Outline

- Performance scalability
- Analytical performance measures
- Amdahl’s law and Gustafson-Barsis’ law
- Chapters 1 and 2 Structured Parallel Programming
Logistics

- Lab organization
  - 9:00-11:00 will be for last lab’s assignments
  - 11:00-12:00 will be for new programming technology

- Programming lab plan
  - 5 weeks of learning programming technology: ISPC, OpenMP, Cilk Plus, MPI, OpenACC
  - 5 weeks of working on projects

- Lab on ISPC seems to go well last week
  - Continue working with exercises

- Lab this week will focus on learning OpenMP
  - Look at the OpenMP tutorials online
What is Performance?

- In computing, performance is defined by 2 factors
  - Computational requirements (what needs to be done)
  - Computing resources (what it costs to do it)
- Computational problems translate to requirements
- Computing resources interplay and tradeoff

\[
\text{Performance} \sim \frac{1}{\text{Resources for solution}}
\]

Hardware  Time  Energy  ... and ultimately  Money
Why do we care about Performance?

- Performance itself is a measure of how well the computational requirements can be satisfied.
- We evaluate performance to understand the relationships between requirements and resources.
  - Decide how to change “solutions” to target objectives.
- Performance measures reflect decisions about how and how well “solutions” are able to satisfy the computational requirements.

“The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible.”

Charles Babbage, 1791 – 1871
What is Parallel Performance?

- Here we are concerned with performance issues when using a parallel computing environment
  - Performance with respect to parallel computation
- Performance is the *raison d’être* for parallelism
  - Parallel performance versus sequential performance
  - If the “performance” is not better, parallelism is not necessary (Is it?)
- *Parallel processing* includes techniques and technologies necessary to compute in parallel
  - Hardware, networks, operating systems, parallel libraries, languages, compilers, algorithms, tools, …
- Parallelism must deliver performance
  - How? How well?
Performance Expectation (Loss)

- If each processor is rated at $k$ MFLOPS and there are $p$ processors, should we see $k^*p$ MFLOPS performance?
- If it takes 100 seconds on 1 processor, shouldn’t it take 10 seconds on 10 processors?
- Several causes affect performance
  - Each must be understood separately
  - But they interact with each other in complex ways
    - Solution to one problem may create another
    - One problem may mask another
- Scaling (system, problem size) can change conditions
- Need to understand performance space
Embarrassingly Parallel Computations

- An embarrassingly parallel computation is one that can be obviously divided into completely independent parts that can be executed simultaneously
  - In a truly embarrassingly parallel computation there is no interaction between separate processes
  - In a nearly embarrassingly parallel computation results must be distributed and collected/combined in some way

- Embarrassingly parallel computations have potential to achieve maximal speedup on parallel platforms
  - If it takes \( T \) time sequentially, there is the potential to achieve \( T/P \) time running in parallel with \( P \) processors
  - What would cause this not to be the case always?
**Scalability**

- A program can scale up to use many processors
  - What does that mean?
- How do you evaluate scalability?
- How do you evaluate scalability goodness?
- Comparative evaluation
  - If double the number of processors, what to expect?
  - Is scalability linear?
- Use parallel efficiency measure
  - Is efficiency retained as problem size increases?
- Apply performance metrics
Performance and Scalability

- Evaluation
  - *Sequential* runtime ($T_{seq}$) is a function of
    ◆ problem size and architecture
  - *Parallel* runtime ($T_{par}$) is a function of
    ◆ problem size and parallel architecture
    ◆ # processors used in the execution
  - Parallel performance affected by
    ◆ algorithm + architecture

- Scalability
  - Ability of parallel algorithm to achieve performance gains proportional to the number of processors and the size of the problem
Performance Metrics and Formulas

