Lecture 4

Characterizing performance
Parallel image processing and generation
Announcements

• The meeting room for section has changed
  **EBU3B 2154** (This room)
Performance measurement and characterization
Measures of Performance

• Why do we measure performance?
• Measure of performance
  – Completion time
  – Processor time product
    Completion time \times \# \text{processors}
  – Throughput: amount of work that can be accomplished in a given amount of time
  – Relative performance: given a reference architecture or implementation
    AKA \textit{Speedup}
Parallel Speedup and Efficiency

• How much of an improvement did our parallel algorithm obtain over the serial algorithm?

• Define the **parallel speedup**, \( S_p \)

\[
S_p = \frac{\text{Running time of the best serial program on 1 processor}}{\text{Running time of the parallel program on } P \text{ processors}}
\]

• \( T_1 \) is defined as the running time of the “best serial algorithm”

• In general: *not* the running time of the parallel algorithm on 1 processor

• **Definition:** Parallel efficiency \( E_p = S_p / P \)
What can go wrong with speedup?

• Speedup is not always an accurate way to compare different algorithms or machines

• We might be able to obtain a better speedup at the cost of a longer running time

• If we have a processor-time allocation the bottom line is running time $T_p$ or the 
  \textit{space time cost} $P T_p$
Superlinear speedups

• We have a *super-linear* speedup when
  \[ S_p > P \Rightarrow E_p > 1 \]

• Super-linear speedups are often an artifact of inappropriate measurement technique

• Where there is a super-linear speedup, a better serial algorithm may be lurking

• JIT effect – spurious?
What’s wrong with speedup?

• Not always an accurate way to compare different algorithms....

• .. or the same algorithm running on different machines

• For an individual user the bottom line is running time $T_p$ or the space time cost $P T_p$

• Can we obtain a better speedup at the cost of a longer running time?
Scalability

• We want performance to scale linearly with the number of processors

• Difficulties
  ► Serial sections: code that runs on only one processor
  ► “Non-productive” work associated with parallel execution, e.g. communication
  ► Load imbalance: uneven work assignments over the processors

• Some algorithms present intrinsic barriers to scalability leading to alternatives
  \[ \text{for } i=0:n-1 \text{ } \text{sum} = \text{sum} + x[i] \]
• Limits scalability

• Let $f$ = the fraction of $T_1$ that runs serially

• $T_1 = f \times T_1 + (1-f) \times T_1$

• $T_P = f \times T_1 + (1-f) \times T_1 / P$

  Thus $S_P = 1/[f + (1 - f)/p]$

• As $P \to \infty$, $S_P \to 1/f$

• This is known as Amdahl's Law (1967)
Amdahl’s law (1967)

- A serial section limits scalability
- Let $f = \text{fraction of } T_1 \text{ that runs serially}$
- *Amdahl's Law (1967)*: As $P \to \infty$, $S_P \to 1/f$
Weak scaling

- Is Amdahl’s law pessimistic?
- Observation: Amdahl’s law assumes that the workload \( W \) remains fixed
- But parallel computers are used to tackle more ambitious workloads
- Let’s increase \( W \) with \( P \); we have weak scaling (strong scaling if \( W = \text{constant} \))
  \( W \) increases with \( P \)
  \( f \) often decreases with \( W \)
Measuring performance
Challenges to measuring performance

• Reproducibility
  – Transient system operating conditions
  – Differing systems or program configuration

• Measurements are imprecise
  – “Heisenberg uncertainty principle:” measurement technique may affect performance
  – Overheads and inaccuracy

• Explain anomalous behavior, but ignore anomalies that are not significant
Complications

• Cost of measuring a full run is prohibitive
  – Ignore startup code if you plan to run for a much longer time in production

• Transient behavior
  – Repeat your measurements
  – “Warm up” the code before collecting measurements
  – Ignore outliers unless their behavior is important to you
  – Average time, maximum time, minimum time?
Measurement collection

