Parallel computing

• What is it?
  – multiple processors cooperating to solve a single problem
  – hopefully faster than using a single processor!

• Why is it needed?
  – greater compute performance
  – shorter time to solution
Where is performance needed?

- sometimes performance is required in time-critical tasks
  - timely and accurate weather forecast
  - obstacle detection for self driving cars

- sometimes performance gives a competitive advantage
  - from Walmart to Wall Street
    - data mining of trends
    - delivery logistics
    - real-time analytics (high frequency trading)
  - engineering, manufacturing, and pharmaceuticals
    - vehicle crash simulations, material properties prediction, drug design

- sometimes performance is the only way to answer a question
  - scientific progress using mathematical modeling and numerical simulation
    - human genome assembly
    - computational science and the timely Nobel prize
Why can’t we just build a faster single processor?

- Moore’s “Law”
  - processor performance per $ doubles every two years!
Transistor miniaturization and performance

- **Dennard scaling**
  - transistor switching power $\propto$ transistor size
  - shrinking transistor size
    - decreases switching power
    - decreases switching time (higher clock frequency)
    - increases number of transistors per unit area
  - so for the same power and space budget we get
    - faster arithmetic operations
    - pipelined arithmetic
    - more and larger caches
  $\Rightarrow$ increased performance

- **Limits to Dennard Scaling**
  - as transistor size approaches quantum mechanical limits
    - increasing leakage current
    - exponential power increase!

Source: Patrick Gelsinger, Intel®
Parallelism is now the principal source of performance

- Processor evolution after 2004 (Intel)
  - multiple cores per socket
  - lower per-core performance
  - similar power per chip
    - per-core “turbo” mode
  - vector units and larger caches
  - multiple and higher performance off-chip memory interfaces

- Moore’s “law”
  - performance per socket is still increasing but no longer exponentially
  - power/cooling per socket is the limiting factor

- Factors limiting parallel computing
  - overall system power
  - inconveniently slow speed of signal propagation!
Parallel computing at various scales

- **Modern processor core**
  - pipelined, superscalar, multiword ALUs
  - L1 and L2 caches

- **Socket**
  - multiple cores (4 – 64)
  - L3 cache

- **Accelerators**
  - Nvidia V100 GPU (2560 arithmetic units)

- **Node**
  - up to 4 sockets
  - up to 8 accelerators
  - fast local interconnect

- **Cluster**
  - tens to thousands of nodes
  - high speed interconnection network

---

64-bit floating point ops per second (FLOPS)

- **Giga** $10^9$
- **Tera** $10^{12}$
- **Peta** $10^{15}$
- **Exa** $10^{18}$
# Top supercomputers (2022)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Rmax $\times 10^{18}$</th>
<th>Rpeak $\times 10^{18}$</th>
<th>Location</th>
<th>Manufacturer</th>
<th>Cores</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Frontier</td>
<td>1.1</td>
<td>1.7</td>
<td>Oak Ridge Natl Lab</td>
<td>HPE CRAY</td>
<td>8,730,112</td>
<td>2022</td>
</tr>
<tr>
<td>2</td>
<td>Fugaku</td>
<td>0.4</td>
<td>0.6</td>
<td>RIKEN, Japan</td>
<td>Fujitsu</td>
<td>7,630,848</td>
<td>2020</td>
</tr>
</tbody>
</table>

**Sunway TaihuLight**
National Research Center for Parallel Computer Engineering and Technology in Wuxi, CN
What are the parallel computing challenges?

- Parallel computing involves many aspects of computer science
  - new algorithms must be designed
  - new algorithm analysis techniques must be used
  - new programming models and languages must be learned
  - memory operation and performance must be understood
  - communication costs and network behavior must be considered
  - different operating systems, services, and I/O
  - different debugging and performance monitoring
  - novel and continuously changing hardware
  - …
Summary: Why study parallel computing?

- It is *useful* and it is *used*
- It involves *new algorithms* and *analytic techniques*
- **Future computing** will increasingly be predicated on the use of parallelism
- To understand *what is feasible and what is not*
How else is parallelism used?

• Parallelism may improve reliability
  – high availability
  – high assurance

• Parallelism may be inherent in the problem
  – (G)UIs
  – distributed systems
    • >80 processors in a modern luxury car

• Parallelism is a simple load scaling approach
  – server farms

... but these are not the focus of this course!
Parallel Computing vs. Distributed Computing

• **Parallel Computing (COMP 633)**
  – Multiple processors cooperating to solve a single problem
  – Key concepts
    • Design and analysis of scalable parallel algorithms
    • Programming models
    • Systems architecture and hardware characteristics
    • Performance analysis, prediction, and measurement

• **Distributed Systems (COMP 734)**
  – Providing reliable services to multiple users via a system consisting of multiple processors and a network
  – Key concepts
    • Services & protocols
    • Reliability
    • Security
    • Scalability
Parallel Computing vs. Concurrent Algorithms

