOUD COMPUTING

- Syllabus online at:
  http://faculty.washington.edu/wlloyd/courses/tcss562/

- Grading

- Schedule

- Assignments

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.4 |
|---|---|---|

4

# REFERENCES

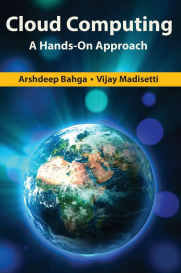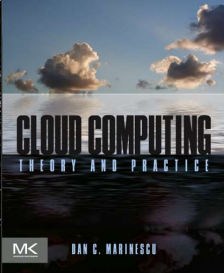- [1] Cloud Computing: Concepts, Technology and Architecture *
- Thomas Erl, Prentice Hall 2013

- [2] Cloud Computing - Theory and Practice
- Dan Marinescu, First Edition 2013 *, Second Edition 2018

- [3] Cloud Computing: A Hands-On Approach
- Arshdeep Bahga 2013

*  - available online via UW library

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.5 |

5

# REFERENCES - 2

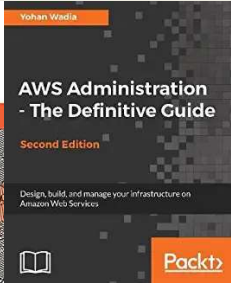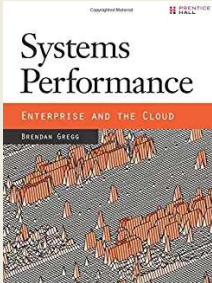- [4] Systems Performance: Enterprise and the Cloud *
- Brendan Gregg, First Edition 2013

- [5] AWS Administration – The Definitive Guide *
- Yohan Wadia, First Edition 2016

- Research papers

*  - available online via UW library

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.6 |

6

## TCSS 562 – Fall 2019

- **Mondays/Wednesdays**
  - Lecture, midterm, quiz, activities
  - No class: Monday Nov 11
    Class online: Wednesday Nov 27
  - Key Topics:
  - IaaS, Virtualization, Serverless computing, FaaS, Containerization

- **Tutorials**
  - Lab Day: Tutorials, group project work

- **No Final exam**
- **Midterm Wednesday October 30th** *(tentative)*
- **Term Project: Build and evaluate alternate implementations of a native cloud serverless application; or group proposed cloud research project**

TCSS 562
FALL
2019

September 25, 2019        TCSS562: Software Engineering for Cloud Computing [Fall 2019]
School of Engineering and Technology, University of Washington - Tacoma          L1.7

7

## TCS562 COURSE WORK

- **Project Proposal**

- **Project Status Reports / Activities / Quiz**
  - ~ 2-4 total items  (??)
  - Variety of formats: in class, online, reading, activity

- **Midterm**
  - Open book, note, etc.

- **Class Presentation**

- **Term Project / Paper / Presentation**

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.8 |

8

# CLASS PRESENTATION

- **Each student will make one presentation in a <u>team of ~3</u>**

- **<u>Technology sharing presentation</u>**
    - **PPT Slides, demonstration**
    - **Provide technology overview of one cloud service offering**
    - **Present overview of features, performance, etc.**

- **<u>Cloud Research Paper Presentation</u>**
    - **PPT slides, identify research contributions, strengths and weaknesses of paper, possible areas for future work**

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.9 |
|---|---|---|

9

# TCS562 TERM PROJECT

- **Project description to be posted**
- **Teams of ~3, self formed, one project leader**
- **Proposal due: Friday October 11, 11:59pm (tentative)**

- **Focus:**
    - **Build a native cloud serverless application**
    - **Compose multiple FaaS functions (services)**
    - **Compare alternate implementation of:**
        - **Service compositions**
        - **Application flow control - AWS Step Functions, laptop client, etc.**
        - **External cloud components (e.g. database, key-value store)**
    - **How does application design impact cost and performance?**

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.10 |
|---|---|---|

10

# TCSS562 TERM PROJECT - 2

- **Deliverables**
  - **Demo in class at end of quarter (TBD)**
  - **Project report paper (4-6 pgs IEEE format, template provided)**
  - **GitHub (project source)**
  - **How-To document (via GitHub markdown)**

