Distributed and Parallel Computing

Lecture 1

Alan P. Sexton

University of Birmingham

Autumn 2019

Introduction

Three Topics:

- Multicore, shared memory parallelism
  - About 3 weeks
  - Unassessed exercises
  - Online Canvas class test (5%), which may be partially based on unassessed assignments
  - One question in final exam
- General Purpose GPU programming
  - CUDA programming on NVidia GPUs
    - About 5 weeks
    - Unassessed exercises plus 1 assessed exercise (10%)
    - Two questions in final exam
- Distributed Algorithms
  - About 3 weeks
  - Unassessed exercises
  - Online Canvas class test (5%), which may be partially based on unassessed assignments
  - One question in final exam

Class Discussion

Parallel, and distributed, programming is difficult. Even small programs in this domain can be very hard to compose, very hard to prove correct and very hard to debug - MUCH harder than serial programs.

Class Discussion

Parallel, and distributed, programming is difficult. Even small programs in this domain can be very hard to compose, very hard to prove correct and very hard to debug - MUCH harder than serial programs.

1. What is the relationship, if any, between parallel programming and distributed programming?

Class Discussion

Parallel, and distributed, programming is difficult. Even small programs in this domain can be very hard to compose, very hard to prove correct and very hard to debug - MUCH harder than serial programs.

1. What is the relationship, if any, between parallel programming and distributed programming?
2. Why do we do parallel programming?

Class Discussion

Parallel, and distributed, programming is difficult. Even small programs in this domain can be very hard to compose, very hard to prove correct and very hard to debug - MUCH harder than serial programs.

1. What is the relationship, if any, between parallel programming and distributed programming?
2. Why do we do parallel programming?
3. Why do we do distributed programming?

Moore's Law

> *The number of transistors in a dense integrated circuit double approximately every two years*
>
> *[Gordon Moore, 1965]*

- Not technically a law: just an observation and a prediction
- Has been more or less true until about 2012, now slowing down
- Processor clock rates stopped increasing in the early 2010s due to heat dissipation problems

Cannot improve performance by increasing clock rate, so use the extra transistors to put multiple processors on the same chip to get more work done in the same time using parallelism:

- Intel Core i9 Extreme i9-7980XE: 18 cores, 36 threads (Hyperthreading)
- AMD Ryzen Threadripper 2990WX: 32 cores, 64 threads

Cores vs Hyperthreads

- Each core is a processing unit containing Arithmetic Logic Unit, Floating point unit, and a number of caches, as well as the usual program counter, instruction register and register set.
- There are enough transistors on chips today that we can double up on the registers and program counter in a core, while sharing the ALU and FPU so that we can run two different threads of execution on the same core.
- Caches are in multiple levels: L1 being smallest and closest to the core. Most CPUs have an L2 and L3.
- Typically L1 is local to a core (shared between the 2 hyperthreads of the core), L2 is possibly shared between 2 cores, and L3 shared by all cores on the chip, though other arrangements are possible.
- Where there is contention for a resource between two threads, one may have to wait till the other releases the resource

Measuring Parallel Speedup

- **Latency**: the time from initiating to completing a task
  - Units of time
- **Work**: a measure of what has to be done for a particular task
  - number of floating point operations
  - number of images processed
  - number of pixels processed
  - number of simulation steps
  - number of comparison operations
- **Throughput**: work done per unit time

Speedup and Efficiency

- **Speedup$_P$**, S_P: The ratio of the latency for solving a problem with 1 hardware unit to the latency of solving it with P units
  - $S_P = \frac{T_1}{T_P}$
  - Perfect linear speedup: $S_P = P$
- **Efficiency**, E_P: The ratio of the latency for solving a problem with 1 hardware unit to P times the latency of solving it on P units
  - This measures how well the individual hardware units are contributing to the overall solution
  - $E_P = \frac{T_1}{P \times T_P} = \frac{S_P}{P}$
  - Perfect linear efficiency: $E_P = 1$

