CS 5220 – 2015-08-27 Preclass Questions

Michael Whittaker (mjw297)

August 26, 2015

1. Refer to Figure 1 and Figure 2. These plots were generated by `amdahl.py`.

2. We have a central server serially processing and sending a set of tasks $\{t_1, t_2, \ldots\}$ to p workers $\{w_1, w_2, \ldots, w_p\}$. Assume that the server sends task t_i to worker $w_{i \bmod p}$. For example, if we had three workers, then the central server would send t_1 to w_1, t_2 to w_2, t_3 to w_3, t_4 to w_1, t_5 to w_2, etc. Also assume that each worker has a task buffer such that the central server can send a task to a worker even if it is already working on a task. Now, consider the possible values of α.

   - Case 1: $\alpha \geq \frac{\tau}{p}$. At time α, w_1 receives t_1. Then at time $\alpha + p\alpha$, w_1 receives t_{p+1}. If $\alpha \geq \frac{\tau}{p}$, then $p\alpha \geq \tau$, so between time α and time $\alpha + p\alpha$, w_1 has completed t_1. It then sits idle until it receives t_{p+1}, at which point it immediately starts working. Here, the central server is the bottle neck, so the throughput of the program is bounded from above by $\boxed{\frac{1}{\alpha}}$.
   - Case 2: $\alpha < \frac{\tau}{p}$. As before, w_1 receives t_1 at time α. At time $\alpha + p\alpha$, the central server sends task t_{p+1} to w_1. Since $\alpha < \frac{\tau}{p}$, w_1 is still processing t_1 at time $\alpha + p\alpha$, so t_{p+1} is queued in the task buffer of w_1. w_1 will continue to work on t_1 until it finishes at which point it will *immediately* begin work in item t_{p+1}. Here, the p workers are the bottleneck, so the throughput of the program is bounded from above by somewhere between $\boxed{\frac{1}{\tau} \text{ to } \frac{p}{\tau}}$.

3. You should not tune a piece of code if

   - the time taken to tune the code is greater than the time gained from tuning,
   - the tuning significantly decreases the readability or maintainability of the code, or
   - the code you're tuning isn't a bottleneck of the program.

4. According to this intel spec sheet, each Xeon Phi board runs a theoretical maximum of 1,011 GFLOPS. With 15 such boards, we get $15 \times 1,011 = 15.165$ TFLOPS.

5. I have four Intel Core i5-4210U CPUs each running at 1.70GHz. According to this StackOverflow answer, each processor can perform 4 double precision floating operations per second for a total of $4 \times 4 \times 1.7 = 27.2$ GFLOPS.