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.11 |
|---|---|---|

11

# ALTERNATE TERM PROJECT IDEAS

- **GOAL: propose cloud development project that serves as a vehicle to compare and contrast the use of alternative cloud services**

- **Examples:**
- **Object/blob storage services**
  - **Amazon S3, Google blobstore, Azure blobstore, vs. self-hosted**
- **Cloud Relational Database services**
  - **Amazon Relational Database Service (RDS), Aurora, Self-Hosted DB**
- **Platform-as-a-Service hosting (PaaS) alternatives**
  - **Amazon Elastic Beanstalk, Heroku, others**
- **Function-as-a-Service platforms**
  - **Google Cloud Functions, Azure Functions, IBM Cloud Functions**

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.12 |
|---|---|---|

12

# TERM PROJECT IDEAS - 2

- File-based storage systems
  - Amazon EBS, Amazon EFS, others
- Container orchestration services
  - Amazon ECS, AKS, Azure Kubernetes Service
- Queueing services comparison
  - Amazon SQS, Amazon MQ, Apache Kafka, RabbitMQ, 0Mq, others
- NoSQL database services comparison
  - DynamoDB, Google BigTable, MongoDB, Cassandra

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.13 |
|---|---|---|

13

# TERM PROJECT: RESEARCH

- Alternative: conduct a cloud-related research project on any topic to answer a set of research questions
  - Can be used to help spur MS Capstone/Thesis work

- If you're interested in this option, please talk with the instructor

- First step is to identify 1 – 2 research questions

- Instructor will help guide projects throughout the quarter

- Approval based on team preparedness to execute project

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.14 |
|---|---|---|

14

## PROJECT SUPPORT

- Project cloud infrastructure support:
- Sign up for the Github Student Developer Pack:
  - https://education.github.com/pack
  - Includes up to $150 in Amazon Cloud Credits
  - Includes up to $100 in Microsoft Azure Credits
  - AWS credit extensions provided as needed
- Microsoft Azure for Students
  - $100 free credit per account valid for 1 year
  - https://azure.microsoft.com/en-us/free/students/
  - Also: $200 free credit option for 1 month
- Google Cloud
  - $300 free credit for 1 year
  - https://cloud.google.com/free/
- Chameleon / CloudLab
  - Bare metal NSF cloud - free

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.15 |

15

## TCSS562 TERM PROJECT OPPORTUNITIES

- **Projects can lead to papers or posters presented at ACM/IEEE/USENIX conferences, workshops**
  - Networking and travel opportunity
  - Conference participation (posters, papers) helps differentiate your resume from others

- **Project can support preliminary work for: UWT - MS capstone/thesis project proposals**

- **Research projects provide valuable practicum experience with cloud systems analysis, prototyping**

- **Publications are key for building your resume/CV, Also key if applying to PhD programs**

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.16 |

16

## TCSS562 TERM PROJECT - 3

- Project status report / term project check-ins
  - Written status report
  - 2-3 times in quarter
  - Part of: *"Project Status Reports / Activities / Quizzes"* category
  - 10% of grade

- Project meetings with instructor
  - After class, end half of class, office hours