- $T_1$ is the execution time on a single processor
- $T_p$ is the execution time on a $p$ processor system
- $S(p)$ ($S_p$) is the speedup
  \[ S(p) = \frac{T_1}{T_p} \]
- $E(p)$ ($E_p$) is the efficiency
  \[ \text{Efficiency} = \frac{S_p}{p} \]
- $Cost(p)$ ($C_p$) is the cost
  \[ Cost = p \times T_p \]
- Parallel algorithm is cost-optimal
  - Parallel time = sequential time ($C_p = T_1$, $E_p = 100\%$)
Amdahl’s Law (Fixed Size Speedup)

- Let $f$ be the fraction of a program that is sequential
  - $1-f$ is the fraction that can be parallelized
- Let $T_1$ be the execution time on 1 processor
  $$T_1 = fT_1 + (1-f)T_1$$
- Let $T_p$ be the execution time on $p$ processors
- $S_p$ is the speedup
  $$S_p = T_1 / T_p$$
  $$= T_1 / (fT_1 + (1-f)T_1 /p))$$
  $$= 1 / (f + (1-f)/p))$$
- As $p \to \infty$
  $$S_p = 1 / f$$
Amdahl’s Law and Scalability

- **Scalability**
  - Ability of parallel algorithm to achieve performance gains proportional to the # processors and the size of the problem

- **When does Amdahl’s Law apply?**
  - When the problem size is **fixed**
  - *Strong scaling* ($p\to\infty$, $S_p = S_\infty \to 1/f$)
  - Speedup bound is determined by the degree of sequential execution time in the computation, not # processors!!!
  - Uhh, this is not good … Why?
  - Perfect efficiency is hard to achieve

- See original paper by Amdahl on webpage
Amdahl’s Law (Fixed Size Speedup)

Only 20x speedup with 64K Processors!

Diminishing returns … Why?
Gustafson-Barsis’ Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time

- Assume parallel time is kept constant
  - \( T_p = C = (f+(1-f)) \times C \)
  - \( f_{seq} \) is the fraction of \( T_p \) spent in sequential execution
  - \( f_{par} \) is the fraction of \( T_p \) spent in parallel execution

- What is the execution time on one processor?
  - Let \( C=1 \), then \( T_s = f_{seq} + p(1-f_{seq}) = 1 + (p-1)f_{par} \)

- What is the speedup in this case?
  - \( S_p = T_s / T_p = T_s / 1 = f_{seq} + p(1-f_{seq}) = 1 + (p-1)f_{par} \)
Gustafson-Barsis’ Law and Scalability

- **Scalability**
  - Ability of parallel algorithm to achieve performance gains proportional to the number of processors and the size of the problem

- **When does Gustafson’s Law apply?**
  - When the problem size can increase as the number of processors increases
  - *Weak scaling* \( S_p = 1 + (p-1)f_{par} \)
  - Speedup function includes the number of processors!!!
  - Can maintain or increase parallel efficiency as the problem scales

- See original paper by Gustafson on webpage
Amdahl versus Gustafson-Baris

Total time = $T_{seq} + T_{par}$

= $T_{seq} + (1-f)T_1/p$

does not change
Amdahl versus Gustafson-Baris

Gustafson-Baris

\[ T_{seq} = f_{seq} T_1 + P f_{par} T_1 \]

\[ T_{par} = ??? \]

\( T_{seq} \text{ recomputed for every } P !!! \)
Amdahl versus Gustafson-Baris

Gustafson-Baris

$T_{seq} = f_{seq} T_1 + P f_{par} T_1$

$T_{par} \sim f_{seq} T_1 + f_{par} T_1$

Why?

$T_{seq}$ recomputed for every $P$ !!!
Issues with Amdahl and G-B Laws

- Do you see any issues?
- Think about what is actually being computed
  - What do you know about it?
- Is there any merit in reality?
  - How realistic is fixed speedup?
  - Are there problems where you see scaled speedup being beneficial?
Top 500 Benchmarking Methodology

- Listing of the world’s 500 most powerful computers
- Yardstick for high-performance computing (HPC)
  - $R_{\text{max}}$: maximal performance Linpack benchmark
    - dense linear system of equations ($Ax = b$)
- Data listed
  - $R_{\text{peak}}$: theoretical peak performance
  - $N_{\text{max}}$: problem size needed to achieve $R_{\text{max}}$
  - $N_{1/2}$: problem size needed to achieve 1/2 of $R_{\text{max}}$
  - Manufacturer and computer type
  - Installation site, location, and year
- Updated twice a year at SC and ISC conferences
### Top 10 (November 2015)

