Mitigating performance and progress variability in iterative asynchronous algorithms

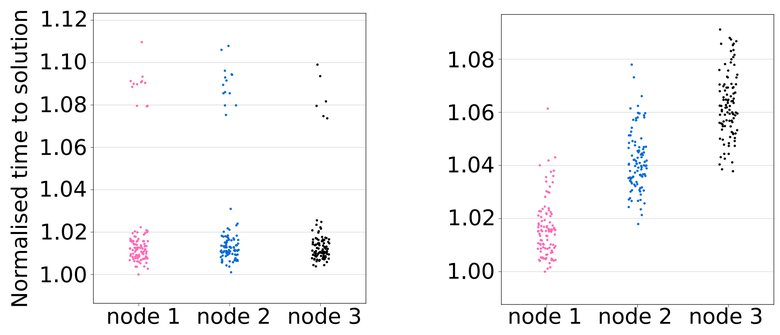
Justs Zarins
j.zarins@ed.ac.uk

Michèle Weiland
m.weiland@epcc.ed.ac.uk

Performance variability (noise) hinders HPC efficiency of synchronous applications.

1

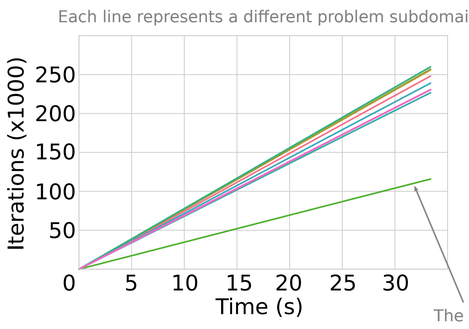
Same code
Same input
Different runtime



Each dot represents a repeated run of Jacobi's algorithm in shared memory.
Results are from two different HPC systems, ARCHER[3] (left) and Cirrus[4] (right).

Asynchronous algorithms do better but they still suffer from noise.

2

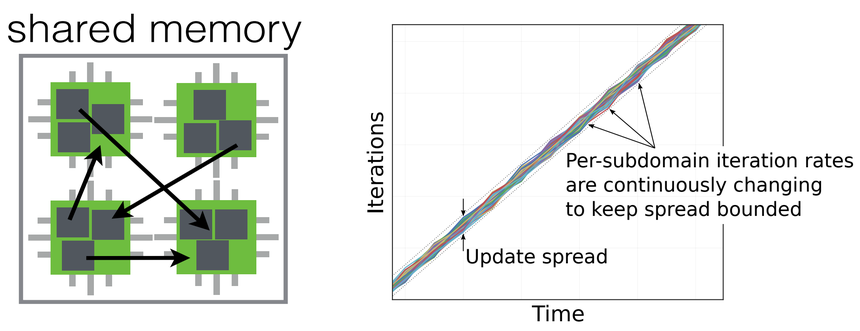


Update staleness
grows over time.
Efficiency decreases.

We propose load balancing specifically designed for asynchronous algorithms.

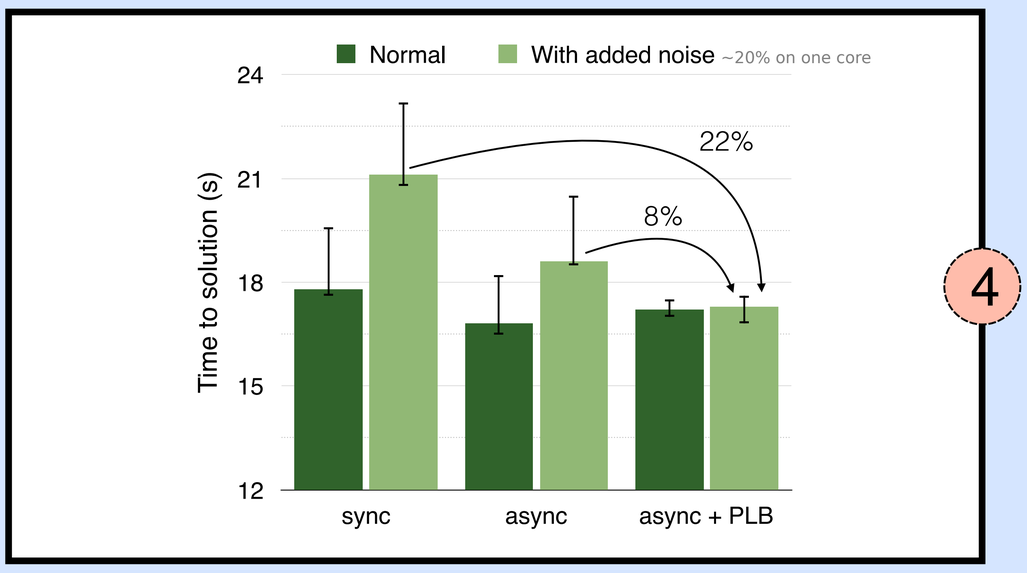
Progressive Load Balancing (PLB).