Interpreting speedup and efficiency

- Sub-linear speedup and efficiency is normal
  - Overhead in parallelizing a problem
- Super-linear speedup is possible, but usually due to special conditions
  - e.g. Serial version does not fit in CPU cache but each of the parallel sub-problems do.
- Important to compare the best serial version of the program with the parallel version
  - Serial algorithm A is fast but hard to parallelize
  - Serial algorithm B is slow but gives Parallel algorithm B
  - To measure speedup/efficiency, compare Serial A to Parallel B
  - Both must solve the same problem, but allow for minor differences
    - e.g. small round-off differences (but be aware of differing precision on host and on GPU!)
    - differences due to different execution order

Strong Scalability

Gene Amdahl, in 1967, argued that the time spent executing a program is composed of the time spent doing non-parallelizable work plus the time spent doing parallelizable work:

$$T_1 = T_{\text{ser}} + T_{\text{par}}$$

Therefore, if the speedup on P units of the parallel part only is s

$$T_P = T_{\text{ser}} + \frac{T_{\text{par}}}{s}$$

Then the overall speedup given the speedup of the parallel part is s is:

$$S_P = \frac{T_{\text{ser}} + T_{\text{par}}}{T_{\text{ser}} + \frac{T_{\text{par}}}{s}}$$

If we let f be the fraction of a program that is parallelizable, then $T_{\text{ser}} = (1-f)T_1$ and $T_{\text{par}} = fT_1$. Hence

$$S_P = \frac{(1-f)T_1 + fT_1}{(1-f)T_1 + \frac{fT_1}{s}} = \frac{1}{1-f+\frac{f}{s}} \qquad \text{(Amdahl's Law)}$$

Ahmdahl's Law Example

5 painters are painting a house with 5 rooms, 1 painter per room

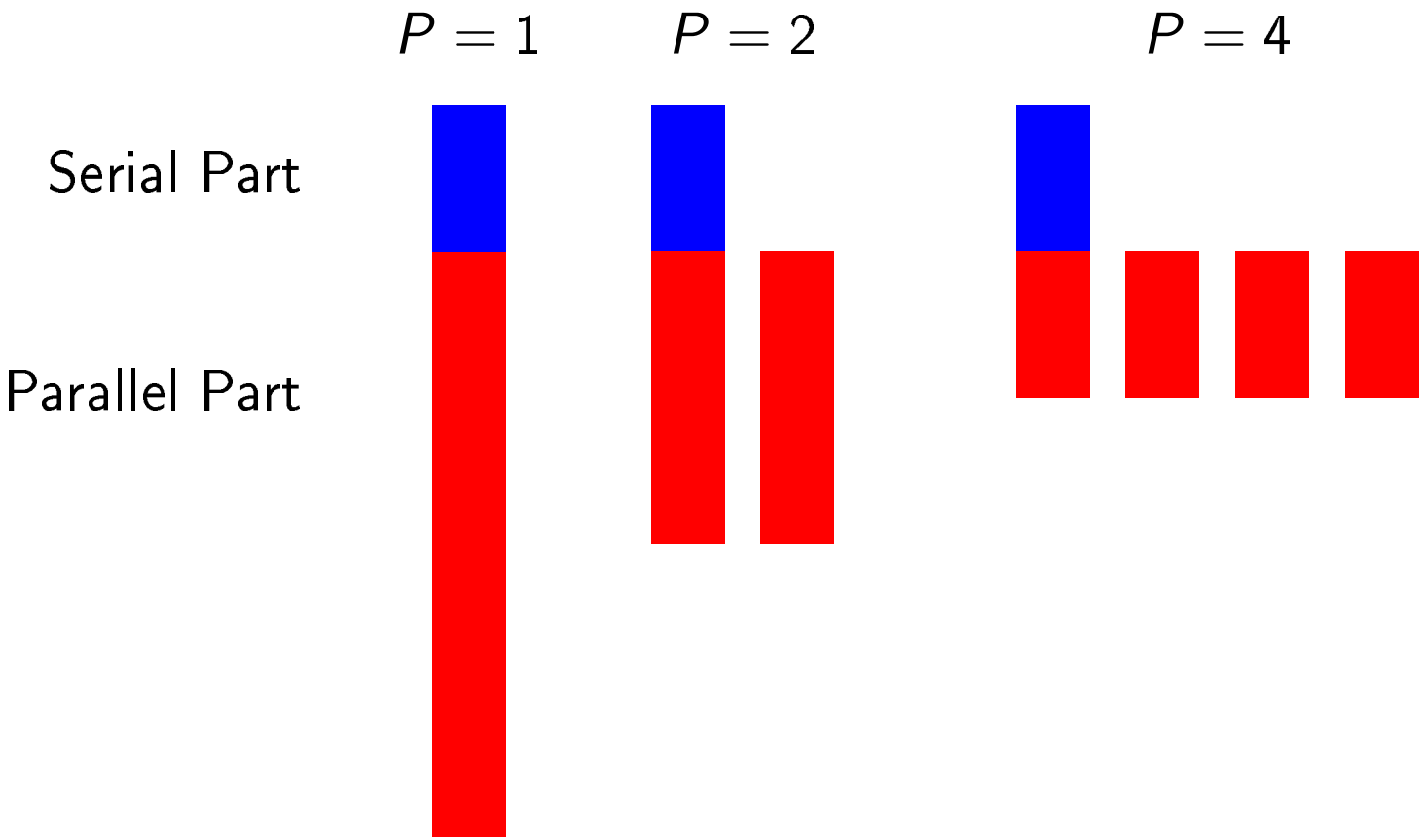
- Speedup of 5?
- What if one room is twice the size of the other rooms?
- One unit of work, and one unit of time, to paint the small rooms, 2 of each for the large
- 6 units of time for a single painter to paint all the rooms
- 1 unit of time for 5 painters to paint the small rooms and half the large
- 1 extra unit of time for last painter to finish the large room
- Speedup $S_5 = \frac{6}{2} = 3$

