Dustin Cook

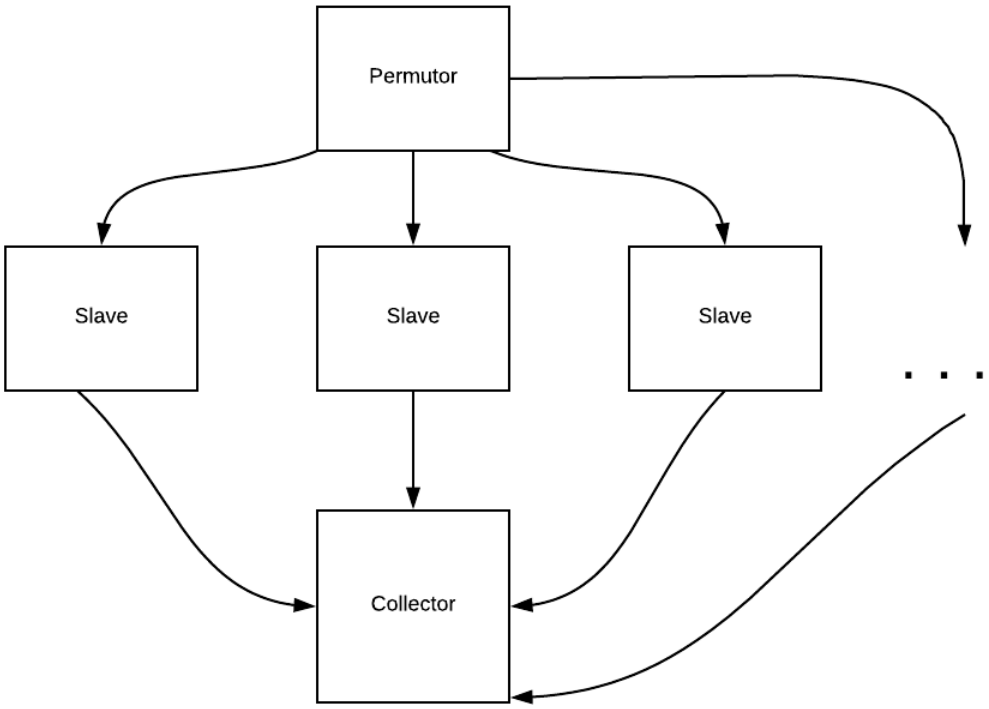
CSC 410 final

Christer Karlsson

12/10/2018

Program Description

MPI solution for the N Queens problem in parallel. There are three different types of threads for this application.



- Permutor – The Permutor rotates a tuple and sends the rotations to a slave.
- Collector – Waits for a slave to find all solutions for a given rotation.
- Slave – After receiving a rotation from Permutor, finds all solutions in lexicographical order up to but not including the next rotation of the given tuple.

Compile:

make parallel

Usage:

mpirun -np <threads> -hostfile hosts ./nqueens <n> <print> <omp threads>

- threads – 3 is required to run, its recommended to have 2 + n.
- n – nxn board
- print – 1 to print valid solitions 0 to just print number of solutions.
- omp threads – number of inner omp threads to use, 1 or 6 recommended. (Does not appear to give significant performance boost), used in slave and permutor.

Algorithms and Libraries Uses

- std::next_permutation
- std::rotate
- std::vector
- MPI
- OMP
- Python for testing
- Make for building

Functions and Structure

Single source file: **nqueens.cpp**.

There are three major functions: permutor, collector, and slave. Immediately a thread determines what kind of thread it is and calls its appropriate callback function e.g. if permutor calls permutor. OMP is in the program, but I have found that it gives no real benefit.

The permutor rotates and sends tuples to the salves in the order 1 – p- 2, wrapping around if it still has rotations to send after hitting the last slave. Once the permutor has sent all rotations, it sends a shutdown message to all slaves.

