Project 4 : MPI Hotplate

Duane Johnson

Feb 24, 2009

Introduction

This paper investigates how 3 popular libraries used for parallelization can be used to solve the "hotplate" problem. Speeds and methods are compared so that the various implementation possibilities can be chosen in advance for future work. The libraries investigated are OpenMP, PThreads, and MPI.

Approach

While there are many ways in which the hotplate problem can be solved using more efficient algorithms, the implementations in this study do not exploit algorithmic speed-ups so that the libraries themselves can be compared with similar assumptions.

Experimental Setup

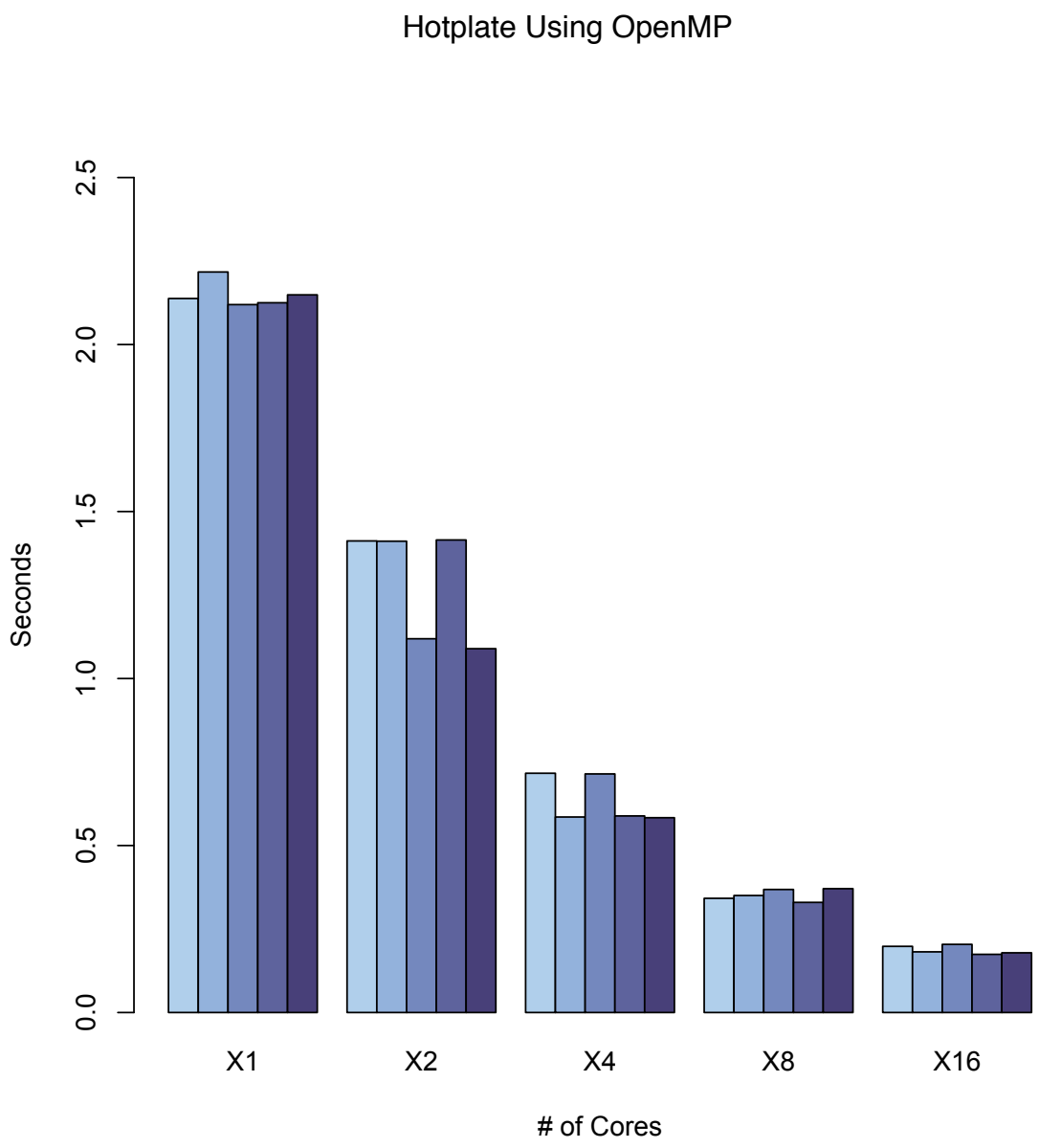
All implementations of the hotplate solution in this study use a two-buffer 2D matrix of 32-bit floating-point numbers. The "active" buffer is swapped between the two buffers so that a "source matrix" and a "destination matrix" are available at all times. In this way, cells in the matrix have their *next state* values computed from *previous state* values in a consistent manner.

