Performance Prediction of Parallel Scientific Applications

Rodrigo Escobar and Rajendra V. Boppana
Computer Science Department, University of Texas at San Antonio, San Antonio, TX, 78249, USA
rodrigod.escobar@gmail.com,rajendra.boppana@utsa.edu

ABSTRACT

Performance prediction of parallel applications has benefits across various domains in computing. It can provide data to developers for fine tuning specific code regions, and it can give information to job schedulers in high performance computing environments to improve resource utilization. In this work, we present an approach to performance prediction of parallel scientific applications using fractals. The fractal theory has been used before to model the cache miss ratios experienced by programs and the amount of traffic in access networks, among others. For four well-known parallel scientific applications, our technique gave good results with errors of less than 6% in most cases.

KEYWORDS

Performance modeling, Fractal theory, Parallel applications

1 INTRODUCTION

Performance prediction of parallel workloads on High Performance Computing (HPC) platforms has several applications. From code optimization to efficient job scheduling, having good estimates of the expected runtime of an application can help developers to know what parts of their programs need special attention, or help resource schedulers to dispatch jobs in order to improve overall load balance and throughput [6]. It can also help HPC users to determine the resources they need to request for their workloads since inaccurate estimations might result in early cancellation of jobs [14].

Most HPC application performance or runtime prediction techniques start by identifying one or more phases (or code blocks) that dominate the execution time of the target application using methods such as manual instrumentation of source code or automatic analysis of traces. The phase runtimes are predicted and aggregated for the overall application runtime prediction.

Known techniques to predict the performance of application phases include the execution of synthetic code or skeletons [11], static analysis, curve fitting [7], regression and machine learning [10]. Recently, we proposed a prediction method that uses small-scale executions along with statistical analyses to match each application phase to a scientific kernel from a previously built reference database and using the kernel runtime as a proxy for the application phase runtime [5]. The current techniques require extensive source code analysis [7, 11], pre-construction of reference kernel databases [5, 7], or large number of samples [10].

In this work, we develop a mathematical model based on the fractal theory to predict the performance of parallel applications that have major computational phases with polynomial time complexity. A fractal is a pattern that repeats itself at different scales [3]. The fractal theory has been used before to model and predict the cache miss ratios experienced by programs [13], to model the amount of traffic in access networks [1], and to model the burst of workloads in clouds [6], among others. We use the theory of fractal to estimate the runtimes of parallel scientific applications. Our experimental results for four well-known parallel scientific applications show that this method can give good prediction accuracy with errors less than 6% in most cases.

2 METHODOLOGY

We predict an application runtime in three steps: Detection of major computing functions (called phases, in this work) that account for most of the application runtime, prediction of the phase runtimes based on their fractal dimensions, and prediction of the overall application runtime. These steps are described below.

2.1 Detection of major phases of applications

The major phases of an application are determined using a few small-scale (small input) runs of the application. For each small-scale execution, we use the Tuning and Analysis Utilities (TAU [16]) to record, with low (1-2%) overhead, the execution time of each function in the application. Using the execution logs by TAU, we identify the functions which, cumulatively, account for 70% or more of the total runtime. In each of the four applications we used, just one or two functions take up 70% to 90% of the total runtime.

We determine, empirically, using several small-scale runs, a small input size C, beyond which the percentage of the execution time spent in the major phases is stable. Then, the application is run for a few, k, additional small input sizes $N_i = C + i\delta$, $1 \leq i \leq k$, and the runtimes of the phases and the overall application are recorded. In our experiments, about five runs were needed to determine C, and five more runs, $k = 5$, were enough to obtain accurate prediction times. Each small-scale run took significantly less time, often less than a minute, than that for the target input size.

2.2 Phase runtime prediction

For phases with polynomial time complexity, the property of self-similarity holds since the number of operations executed for different input sizes differ only by a constant factor on a log scale. Therefore, the runtime complexity of an application phase may be modeled by fractals, which have been used to model self-similar phenomena.

To predict a phase runtime for $N_{Lrg} > N_k$, we construct the data points $\left(\ln\left(\frac{N_i}{N_{Lrg}}\right), \ln(T_i)\right)$, where $1 \leq i \leq k$ and T_i are the phase runtimes observed for small-scale runs. We determine the regression line $mx + b$ that has the least-squares fit to the data. The slope m is the fractal dimension of the phase runtime polynomial, x is the x-coordinate in the data set, and e^b is the predicted runtime for N_{Lrg}. For instance, Figure 1 shows the linear fit to the experimental data obtained for the *residual* phase of *SMG2000* on 32 cores.

