CpSc 418

Homework 2

Solution Set

72 points.

1. **Poetry Jam** (28 points)

   (a) Implement Erlang functions to implement a lock process. (12 points)
   See the solution in hw2.erl.

   (b) Secret messages (12 points)
   See the solution in hw2.erl.

   (c) Explain your design (2 points)
   See the solution in hw2.erl. Li Bai was the winner of `the favorite_poet()` contest, congratulations!

2. **We All Have Our Moments** (20 points)

   (a) Implement `mp_leaf(List, M, X0) -> {LengthList, SumListM.}` (2 points)
   See the solution in hw2.erl.

   (b) Implement `mp_combine(Left, Right) -> SubtreeSummary`. (2 points)
   See the solution in hw2.erl.

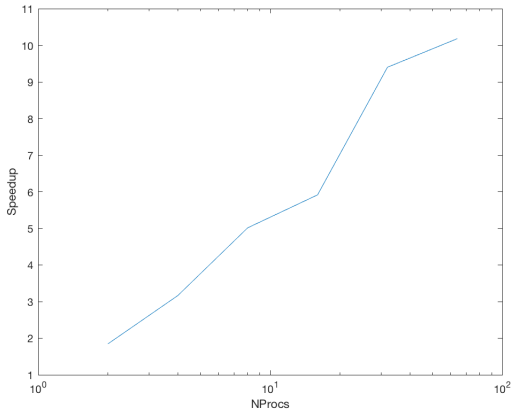
   (c) Implement `mp_Root(RootSummary) -> MomentM`. (2 points)
   See the solution in hw2.erl.

   (d) Test your code. (4 points)
   Many cases are covered in `moment_par_test()`. Notable omission is cases where some nodes are empty

   (e) Speed up versus number of processors (4 points)

NProcs	Time Par	Speedup
2	0.056	1.846
4	0.033	3.165
8	0.021	5.015
16	0.018	5.915
32	0.011	9.400
64	0.010	10.181

Table 1: Speedup data given a fixed `NData` and increasing `NProcs`

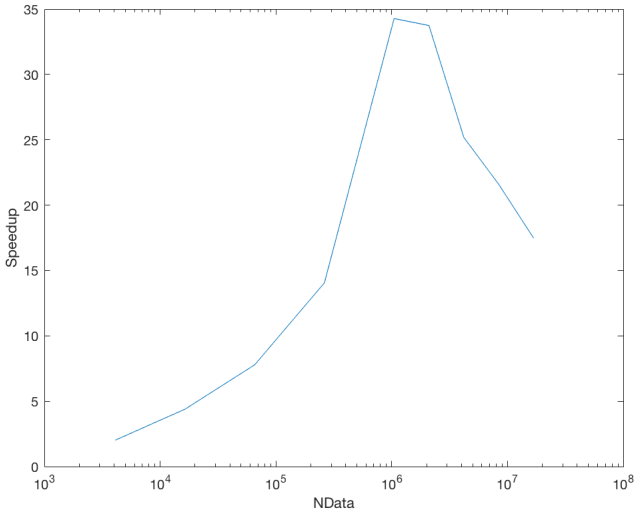


   (f) Speed up versus length of the list (4 points)

   (g) Observations (2 points)

NData	Time Par	Speedup
2^{12}	1.88e-4	2.019
2^{14}	2.86e-4	4.389
2^{16}	5.52e-4	7.784
2^{18}	1.56e-3	14.055
2^{20}	6.24e-3	34.275
2^{21}	0.012	33.741
2^{22}	0.031	25.178
2^{23}	0.074	21.600
2^{24}	0.191	17.485

Table 2: Speedup data given a fixed `NProcs` and increasing `NData`



There are many reasonable observations, so you may have something different than what is given below.
For fixed `NData` and growing `NProcs` we see a pattern of increasing speedup, but the benefit of doubling the number of processors is considerably less than a doubling in speedup. This is expected since we are increasing the amount of overhead of communication that must be done between the combine tree nodes.
For fixed `NProcs` and growing `NData` we see a pattern of increasing speedup. This behaviour is expected because larger `NData` for fixed `NProcs` means more parallel work in the leaf nodes while holding constant the communication needed for the combine tree nodes. However, we see a steep drop in our speedup after a certain point. This is due to the size of our cache. All processors will need to access their data to perform the reduce function, but not all data can be brought into the cache at the same time so some nodes will be forced to wait.

3. **Scan Tree** (10 points)

(a) The upward pass (5 points)

a) `{2,18}`
c) `{2,50}`
d) `{2,13}`
e) `{3,45}`
f) `{3,18}`
g) `{3,66}`
i) `{4,31}`
k) `{6,63}`
l) `{6,92}`
m) `{8,94}`

(b) The downward pass (5 points)

a) `{0,0}`

c) `{4,31}`
d) `{6,81}`
e) `{8,94}`
f) `{11,139}`
g) `{14,157}`
i) `{0,0}`
k) `{8,94}`
l) `{14,157}`
m) `{0,0}`
q) `[12.857142857142858,11.75]`

4. **Scanning Moments** (15 points)

   (a) Implement `moment_fold(List, M, X0) -> List2` (5 points)
   See the solution in hw2.erl.

   (b) Implement `moment_scan(W, KeySrc, KeyDst, M, X0) -> Est_Moment` (5 points)
   See the solution in hw2.erl.

   (c) Tests and Efficiency (5 points)
   Many cases are covered in `moment_scan_test()`. Notable omission is cases where some nodes are empty