- **Parallel Computing (COMP 633)**
  - Multiple processors cooperating to solve a single problem
  - Key concepts
    - Design and analysis of scalable parallel algorithms
    - Programming models
    - Systems architecture and hardware characteristics
    - Performance analysis, prediction, and measurement

- **Distributed and Concurrent Algorithms (COMP 735)**
  - Specification of fundamental algorithms and proofs of their correctness and performance properties
    - Mutual exclusion
    - Readers and writers
  - Key concepts
    - Lower and upper bounds, impossibility proofs
    - Formal methods
    - Wait-free and lock-free methods
Course Introduction

• Organization and content of this course
  – prerequisites
  – source materials
  – course grading
  – what will be studied

• Introductory examples
Organization of the course

• Course web page
  – Syllabus
    • Prerequisites
    • Learning Objectives
    • Honor Code
    • Topics
  – Source materials
  – Computer usage

• Reading assignment for next time
  – Parallel Random Access Machine (PRAM) model and algorithms
    • sections 1, 2, 3.1 (pp 1-8)

• Sign up for Piazza
  – using link on web page
What will we study?

• Course is organized around different models of parallel computation
  – shared memory models [main focus]
    • PRAM
    • Loop-level parallelism, threads, tasks (OpenMP, Cilk)
    • Accelerators (Cuda)
  – distributed memory models [secondary focus]
    • bulk-synchronous processing (BSP, UPC), message passing (MPI)
  – data-intensive models [cursory treatment]
    • MapReduce/Hadoop, spark

• For each model we examine
  – algorithm design techniques
  – cost model and performance prediction
  – how to express programs
  – hardware and software support
  – performance analysis
  – advantages and limitations of the model including realism, applicability and tractability

by studying some examples in detail
Let’s try it right now!

- **Vector summation**
  - given vector $V[1..n]$ compute $s = \sum_{i=1}^{n} V_i$
  
  e.g. for $n = 8$
  
  $$s = V_1 + V_2 + \ldots + V_7 + V_8$$

- **sequential algorithm**
  - $n-1$ additions: optimal
  
  - e.g. sum from left to right
  
  - sequential running time
    
    - $T(n) = O(n)$
Example 1: DAG model of parallel computation

• A program $P = (V, E)$ is a tree where
  
  leaf vertices in $V$  \sim  values
  interior vertices in $V$  \sim  operations
  edges $E$  \sim  evaluation dependences

\[ V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 \]
Execution of a DAG “program”

- **definition**
  - an operation is **ready** if all of its children are leaves

- **parallel execution step**
  - simultaneously evaluate all ready operations and replace each with its value

- **program execution**
  - perform parallel execution steps until no operations remain
Complexity metrics for DAG model

- **Work complexity** of a DAG program
  - total number of operations performed
  - \( \text{work} = \# \text{interior vertices in DAG} \)

- **Step complexity** of a DAG program
  - number of execution steps
  - \( \text{steps} = \text{length of longest path in DAG} \)

<table>
<thead>
<tr>
<th></th>
<th>work</th>
<th>steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog 1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Prog 2</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>
Asymptotic complexity metrics for DAG model

• Asymptotic complexity
  – problem size \( n \)
  – \( W(n) \) asymptotic work complexity
  – \( S(n) \) asymptotic step complexity
  – \( T^*(n) \) optimal asymptotic sequential time complexity

• Definition
  – A DAG program is work efficient if \( W(n) = O( T^*(n) ) \)

<table>
<thead>
<tr>
<th></th>
<th>( W(n) )</th>
<th>( S(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog 1</td>
<td>( O(n) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Prog 2</td>
<td>( O(n) )</td>
<td>( O(\lg n) )</td>
</tr>
</tbody>
</table>
Asymptotic complexity metrics for DAG model

- Asymptotic complexity
  - problem size $n$
  - $W(n)$ asymptotic work complexity
  - $S(n)$ asymptotic step complexity
  - $T^*(n)$ optimal asymptotic sequential time complexity

- Definition
  - A DAG program is work efficient if $W(n) = O(T^*(n))$

<table>
<thead>
<tr>
<th></th>
<th>$W(n)$</th>
<th>$S(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog 1</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Prog 2</td>
<td>$O(n)$</td>
<td>$O(lg\ n)$</td>
</tr>
</tbody>
</table>
Execution of DAG programs with fixed resources

- At most $p$ operations evaluated simultaneously in a DAG program $H$
  - models execution using $p$ “processors”

- **Definition**
  - $T_p(n)$ is the time to execute $H$ using $p$ processors
    - $n$ - problem size
    - $p$ - maximum number of nodes that may be evaluated concurrently in each timestep
  - $T_1(n) = W(n)$
  - $T_∞(n) = S(n)$

But what is $T_2(8)$ for prog 2?
Evaluation order

- Determining evaluation order to minimize $T_p(n)$ is NP-hard!