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.17 |
|---|---|---|

17

## 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
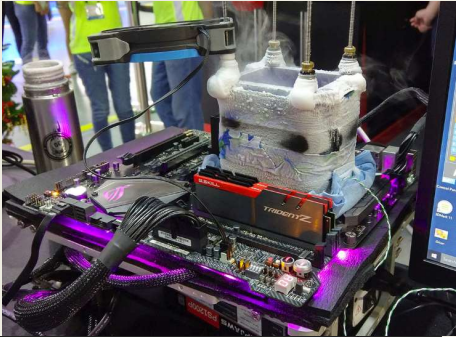  - Modularity

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.18 |
|---|---|---|

18

## 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**

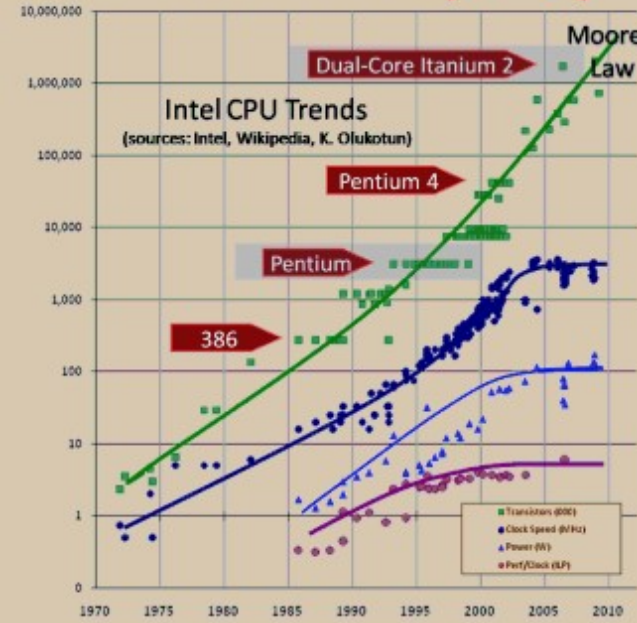| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.19 |
|---|---|---|

19



| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.20 |
|---|---|---|

20

# AMD'S 64-CORE 7NM CPUS

- **Epyc Rome CPUs**
- **Announced August 2019**
- **EPYC 7H12 requires liquid cooling**

| AMD EPYC 7002 Processors (2P) | | | | | | |
|---|---|---|---|---|---|---|
| | Cores Threads | Frequency (GHz) | | L3* | TDP | Price |
| | | Base | Max | | | |
| EPYC 7H12 | 64 / 128 | 2.60 | 3.30 | 256 MB | 280 W | ? |
| EPYC 7742 | 64 / 128 | 2.25 | 3.40 | 256 MB | 225 W | $6950 |
| EPYC 7702 | 64 / 128 | 2.00 | 3.35 | 256 MB | 200 W | $6450 |
| EPYC 7642 | 48 / 96 | 2.30 | 3.20 | 256 MB | 225 W | $4775 |
| EPYC 7552 | 48 / 96 | 2.20 | 3.30 | 192 MB | 200 W | $4025 |

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.21 |
|---|---|---|

21

# 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**

- **Example: → hyperthreads add +32.9%**

4770 with HTT Vs. 4670 without HTT - 25% improvement w/ HTT

CPU Mark Relative to Top 10 Common CPUs
*As of 7th of February 2014 - Higher results represent better performance*

| | |
|---|---|
| Intel Core i7-4770 @ 3.40GHz | 9,965 |
| Intel Core i7-3770K @ 3.50GHz | 9,642 |
| Intel Core i7-3770 @ 3.40GHz | 9,419 |
| AMD FX-8350 Eight-Core | 9,051 |
| Intel Core i7-3820 @ 3.60GHz | 9,015 |
| Intel Core i7-2600K @ 3.40GHz | 8,593 |
| Intel Core i7-2600 @ 3.40GHz | 8,316 |
| AMD FX-8320 Eight-Core | 8,121 |
| Intel Core i5-4670 @ 3.40GHz | 7,513 |

PassMark Software © 2008-2014

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.22 |
|---|---|---|

22

# 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 promoting pure parallel programming efforts have failed
- Enterprise computing world has been *skeptical* and less involved in parallel programming

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.23 |
|---|---|---|

23

# 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.24 |
|---|---|---|

24

## 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.25 |

25

## 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*

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.26 |

26

# SMITH WATERMAN RUNTIME

- **Laptop server and client (2-core, 4-HT): 8.7 hours**

- **AWS Lambda FaaS, 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 Virtual Machine - m5.24xlarge: 96 vCPUs
  - Cloud server: Lambda ~1000 Intel E5-2666v3 2.9GHz CPUs

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.27 |
|---|---|---|

27

# CLOUD COMPUTING:
# HOW DID WE GET HERE? - 3

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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.28 |
|---|---|---|

28

# 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 problems 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.29 |

29

# 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
  - Coordination via specific programming API (MPI)
- *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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.30 |

30

# TYPES OF PARALLELISM

- ▪ **Parallelism:**
  - ▪ **Goal: Perform multiple operations at the same time to achieve a speed-up**

