Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Tom CORNEBIZE

Thèse dirigée par **Arnaud LEGRAND**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de l'École Doctorale **MSTII**

Le Titre de la Thèse
The English Title

Thèse soutenue publiquement le **1er janvier 1970**,
devant le jury composé de :

I dedicate this thesis to my grumpy cat.

> *Elle est où la poulette ?*
>
> — **Kadoc De Vannes**

Remerciements

(Acknowledgments)

I would like to thank everyone, except from Dobby the free elf.

Merci public !

Abstract / Résumé

Abstract

The English abstract.

Résumé

Le résumé en français.

Contents

1 Introduction

To introduce my work, I will write a nice introduction in the following. Citation example for the Top500 website [@top500] and some random paper [Gra69].

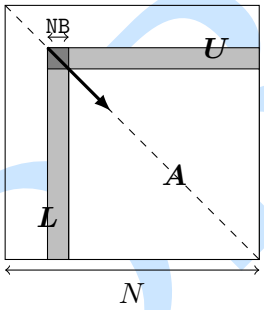
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2 Performance prediction through simulation: the HPL case

The work presented in this chapter has been published at a conference[CLH19] and has been submitted for publication in a journal. The content of this chapter is therefore a near-verbatim copy of these articles.

2.1 High Performance Linpack

2.1.1 The benchmark



allocate and initialize A;
for $k = N$ **to** 0 **step** NB **do**
  allocate the panel;
  factor the panel;
  broadcast the panel;
  update the sub-matrix;
end

Figure 2.1: Overview of High Performance Linpack

In this work, we use the freely-available reference-implementation of HPL[Pet+16], which relies on MPI. HPL implements a matrix factorization based on a right-looking variant of the LU factorization with row partial pivoting and allows multiple look-ahead depths. The principle of the factorization is depicted in Figure2.1. It consists of a series of panel factorizations followed by an update of the trailing sub-matrix. HPL uses a two-dimensional block-cyclic data distribution of A and implements several custom MPI collective communication algorithms to efficiently overlap communications with computations. The main parameters of HPL are:

- `N` is the order of the square matrix A.

- `NB` is the *blocking factor*, i.e. the granularity at which HPL operates when panels are distributed or worked on.

- `P` and `Q` denote the number of process rows and the number of process columns, respectively.

- `RFACT` determines the panel factorization algorithm. Possible values are Crout, left- or right-looking.

- `SWAP` specifies the swapping algorithm used while pivoting. Two algorithms are available: one based on *binary exchange* (along a virtual tree topology) and the other one based on a *spread-and-roll* (with a higher number of parallel communications). HPL also provides a panel-size threshold triggering a switch from one variant to the other.

- `BCAST` sets the algorithm used to broadcast a panel of columns over the process columns. Legacy versions of the MPI standard only supported non-blocking point-to-point communications, which is why HPL ships with in total 6 self-implemented variants to overlap the time spent waiting for an incoming panel with updates to the trailing matrix: `ring`, `ring-modified`, `2-ring`, `2-ring-modified`, `long`, and `long-modified`. The `modified` versions guarantee that the process right after the root (i.e. the process that will become the root in the next iteration) receives data first and does not further participate in the broadcast. This process can thereby start working on the panel as soon as possible. The `ring` and `2-ring` versions each broadcast along the corresponding virtual topologies while the `long` version is a *spread and roll* algorithm where messages are chopped into `Q` pieces. This generally leads to better bandwidth exploitation. The `ring` and `2-ring` variants rely on `MPI_Iprobe`, meaning they return control if no message has been fully received yet, hence facilitating partial overlap of communication with computations. In HPL 2.1 and 2.2, this capability has been deactivated for the `long` and `long-modified` algorithms. A comment in the source code states that some machines apparently get stuck when there are too many ongoing messages.

- `DEPTH` controls how many iterations of the outer loop can overlap with each other.

The sequential complexity of this factorization is

$$\text{flop}(N) = \frac{2}{3}N^3 + 2N^2 + \%_{00}(N)$$

where N is the order of the matrix to factorize. The time complexity can be approximated by

$$T(N) \approx \frac{\left(\frac{2}{3}N^3 + 2N^2\right)}{P \cdot Q \cdot w} + \Theta((P+Q) \cdot N^2)$$

where w is the flop rate of a single node and the second term corresponds to the communication overhead which is influenced by the network capacity and the previously listed parameters (`RFACT`, `SWAP`, `BCAST`, `DEPTH`, ...) and is very difficult to predict.

