Lab Report 1

Parallelism through MATLAB

CSC 452-001

Andrew Almond

09/26/19

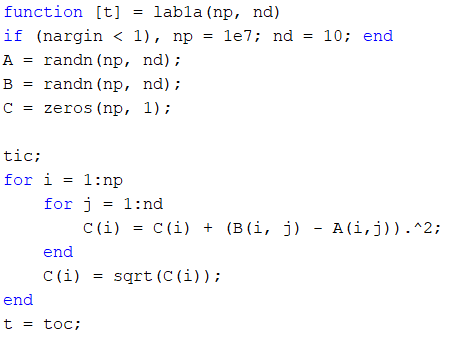
**Introduction**

From the lab, a greater understanding of parallelism can be gathered through its ability to achieve higher efficiency. However, too many parallel instances, in this case workers, can also lead to lower efficiency and longer wait times. To experiment, both a local machine and the LONI cluster will be the test equipment to compare systems with varying processing power and resources.

**Methods & Algorithms**

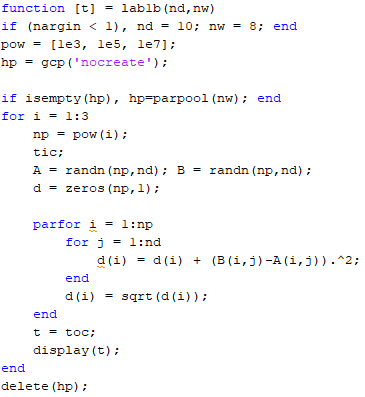
Question 1:

The beginning question gives an answer where parallelism is not used. The program shown below declares variables A and B as random matrices with size “np” (1e3, 1e5 or 1e7) by “nd” (10). C is declared the same way as the prior two; however, it consists of all zeros to make it ready for the function. As shown, each position in C’s matrix is set equal to the function below and the final time is collected.



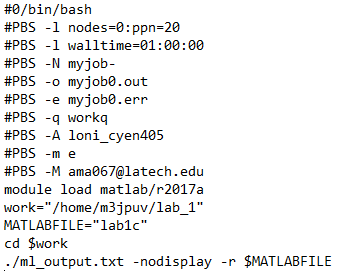
Question 2:

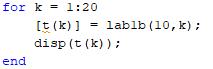
To try and gain efficiency over the solution to question 1, parallelism has to be utilized. To achieve this, parallelism is incorporated through assigning a number of workers, as well as the “parfor” statements instead of “for” statements. A parallel pool needs to be started if there isn’t one already available. Splitting the task between separate workers leads to quicker completion, but too many leads to loses.



Question 3:

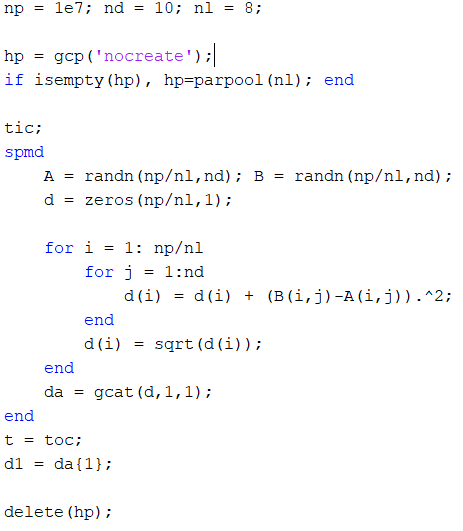
This solution varies from the rest in how it utilizes the LONI cluster to determine its results. The entire premise of this is to repeat the solution to question 2 through 1 to 20 workers. In this instance, the local machine couldn’t complete the task, so a distributed system was needed. This also required a script to run correctly on the LONI cluster.