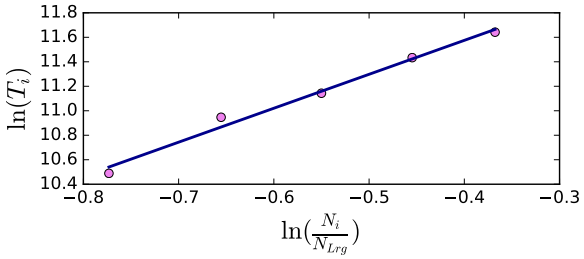


Figure 1: Linear fit for the *residual* phase of application *SMG2000*. $N_{Lrg} = 260$

2.3 Overall prediction

For the overall application runtime prediction, we also need to predict the runtime of the parts of the application that were not in the phases. To this end, we extrapolate, using linear regression, the percentage of time that will be spent by each major phase for the target input size N_{Lrg}.

If the estimated percentages of the total execution time spent in phases $P1, P2, \ldots, Ph$ are $r_{P1}, r_{P2}, \ldots, r_{Ph}$, respectively, and their predicted runtimes are $t_{P1}, t_{P2}, \ldots, t_{Ph}$, respectively, then our prediction for the entire application runtime is $T = (t_{P1} + \ldots + t_{Ph})/(r_{P1} + \ldots + r_{Ph})$.

3 EXPERIMENTAL RESULTS

Results obtained in this work were obtained using a cluster of five nodes at the University of Texas at San Antonio's Institute for Cyber Security (ICS) and four parallel scientific applications—SMG2000 [8], SNAP [2], HPCG [12], and CoMD[9], which were used in the most relevant prior works [5, 7, 11]—to evaluate the proposed prediction technique. Each node has two Intel Xeon X5650 6-core, 2.67GHz processors, 128GB DDR3 RAM, one 800 GB SATA II hard drive, and a 40Gb/s infiniband interconnection interface. The software environment consisted of CentOS 6.5, GCC 4.4.7, Open MPI 1.8, and TAU 2.26. We ran two sets of experiments: one using 8 cores (1 process/core) for application runs, and another using 32 cores.

Table 1: Prediction of application runtimes

App-Phase	$C; N_{Lrg}$	Phase Runtime		$\lvert\%Error\rvert$	
		Predicted	Actual	Phase	Overall
8 Cores					
comd-ljforce	112;320	5109.9	5120.4	0.2%	0.6%
hpcg-mg	56;256	803.2	794.8	1.1	1.4
snap-dim3	96;512	2875.7	2976.2	3.4	5.0
smg-cyclicred	120;260	70.3	78.9	11.0	11.0
-residual		105.7	113.0	6.5	
32 Cores					
comd-ljforce	112;512	5470.6	5516.4	0.8%	1.8%
hpcg-mg	56;256	1727.7	1723.4	0.2	3.3
snap-dim3	96;640	1525.7	1582.0	3.6	6.0
smg-cyclicred	120;260	166.7	167.2	0.3	2.7
-residual		322.5	320.5	0.6	

Table 1 shows the experimental data for various phase runtimes in the four applications. For some applications, the time required for processing increases when more cores are used. Therefore, although the same input size is used, execution time may be different, as in the case of SMG2000. The application SMG2000 has two major phases, while the others have just one major phase. The error in predicting a phase runtime is less than 5% in 8 out of 10 cases. Although the overall application runtime predictions require additional estimates of the non-phase runtimes, the errors were 6% or less in 7 out of 8 cases. We are currently conducting additional experiments using 256 or more cores on the Chameleon bare metal platform [4] at the Texas Advanced Computing Center.

4 CONCLUSIONS

In this work we presented an approach to predict runtimes of scientific HPC applications whose major execution phases have polynomial time complexity. Our approach is simple to use and only requires a few small-scale executions. Our preliminary experiments using different applications and number of processes gave good results with errors of less than 6% in most cases.

In the future work, we will explore how our technique can be expanded to take into account non-polynomial factors.

REFERENCES

[1] M. A. Arfeen, K. Pawlikowski, A. Willig, and D. McNickle. 2016. Fractal renewal process based analysis of emerging network traffic in access networks. In *2016 26th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 265–270. https://doi.org/10.1109/ATNAC.2016.7878820

[2] National Energy Reserach Scientific Computer Center. 2017. SNAP. (2017). http://www.nersc.gov/research-anddevelopment/apex/apex-benchmarks/snap/

[3] M. Chandra, T. Shrimali, and A. Gupta. 2012. A Survey: Recent Development in Fractals. In *2012 Fourth International Conference on Computational Intelligence and Communication Networks*. IEEE, 251–256. https://doi.org/10.1109/CICN.2012.36

[4] Chameleon Cloud. 2017. A configurable experimental environment for large-scale cloud research. (2017). https://www.chameleoncloud.org/

[5] R. Escobar and R. V. Boppana. 2016. Performance Prediction of Parallel Applications Based on Small-Scale Executions. In *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*. IEEE, 362–371. https://doi.org/10.1109/HiPC.2016.049