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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.31 |

31

# 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...***
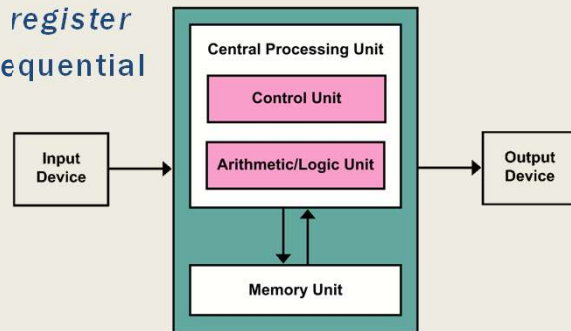
| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.32 |

32

# 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

33

# 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

34

# 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.35 |

35

# 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*

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.36 |

36

# 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.37 |
|---|---|---|

37

# 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*

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.38 |
|---|---|---|

38

## CPU PIPELINING



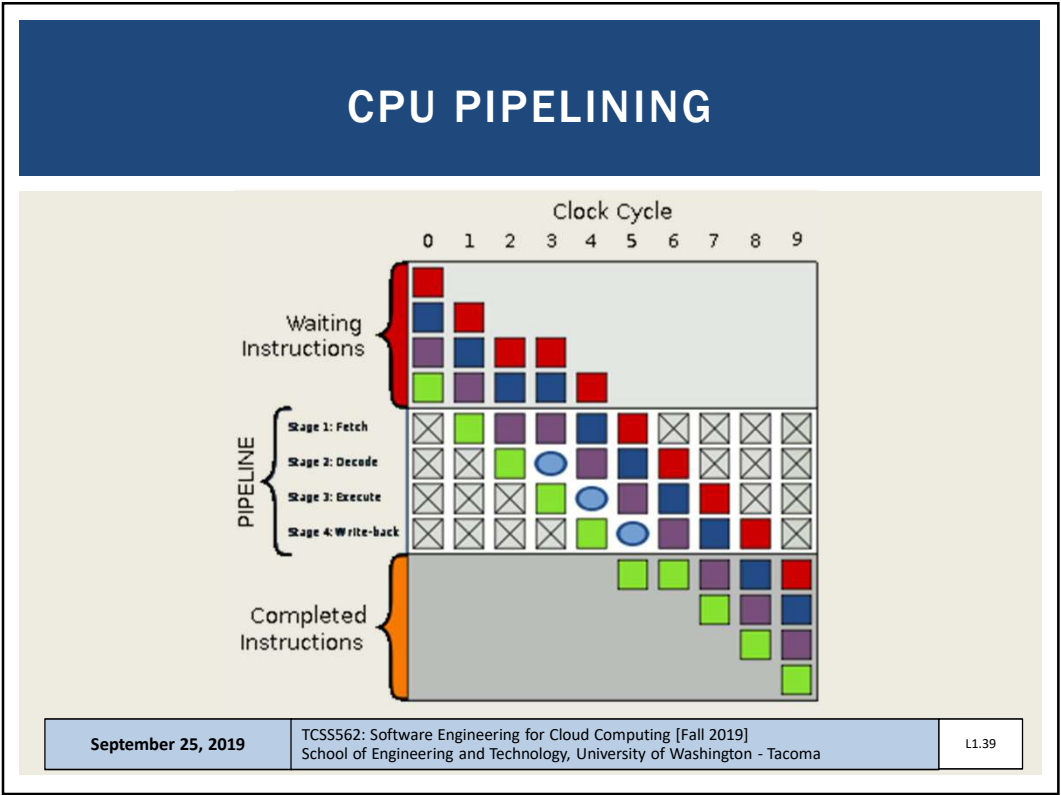| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.39 |

39

## INSTRUCTION LEVEL PARALLELISM - 2

- RISC CPU:
- 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 CPU) – processing pipeline w/ 35 stages!

