Understanding Potential Performance Issues Using Resource-based Alongside Time Models

Nan Ding
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
nanding@lbl.gov

Victor W Lee
Intel Corporation
Santa Clara, USA
victor.w.lee@intel.com

Wei Xue
Weimin Zheng
Tsinghua University
Haidian District, Beijing, China
xuewei@tsinghua.edu.cn

ABSTRACT

Performance analysis has been considered as a necessary step to bridge the widened gap between the actual and the expected performance of scientific computing applications (SCAs). Performance analysis tools are becoming one of the most critical components in today's HPC systems. Performance modeling, the core technology to identify key performance characteristics and predict potential performance bottlenecks, is becoming an indispensable tool to understand the performance behaviors and guide performance optimization of SCAs. Meanwhile, numerous challenges and opportunities are introduced by the complexity and enormous code legacy of SCA's, the diversity of HPC architectures, and the nonlinearity of interactions between SCAs and HPC systems. To address these issues, we propose the ***R***esource-based ***A***longside ***T***ime (RAT) modeling method to help to understand the application run-time performance efficiently. Firstly, we use hardware counter-assisted profiling to identify the key kernels and non-scalable kernels in the application. Secondly, we show how to apply the resource-based profiling into performance models to understand the potential performance issues and predict performance in the regimes of interest to developers and performance analysts. Thirdly, we propose an easy-to-use performance modeling tool for scientists and performance analytics. Our evaluations demonstrate that by only performing a few small-scale profilings, RAT is able to keep the average model error rate around 15% with average performance overheads of 3% in multiple scenarios, including NAS parallel benchmarks, dynamical core of atmosphere model of the Community Earth System Model (CESM), and the ice component of CESM over commodity clusters.

KEYWORDS

Performance modeling, Resource-based alongside time, Hardware counter, performance issues

ACM Reference Format:
Nan Ding, Victor W Lee, Wei Xue, and Weimin Zheng. 2018. Understanding Potential Performance Issues Using Resource-based Alongside Time Models. In *Proceedings of ACM conference (SC'18).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

SC'18, November, 2018, DALLAS, TX, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 SUMMARY

The ever-growing complexity of HPC applications, as well as the computer architectures, cost more efforts than ever to learn application behaviors by massive analysis of applications' algorithms and implementations. To efficiently make projections of applications' scaling run-time performance, designing performance models [1–13] has long been an art only mastered by a small number of experts. Nevertheless, we can still see that performance models can be used to quantify meaningful performance characteristics across applications [2, 3] and to provide performance bottlenecks associated with their implementations [14]; to offer a convenient mechanism for users and developers to learn the scaling performances [8, 9], and even to guide the optimization decisions [11].

We propose a ***R***esource-based ***A***longside ***T***ime (RAT) model (Tab. 1) that starts with an analytical model framework and predicts the computation and communication performance separately by using hardware counter-assisted profiling. We instrument the PMPI interface [15] to profile communication performance and then use the well-known Hockney model [16] to predict the communication performance. Such methods allow us to overcome the disadvantages of manual high-efforts (analytical models [10–13]) and unwarrantable model accuracy (empirical models [1–8]).

To summary, our contributions are as follows: **1. A hardware counter-assisted technique to identify function-level kernels.** As opposed to other typical performance modeling works, such as modeling each loop as a kernel [5, 17], we choose function-level kernels to model for two reasons. First, applications often have thousands of loops and understanding the significance of each loop can be challenging for users, even the well-known NAS Parallel Benchmarks (NPB [18]) contain hundreds of loops. Second, functions usually naturally separate communication and computation in parallel applications to enable us to predict them separately. We identify and model three kinds of kernel candidates by using execution cycle counter from profiling runs with different parallelisms.

- Functions whose share of the application execution cycles are larger than a user-defined threshold.
- Other functions whose run-time is not decreasing. Non-expensive functions may become expensive ones when we conduct the application run with different parallelisms or different inputs. For example, the functions in the sequential part can turn into hot-spots when running the application with more processes.
- The remaining functions except for the first and second set of functions. The reason is that the aggregated kernel can reduce the overhead of building performance models while maintaining good accuracy. Besides, the entire run-time would be slightly affected even if we consider those small functions individually.