2.1.2 Typical runs on a supercomputer

Although the TOP500 reports precise information about the core count, the peak performance and the effective performance, it provides almost no information on how (software versions, HPL parameters, etc.) this performance was achieved. Some colleagues agreed to provide us with the HPL configuration they used and the output they submitted for ranking (see Table2.1). In June 2013, the Stampede supercomputer at TACC was ranked 6th in the TOP500 by achieving $5168.1\,\mathrm{TFlop\,s^{-1}}$. In November 2017, the Theta supercomputer at ANL was ranked 18th with a performance of $5884.6\,\mathrm{TFlop\,s^{-1}}$ but required a 28-hour run on the whole machine. Finally, we ran HPL ourselves on a Grid'5000 cluster named Dahu whose software stack could be fully controlled.

Table 2.1: Typical runs of HPL

	Stampede@TACC	Theta@ANL	Dahu@G5K
`Rpeak`	$8520.1\,\mathrm{TFlop\,s^{-1}}$	$9627.2\,\mathrm{TFlop\,s^{-1}}$	$62.26\,\mathrm{TFlop\,s^{-1}}$
N	3,875,000	8,360,352	500,000
`NB`	1024	336	128
$P \times Q$	77×78	32×101	32×32
`RFACT`	Crout	Left	Right
`SWAP`	Binary-exch.	Binary-exch.	Binary-exch.
`BCAST`	Long modified	2 Ring modified	2 Ring
`DEPTH`	0	0	1
`Rmax`	$5168.1\,\mathrm{TFlop\,s^{-1}}$	$5884.6\,\mathrm{TFlop\,s^{-1}}$	$24.55\,\mathrm{TFlop\,s^{-1}}$
Duration	2 hours	28 hours	1 hour
Memory	120 TB	559 TB	2 TB
MPI ranks	1/node	1/node	1/core

The performance typically achieved by supercomputers (`Rmax`) needs to be compared to the much larger peak performance (`Rpeak`). This difference can be attributed to the node usage, to the MPI library, to the network topology that may be unable to

deal with the intense communication workload, to load imbalance among nodes (e.g. due to a defect, system noise, ...), to the algorithmic structure of HPL, etc. All these factors make it difficult to know precisely what performance to expect without running the application at scale. It is clear that due to the level of complexity of both HPL and the underlying hardware, simple performance models (analytic expressions based on N, P, Q and estimations of platform characteristics) may be able to provide trends but can by no means accurately predict the performance for each configuration (e.g. consider the exact effect of HPL's six different broadcast algorithms on network contention). Additionally, these expressions do not allow engineers to improve the performance through actively identifying performance bottlenecks. For complex optimizations such as partially non-blocking collective communication algorithms intertwined with computations, a very faithful modeling of both the application and the platform is required. One goal of this thesis was to simulate systems at the scale of Stampede. Given the scale of this scenario (3,785 steps on 6,006 nodes in two hours), detailed simulations quickly become intractable without significant effort.

2.2 Related work

2.2.1 Performance prediction

A first approach for estimating the performance of applications like HPL is statistical modeling of the application as a whole[LD12]. By running the application several times with small and medium problem sizes (of a few iterations of large problem sizes) and using simple linear regressions, it is possible to predict its makespan for larger sizes with an error of only a few percents and a relatively low cost. Unfortunately, the predictions are limited to the same application configuration and studying the influence of the number of rows and columns of the virtual grid or of the broadcast algorithms requires a new model and new (costly) runs using the whole target machine. Furthermore, this approach does not allow to study what-if scenarios (e.g. to evaluate what would happen if the network bandwidth was increased or if node heterogeneity was decreased) that are particularly useful when investigating potential performance improvements.

Simulation provides the details and flexibility missing to such black-box modeling approach. Performance prediction of MPI applications through simulation has been widely studied over the last decades but two approaches can be distinguished in the literature: offline and online simulation.

With the most common approach, *offline simulation*, a trace of the application is first obtained on a real platform. This trace comprises sequences of MPI operations and CPU bursts and is given as an input to a simulator that implements performance models for the CPUs and the network to derive predictions. Researchers interested in finding out how their application reacts to changes to the underlying platform can replay the trace on commodity hardware at will with different platform models. Most HPC simulators available today, notably BigSim[ZKK04], Dimemas[Bad+03] and CODES[Mub+16], rely on this approach. The main limitation of this approach comes from the trace acquisition requirement. Not only is a large machine required but the compressed trace of a few iterations (out of several thousands) of HPL typically reaches a few hundred MB, making this approach quickly impractical[Cas+15]. Worse, tracing an application provides only information about its behavior at the time of the run: slight modifications (e.g. to communication patterns) may make the trace inaccurate. The behavior of simple applications (e.g. `stencil`) can be extrapolated from small-scale traces[WM11; CLT13] but this fails if the execution is non-deterministic, e.g. whenever the application relies on non-blocking communication patterns, which is unfortunately the case for HPL.