- Simple non-optimal greedy evaluation order
  - at each step
    - p or fewer operations ready ⇒ evaluate all ready nodes
    - more than p operations ready ⇒ evaluate any p ready nodes

- Running time using greedy strategy can be bounded

$$\left\lfloor \frac{W(n)}{p} \right\rfloor \leq T_p(n) \leq \left\lfloor \frac{W(n)}{p} \right\rfloor + S(n)$$
“fast” parallel programs give good speedup

- Definition
  - a fast parallel program has step complexity $S(n)$ that is asymptotically smaller than work complexity $W(n)$
  \[
  S(n) = o(W(n)) \quad \text{means} \quad \lim_{n \to \infty} \frac{S(n)}{W(n)} = 0
  \]

- For a fixed number of processors $p$, a fast parallel program gives better speedup as problem size $n$ is increased
  \[
  \left[\frac{W(n)}{p}\right] \leq T_p(n) \leq \left[\frac{W(n)}{p}\right] + S(n)
  \]
  \[
  \lim_{n \to \infty} T_p(n) = O\left(\frac{W(n)}{p}\right)
  \]
  - asymptotically optimal speedup on large problems!
But can’t speedup indefinitely

- You can’t speed up a parallel algorithm indefinitely using more processors
  - for a fixed problem size $n$, step complexity limits speedup
    \[ T_P(n) = O\left(\frac{W(n)}{p} + S(n)\right) \]

- prog 1 cannot be sped up at all using more processors!
  - $W(n) = \Theta(n)$
  - $S(n) = \Theta(n)$

- prog 2 requires $\Omega(\lg n)$ steps regardless of the number of processors
  - $W(n) = \Theta(n)$
  - $S(n) = \Theta(\lg n)$
Consequences: work efficiency is paramount

- A parallel program H that is \textit{not} work efficient loses asymptotically!
  - for any given p, there exists a problem size \( n_0 \) such that
    - an efficient sequential program using one processor on problems of size \( n > n_0 \) is faster than the parallel program H using p processors!
  - it doesn’t help if H is \textit{fast}
  - worst results on large problems!

\[
T_p(n) = O\left(\left\lceil \frac{W(n)}{p} \right\rceil + S(n) \right)
\]
Example 2: Message-passing model

- p processors connected in a ring
  - each processor
    - runs the same program
    - has a unique processor id $0 \leq i < p$
    - can send a value to its left neighbor

- summation of $V[0..p-1]$ using p processors
  - assume $V_i$ is in $s$ on processor $i$ at start
  - program terminates with $s = \sum_{j \in 0..p-1} V_j$ on processor 0
Summation program

```
for h := 1 to (\lg p) 
  x := s
  for j := 1 to 2^{h-1} do 
    send value of x to left and receive new value for x from right 
    s := s + x
```

Example: p = 4

```
s = \[ V_0 \quad V_1 \quad V_2 \quad V_3 \]
```

\[ h = 1, \quad s = \quad V_0 + V_1 \quad V_2 + V_3 \]

\[ h = 2, \quad s = \quad V_0 + V_1 + V_2 + V_3 \]
Analysis of summation program

```
for h := 1 to (lg p)
x := s
    for j := 1 to 2^{h-1} do
        send value of x to left and receive new value for x from right
s := s + x
```

• Let
  – $t_a$ time to perform addition
  – $t_c$ time to perform communication

$$T_p(n) = \sum_{h=1}^{\lg p} (t_a + 2^{h-1} t_c)$$

$$= (\lg p) \cdot t_a + (p - 1) \cdot t_c$$

• Is this good performance?
What’s wrong?

• poor network?
  – network *diameter* is large thus values have to travel far
  – so communication time is huge compared to addition time
  – a smaller diameter network might do better

• bad communication strategy?
  – “cut-through” routing would be superior

• poor utilization of the processors?
  – only a few processors are performing useful additions!

• problem size too small?
  – this is the real problem!
Summation of $n$ values with $p$ processors

- Each processor holds $n/p$ values

```
s := sum of $n/p$ values in this processor
for h := 1 to (lg p)
x := s
    for j := 1 to $2^{h-1}$ do
        send value of $x$ to left and receive new value for $x$ from right
    s := s + x
```

Example:

$n = 8$
$p = 4$

```
\begin{align*}
V_0 & \quad V_2 & \quad V_4 & \quad V_6 \\
V_1 & \quad V_3 & \quad V_5 & \quad V_7
\end{align*}
```
Summation of $n$ values using $p$ processors

- **Analysis**

  $$T_p(n) = \left(\frac{n}{p} - 1\right) \cdot t_a + (\lg{p}) \cdot t_a + (p-1) \cdot t_c$$

  $$\approx \left(\frac{n}{p}\right) \cdot t_a + (\lg{p}) \cdot t_a + p \cdot t_c$$

  \begin{align*}
  &\text{speedup} & \text{overhead}
  \end{align*}

- **excellent performance can be achieved**
  - for arbitrary $p$, $t_a$, $t_c$
  - asymptotically optimal speedup with sufficiently large $n$
    - overheads and inefficiencies can be amortized!