• Report the *best* timings
  ► Repeat results 3 to 5 times until at least 2 measures agree to within... 5%, 10%
  ► Report the minimum time
• Also report outliers
• A scatter plot or error bar can be useful
Timing collection

• Measures of time
  ▶ Elapsed, or “wall clock” time
  ▶ CPU time = system + user time
  ▶ Overhead, resolution, and quantization effects

• Measurement tools
  ▶ Unix `time` command does a reasonable job for long-running programs
  ▶ `System.currentTimeMillis()`
  ▶ Often platform dependent, especially library routines
Enable others to reproduce your results

• Builds confidence within a community
• Report where you ran, software versions, processor, etc.
  ➤ `uname -a`
    - Linux ieng6-203.ucsd.edu 2.6.18-164.el5PAE #1 SMP Thu Sep 3 04:10:44 EDT 2009
      i686 i686 i386 GNU/Linux
  ➤ `javac -version`
    - javac 1.5.0_1
  ➤ **Access processor configuration information**
    - On linux: `/proc/cpuinfo`
    - `vendor_id` : GenuineIntel
    - `cpu family` : 6
    - `model` : 15
    - `model name` : Intel(R) Xeon(R) CPU X5355 @ 2.66GHz
    - `stepping` : 7
    - `cpu MHz` : 2660.237
    - `cache size` : 4096 KB
    - See http://www.intel.com/products/processor_number
Computing the Mandelbrot set
The Mandelbrot set

- For which points \( c \) in the complex plane does the following iteration remain bounded?
  \[ z_{k+1} = z_k^2 + c \]
  \( c \) is a complex number, \( z_0 = 0 \)
- When \( c=0 \), all points lay within a unit disk \( |z| \leq 1 \)
- If \( |z| \geq 2 \), the iteration is guaranteed to diverge to \( \infty \)
- Plot the rate at which points in a given region diverge, encoding iterations as colors
- Stop the iterations when \( z_{k+1} \geq 2 \) or \( k \) reaches some limit
- Plot \( k \) at each position
- Points that converge are in the Mandelbrot set, named after B. Mandelbrot
A quick review of complex numbers

• We define \( i = \sqrt{-1} \)

• A complex number \( z = x + i \cdot y \)
  – \( x \) is called the real part
  – \( y \) is called the imaginary part

• We associate each complex number with a point in the \( x-y \) plane

• The magnitude of a complex number is the same as vector length: \( |z| = \sqrt{x^2 + y^2} \)

• \( z^2 = (x+iy)(x+iy) = (x^2 - y^2) + 2xyi \)
A load balancing problem

- Some points iterate longer than others
- If we use block decomposition, some processors finish later than others
- We have a load imbalance
- Will return to this next week
Image representation and processing
Digital Image Representation

Photos by Martin Juell

Red
Green
Blue

RGB representation
Ryan Cuprak

wikipedia

Photos by Martin Juell
Image processing example: image smoothing

Original 15 iter 50 iter
100 iter 300 iter 1000 iter
Image smoothing algorithm

• Repeat as many times as needed

\[
\text{for } (i,j) \text{ in } 0:\text{N}-1 \times 0:\text{N}-1
\]

\[
u'[i,j] = \frac{(u[i-1,j] + u[i+1,j]+ u[i,j-1]+ u[i, j+1])}{4}
\]

\[u = u'\]
Parallel Implementation

- Partition data into parts, assigning each to a unique thread
- Dependences on values found on neighboring processes
- Threads share boundary values

```
P0  P1  P2  P3
0   0
1   1
2   2
3   3
```
Java Implementation

• To represent images we’ll use
  `java.awt.image.BufferedImage` for I/O (ImageIO)
  `java.awt.image.WritableRaster` for display

• We’ll convert to a p3D int array when manipulating images in memory (conversion 1D when displaying)

• \( W \times H \times 3 \)