The second approach discussed in the literature is *online simulation*. Here, the application is executed (emulated) on top of a simulator that is responsible to determine when each process is run. This approach allows researchers to study directly the behavior of MPI applications but only a few recent simulators such as SST Macro[Jan+10], SimGrid/SMPI[Cas+14] and the closed-source xSim[Eng14] support it. To the best of our knowledge, only SST Macro and SimGrid/SMPI are mature enough to faithfully emulate HPL. This work relies on SimGrid as its performance models and its emulation capabilities seemed quite solid but the proposed developments would a priori also be possible with SST. Note that the HPL emulation described in Section2.3 should not be confused with the application skeletonization[Bir+13] commonly used with SST. Skeletons are code extractions of the most important parts of a complex application whereas we only modify a few dozens of lines of HPL before emulating it with SMPI. Finally, it is important to understand that the proposed approach is intended to help studies at the level of the whole machine and application, not the influence of microarchitectural details as intended by MUSA[Gra+16].

2.2.2 Simgrid/SMPI

SimGrid[Cas+14] is a flexible and open-source simulation framework that was originally designed in 2000 to study scheduling heuristics tailored to heterogeneous

grid computing environments but has later been extended to study cloud and HPC infrastructures. The main development goal for SimGrid has been to provide validated performance models particularly for scenarios making heavy use of the network. Such a validation usually consists of comparing simulation predictions with results from real experiments to confirm or debunk network and application models.

SMPI, a simulator based on SimGrid, has been developed and used to simulate unmodified MPI applications written in C/C++ or FORTRAN[Deg+17]. The complex network optimizations done in real MPI implementations need to be considered when predicting the performance of MPI applications. For instance, the "eager" and "rendez-vous" protocols are selected based on the message size, with each protocol having its own synchronization semantics, which strongly impact performance. SMPI supports different performance modes through a generalization of the LogGPS model. Another difficult issue is to model network topologies and contention. SMPI relies on SimGrid's communication models where each ongoing communication is represented as a whole (as opposed to single packets) by a *flow*. Assuming steady-state, contention between active communications can then be modeled as a bandwidth sharing problem that accounts for non-trivial phenomena (e.g. cross-traffic interference[Vel+13]). If needed, communications that start or end trigger a re-computation of the bandwidth share. In this fluid model, the time to simulate a message passing through the network is independent of its size, which is advantageous for large-scale applications frequently sending large messages and orders of magnitude faster than packet-level simulation. SimGrid does not model transient phenomena incurred by the network protocol but accounts for network topology and heterogeneity. Special attention to the modeling of collective communication algorithms has also been paid in SMPI, but this is of little significance in this article as HPL ships with its own implementation of collective operations.

SMPI maps every MPI rank of the application onto a lightweight simulation thread. These threads are then run one at a time, i.e. in mutual exclusion. Every time a thread enters an MPI call, SMPI takes control and the time that was spent computing (isolated from the other threads) since the previous MPI call is injected into the simulator as a virtual delay. This time may be scaled up or down depending on the speed of the simulated machine with respect to the simulation machine. Recent results report consistent performance predictions within a few percent for standard benchmarks on small-scale clusters (up to 12×12 cores [Hei+17] and up to 128×1 cores [Deg+17]). In this thesis, I validate this approach at a much larger scale with HPL, whose emulation comes with at least two challenges:

- The time-complexity of the algorithm is $\Theta(N^3)$ and $\Theta(N^2)$ communications are performed, with N being very large. The execution on the Stampede cluster took roughly two hours on 6006 compute nodes. Using only a single node, a naive emulation of HPL at the scale of the Stampede run would take about 500 days if perfect scaling was reached.
- The tremendous memory consumption and amount of memory accesses need to be drastically reduced.

2.3 Emulating HPL at large scale

some text...

2.4 Modeling HPL kernels and communications

some text...

2.5 Validation

some text...

2.6 Sensibility analysis

some text...

3 Experimental control

3.1 Experimental Testbed and Experiment Engines

some text...

3.2 Randomizing matters!

some text...

3.3 Performance non-regression tests

some text...