<table>
<thead>
<tr>
<th>RANK</th>
<th>SITE</th>
<th>SYSTEM</th>
<th>CORES</th>
<th>RMAX (TFLOP/S)</th>
<th>RPEAK (TFLOP/S)</th>
<th>POWER (KW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>National Super Computer Center in Guangzhou, China</td>
<td>Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT</td>
<td>3,120,000</td>
<td>33,862.7</td>
<td>54,902.4</td>
<td>17,808</td>
</tr>
<tr>
<td>2</td>
<td>DOE/SC/Oak Ridge National Laboratory, United States</td>
<td>Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.</td>
<td>560,640</td>
<td>17,590.0</td>
<td>27,112.5</td>
<td>8,209</td>
</tr>
<tr>
<td>3</td>
<td>DOE/NN5A/LLNL, United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM</td>
<td>1,572,864</td>
<td>17,173.2</td>
<td>20,132.7</td>
<td>7,890</td>
</tr>
<tr>
<td>4</td>
<td>RIKEN Advanced Institute for Computational Science (AICS), Japan</td>
<td>K computer, SPARC64 VIII fx 2.0GHz, Tofu interconnect Fujitsu</td>
<td>705,024</td>
<td>10,510.0</td>
<td>11,280.4</td>
<td>12,660</td>
</tr>
<tr>
<td>5</td>
<td>DOE/SC/Argonne National Laboratory, United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM</td>
<td>786,432</td>
<td>8,586.6</td>
<td>10,066.3</td>
<td>3,945</td>
</tr>
<tr>
<td>6</td>
<td>DOE/NN5A/LANL/SNL, United States</td>
<td>Trinity - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.</td>
<td>301,056</td>
<td>8,100.9</td>
<td>11,078.9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Swiss National Supercomputing Centre (CSCS), Switzerland</td>
<td>Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect, NVIDIA K20x Cray Inc.</td>
<td>115,984</td>
<td>6,271.0</td>
<td>7,788.9</td>
<td>2,325</td>
</tr>
<tr>
<td>8</td>
<td>HLRS - Höchstleistungsrechenzentrum Stuttgart, Germany</td>
<td>Hazel Hen - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect Cray Inc.</td>
<td>185,088</td>
<td>5,640.2</td>
<td>7,403.5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>King Abdullah University of Science and Technology, Saudi Arabia</td>
<td>Shaheen II - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.</td>
<td>196,608</td>
<td>5,537.0</td>
<td>7,235.2</td>
<td>2,834</td>
</tr>
<tr>
<td>10</td>
<td>Texas Advanced Computing Center/Univ. of Texas, United States</td>
<td>Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell</td>
<td>462,462</td>
<td>5,168.1</td>
<td>8,520.1</td>
<td>4,510</td>
</tr>
</tbody>
</table>
# Top 10 (November 2014)

<table>
<thead>
<tr>
<th>RANK</th>
<th>SITE</th>
<th>SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>DOE/SC/Oak Ridge National Laboratory, United States</td>
<td>Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x</td>
</tr>
<tr>
<td>3</td>
<td>DOE/NNSA/LLNL, United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM</td>
</tr>
<tr>
<td>4</td>
<td>RIKEN Advanced Institute for Computational Science [AICS], Japan</td>
<td>K computer, SPARC64 VIII fx 2.0GHz, Toha interconnect, Fujitsu</td>
</tr>
<tr>
<td>5</td>
<td>DOE/SC/Argonne National Laboratory, United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM</td>
</tr>
<tr>
<td>6</td>
<td>Swiss National Supercomputing Centre [CSCS], Switzerland</td>
<td>Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect, NVIDIA K20x, Cray Inc.</td>
</tr>
<tr>
<td>7</td>
<td>Texas Advanced Computing Center/Univ. of Texas, United States</td>
<td>Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell</td>
</tr>
<tr>
<td>8</td>
<td>Forschungszentrum Juelich [FZJ], Germany</td>
<td>JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect, IBM</td>
</tr>
<tr>
<td>9</td>
<td>DOE/NNSA/LLNL, United States</td>
<td>Vulcan - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect, IBM</td>
</tr>
<tr>
<td>10</td>
<td>Government, United States</td>
<td>Cray CS-Storm, Intel Xeon E5-2660v2 10C 2.20GHz, Infiniband FDR, Nvidia K40 Cray Inc.</td>
</tr>
</tbody>
</table>