Apply Amdahl's law directly:

- $s = 5$
- 6 units of work to complete, 5 can done in parallel: $f = \frac{5}{6}$

$$S_P = \frac{1}{\left(1 - \frac{5}{6} + \frac{\left(\frac{5}{6}\right)}{5}\right)} = \frac{1}{\left(\frac{2}{6}\right)} = 3$$

Amdahl's Law Graphically



Interpreting Amdahl's Law

Amdahl's law says that there is a limit to parallel speedup

$$\lim_{s\to\infty} S_P = \lim_{s\to\infty} \frac{1}{1-f+\frac{f}{s}} = \frac{1}{1-f}$$

or, in other words

$$\lim_{s\to\infty} \frac{T_1}{T_P} = \frac{1}{1-f}$$

$$\Rightarrow \lim_{P\to\infty} T_P = T_{\text{ser}} \qquad \text{assuming } P \to \infty \Rightarrow s \to \infty$$

Weak Scalability

John Gustafson and Edwin Barsis, in 1988, argued that Amdahl's law did not give the full picture

- Amdahl kept the task fixed and considered how much you could shorten the processing time by running in parallel
- Gustafson and Barsis kept the processing time fixed and considered how much larger a task you could handle in that time by running in parallel
- This was motivated by observing that, as computers increase in power, the problems that they are applied to often increase in size

Gustafson-Barsis Law

Assume that W is the workload that can be executed without parallelism in time T. If f is the fraction of the workload that is parallelizable, then