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.40 |

40

## 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.41 |

41

## 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 instructions are 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.42 |

42

## (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 reduce total number of instructions for large vector operations

- Provides higher potential speedup vs. MIMD architecture

- Developers can think sequentially; not worry about parallelism

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.43 |

43

## 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)

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.44 |

44

# ARITHMETIC INTENSITY

- **Arithmetic intensity**:      Ratio of work (W) to memory traffic r/w (Q)    $I = \frac{W}{Q}$
  Example: # of floating point ops per byte of data read
- Characterizes application scalability with SIMD support
  - *SIMD can perform many fast matrix operations in parallel*

- *High arithmetic Intensity:*
  *P*rograms with dense matrix operations scale up nicely
  (many calcs vs memory RW, supports lots of parallelism)

- *Low arithmetic intensity:*
  Programs with sparse matrix operations do not scale well
  with problem size
  (memory RW becomes bottleneck, not enough ops!)
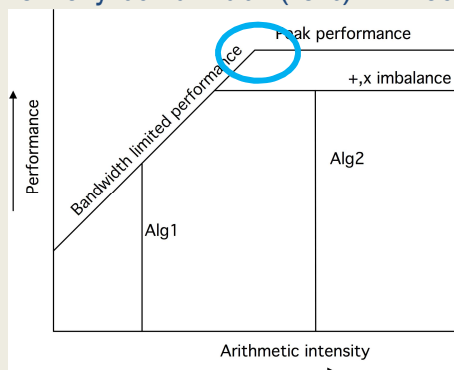
| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.45 |

45

# 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)



Key take-aways:
When a program's has **low** Arithmetic Intensity, memory bandwidth limits performance..

With **high** Arithmetic intensity, the system has peak parallel performance…
→ *performance is limited by??*

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.46 |

46

# 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.47 |

47

# 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

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.48 |

48

## 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 "<u>perfect scaling</u>"

- Must consider data transfer and computation setup time

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.49 |

49

## AMDAHL'S LAW

- Portion of computation which cannot be parallelized determines the overall speedup
- For an embarrassingly parallel job of fixed size
- Assuming no overhead for distributing the work, and a perfectly even work distribution

  α: fraction of program run time which can't be parallelized (e.g. must run sequentially)

- Maximum speedup is:

$$S = 1/\alpha$$

- **<u>Example:</u>**
  Consider a program where 25% cannot be parallelized
  **<u>Q:</u> _What is the maximum possible speedup of the program?_**

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.50 |

50

# GUSTAFSON'S LAW

- Calculates the *__scaled speed-up__* using "N" processors

$$S(N) = N + (1 - N) \alpha$$

N: Number of processors

α: fraction of program run time which can't be parallelized (e.g. must run sequentially)

- Example:

  Consider a program that is embarrassingly parallel, but 25% cannot be parallelized. α=.25

  QUESTION: *If deploying the job on a 2-core CPU, what scaled speedup is possible assuming the use of two processes that run in parallel?*

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.51 |
|---|---|---|

51

# GUSTAFSON'S EXAMPLE

- QUESTION:

  What is the maximum theoretical speed-up on a **2-core CPU** ?

  $$S(N) = N + (1 - N) \alpha$$

  N=2, α=.25

  $$S(N) = 2 + (1 - 2) .25$$

  S(N) = ?

- What is the maximum theoretical speed-up on a **4-core CPU**?

  $$S(N) = N + (1 - N) \alpha$$

  N=4, α=.25

  $$S(N) = 4 + (1 - 4) .25$$

  S(N) = ?

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.52 |
|---|---|---|

52

# MOORE'S LAW

- Transistors on a chip doubles approximately every 1.5 years
- CPUs now have billions of transistors
- Power dissipation issues at faster clock rates leads to heat removal challenges
  - Transition from: increasing clock rates → to adding CPU cores

- *Symmetric core processor* –multi-core CPU, all cores have the same computational resources and speed
- *Asymmetric core processor* – on a multi-core CPU, some cores have more resources and speed
- *Dynamic core processor* – processing resources and speed can be dynamically configured among cores

