Administrivia

- HW0 reviews by Tomorrow.
- HW1 out.
  - AWS setup - who has / hasn’t gotten it working?
- Next Readings posted, plan to discuss on Friday.
Work-Span Model & Caches
Speedup

- $T_S = \text{Time to run a program on a single processor.}$
- $T_P = \text{Time to run on P processors.}$
- Speedup for P processors is $T_S / T_P$
Amdahl’s Law

If \( \frac{1}{N} \) of compute time is in serial code, maximum speedup is \( N \) on any number of processors.
Gustafson-Barsis’s Law

If $1/N$ of compute time is in serial code,

speedup on $P$ processors is

$$S(P) = P - \frac{1}{N} \cdot (P - 1)$$
Work-Span
A computation, represented as a graph of dependencies.
Amdahl’s Law only talks about the serial nodes. This is far too simple.
Work = all computations.

Proportional to $T_s$
(time to run on a single processor)
Span = Critical Path computations

Proportional to $T_\infty$
(time to run on infinite processors)
Upper Bounds on Speedup

If we ignore bandwidth & caching, we have two upper bounds for Speedup on P processors:

Speedup(P) = \( \frac{T_1}{T_p} \)

\( \frac{T_1}{T_p} \leq P \)

\( \frac{T_1}{T_p} \leq \frac{T_1}{T_\infty} \)

Why?

\( T_s = \text{work} \)

\( T_\infty = \text{span} \)
Upper bounds

\[
\frac{T_1}{T_p} \leq P \\
\frac{T_1}{T_p} \leq \frac{T_1}{T_\infty}
\]

“We have to do all the work.”

“We can’t do the critical-path faster on P processors than on \( \infty \) processors.”
Blind spots of Amdahl, Gustafson-Barsis & Work-Span

• Where is the data?
• How is it moved?
• How long does that take?
Caches
Why Cache?

Memory Latency vs. Access Range (Sandra 2013 SP3)
Levels of Cache

- (registers)
- L1 - small, per-core
- L2 - medium, per-core
- L3 - shared
- (Memory)
## Numbers for Xeon

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>~64 ints</td>
<td>0</td>
</tr>
<tr>
<td>L1</td>
<td>32 KB / core</td>
<td>4-5 cycles</td>
</tr>
<tr>
<td>L2</td>
<td>256K / core</td>
<td>10 cycles</td>
</tr>
<tr>
<td>L3</td>
<td>4-12 MB / shared</td>
<td>40 cycles, unshared 65-100, other core</td>
</tr>
<tr>
<td>Memory</td>
<td>??? GB</td>
<td>Local, 180 cycles Remote, 300 cycles</td>
</tr>
</tbody>
</table>

Remember, each core has 2 threads.
Also, L1 cache is 32KB Instruction, 32KB data.
Cache lines

- Caches are organized into “lines” of 64 Bytes.
- If you read offset X, offset X+1 is probably in-cache.
- Coherent reads are good.
Memory Bottleneck

- ~200 cycles to go to memory.
- Get 64 Bytes per read.
- **Need** to perform at least 3 operations per byte.
  - (remember, ints are 8 bytes, floats 8, doubles 16)
Example

• Sum a 2D array, taking elements row-by-row or column-by-column?
Cache hit rates

- 90-97% for L1
- 99% for L2

  - But... ~30% for accesses that actually get to L2.
  - Better to make L2 bigger rather than faster.
  - (And even more important for L3)
• Google

“Gallery of Processor Cache Effects”
Coming up

- Multiprocessing in Python
- The GIL, Cython, & Threading