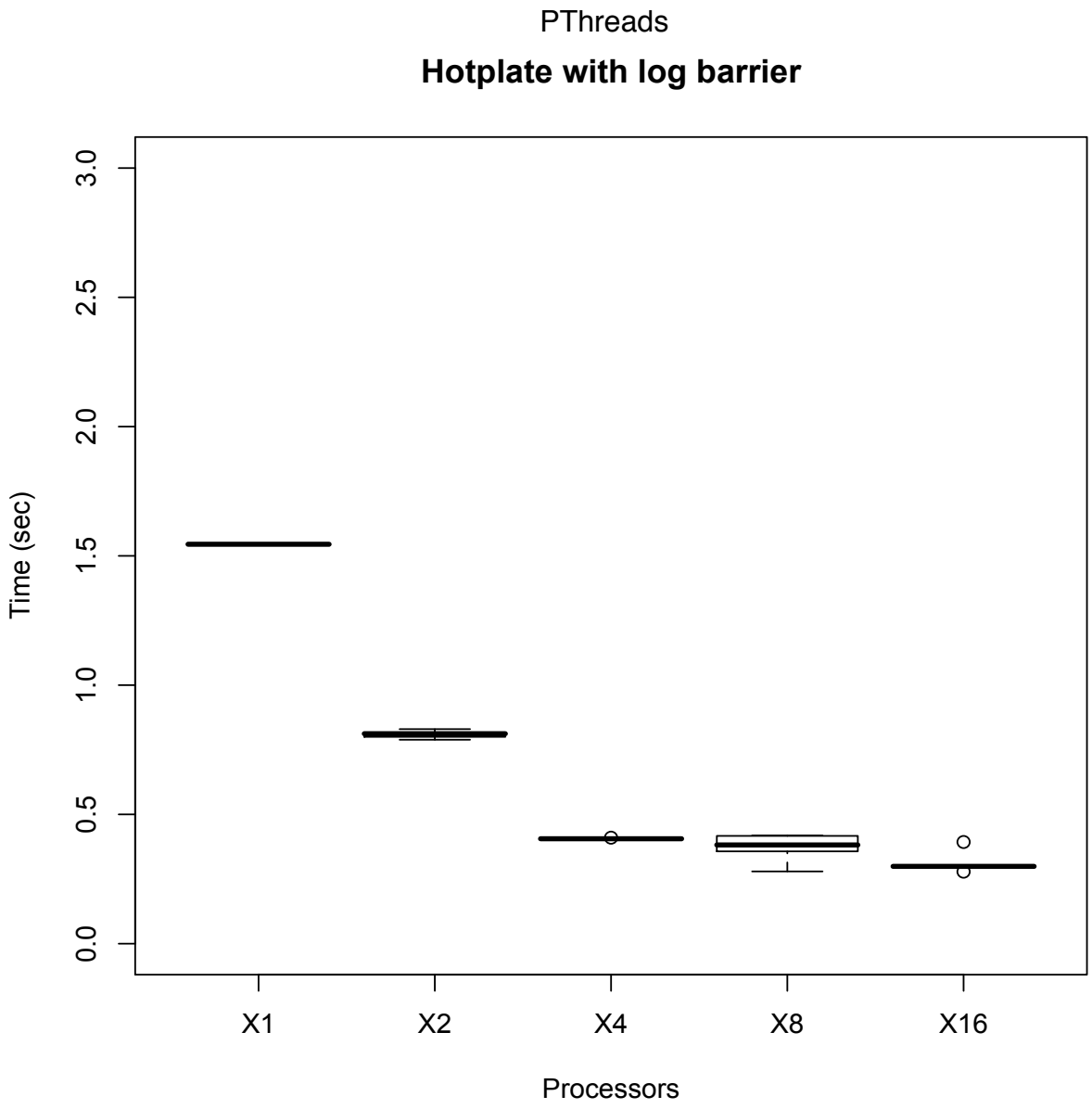
Timing of the solutions were performed on a supercomputer at BYU configured with 2.6GHz Xeon EM64T dual-core processors and 8 GB of fully-buffered RAM, with some 4x quad-core processors where necessary. All processes were resident on the same "node" so that no communication outside of the bus was needed (including in the case of MPI).