- *Observation: asymmetric processors offer a higher speedup*

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.53 |
|---|---|---|

53

# DISTRIBUTED SYSTEMS

- Collection of autonomous computers, connected through a network with distribution software called "middleware" that enables coordination of activities and sharing of resources
- Key characteristics:
- Users perceive system as a single, integrated computing facility.
- Compute nodes are autonomous
- Scheduling, resource management, and security implemented by every node
- Multiple points of control and failure
- Nodes may not be accessible at all times
- System can be scaled by adding additional nodes
- Availability at low levels of HW/software/network reliability

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.54 |
|---|---|---|

54

# DISTRIBUTED SYSTEMS - 2

- **Key non-functional attributes**
  - **Known as "ilities" in software engineering**

- **Availability – 24/7 access?**
- **Reliability - Fault tolerance**
- **Accessibility – reachable?**
- **Usability – user friendly**
- **Understandability – can under**
- **Scalability – responds to variable demand**
- **Extensibility – can be easily modified, extended**
- **Maintainability – can be easily fixed**
- **Consistency – data is replicated correctly in timely manner**

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.55 |

55

# TRANSPARENCY PROPERTIES OF DISTRIBUTED SYSTEMS

- **Access transparency**: local and remote objects accessed using identical operations
- **Location transparency**: objects accessed w/o knowledge of their location.
- **Concurrency transparency**: several processes run concurrently using shared objects w/o interference among them
- **Replication transparency**: multiple instances of objects are used to increase reliability
  *- users are unaware if and how the system is replicated*
- **Failure transparency**: concealment of faults
- **Migration transparency**: objects are moved w/o affecting operations performed on them
- **Performance transparency**: system can be reconfigured based on load and quality of service requirements
- **Scaling transparency**: system and applications can scale w/o change in system structure and w/o affecting applications

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.56 |

56

## TYPES OF MODULARITY

- *Soft modularity:* TRADITIONAL
- Divide a program into modules (classes) that call each other and communicate with shared-memory
- A procedure calling convention is used (or method invocation)

- *Enforced modularity:* CLOUD COMPUTING
- Program is divided into modules that communicate only through message passing
- The ubiquitous client-server paradigm
- Clients and servers are independent decoupled modules
- System is more robust if servers are stateless
- May be scaled and deployed separately
- May also FAIL separately!

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.57 |

57

## CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS

- Multi-core CPU technology and hyper-threading
- What is a
  - Heterogeneous system?
  - Homogeneous system?
  - Autonomous or self-organizing system?
- **Fine grained vs. coarse grained parallelism**
- Parallel message passing code is easier to debug than shared memory (e.g. p-threads)
- Know your application's max/avg **Thread Level Parallelism** (*TLP*)
- **Data-level parallelism**: Map-Reduce, (SIMD) Single Instruction Multiple Data, Vector processing & GPUs

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.58 |

58

## CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS - 2

- **Bit-level parallelism**
- **Instruction-level parallelism** (CPU pipelining)
- **Flynn's taxonomy**: computer system architecture classification
  - **SISD** – Single Instruction, Single Data (modern core of a CPU)
  - **SIMD** – Single Instruction, Multiple Data (Data parallelism)
  - **MIMD** – Multiple Instruction, Multiple Data
  - MISD is RARE; application for fault tolerance…
- **Arithmetic intensity**: ratio of calculations vs memory RW
- **Roofline model:**
  Memory bottleneck with low arithmetic intensity
- **GPUs**: ideal for programs with high arithmetic intensity
  - SIMD and Vector processing supported by many large registers

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.59 |

59

## CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS - 3

- **Speed-up (S)**
  $S(N) = T(1) / T(N)$
- **Amdahl's law:**
  $S = 1/ \alpha$
  $\alpha$ = percent of program that must be sequential
- **Scaled speedup with N processes:**
  $S(N) = N – \alpha( N-1)$
- Moore's Law
- Symmetric core, Asymmetric core, Dynamic core CPU
- Distributed Systems Non-function quality attributes
- Distributed Systems – Types of Transparency
- Types of modularity- Soft, Enforced

| September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019]<br>School of Engineering and Technology, University of Washington - Tacoma | L1.60 |

60

# QUESTIONS

September 25, 2019

TCSS562: Software Engineering for Cloud Computing [Fall 2019]
School of Engineering and Technology, University of Washington - Tacoma

L1.61

61