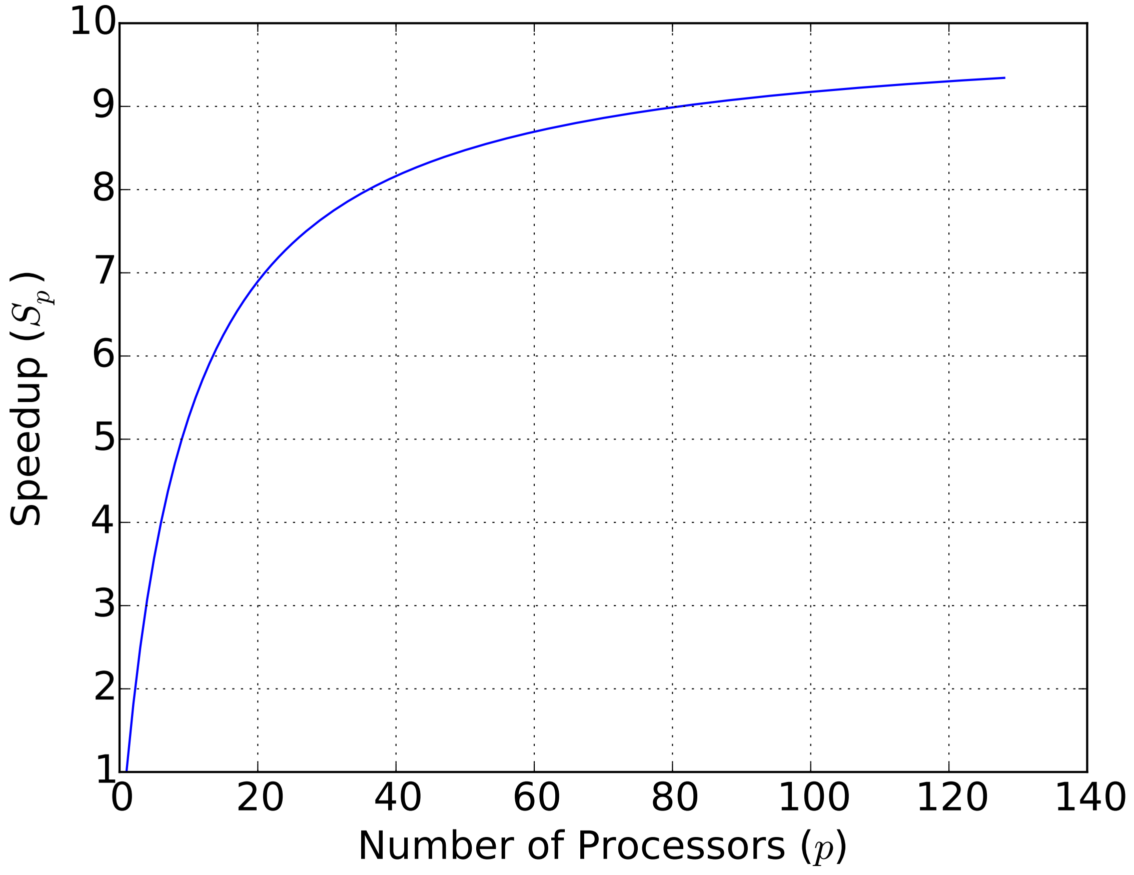

Figure 1: Speedup according to Amdahl's Law

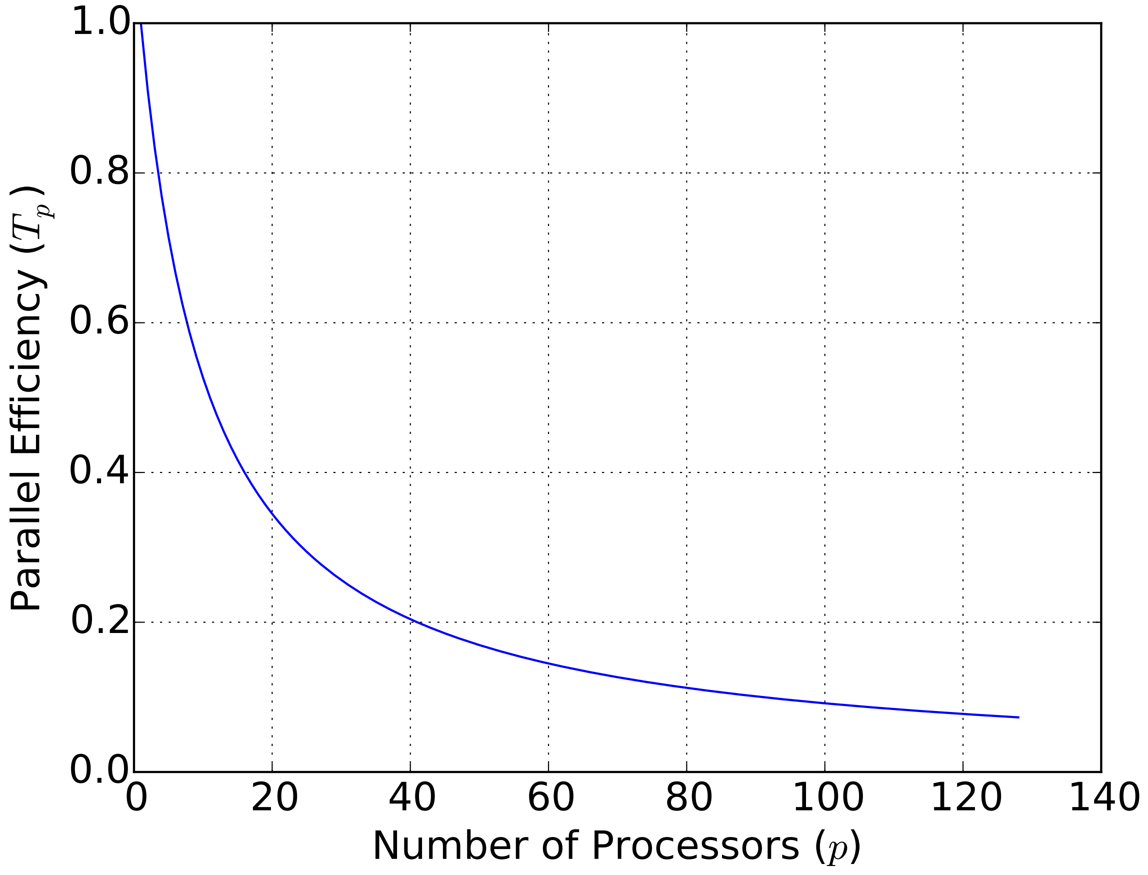

Figure 2: Parallel efficiency according to Amdahl's Law