Same systems!!!

Different architectures
# Top 10 (November 2013)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Cores</th>
<th>Rmax (TFlop/s)</th>
<th>Rpeak (TFlop/s)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>National Super Computer Center in Guangzhou, China</td>
<td>Tianhe-2 (MilkyWay-2) - TH-V8-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P</td>
<td>3,120,000</td>
<td>33,662.7</td>
<td>54,902.4</td>
<td>17,808</td>
</tr>
<tr>
<td>2</td>
<td>DOE/SC/Oak Ridge National Laboratory, United States</td>
<td>Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x</td>
<td>560,640</td>
<td>17,590.0</td>
<td>52,171.2</td>
<td>8,209</td>
</tr>
<tr>
<td>3</td>
<td>DOE/NNSA/LLNL, United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM</td>
<td>1,572,664</td>
<td>17,173.2</td>
<td>20,132.7</td>
<td>7,890</td>
</tr>
<tr>
<td>4</td>
<td>RIKEN Advanced Institute for Computational Science (AICS), Japan</td>
<td>K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect</td>
<td>705,024</td>
<td>10,510.0</td>
<td>11,280.4</td>
<td>12,660</td>
</tr>
<tr>
<td>5</td>
<td>DOE/SC/Argonne National Laboratory, United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM</td>
<td>786,432</td>
<td>8,586.6</td>
<td>10,066.3</td>
<td>3,945</td>
</tr>
<tr>
<td>6</td>
<td>Swiss National Supercomputing Centre (CSCS), Switzerland</td>
<td>Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect, NVIDIA K20x</td>
<td>115,984</td>
<td>6,271.0</td>
<td>7,788.9</td>
<td>2,325</td>
</tr>
<tr>
<td>7</td>
<td>Texas Advanced Computing Center/Univ. of Texas, United States</td>
<td>Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P</td>
<td>462,482</td>
<td>5,168.1</td>
<td>8,520.1</td>
<td>4,510</td>
</tr>
<tr>
<td>8</td>
<td>Forschungszentrum Juelich (FZJ), Germany</td>
<td>JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom interconnect</td>
<td>458,752</td>
<td>5,008.9</td>
<td>5,872.0</td>
<td>2,301</td>
</tr>
<tr>
<td>9</td>
<td>DOE/NNSA/LLNL, United States</td>
<td>Vulcan - BlueGene/Q, Power BQC 16C 1.600GHz, Custom interconnect</td>
<td>393,216</td>
<td>4,293.3</td>
<td>5,033.2</td>
<td>1,972</td>
</tr>
<tr>
<td>10</td>
<td>Leibniz Rechenzentrum, Germany</td>
<td>SuperMUC - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR</td>
<td>147,456</td>
<td>2,897.0</td>
<td>3,185.1</td>
<td>3,423</td>
</tr>
</tbody>
</table>

Same as 2014!
Top 500 – Performance (November 2015)
DAG Model of Computation

- Think of a program as a directed acyclic graph (DAG) of tasks
  - A task can not execute until all the inputs to the tasks are available
  - These come from outputs of earlier executing tasks
  - DAG shows explicitly the task dependencies
- Think of the hardware as consisting of workers (processors)
- Consider a greedy scheduler of the DAG tasks to workers
  - No worker is idle while there are tasks still to execute
Work-Span Model (see SPP book)