After receiving a rotation, the slave runs std::next_permute (n-1)! times finding all permutes for its lexicographical order and solves each permutation. The slave then listens for another message from the permutor and shutdown when the permutor sends the shutdown message. Once the slave receives the

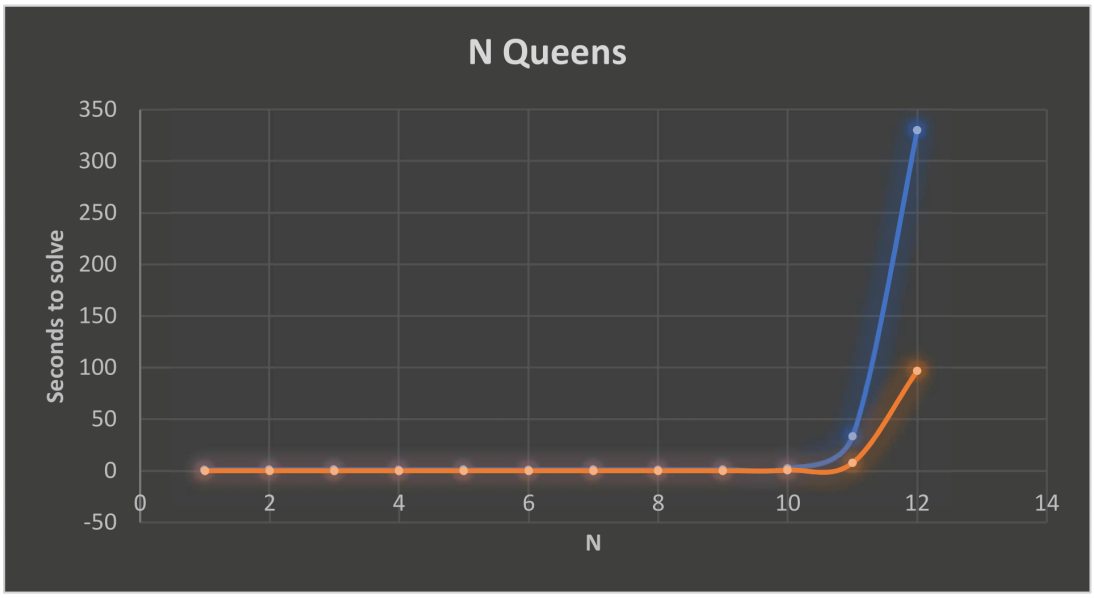
shutdown message, it sends its total solutions found to the collector as well as a shutdown signal for the collector to know that this slave has shutdown.

While there is still a slave that hasn't shutdown, the collector listens for a message from a slave. If the message was a shutdown, the collector makes a note that that slave shutdown, otherwise it is the slaves total solutions found, so it adds it to a cumulative total solution. Once all slaves have given the shutdown message, the collector prints all total solutions received.

Testing

- test.py – runs serial and parallel code versions and displays run times / results
- kfm.py – runs karp flat metric on nqueens with n = 11 and increasing np.
- plot.py – plots csv file "plots.csv" with run times for serial and parallel
- gboris.py – reads plot.csv and returns estimates serial execution for the program per n.

Data



Parallel – Blue, Sequential - Orange

The graph above was plotted using the following data:
Time is in seconds.

N	Parallel Time	Sequential Time
1	1.363989	0.00139
2	1.378785	0.001404
3	1.38989	0.001378
4	1.302628	0.001371
5	1.427016	0.001373
6	1.324403	0.001496
7	1.383304	0.002242
8	1.368139	0.008253
9	1.454466	0.06555
10	2.625558	0.662289
11	33.59888	7.721119
12	330.0054	96.94656

Analysis

Its unfortunate, but my parallel solution for this problem ended up being slower than the serial version I wrote to test its correctness with. I did try to perform an analysis on it with the standard parallel analysis methods.

Amdahl's Law

Using Amdahl's law to estimate the serial portion of my algorithm per given problem size.

N	Serial
1	-9%
2	-9%
6	-9%
7	-9%
8	-8%
9	-4.1%
10	18%
11	16%

Efficiency (not good)

N	Serial
1	.0085%
2	.0084%
6	.0094%
7	.013%
8	.05%
9	.38%
10	2.1%
11	24%
12	2.4%

Gustafson-Barsis's Law

When using the law to solve for serial time of my application, it returns 1 on average indicating that I am not utilizing my resources properly.

Karp Flat Metric

Selected problem size: 11

F_e	P
4.95362	3
82.29057	4
3.176658	5
2.765983	6
2.054255	7
1.579743	8
1.694941	9
1.524324	10
7.03669	11
6.635921	12
1.474196	13

Overall judgement: Not a valid parallel solution

It's kind of a shame. I didn't think that rotating and sending out the table with MPI would be so much worse than just doing it in serial. By the time I realized it was a bad solution it was far to late to go back to the drawing board. I was only able to reach a size of 13 for my parallel version.

Resources

- Wikipedia
- Your notes
- Stack Overflow