[6] M. Ghorbani, Y. Wang, Y. Xue, M. Pedram, and P. Bogdan. 2014. Prediction and control of bursty cloud workloads: A fractal framework. In *2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. IEEE, 1–9. https://doi.org/10.1145/2656075.2656095

[7] A. Jayakumar, P. Murali, and S. Vadhiyar. 2015. Matching Application Signatures for Performance Predictions Using a Single Execution. In *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 1161–1170. https://doi.org/10.1109/IPDPS.2015.20

[8] Lawrence Livermore National Laboratory. 2001. SMG2000. (2001). https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/smg/

[9] ExMatEx Extreme Materials at Extreme Scale. 2017. CoMD Proxy Application. (2017). http://www.exmatex.org/comd.html

[10] Karan Singh, Engin Ipek, Sally A. McKee, Bronis R. de Supinski, Martin Schulz, and Rich Caruana. 2007. Predicting parallel application performance via machine learning approaches: Research Articles. *Concurr. Comput. : Pract. Exper.* 19, 17 (2007), 2219–2235. https://doi.org/10.1002/cpe.v19:17

[11] S. Sodhi and J. Subhlok. 2004. Skeleton based performance prediction on shared networks. In *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004.* IEEE, 723–730. https://doi.org/10.1109/CCGrid.2004.1336704

[12] HPCG The High Performance Conjugate Gradients project. 2017. The high performance conjugate gradients (hpcg) benchmark. (2017). http://www.hpcg-benchmark.org/

[13] D. Thiebaut. 1989. On the fractal dimension of computer programs and its application to the prediction of the cache miss ratio. *IEEE Trans. Comput.* 38, 7 (1989), 1012–1026. https://doi.org/10.1109/12.30852

[14] L. T. Yang, Xiaosong Ma, and F. Mueller. 2005. Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*. IEEE, 40–40. https://doi.org/10.1109/SC.2005.20

Performance Prediction of Parallel Scientific Applications

Rodrigo D. Escobar and Rajendra V. Boppana, CS Department, UT San Antonio

UTSA Computer Science

Introduction

Benefits

- Determine resources needed for the execution of workloads.
- Identify the code regions for fine-tuning and performance improvement.
- Provide information for resource schedulers to improve overall load balance and throughput.

Approach

- Identify the major computing phases.
- Predict phase runtimes.
- Aggregate phase predictions to give an overall prediction.

Limitations

- Suitable for applications with polynomial runtimes.

Phase Prediction

- Profile the application for a few small-scale input sizes N_1, N_2, ..., N_k, where k is a small integer.
- Use the Tuning and Analysis Utilities (TAU) profiler configured with the profile only option.
- Use the profile logs to detect the functions (called phases in this work) that comprise most of the execution time.
- For phases with polynomial time complexity, the property of self-similarity holds.
- The number of operations executed for different input sizes differ only by a constant factor on a log scale.

Modeling with Fractals

Mathematical model based on the fractal theory to predict the performance of parallel applications that have major computational phases with polynomial time complexity.

Phase runtime prediction for input size N_{Lrg}:

- Use the profiles captured for the k small input executions to construct the data points:

$$\left(\ln\left(\frac{N_i}{N_{Lrg}}\right), \ln(T_i)\right), i \le i \le k$$

T_i: Phase runtimes observed for the small-scale runs.

- Fit a line $mx + b$ to the data points.

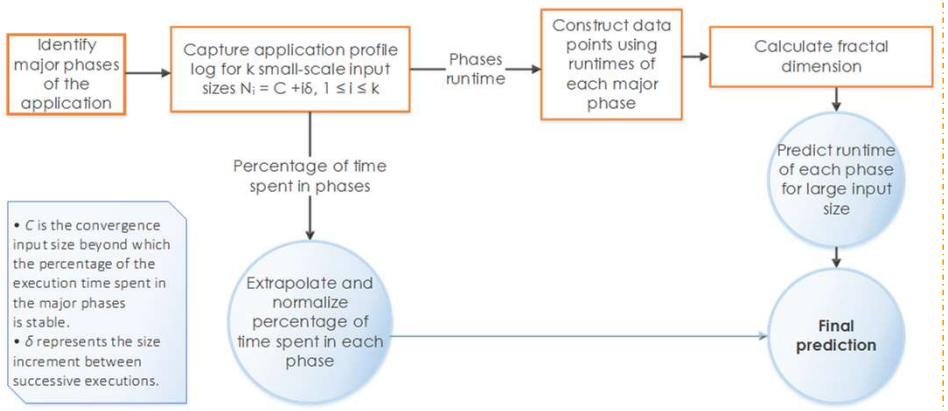
m: Fractal dimension of the phase runtime polynomial.

x: x-coordinate in the data set

b: Line intercept with the y-axis.

e^b: Predicted runtime for N_{Lrg}.

