Homework: Linpack Scaling and Analysis

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

November 29th, 2021

1. Exercise 1: Linpack Scaling

Explore simulation scaling in the context of the Linpack benchmark, that is, Gaussian Elimination. Ignore the system solving part and only consider the factorization part.

Question 1: Suppose you have a single core machine, and your benchmark run takes time T with M words of memory Now you buy a processor that is twice as fast, and you want to do a benchmark run that again takes time T. How much memory do you need?

Solution:

The total memory needed can be determined through an application of simulation scaling where an assumption is made that the simulated time, "S", and the running time, "T" are constant. Then, it follows that the total memory "M", memory per processor, "m", and the total number of processors are related through the following equation

$$M = Pm$$

First, we are given that the runtime of the first benchmark run is "T" and the total memory the first benchmark run takes is "M". Now, implementing a new processor which is twice as fast as the first processor implies that the new processor will have twice the clock speed, and since the clock speed increases by a factor of two, the number of clock cycles will increase by a factor of two. It follows that the benchmark time of the second run will be half that of the first run, namely, "T/2", and according to the equation above that. relates the total memory used to the processor number, and memory per processor, the total memory for the second benchmark run should decrease by a factor of 2. It follows that the total memory needed (in words) for the second benchmark run is "M/2".

Question 2: Now suppose you have a machine with P processors, each with M memory, and your benchmark run takes time T. You buy a machine with 2P processors, of the same clock speed and core count, and you want to do a benchmark run, again taking time T. How much memory does each node take?

Solution:

From the problem statement, we note that each processor has a single core, and that the number of words of total memory per core equals "M", the benchmark runtime equals "T", and the number of processor cores for the first and second runtime are "P" and "2P" respectively. Since the new processor contains twice the number of cores as the old processor, the clock speed of the cores doubles. As a consequence, the new machine with twice as many processors can double the amount of clock cycles, and execute twice as many instructions as the old in the same period of time. Since the number of processor cores increases by a factor of two while the benchmark runtime stays the same, the total memory (in words) must decrease by a factor of two. Therefore, the amount of memory required per processor node should equal to "M/2".

2. Exercise 2: Linpack Analysis

Apply Brent's theorem to Gaussian elimination, assuming that add/multiply/-division all take one unit time.

Question 1:What is the critical path; what is its length; what is the resulting upper bound on the parallel runtime?

Solution:
The critical path is defined as the chain of dependencies (sequential events) of maximum length in a problem. Since they must be executed sequentially, the length of the critical path, "t", is a lower bound on the parallel execution time of an algorithm. Now, according to Brent's Theorem, an upper bound on the parallel execution time can be characterized by the following inequality (where "m" is the total number of tasks, "p" is the total number of processors, and "t" is the length of a critical path):

$$T\_p \leq t + \frac{m-t}{p}$$

To determine the length of the critical path, a Gaussian elimination algorithm must be defined. Suppose we generalize the algorithm for a square n by n matrix characterized by the following system of equations:

$$Ax = b$$

We can consider implementing Gaussian Elimination with partial pivoting. Now, Gaussian elimination is by definition equivalent to triangular factorization. In particular, Gaussian Elimination with partial pivoting will produce an upper triangular matrix, "U", and a lower triangular matrix, "L" satisfying the equation:

$$PA = LU$$

Here, "P" is a permutation matrix, and "A" is overwritten by "L" and "U". Now, it also follows follows from partial pivoting that the system, "Ax=b" becomes,

$$Ux = g$$

Now, scaling analysis can be used to generalize the Gaussian Elimination algorithm using concepts such as broadcasting, scatter, allgather, gather, and associativity. For a coefficient matrix "A" with "n" rows, there are five main steps that must be repeated "n-1" times, where the "kth" iteration performs computation of the "kth" column of the "U" matrix. These steps are as follows:

1) Find the index of the next pivot row by locating an element of maximum magnitude in the current (kth) column while considering only elements on an below the associated diagonal. Through scattering, this involves "P" processors where "P" is the maximum number of processors on the grid. Each must find a local maximum and apply the associative operator. 2) Next, the pivot row must be broadcasted vertically. 3) Then, the current "kth" row must be exchanged with the pivot row, and a record of the row permutation must be stored. Only the current "kth" row must be sent since the pivot row has been broadcast. 4) Then, the elements of "L" from the "kth" column are calculated and broadcast horizontally. 5) Lastly, Gaussian Elimination is performed using the pivot row and the other rows defined in each processor.