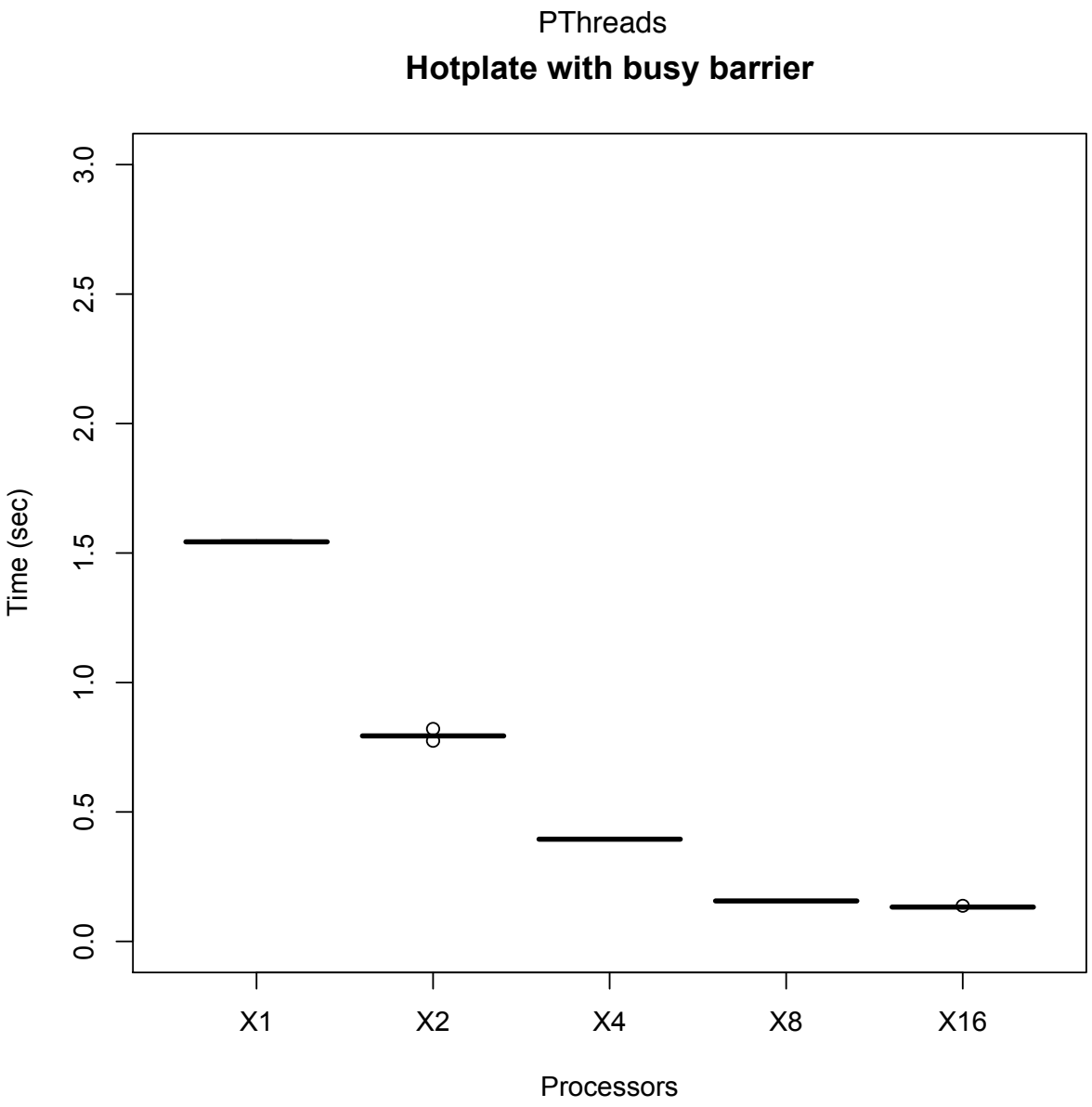
Results

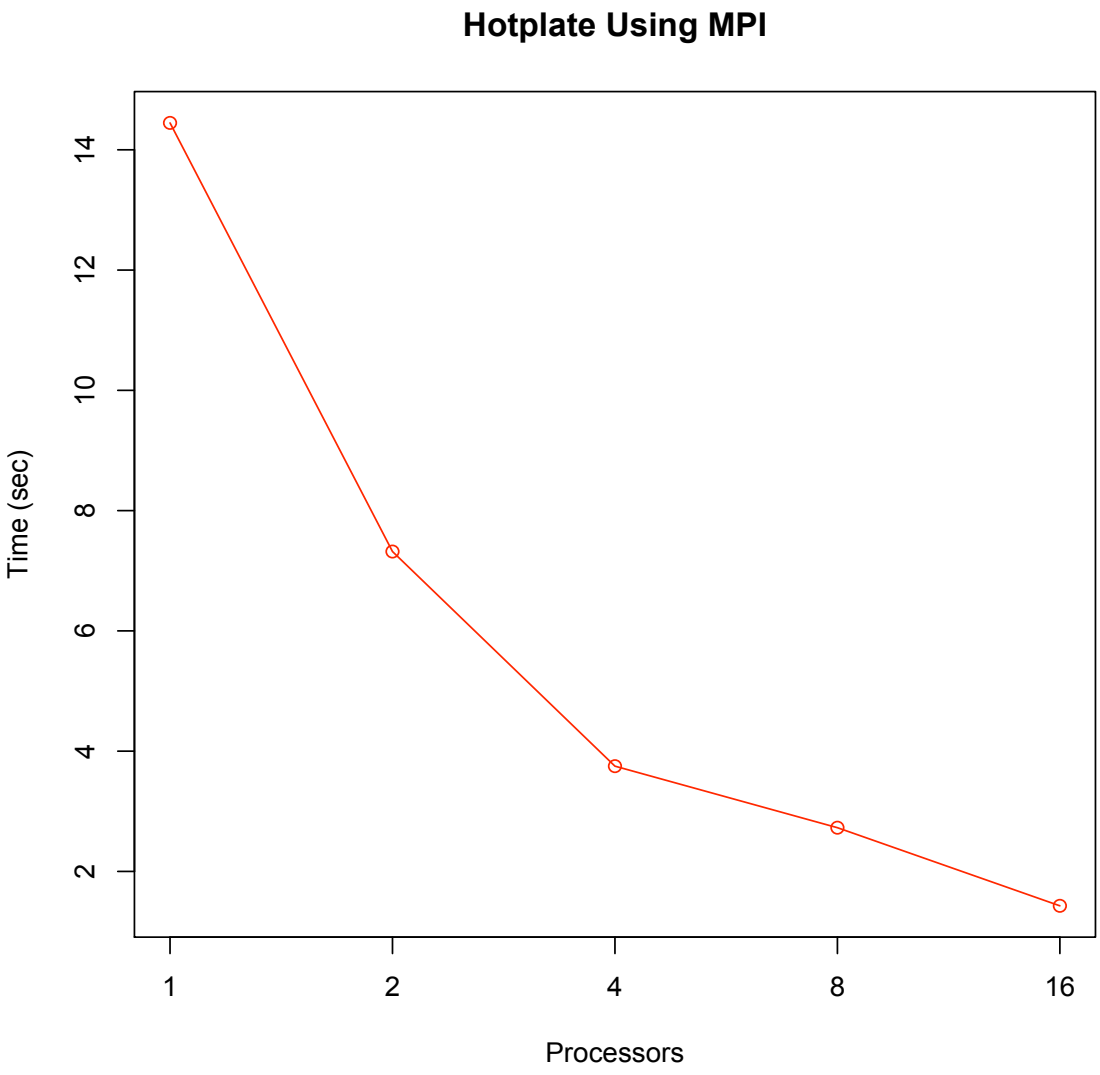
The scale on the following graphs should be noted carefully, as for example the MPI graph shows a range from 0 to 15 seconds while the others show a range from 0 to about 2 or 3 seconds.

While the PThreads solution was the most tedious of the group in terms of programming time and the required thought that went in to "thinking like a process", it was clearly the winner in terms of wall-to-wall speed, especially when using a busy-barrier on multiple CPUs. PThreads achieved a solution time of 0.2 seconds for 16 processors. OpenMP was close behind on this scale, closing in on 0.2 seconds as well. Slowest of the group was MPI which took nearly 1.5 seconds for 16 processors to complete the calculation:


Hotplate Using OpenMP
Seconds
0.0
0.5
1.0
1.5
2.0
2.5
X1
X2
X4
X8
X16
# of Cores


PThreads
Hotplate with log barrier
Time (sec)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X1
X2
X4
X8
X16
Processors


PThreads
Hotplate with busy barrier
Time (sec)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X1
X2
X4
X8
X16
Processors


Hotplate Using MPI
Time (sec)
2
4
6
8
10
12
14
1
2
4
8
16
Processors

Conclusion

While the PThreads and OpenMP solutions are the clear winners for multiple processors on the same node, MPI may show promise when more processing power than is available on a single node is required. For example, in the supercomputer used for this project, there are no nodes with more than 16 processors. In order to solve a problem with greater than 16 nodes working concurrently, only the MPI solution would work "out of the box". The other two would require further communication libraries in order to capitalize on such processing power.

Alternatively, OpenMP seems to be the best and fastest solution for general-purpose parallel processing on a single node, since it requires less work and produces an output similar to that which was achieved by hand-crafting a PThreads solution.