CPSC 424/524 Spring 2018 Assignment 3: Triangular Matrix Multiplication

Ezra Davis

Building and files

- `build-run-mpi.sh` has instructions for running and building all non-serial versions of the program. It needs to be slightly modified to run different test cases.
- `build-run.sh` the original serial build and run script
- `Makefile` makefile for running on the cluster. `module load Langs/Intel/15 MPI/OpenMPI/2.1.1-intel15` must be performed before running it.
- `matmul.c` barely modified serial version of the matrix multiplication program
- `matmul_parallel_*.c` contain the versions of matrix multiplication for tasks 2,3, and 4. Task 5 can use `matmul_parallel_4.c`.
    - Define the preprocessor symbol `#DEBUG` to print out additional timing information - both amount for communication and amount for calculation.
- `parallel.c` is the main function for tasks 2 through 5. It can also run with `matmul.c` in serial (if you run `mpirun -c 1`)
- `readme.md` this readme file
- `serial.c` barely modified version of the serial main program
- `test_local.c` main program for testing small cases of the matrix multiplication program without access to the large `.dat` files on the cluster.

All building and running is done in slightly modified versions of `build-run-mpi.sh`, which is then run with `sbatch build-run-mpi.sh`.

Task 1:

Here is the output of the serial program which is run with `srun build-run.sh`:

```
Matrix multiplication times:
   N      TIME (secs)    F-norm of Error
 -----   -------------  -----------------
   1000       0.1067     0.0000000000
   2000       0.8078     0.0000000001
   4000      16.7762     0.0000000003
   8000     141.0507     0.0000000014

real    2m40.429s
user    2m39.549s
sys 0m0.545s
```

Task 2:

Task 2a:

From the table for 2a, it is clear that there is a large imbalance in the amount of work processed by the lower numbered tasks compared to the higher numbered ones - this is almost certainly because while each task is assigned an equal number of rows, the first rows are much shorter and require less work to be done. Despite this, even task the last task (`N=8000, #p=8, p=7`) spends 10 seconds waiting for communication to complete, implying that we may be able to speed up the length of time it takes to finish the entire problem by streamlining the communication process.

The amount of time spent communicating does increase each time we double the number of processes, so eventually we will stop getting performance benefits from adding more tasks, but we the program would probably still be faster if we used `p=16` compared to `p=8`.

Somehow, my code is faster than the serial program for large matrices even when `p=1` and I have no idea why. That said, the times are roughly comparable, so I'm not too worried.

Here are the results on one node:

p=1

```
   N      TIME (secs)    F-norm of Error
   1000       0.1543     0.0000000000
   2000       1.1806     0.0000000000
   4000      14.0403     0.0000000000
   8000     118.0142     0.0000000000
```

p=2

```
N      TIME (secs)    F-norm of Error
1000       0.1093     0.0000000000
2000       0.8251     0.0000000000
4000      10.1081     0.0000000000
8000      85.0111     0.0000000000
```

p=4

```
N      TIME (secs)    F-norm of Error
1000       0.0783     0.0000000000
2000       0.5676     0.0000000000
4000       5.8001     0.0000000000
8000      57.9622     0.0000000000
```

p=8

```
N      TIME (secs)    F-norm of Error
1000       0.0496     0.0000000000
2000       0.3318     0.0000000001
4000       3.0803     0.0000000002
8000      33.4936     0.0000000000
```

Task 2a: Per process calculation vs communication tradeoff table

- `p` : total number of tasks.
- `N` : Size of matrix
- `#p` : Task id
- `calculating` : Seconds that task spent calculating
- `communicating` : Seconds a task spent waiting for communication to complete