Prediction Methodology



Experimental Setup

Five nodes with the following characteristics:

Component	Characteristics
Processors	2 x Intel Xeon X5650 @2.67GHz per node
Cores/Threads	12/24 per node
Cache	L1 192 KB I + 192KB D L2 1.5MB I+D L3 12MB I+D per processor
Memory	24GB RAM DDR3
Disk	800 GB SATA II
Network	Infiniband Mellanox MT26428
Operating system	CentOS 6.5
Compiler	GCC 4.4.7
MPI	Open MPI v1.8

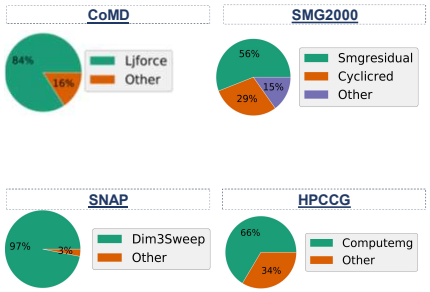
Applications

We used four well-known parallel scientific applications to evaluate our methodology:

- SNAP: A proxy application to model the performance of a modern discrete ordinates neutral particle transport application.
- HPCG: An application that implements the multigrid preconditioned conjugate gradient algorithm with a local symmetric Gauss-Seidel smoother.
- SMG2000: A parallel semi-coarsening multigrid solver for the linear systems arising from finite difference, finite volume, or finite element discretizations of a diffusion equation on rectangular grids.
- CoMD: A proxy application for the simulation of classical molecular dynamics (MD).

These applications were used in the most relevant prior works.

Time Distribution



Overall Prediction

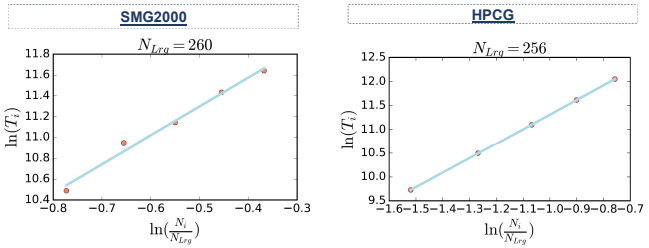
For the overall application runtime prediction, we also need to predict the runtime of the parts of the application that were not in the phases.

- Extrapolate, using linear regression, the percentage of time that will be spent by each major phase for the target input size N_{Lrg}.
- If the estimated percentages of the total execution time spent in phases $P_1, P_2, \ldots, P_h$ are $r_{P1}, r_{P2}, \ldots, r_{Ph}$, respectively, and their predicted runtimes are $t_{P1}, t_{P2}, \ldots, t_{Ph}$, respectively, then our prediction for the entire application runtime is:

$$T = \frac{\sum_i^h t_{Pi}}{\sum_i^h r_{Pi}}$$

Results

Linear Fit to Runtime Data Points



Predictions

App	Phase	C	N_k	N_{Lrg}	m	b	Phase Runtime (s) Predicted	Phase Runtime (s) Actual	|%Error| Phase	|%Error| Overall
8 Cores										
comd	ljforce	112.0	176.0	320.0	3.0	15.4	5109.9	5120.4	0.2%	0.6%
hpcg	computemg	56.0	120.0	256.0	3.1	13.6	803.2	794.8	1.1	1.4
smg2000	cyclicred	120.0	180.0	260.0	2.9	11.2	70.3	78.9	11.0	11.0
	smgresidual	120.0	180.0	260.0	3.2	11.6	105.7	113.0	6.5	
snap	dim3sweep	96.0	160.0	512.0	3.1	14.9	2875.7	2976.2	3.4	5.0
32 Cores										
comd	ljforce	112.0	176.0	512.0	3.0	15.5	5470.6	5516.4	0.8%	1.8%
hpcg	computemg	56.0	120.0	256.0	3.0	14.4	1727.7	1723.4	0.2	3.3
smg2000	cyclicred	120.0	180.0	260.0	2.9	12.0	166.7	167.2	0.3	2.7
	smgresidual	120.0	180.0	260.0	2.8	12.7	322.5	320.5	0.6	
snap	dim3sweep	96.0	160.0	640.0	3.0	14.2	1525.7	1582.0	3.6	6.0

Conclusions

- Presented an approach to predict runtimes of scientific HPC applications whose major execution phases have polynomial time complexity.
- Our approach is simple to use and only requires a few small-scale executions.
- Preliminary experiments gave good results with errors less than 6% in most cases.
- In our future work we will explore how our technique can be expanded to take into account nonpolynomial factors.

Contact:
Rodrigo D. Escobar: rodrigod.escobar@gmail.com
Rajendra V. Boppana: rajendra.boppana@utsa.edu