- \( T_P = \text{time to run with } P \text{ workers} \)
- \( T_1 = \text{work} \)
  - Time for serial execution
    - execution of all tasks by 1 worker
  - Sum of all work
- \( T_\infty = \text{span} \)
  - Time along the critical path
- Critical path
  - Sequence of task execution (path) through DAG that takes the longest time to execute
  - Assumes an infinite # workers available
  - Must execute the critical path in a serial manner
  - Critical path does not parallelize (by definition)
Work-Span Example

- Let each task take 1 unit of time
- DAG at the right has 7 tasks
  - \( T_1 = 7 \)
    - All tasks have to be executed
    - Tasks are executed in a serial order
    - Can they execute in any order? Why?
  - \( T_\infty = 5 \)
    - Time along the critical path
    - In this case, it is the longest pathlength of any task order that maintains necessary dependencies
- What is \( T_p \)?
Lower/Upper Bound on Greedy Scheduling

- Suppose we only have $P$ workers
- We can write a work-span formula to derive a lower bound on $T_P$
  - $\max(T_1/P, T_\infty) \leq T_P$
- $T_\infty$ is the best possible execution time!
  - Why? When?
- **Brent’s Lemma** derives an upper bound
  - Capture the additional cost executing the other tasks not on the critical path
  - Assume can do so without overhead
  - $T_P \leq (T_1 - T_\infty) / P + T_\infty$
Consider Brent’s Lemma for 2 Processors

- $T_1 = 7$
- $T_\infty = 5$
- $T_2 \leq (T_1 - T_\infty) / P + T_\infty$
  \[ \leq (7 - 5) / 2 + 5 \]
  \[ \leq 6 \]
- What is $T_2$ really here?
Amdahl was an optimist!

Brent’s Lemma under estimates speedup
Estimating Running Time

- Scalability requires that $T_\infty$ be dominated by $T_1$

$$T_P \approx T_1 / P + T_\infty \text{ if } T_\infty << T_1$$

- Increasing work ($T_1$) hurts parallel execution proportionately
- The span impacts scalability, even for finite $P$
Parallel Slack

- Sufficient parallelism implies linear speedup

\[ T_p \approx \frac{T_1}{P} \quad \text{if} \quad \frac{T_1}{T_\infty} \gg P \]

- Scaling the problem is a way to increase the amount of parallelism available
Little’s Law

- C: concurrency (not parallelism here)
- R: throughput
- L: latency
- Little’s Law relates C, R, and L

\[ C = R \times L \]

- Extra concurrency can be used to improve throughput when there are long latency tasks
  - Hiding latency
- Parallelizing to hide latency and increase throughput requires over-decomposing the problem to generate extra concurrency per physical unit
  - Hardware multithreading
Asymptotic Complexity

- Time complexity of an algorithm summarizes how the execution time grows with input size.
- Space complexity summarizes how memory requirements grow with input size.
- Standard work-span model considers only computation, not communication or memory.
- Asymptotic complexity is a strong indicator of performance on large-enough problem sizes and reveals an algorithm’s fundamental limits.
Definitions for Asymptotic Notation

- Let \( T(N) \) mean the execution time of an algorithm.

- **Big O notation** (set of \( f() \) with upper bound)
  - \( T(N) \) is a member of \( O(f(N)) \) means that
    \[ T(N) \leq c \cdot f(N) \text{ for constant } c \text{ and } N \geq N_0 \text{ for some } N_0 \]

- **Big Omega notation** (set of \( f() \) with lower bound)
  - \( T(N) \) is a member of \( \Omega(f(N)) \) means that
    \[ T(N) \geq c \cdot f(N) \text{ for constant } c \text{ and } N \geq N_0 \text{ for some } N_0 \]

- **Big Theta notation** (set of \( f() \) with upper/lower)
  - \( T(N) \) is a member of \( \Theta(f(N)) \) means that
    \[ c_1 \cdot f(n) \leq T(N) < c_2 \cdot f(N) \text{ for constants } c_1 \text{ and } c_2 \text{ and } N \geq N_0 \text{ for some } N_0 \]