p	N	#p	calculating	communicating
1	1000	0	0.154257	0.000017
1	2000	0	1.180626	0.000017
1	4000	0	14.040238	0.000019
1	8000	0	118.014200	0.000019
2	1000	0	0.051110	0.058181
2	1000	1	0.104436	0.005574
2	2000	0	0.383503	0.441595
2	2000	1	0.800137	0.025683
2	4000	0	4.069784	6.038318
2	4000	1	9.740579	0.368602
2	8000	0	39.813674	45.197437
2	8000	1	83.744027	1.269418
4	1000	0	0.015565	0.062778
4	1000	1	0.036512	0.040705
4	1000	2	0.049549	0.028352
4	1000	3	0.055626	0.023159
4	2000	0	0.112637	0.454927
4	2000	1	0.272792	0.289623
4	2000	2	0.376180	0.188934
4	2000	3	0.427465	0.140764
4	4000	0	0.846208	4.953861
4	4000	1	2.255041	3.521447
4	4000	2	3.566092	2.222135
4	4000	3	4.304769	1.496251
4	8000	0	10.972409	46.989782
4	8000	1	29.324692	28.540916
4	8000	2	38.852703	19.060007
4	8000	3	44.075137	13.889009
8	1000	0	0.004357	0.045225
8	1000	1	0.011039	0.036936
8	1000	2	0.016431	0.031623
8	1000	3	0.020570	0.028138
8	1000	4	0.023934	0.024864
8	1000	5	0.026235	0.023224
8	1000	6	0.027858	0.021690
8	1000	7	0.028416	0.021403
8	2000	0	0.032583	0.299248
8	2000	1	0.082960	0.240665
8	2000	2	0.121056	0.203988
8	2000	3	0.153598	0.172996
8	2000	4	0.179858	0.147982
8	2000	5	0.198630	0.130860
8	2000	6	0.211002	0.119658
8	2000	7	0.216788	0.115520
8	4000	0	0.238107	2.842225
8	4000	1	0.624450	2.420885
8	4000	2	0.931620	2.119730
8	4000	3	1.220106	1.836653
8	4000	4	1.552247	1.510751
8	4000	5	1.838238	1.230892
8	4000	6	1.956883	1.117819
8	4000	7	2.070927	1.010045
8	8000	0	2.507068	30.986574
8	8000	1	8.270425	25.077264
8	8000	2	12.808546	20.563267
8	8000	3	17.372915	16.023761
8	8000	4	19.253519	14.167156
8	8000	5	21.606345	11.839339
8	8000	6	22.385881	11.083529
8	8000	7	23.038996	10.455570

Task 2b: (Round robin accross the sockets SLURM commands)

Spreading the processes across more nodes ends up creating some serious communication latency - adding about 10 seconds for changing from 2 to 4 nodes for 8 processes.

The added latency to each communication call does suggest that we might want to look at making our requests asynchronous to speed up total time, which will reduce apparent latency (but won't increase throughput or help us if we're waiting for another process to compute).

The load balancing strategy described in task 4 would be a good idea - there is some serious discrepancies between the calculation time of task 0 and of task 7 (or 3 when there are 4

total tasks), and load balancing would even that out.

Total time of additional runs

```
p, nodes      TIME (secs)    F-norm of Error
 -----   -------------  -----------------
   4,2      57.1226     0.0000000000
   8,2      34.2154     0.0000000000
   8,4      45.0222     0.0000000000
```

Per process calculation vs communication tradeoff table

- `p` : total number of tasks.
- `node` : Number of nodes the tasks are spread across
- `#p` : Task id
- `calculating` : Seconds that task spent calculating
- `communicating` : Seconds a task spent waiting for communication to complete

`p node #p calculating communicating -- ---- -- ----------- ------------- 4 1 0 10.972409 46.989782 4 1 1 29.324692 28.540916 4 1 2 38.852`

Task 3:

Interesting aside:

I've been developing my program on my laptop and running on the server only to get final times. Interestingly, I get a segmentation fault when running `MPI_Barrier` on the server with an unfinished `Isend` but not on my laptop - it's an interesting difference in the implmentation of OpenMPI on the two computers. I think that `MPI_Gatherv` on my laptop functions more like a barrier than it does on the `grace` cluster.

Task 3a:

There wasn't as much of an improvement between task 2 and task 3 as I'd hoped. The program is perhaps more scalable because the time spent communicating for the high rank processes is significantly shorter than task 2, but the low rank processes show only a few percentage points of difference (probably because they are waiting for the poorly load balanced higher rank processes to finish calculating).