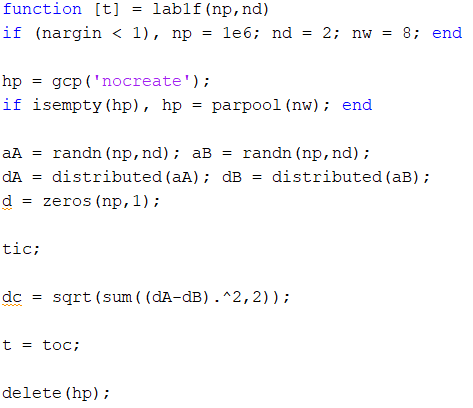
Question 4:

Like in question 2, parallelism will be used; however, in this case, parallelism will be enacted through the “spmd” statement rather than the “parfor” statement. This decision leads to greater flexibility in what parts of the task are parallelized, and the machine is given more control over how the data is allocated. “Parfor” typically is only for a loop and allows MATLAB to divide the resources out. As shown below, the variables are divided out by the number of labs (or workers) since MATLAB isn’t in charge. The “gcat” function concatenates the data together.



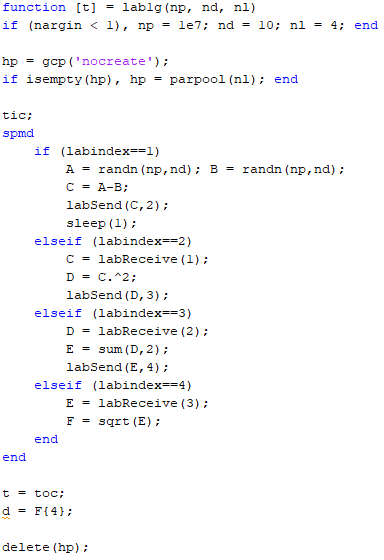
Question 5:

The distributed function allows for even greater flexibility than “spmd” in that it lessens the restrictions on parallelizing the task(s). This should give us better times than the method through “spmd” and through “parfor”.



Question 6:

The solution implements pipelining through “spmd” where the work is divided specifically into the sections shown. One process must wait for the previous process before it starts. Labs 1, 2 and 3 send their calculated data to the next lab, while all receive.



**Experiments**

For all the following tests besides Lab1c, the programs were executed on a PC with an Intel i7 6700HQ (4 Physical Cores) with 12 GB of memory. Lab1c was run on the LONI cluster with 1 node, 20 physical cores and 64 GB of RAM. The only changes to the programs between testing were value changes for the number of workers (or labs) and/or number of vectors.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Constant Variables** | **Varied Variables** | **# of Experiments** |
| **Lab1a** | nd = 10 | np = 1E3, 1E5, 1E7 | 3 |
| **Lab1b** | nd = 10 np = 1E3 | nw = 1:8 | 8 |
| **Lab1b** | nd = 10 np = 1E5 | nw = 1:8 | 8 |
| **Lab1b** | nd = 10 np = 1E7 | nw = 1:8 | 8 |
| **Lab1c with Lab1b** | nd = 10 np = 1E3 | nw = 1:20 | 20 |
| **Lab1c with Lab1b** | nd = 10 np = 1E5 | nw = 1:20 | 20 |
| **Lab1c with Lab1b** | nd = 10 np = 1E7 | nw = 1:20 | 20 |
| **Lab1d** | nd = 10 np = 1E7 nl = 8 | n/a | 1 |
| **Lab1f** | nd = 2 np = 1E6 nw = 8 | n/a | 1 |
| **Lab1g** | nd = 10  nl = 4 | n/a | 1 |

**Results**

For question 1, all that was needed was to change the value of vectors. As the vectors grew larger, the time of completion grew as well. In this case, the machine is completely free to divide the whole workload however it chooses.

|  |  |  |
| --- | --- | --- |
| Lab1a |  |  |
| 1E+03 | 1E+05 | 1E+07 |
| 0.5627 | 0.6408 | 0.6744 |