$$W = (1 - f)W + fW$$

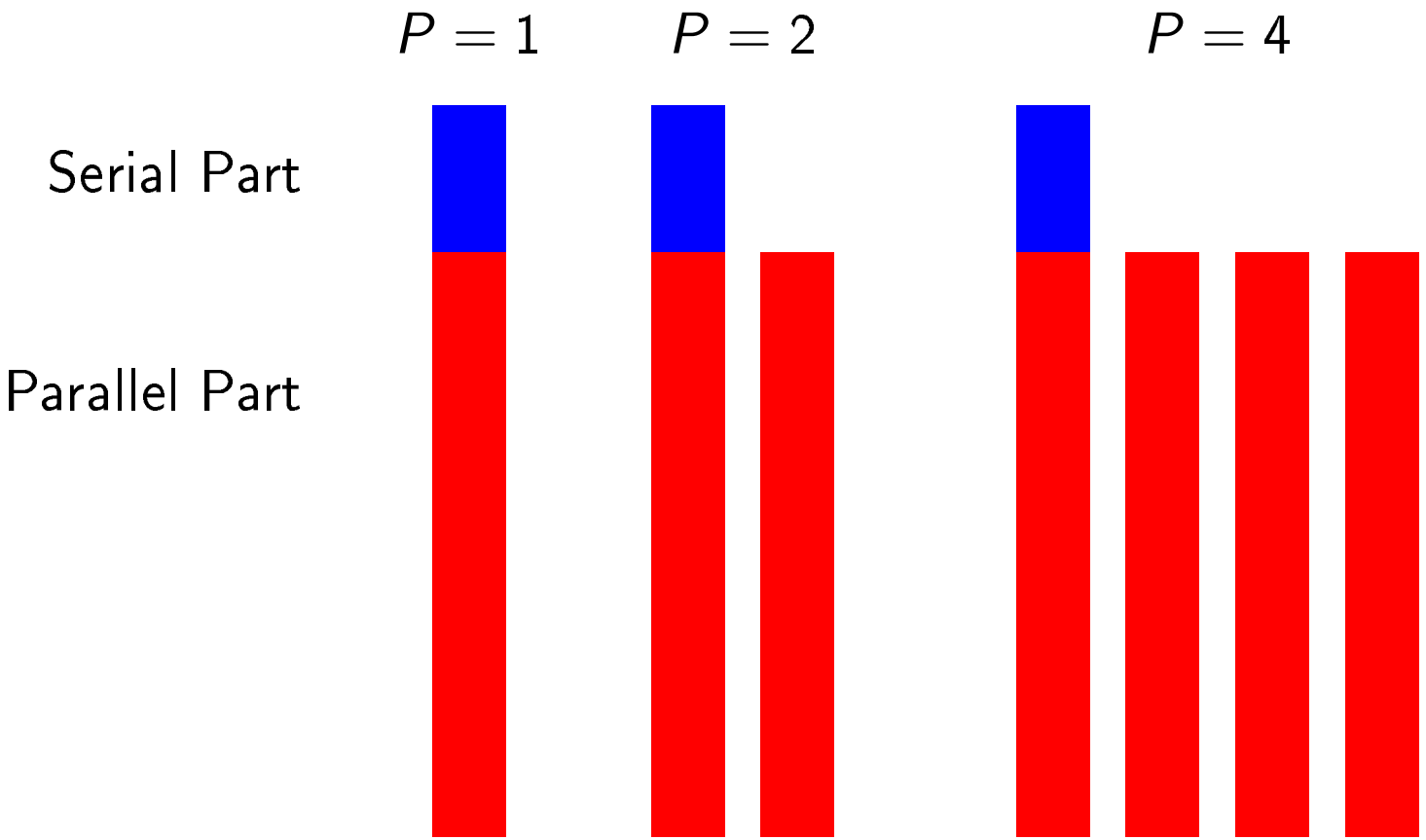
With speedup s, we can run s times the parallelizable part in the same time, although we don't change the amount of work done in the non-parallelizable part:

$$W_s = (1 - f)W + sfW$$

If we do W_s in time T, we are, on average, doing W amount of work in time $\frac{TW}{W_s}$. Hence the total speedup is:

$$S_s = \frac{T}{TW/W_s} = \frac{W_s}{W} = 1 - f + fs \qquad \text{(Gustafson-Barsis)}$$

Gustafson-Barsis Law Graphically



Conclusions

- Both Amdahl and Gustafson-Barsis are correct
- The two together give guidance on what kinds of problems can benefit from parallelization and how
- You can only go so much faster on a fixed problem by parallelization
  - i.e. you cannot avoid Amdahl's limit on a fixed size problem
- However, if, when growing the size of the task, you can increase the size of the parallelizable part of the problem faster than you increase the size of the non-parallelizable part, then Gustafson-Barsis gives opportunities for speedups that are not available if you keep the task size fixed.