Honestly, load balancing is probably more important than making communication asynchronous for this problem with this many processes.

p=1

```
   N      TIME (secs)    F-norm of Error
   1000       0.1540     0.0000000000
   2000       1.1786     0.0000000000
   4000      14.0739     0.0000000000
   8000     117.9903     0.0000000000
```

p=2

```
   N      TIME (secs)    F-norm of Error
   1000       0.1074     0.0000000000
   2000       0.8238     0.0000000000
   4000       9.4087     0.0000000000
   8000      83.8489     0.0000000000
```

p=4

```
   N      TIME (secs)    F-norm of Error
   1000       0.0691     0.0000000000
   2000       0.5033     0.0000000000
   4000       5.2822     0.0000000000
   8000      51.0482     0.0000000000
```

p=8

```
N      TIME (secs)    F-norm of Error
1000       0.0413     0.0000000000
2000       0.2825     0.0000000001
4000       2.7062     0.0000000002
8000      29.7321     0.0000000000
```

Task 3a: Per process calculation vs communication tradeoff table

- `p` : total number of tasks.
- `N` : Size of matrix
- `#p` : Task id
- `calculating` : Seconds that task spent calculating
- `communicating` : Seconds a task spent waiting for communication to complete

```
p  N     #p     calculating communicating
-- ----  --     ----------- -------------
1  1000  0      0.154022        0.000016
1  2000  0      1.178573        0.000009
1  4000  0      14.073851       0.000015
1  8000  0      117.990323      0.000014
2  1000  0      0.051080        0.056288
2  1000  1      0.104701        0.003424
2  2000  0      0.383458        0.440373
2  2000  1      0.808689        0.015969
2  4000  0      4.149853        5.258834
2  4000  1      9.345853        0.064124
2  8000  0      39.853979       43.994954
2  8000  1      83.595920       0.255661
4  1000  0      0.015381        0.053669
4  1000  1      0.036668        0.011069
4  1000  2      0.049572        0.019173
4  1000  3      0.056145        0.013480
4  2000  0      0.112228        0.391119
4  2000  1      0.272539        0.063865
4  2000  2      0.376729        0.124515
4  2000  3      0.426019        0.078087
4  4000  0      0.847508        4.434644
4  4000  1      2.407402        1.333655
4  4000  2      3.453503        1.816430
4  4000  3      4.522799        0.760413
4  8000  0      10.978843       40.069344
4  8000  1      29.417425       5.842403
4  8000  2      39.835418       11.163131
4  8000  3      44.858624       6.191687
8  1000  0      0.004699        0.036638
8  1000  1      0.011167        0.007971
8  1000  2      0.016560        0.007659
8  1000  3      0.020629        0.009523
8  1000  4      0.024134        0.012360
8  1000  5      0.027080        0.013844
8  1000  6      0.027915        0.013137
8  1000  7      0.028445        0.013179
8  2000  0      0.032533        0.250013
8  2000  1      0.081673        0.032853
8  2000  2      0.122260        0.028986
8  2000  3      0.153922        0.045857
8  2000  4      0.181195        0.063588
8  2000  5      0.202200        0.078424
8  2000  6      0.212499        0.069333
8  2000  7      0.218251        0.065063
8  4000  0      0.239247        2.466912
8  4000  1      0.620603        0.550439
8  4000  2      0.930059        0.599245
8  4000  3      1.209316        0.704949
8  4000  4      1.550454        0.813007
8  4000  5      1.857511        0.837457
8  4000  6      1.965945        0.735021
8  4000  7      2.158771        0.548249
8  8000  0      2.319516        27.412585
8  8000  1      8.217945        4.101042
8  8000  2      12.054720       3.617743
8  8000  3      17.198141       3.838697
8  8000  4      19.167104       6.533131
8  8000  5      22.405130       7.279827
8  8000  6      22.441349       7.266907
8  8000  7      23.820234       5.913296
```

Task 3b:

The table(s) below confirm my analysis in task 3a. The high amount of variance (and low minimum) in time spent waiting for communication to finish by each process indicates that almost all of the time spent waiting for communication could be fixed by load balancing, which would probably result in a dramatic improvement.

Total times of additional runs

```
p, nodes      TIME (secs)    F-norm of Error
 -----   -------------  -----------------
   4,2      51.0757     0.0000000000
   8,2      39.4410     0.0000000000
   8,4      39.4410     0.0000000000
```

