COMPSCI 590S Systems for Data Science **Fall 2016**

Lecture 3: Parallelism

Lecturer: Emery Berger *Scribe(s): Jun Wang,Omar Ismail*

3.1 Terminology

Speedup is an important metric measuring the performance of parallelism. It compares the latency for solving the identical computational problem on one processor versus on N processors:

$$\text{speedup}(N) = \frac{T_1}{T_N}, \tag{3.1}$$

where T_1 is the latency, the time it takes to complete a task, of the program with one processor and T_N is the latency on N processors. In other words, T_1 is serial execution time, and T_N is the parallel execution time.

A speedup of more than N with N processors is called *super-linear speedup. Embarrassingly parallel* is another case, which means when solving multiple independent task, little or no coordination between tasks is needed.

When introducing parallelism, a straight-forward question is that if parallelism can increase speedup unlimitedly. The quite answer is no. To deliberate this question, we introduce *critical path length*, which is length of the event path in the execution history of the program that has the longest duration. Alternatively, it is the parallel execution time with infinite processor, so we denote it as T_∞. Fig-3.1 shows a system with 100 parallel tasks. Each circle is a task with latency of 1 time unit. In this system,

$$\text{speedup}(100) = \frac{T_1}{T_{100}} = \frac{102}{3},$$

and the parallel execution time is lower bounded by 3 even with infinite processors.

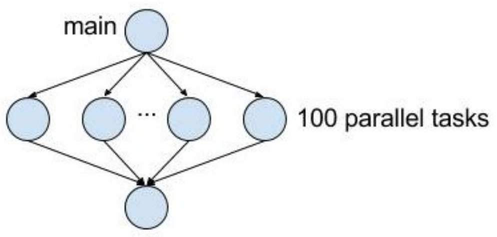

Figure 3.1: A system with 100 parallel tasks

Mathematically, speedup with N processors is upper bounded as

$$\text{speedup}(N) \leq \frac{T_1}{T_\infty + \frac{T_1 - T_\infty}{N}}. \tag{3.2}$$

3.2 Performance theories

Amdahl's law and Gustafson's law are two performance theories illustrating the trade-offs of parallel optimization.

3.2.1 Amdahl's law

Amdahl argued that the execution time T_1 falls into two categories:

- Time spent doing non-parallelizable serial work, denoted as s
- Time spent doing parallelizable work, denoted as p

Then,

$$T_1 = s + p \tag{3.3}$$
$$T_N = s + p/N \tag{3.4}$$

and Amdahl's law states that

$$\text{speedup}(N) \leq \frac{s+p}{s+p/N}. \tag{3.5}$$

Amdahl's law indicates that speedup is limited by the fraction of the work that is not parallelizable even using infinity number of processors. It consists with our observation is last section.

3.2.2 Gustafson's law

Gustafson provides a more optimistic point of view. Different from Amdahl,Gustafson disputed that in really, the workload is not constant; the work for parallel part scales with more processors. For example, if the completion time of a program is accelerated from 100 seconds to 1 second, then the more 99 seconds should be utilized for more work.

Gustafson's law states that

$$\begin{aligned} \text{scaled speedup}(N) &= \frac{s+pN}{s+p} \\ &= s + pN \\ &= N + (1-N)s \end{aligned} \tag{3.6}$$

Gustafson's law is limited by the assumptions that serial work grows much more slowly than parallel work.

3.3 Parallelism/super-linear

Achieving super-linear results from parallelism is possible because of different factors. If you have 1.5GB of data to work with and only 1 GB of memory, one machine will have to hit the disk to finish processing, which takes more time. But if you have two machines (each with 1GB of memory) vs one machine, that means you can fit all the data (1.5GB) in memory (.75GB in each machine), which is much faster than looking at disk for data. So this achieves super-linear processing.

The important thing to note is that the equations above from Amdhal's law and Gustafson's law are overly simplified, and should only be used as rough estimations and reference points. They ignore things like memory.

One of the goals of data science is to fit the work/data in RAM. There is a concept of a ”working set” which is ’what stuff/data are you working on now?’.

3.4 Garbage Collection

Garbage collection is the mechanism in a programming language for cleaning out objects in the memory that have no pointers referencing/pointing to it unreachable objects.

Some collectors have a ”stop the world” approach, where they freeze everything and clean up unreachable objects. Other collectors do clean up unreachable objects concurrently.

One way is to keep a reference count: keep track of the count of objects that have another pointing to it. The issue with this is cycles: where you have two objects pointing to each other but nothing else pointing to either of the two objects.

Another technique is to use Recursive Marking. That is a graph traversal of white/black marking of objects to see if there are any pointers. If white, remove, if black, then keep.

The amount of memory you use is a fraction of the total memory because the other portion is used by garbage collector. Works like real life garbage. You wouldnt go to dump every time you want to throw away one wrapper. Instead you wait till cans are full, then you go once.

Increasing performance of garbage collecting is a skill in and of itself. You can use a profiler to figure out how much memory you need to get it faster. If fast enough, then you are good. If not, then you have to dig into performance issues and use a variety of techniques to improve the garbage collecting. Performance of program depends on disk, RAM, what kind of data, etc. You cannot tell performance analytically, can only tell empirically.

References