4 Conclusion

Your beautiful conclusion. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Bibliography

[@top500] *TOP500 Website*. URL: `https://www.top500.org/` (visited on Sept. 7, 2020). cit. on p. 1

[Bad+03] Rosa M. Badia, Jesús Labarta, Judit Giménez, and Francesc Escalé. "Dimemas: Predicting MPI Applications Behaviour in Grid Environments". In: *Proc. of the Workshop on Grid Applications and Programming Tools*. June 2003. cit. on p. 7

[Bir+13] R. F. Bird, S. A. Wright, D. A. Beckingsale, and S. A. Jarvis. "Performance Modelling of Magnetohydrodynamics Codes". In: *Computer Performance Engineering*. Ed. by Mirco Tribastone and Stephen Gilmore. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 197–209. cit. on p. 7

[Cas+14] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms". In: *Journal of Parallel and Distributed Computing* 74.10 (2014), pp. 2899–2917. cit. on p. 7

[Cas+15] Henri Casanova, Frédéric Desprez, George S. Markomanolis, and Frédéric Suter. "Simulation of MPI applications with time-independent traces". In: *Concurrency and Computation: Practice and Experience* 27.5 (Apr. 2015), p. 24. cit. on p. 7

[CLH19] Tom Cornebize, Arnaud Legrand, and Franz C Heinrich. "Fast and Faithful Performance Prediction of MPI Applications: the HPL Case Study". In: *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. 2019 IEEE International Conference on Cluster Computing (CLUSTER). Albuquerque, United States, Sept. 2019. cit. on p. 3

[CLT13] Laura Carrington, Michael Laurenzano, and Ananta Tiwari. "Inferring Large-scale Computation Behavior via Trace Extrapolation". In: *Proc. of the Workshop on Large-Scale Parallel Processing*. 2013. cit. on p. 7

[Deg+17] Augustin Degomme, Arnaud Legrand, Georges Markomanolis, et al. "Simulating MPI applications: the SMPI approach". In: *IEEE Transactions on Parallel and Distributed Systems* 28.8 (Feb. 2017), p. 14. cit. on p. 8

[Eng14] Christian Engelmann. "Scaling To A Million Cores And Beyond: Using Light-Weight Simulation to Understand The Challenges Ahead On The Road To Exascale". In: *FGCS* 30 (Jan. 2014), pp. 59–65. cit. on p. 7

[Gra+16] Thomas Grass, César Allande, Adrià Armejach, et al. "MUSA: A Multi-level Simulation Approach for Next-generation HPC Machines". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '16. Salt Lake City, Utah: IEEE Press, 2016, 45:1–45:12. cit. on p. 7

[Gra69] Ronald L. Graham. “Bounds on multiprocessing timing anomalies”. In: *SIAM journal on Applied Mathematics* 17.2 (1969), pp. 416–429. cit. on p. 1

[Hei+17] Franz C. Heinrich, Tom Cornebize, Augustin Degomme, et al. “Predicting the Energy Consumption of MPI Applications at Scale Using a Single Node”. In: *Proc. of the 19th IEEE Cluster Conference*. 2017. cit. on p. 8

[Jan+10] Curtis L. Janssen, Helgi Adalsteinsson, Scott Cranford, et al. “A Simulator for Large-scale Parallel Architectures”. In: *International Journal of Parallel and Distributed Systems* 1.2 (2010). `http://dx.doi.org/10.4018/jdst.2010040104`, pp. 57–73. cit. on p. 7

[LD12] Piotr Luszczek and Jack Dongarra. “Reducing the Time to Tune Parallel Dense Linear Algebra Routines with Partial Execution and Performance Modeling”. In: *Parallel Processing and Applied Mathematics*. Springer Berlin Heidelberg, 2012, pp. 730–739. cit. on p. 6

[Mub+16] M. Mubarak, C. D. Carothers, Robert B. Ross, and Philip H. Carns. “Enabling Parallel Simulation of Large-Scale HPC Network Systems”. In: *IEEE Transactions on Parallel and Distributed Systems* (2016). cit. on p. 7

[Pet+16] Antoine Petitet, Clint Whaley, Jack Dongarra, Andy Cleary, and Piotr Luszczek. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. `http://www.netlib.org/benchmark/hpl`. Version 2.2. Feb. 2016. cit. on p. 3

[Vel+13] Pedro Velho, Lucas Schnorr, Henri Casanova, and Arnaud Legrand. “On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations”. In: *ACM Transactions on Modeling and Computer Simulation* 23.4 (Oct. 2013), p. 23. cit. on p. 8

[WM11] Xing Wu and Frank Mueller. “ScalaExtrap: Trace-Based Communication Extrapolation for SPMD Programs”. In: *Proc. of the 16th ACM Symp. on Principles and Practice of Parallel Programming*. 2011, pp. 113–122. cit. on p. 7

[ZKK04] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant Kale. “BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines”. In: *Proc. of the 18th IPDPS*. 2004. cit. on p. 7

List of Figures

List of Tables

Abstract

The English abstract.

Résumé

Le résumé en français.