From each of these steps, it follows that the total number of additions/subtractions from A to U equals

$$\frac{n(n-1)(2n-1)}{6}$$

Additionally, it follows that the total number of multiplications/divisions from A to U equals

$$\frac{n(n^2-1)}{3}$$

The total number of additions/subtractions from b to g equals

$$\frac{n(n-1)}{2}$$

The total number of additions/subtractions from g to x through back-substitution in finding the solutions equals

$$\frac{n(n-1)}{2}$$

The total number of multiplications/divisions from g to x through back-substitution in finding the solutions equals

$$\frac{n(n+1)}{2}$$

Adding each of the terms gives the following for the total number of operations needed to obtain the solution for a general system of n linear equations in n variables through Gaussian Elimination:

$$\frac{n(n-1)(2n+5)}{6} + \frac{n(n^2+3n-1)}{3}$$

Now, in assuming that each of the arithmetic operations, "addition, subtraction, multiplication, and division" all take one unit time to execute, it follows that the time it takes for a single processor to execute each of the steps in the algorithm is equal to the total number of steps, namely,

T·p = m

Now, there are two main contributions to the lower bound on the parallel runtime, namely, a floating-point contribution, and a communication contribution. The floating-point contribution (determined above) can be reduced into the following expression:

$$\frac{2n^3}{3} + O(n^2)$$

In scattered representation, each of the

$$P = s^2$$

processors performs an equal amount of arithmetic, and thus, the floating-point arithmetic contributes the following to the total computation time:

$$O(\frac{n^3}{s^2})$$

In the communication contribution, each processor sends or receives "O(n/s)"

words at each iteration of the algorithm defined above. Under this assumption it follows that the total communication time contribution equals

$$O(\frac{n^2}{s} + O(n)$$

Then, it follows that the resultant upper bound on the parallel runtime equals

$$O(\frac{n^3}{s^2} + O(\frac{n^2}{s} + O(n)$$

From this, it can be shown that with an increase in the number of processors, the length of the critical path decreases. The minimum number of processors can be expressed as a function of the size of the matrix, namely, the number of rows "P(n)". Then, we can define an upper and lower bound for the optimal number of processors, and then perform Gaussian Elimination under these conditions. If it's determined that the critical path length is less than the value of "P" in the current iteration, then "P" is set to be the new upper bound, and "P" is instantiated as the lower bound for "Pmin".It the value is larger, we can set "P" to be the new lower bound. Then, we can iterate through the algorithm until the lower and upper bounds converge to the true optimal value of "P". This can be based off of Ahmadal's Law. Now, the computation can ultimately be performed with S = O(log(s(n)) providing that the upper bound remains S = O(s(n)).

Question 2: How many processors could you theoretically use? What speedup and efficiency does that give?

Solution:
Now, according to Brent's Theorem, since the critical path doesn't depend on the number of processors used, there can be an infinite number of processors used in the Gaussian Elimination algorithm. By definition, the speedup is related to the critical path and the granularity, which measures the amount of operations that can be performed between synchronizations. When the number of tasks is less than the minimum number of tasks in the algorithm for a given number of processors, the speedup was reduced through an increase in granularity. The speedup is expressed mathematically as

$$S\_p = \frac{T\_1}{T\_p}, S\_p \leq p$$

The efficiency is measure of the average utilization of the processors in a machine and the communication of information between the processors. The lesser the amount of communication between the processors, the more independent and parallel operations, the higher the efficiency of the algorithm. The efficiency can be expressed mathematically as

$$E\_p = \frac{S\_p}{p}, 0 < E\_p \leq 1$$

Thus, the efficiency is directly related to the speedup from a single processor, and the number. of processors in the machine.