3



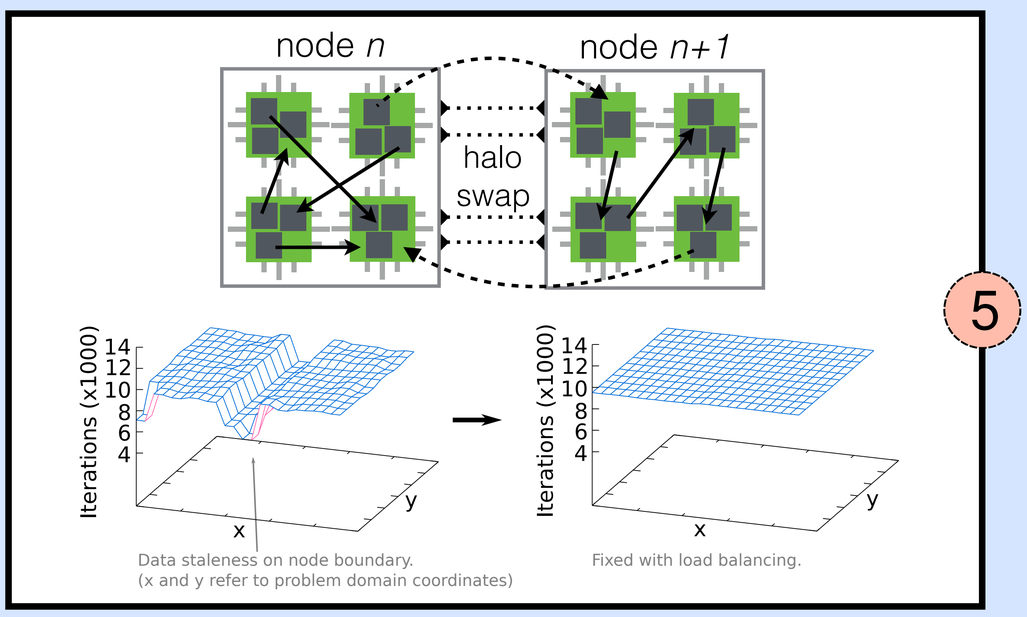
Noise is absorbed! Time to solution improves!

4



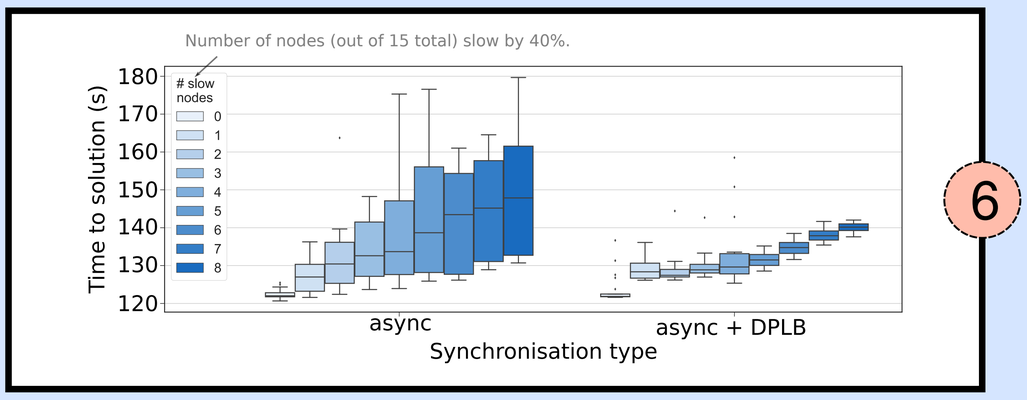
What about multiple nodes?
Add a cross-node layer: Distributed Progressive Load Balancing (DPLB).

5



Reduce progress imbalance and time to solution variability!

6



1) Large HPC machines are susceptible to irregular performance. Factors like chip manufacturing differences, heat management and network congestion combine to result in varying execution time for the same code. These issues are independent of the regularity of a problem.

2) Asynchronous, or chaotic, algorithms (used in linear algebra, machine learning, graph processing) offer a partial solution. In these algorithms fast workers are not forced to synchronise with slow ones. Instead they continue computing updates using the data available to them, which may have become stale (i.e. it is a number of iterations out of date compared to the most recent data). Consequently, the convergence rate of asynchronous algorithms tends to be lower.

3) To address this problem, we are developing load balancing strategies for iterative asynchronous algorithms in both shared and distributed memory settings. The method, called Progressive Load Balancing (PLB), works by exploiting the fact that an asynchronous algorithm allows some staleness without breaking down. Thus load can be balanced over time, as opposed to balancing instantaneously. Problem subdomains are periodically moved between workers to increase or decrease their load (for more details see [1]). Previous approaches are not suitable to asynchronous algorithms, are less robust against dynamic noise or are application specific.

4) We test PLB using Jacobi's algorithm. PLB shows a clear advantage, by redistributing the effect of the noise across available cores, thus avoiding any significant slowdown.

5) In distributed memory PLB is still running on each node, but now a cross-node balancing layer also periodically query the average progress of nodes. Given this information, problem subdomains are sent from nodes that are falling behind to nodes that are running ahead.

6) We observe a reduction in global progress imbalance of 1.08x–4.05x and a reduction in time to solution variability of 1.11x–2.89x on 15 nodes with up to half of them running with a 40% slowdown (level set according to [2])

[1] doi.org/10.1145/3149704.3149765
[2] doi.org/10.1145/2807591.2807638
[3] www.archer.ac.uk/
[4] www.cirrus.ac.uk/

EPSRC Centre for Doctoral Training in Pervasive Parallelism
THE UNIVERSITY of EDINBURGH
informatics
epcc
EPSRC Engineering and Physical Sciences Research Council