In question 2, MATLAB isn’t in full control of how the workload is divided out. This leads to greater performance in lower vectors but a weakness for larger vectors as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| Lab1b |  |  |  |
| Num of Workers | 1E+03 | 1E+05 | 1E+07 |
| 1 | 0.1766 | 0.484 | 17.1184 |
| 2 | 0.666 | 0.2399 | 16.4573 |
| 3 | 0.3016 | 0.3131 | 16.9539 |
| 4 | 0.2871 | 0.2907 | 14.4291 |
| 5 | 0.3967 | 0.3035 | 17.3024 |
| 6 | 0.4927 | 0.2957 | 18.6575 |
| 7 | 0.4907 | 0.4109 | 21.535 |
| 8 | 0.5351 | 0.4318 | 21.1421 |

The solution for question 3 uses question 2’s solution and places it on the LONI cluster for more workers to be available. As shown below, the completion times do become faster as the number of workers increased to an extent. This shows that a task can have too great a number of workers and lose efficiency. Overall, looking at the data shows that an appropriate number of workers to assign would be 4.

|  |  |  |  |
| --- | --- | --- | --- |
| Lab1c |  |  |  |
| Num of Workers | 1E+03 | 1E+05 | 1E+07 |
| 1 | 0.2048 | 0.161 | 10.1177 |
| 2 | 0.138 | 0.1432 | 7.4894 |
| 3 | 0.1518 | 0.1049 | 7.0719 |
| 4 | 0.156 | 0.0993 | 6.997 |
| 5 | 0.1366 | 0.1019 | 7.0736 |
| 6 | 0.167 | 0.1037 | 7.0354 |
| 7 | 0.1667 | 0.1086 | 7.281 |
| 8 | 0.156 | 0.0993 | 6.997 |
| 9 | 0.2011 | 0.1104 | 7.3501 |
| 10 | 0.1856 | 0.1167 | 7.1577 |
| 11 | 0.1908 | 0.1205 | 7.3628 |
| 12 | 0.2279 | 0.1224 | 7.041 |
| 13 | 0.2144 | 0.1365 | 7.0221 |
| 14 | 0.2058 | 0.1376 | 7.0959 |
| 15 | 0.2649 | 0.1368 | 6.8955 |
| 16 | 0.245 | 0.1539 | 8.2458 |
| 17 | 0.2484 | 0.1528 | 6.8715 |
| 18 | 0.3258 | 0.157 | 6.9007 |
| 19 | 0.2763 | 0.1763 | 6.9395 |
| 20 | 0.277 | 0.1665 | 9.0402 |

Given 8 workers, the solution for question 4 shows a much faster completion time than the solution to question 2. This gives more control over to the machine (MATLAB), allowing for more optimized loads for larger vectors.

|  |  |
| --- | --- |
| Lab1d |  |
| t | 5.1961 |

Like in question 2 and 4, the solution to question 5 utilizes a function to allow the machine greater control of the allocation of resources. This solution below advances further past question 4’s solution as well. The result of such happens to be an even faster completion.

|  |  |
| --- | --- |
| Lab1f |  |
| t | 1.879 |

For the solution to question 6, pipelining was used; however, this case has its flaws. Pipelining is typically meant for high throughput, large data situations. This method’s speed is defined by the slowest process(es) tied to one of the workers. This worker will slow down the overall process. Pipelining can be further improved by making the processes closer to the same length. Because the processes were divided out to each work in this manner, the completion time happens to be quite large.

|  |  |
| --- | --- |
| Lab1g |  |
| t | 107.1608 |

**Conclusion:**

Based on the results from our 6 problems and 90 cumulative experiments, it’s easy to see that parallelizing processes like each function here could lead to quicker results depending on the actual task. According to the data, use of too many workers in MATLAB could actually have a negative impact on the performance and decrease efficiently like previously believed. Though the tests weren’t bottlenecked by this local machine’s hardware too much in the given examples, other local machines could have been, depending on the performance and resources. For this purpose, usage of distributed systems like the provided LONI cluster definitely would be necessary. If given further testing with these forms of achieving parallelism (spmd, parfor, distributed and pipelining), an even greater understanding of these methods’ advantages and disadvantages could be formed.

**Github Link:**

<https://github.com/M3JPUV/CSC452-Lab1.git>