Per process calculation vs communication tradeoff table

- `p` : total number of tasks.
- `node` : Number of nodes the tasks are spread across
- `#p` : Task id
- `calculating` : Seconds that task spent calculating
- `communicating` : Seconds a task spent waiting for communication to complete

```
p  node  #p   calculating communicating
-- ----  --   ----------- -------------
4  1     0    10.978843     40.069344
4  1     1    29.417425     5.842403
4  1     2    39.835418     11.163131
4  1     3    44.858624     6.191687
4  2     0    9.159979      41.915732
4  2     1    26.346534     13.454560
4  2     2    39.721998     11.300924
4  2     3    44.854408     6.235647
8  1     0    2.319516      27.412585
8  1     1    8.217945      4.101042
8  1     2    12.054720     3.617743
8  1     3    17.198141     3.838697
8  1     4    19.167104     6.533131
8  1     5    22.405130     7.279827
8  1     6    22.441349     7.266907
8  1     7    23.820234     5.913296
8  2     0    1.765878      37.675139
8  2     1    4.785282      15.772475
8  2     2    10.535894     16.404557
8  2     3    14.108697     19.088410
8  2     4    33.422029     5.939334
8  2     5    37.347937     2.068089
8  2     6    22.300875     17.137172
8  2     7    22.539932     16.929151
8  4     0    1.765878      37.675139
8  4     1    4.785282      15.772475
8  4     2    10.535894     16.404557
8  4     3    14.108697     19.088410
8  4     4    33.422029     5.939334
8  4     5    37.347937     2.068089
8  4     6    22.300875     17.137172
8  4     7    22.539932     16.929151
```

Task 4:

Task 2 and task 3 clearly have load balancing issues. I decided that the simplist way to balance the load better is to ensure that each process has roughly the same number of elements of the matrix, which is easy enough to do by adding rows to the first process until it has `(N*(N+1)/2) * num_processes)` rows, and repeat that with the later processes, leaving any extra matrix rows to the last process.

As you can see, this decreased the running time dramatically (almost a factor of 2 compared to task 3). Intuitively, this process also seems more scalable - we waste far less running time waiting the last few processes to complete.

I'm not entirely sure why running 4 tasks on 1 node took longer than spreading it across 2 nodes.

Total times of matrix multiplications

```
p,nodes   TIME (secs)    F-norm of Error
 -----   -------------  -----------------
   4,1      48.3193     0.0000000007
   4,2      38.2903     0.0000000007
   8,1      21.3519     0.0000000009
   8,2      26.4483     0.0000000009
   8,4      26.4483     0.0000000009
```

Per process calculation vs communication tradeoff table

- `p` : total number of tasks.
- `node` : Number of nodes the tasks are spread across
- `#p` : Task id
- `calculating` : Seconds that task spent calculating
- `communicating` : Seconds a task spent waiting for communication to complete

```
p  node  #p   calculating communicating
-- ----  --   ----------- -------------
4  1     0    48.071516   0.247734
4  1     1    40.917110   7.346713
4  1     2    35.727831   12.567271
4  1     3    31.566219   16.754695
4  2     0    38.019593   0.270688
4  2     1    33.210402   5.031797
4  2     2    28.178327   10.083532
4  2     3    24.931262   13.373344
8  1     0    20.997349   0.354565
8  1     1    20.742037   0.522906
8  1     2    18.897301   2.386293
8  1     3    17.617265   3.686062
8  1     4    16.078185   5.228097
8  1     5    15.317710   6.019837
8  1     6    14.404397   6.945837
8  1     7    13.525022   7.841147
8  2     0    15.992409   10.455903
8  2     1    16.232806   3.881858
8  2     2    16.223746   5.810235
8  2     3    15.353427   8.564842
8  2     4    25.981504   0.429699
8  2     5    24.702734   1.732883
8  2     6    12.867661   13.578189
8  2     7    12.160585   14.315729
8  4     0    15.992409   10.455903
8  4     1    16.232806   3.881858
8  4     2    16.223746   5.810235
8  4     3    15.353427   8.564842
8  4     4    25.981504   0.429699
8  4     5    24.702734   1.732883
8  4     6    12.867661   13.578189
8  4     7    12.160585   14.315729
```

Task 5:

I didn't need to change anything from my task 4 to get this to work as I had planned for this situation from the very beginning of the assignment.

```
   N      TIME (secs)    F-norm of Error
 -----   -------------  -----------------
   7633      19.3182     0.0000000008
Process 0 spent 16.548526 time calculating and 2.769722 time communicating
Process 1 spent 16.662782 time calculating and 2.530805 time communicating
Process 2 spent 19.064767 time calculating and 0.183009 time communicating
Process 3 spent 16.645378 time calculating and 2.623345 time communicating
Process 4 spent 13.903794 time calculating and 5.382887 time communicating
Process 5 spent 13.010575 time calculating and 6.281878 time communicating
Process 6 spent 10.668788 time calculating and 8.649129 time communicating
```