Progressive Load Balancing in Shared and Distributed Memory

JUSTS ZARINS, University of Edinburgh
MICHÈLE WEILAND, EPCC

CCS Concepts: • **Computing methodologies → Massively parallel algorithms**;

Additional Key Words and Phrases: asynchronous algorithm, load balancing, performance variability

ACM Reference Format:
Justs Zarins and Michèle Weiland. 2018. Progressive Load Balancing in Shared and Distributed Memory. 1, 1, Article 1 (August 2018), 2 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 POSTER SUMMARY

Large HPC machines of today and of tomorrow 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 and input sets [4–6]. These issues are independent of the regularity of a problem - even an application with even workload distribution would face performance variation under the stated circumstances. As a result, tightly coupled synchronous applications suffer large efficiency losses when the whole application is forced to wait for the slowest parallel workers. This significantly hinders the efforts to achieve efficient utilisation of petascale and exascale machines.

Asynchronous, or chaotic [3], algorithms offer a partial solution. In these algorithms, usually iterative convergent in nature, fast workers are not forced to synchronise with slow ones. Instead they continue computing updates, and moving towards the solution, using the latest data available to them, which may have become stale (i.e. it is a number of iterations out of date compared to the most recent data). While this allows for high computational efficiency, the convergence rate of asynchronous algorithms tends to be lower. Thus the problem of performance variability can be seen to translate into one of progress variability.

To address this problem, we are using the unique properties of asynchronous algorithms to develop load balancing strategies for iterative asynchronous algorithms in both shared and distributed memory settings. The method, called Progressive Load Balancing (PLB), works by recognising the fact that an asynchronous algorithm by definition 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. This leads to varying update rates for each subdomain, which can be controlled to estabilsh a dynamic bound on the progress variation within the system (see Figure 1). Doing load balancing in this way allows for greater computational efficiency and less frequent balancing. In addition, good balance can be achieved using coarser problem subdivision which reduces the number of communication links required.

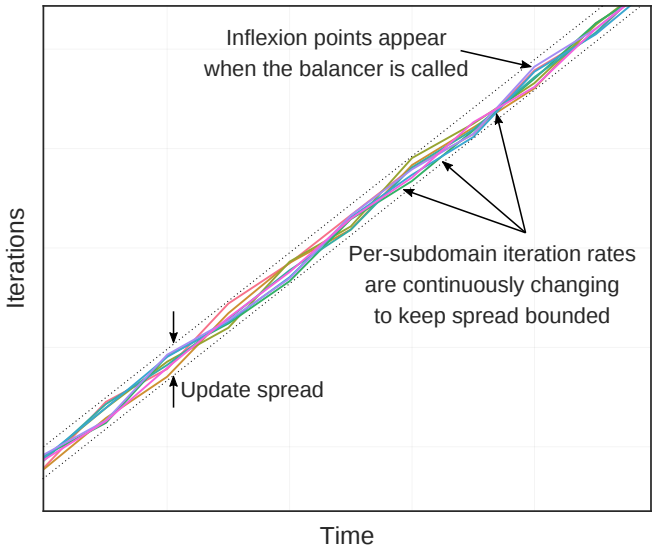

Fig. 1. An example of progressive load balancing. Problem subdomains (represented by one coloured line each) are being moved between slow and fast running threads to ensure an overall even progress towards the solution. Load balancing in this style is only possible with asynchronous algorithms. This picture is drawn using data from a real experiment.

We test PLB using Jacobi's algorithm in its asynchronous variant. In the shared memory setting we are able to reduce data staleness while maintaining a high iteration rate in comparison to synchronous and semi-synchronous (staleness tolerated up to a set limit) counterparts [7]. To further test the method, we want to evaluate the case of faulty hardware. System noise, and thus performance variability, is unpredictable. In our experiments, we therefore simulate the situation of noise being generated by a slow processing core. Under these circumstances PLB shows a clear advantage, by redistributing the effect of the noise across available cores, thus avoiding any significant slowdown. In terms of time to solution this results in a 5%–25% speedup over other synchronisation times (this is with 19% noise added to one core).

To go beyond shared memory we add a separate balancing layer that deals with cross-node data movement; the extended scheme is called Distributed Progressive Load Balancing (DPLB). PLB is still running on each node, but now we 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. Again, exploiting the available slack in asynchronous algorithms makes this method efficient. We observe a reduction in global progress imbalance of 1.08x–4.05x and a reduction in time to

XXXX-XXXX/2018/8-ART1
https://doi.org/10.1145/nnnnnnn.nnnnnnn

solution variability of 1.11x–2.89x on 15 nodes with up to half of them running with a 40% slowdown.

Performance variability is a serious performance issue facing modern HPC machines. Asynchronous algorithms have great promise to solve this issue. However, we argue that, to fully exploit their advantages, load balancing is still required. We present a solution specific to asynchronous algorithms and show that it can deal with a large amount of performance variation and results in better performance overall. Currently we are working towards evaluating PLB and DPLB on other asynchronous algorithms, for example asynchronous stochastic gradient descent.

ACKNOWLEDGMENTS

This work was supported by grant EP/L01503X/1 for the University of Edinburgh School of Informatics Centre for Doctoral Training in Pervasive Parallelism (pervasiveparallelism.inf.ed.ac.uk) from the UK Engineering and Physical Sciences Research Council (EPSRC).

This work used the ARCHER UK National Supercomputing Service [1] and EPCC's Cirrus HPC Service [2].

REFERENCES

[1] 2018. ARCHER. Retrieved 17-5-2018 from http://www.archer.ac.uk
[2] 2018. Cirrus. Retrieved 17-5-2018 from https://www.epcc.ed.ac.uk/cirrus
[3] D. Chazan and W. Miranker. 1969. Chaotic relaxation. *Linear Algebra Appl.* 2, 2 (apr 1969), 199–222. https://doi.org/10.1016/0024-3795(69)90028-7
[4] Pedro J Garcia, J Flich, J Duato, I Johnson, Francisco J Quiles, and F Naven. 2005. Dynamic evolution of congestion trees: Analysis and impact on switch architecture. *Lecture Notes in Computer Science (HiPEAC 2005)* 3793 (2005), 266–285.
[5] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. 2015. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. 896–904. https://doi.org/10.1109/IPDPSW.2015.70
[6] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, et al. 2015. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 78.
[7] Justs Zarins and Michèle Weiland. 2017. Progressive Load Balancing of Asynchronous Algorithms. In *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms (IA3'17)*. ACM, New York, NY, USA, Article 5, 9 pages. https://doi.org/10.1145/3149704.3149765