Table 1: How is the model item derived?

	how is the model item derived?
T_comp_i	$\frac{instructions_i * CPI\_core_i}{CPUfrequecy * P}$
T_comm	$\sum_{i=1}^{r} T\_p2p + \sum_{i=1}^{l} T\_collective$
T_mem_i	$T\_L1_i + T\_L2_i + T\_LLC_i + T\_mm_i$
BF_mem_i	$\frac{T\_stall_i}{T\_mem_i}$
CPI_core_i	$CPI - \sum_{m=L1}^{Mem} Miss_m \cdot L_m$
BF_comm	$\frac{T\_mapp - T\_mcomp}{T\_mcomm}$
T_stall_i	fitting from RESOURCE_STALLS.LB(ST) counter
T_L1_i	fitting from MEM_LOAD_UOPS_RETIRED.L1_HIT_PS counter
T_L2_i	fitting from MEM_LOAD_UOPS_RETIRED.L2_HIT_PS counter
T_LLC_i	fitting from MEM_LOAD_UOPS_RETIRED.LLC_HIT_PS counter
T_mm_i	fitting from MEM_UOPS_RETIRED.ALL_LD(ST)_PS counter
$instructions_i$	fitting from INST_RETIRED.ANY_P counter
$T\_collective$	fitting from P and operation type
T_p2p	fitting from S
T_others	fitting from P

2. A Resource-based Alongside Time (RAT) model. The total run-time (T_app) equals to the accumulation of the compute kernels' run-times ($\sum T\_comp_i$, i refers to the identification of compute kernels) and the non-overlapped communication run-time (T_comm). Our model predicts T_app of a given parallel application on a target scale P_t by using several profiling runs with q processes, where $q \in \{2, ..P_0\}$, $P_0 < P_t$. Within the profiling runs, we use regression-based method ($f = a \cdot (\log P)^b \cdot P^c + d$) to fit each counter profiling results with number of processes (P). The point-to-point (p2p) communication time cost t of sending a certain number of message m of size s equals to $t = m \cdot (a \cdot s + b)$. According to the well-known Hockney model [16], we modeling the p2p communication time t with total communication size ts as $a \cdot ts^b + c$. We consider the *MPI_Bcast*, *MPI_Alltoall*, and *MPI_Allreduce* in the subset of MPI collective operations. Take *MPI_Bcast* as an example, the time cost t of a broadcast a message of size s among all processes P equals to $t = a \cdot log(P) + b \cdot s^c + d$. For a sake for simplicity, we use an average message size among processes rather than modeling each individual message.

The computing platform is a 4-node Intel Xeon cluster that contains two Intel Xeon E5-2698v3 processors running at 3.0GHz with 64GB of DDR3-1600 memory. In the case study of the Los Alamos sea-ice model (CICE [19]), RAT model guides us to find `Limited_gradient` (`L`) and `transport_integrals` (`T`) have similar CPIs. However, `L` has a higher *BF_mem* than `T` which indicts that `L` suffers from lower memory traffic as Fig. 1 shows. In NPB [18], Fig. 2 shows that RAT model can help to capture the performance insights that SP has a relatively bad memory behavior than BT. By looking into the SP code, it has some non-continuous memory accesses. Similar to the LU comparing to other workloads. Fig. 3 shows the comparison between the RAT model and the well-known model [20] based on Amdahl's law in multiple scenarios. Rat model can lower the model error rate while capturing the critical performance characteristics.

Calotoiu et al. [8] use performance models to find performance scalability bugs. This is probably the most similar work with ours. The main differences are that (1) they aim to report the kernel rankings while they do not separate the computation and communication. Thus the communication time reveals in strong scaling runs. However, the computation code sections that do not scale well are still hidden in the complex codeïijŇand (2) they use communications and floating point operations as metrics to evaluate the large-scale performance issues, while we provide the possible causes of the potential scaling issues by separating the memory effect from computations. To better understand the fine-grained performance, Bhattacharyya et al. [5] break the whole program into several loop kernels with the assumption that kernels can have simpler performance behaviors. However, this can be hundreds of kernels even for the NAS parallel benchmarks, and it is not effective to handle the complex loops and functions in real applications. Chatzopoulos et al. use the hardware counters to extrapolating the scalability of in-memory applications [21]. There is a consensus that performance modeling technique can be an effective approach for understanding the resource consumption and scalability.

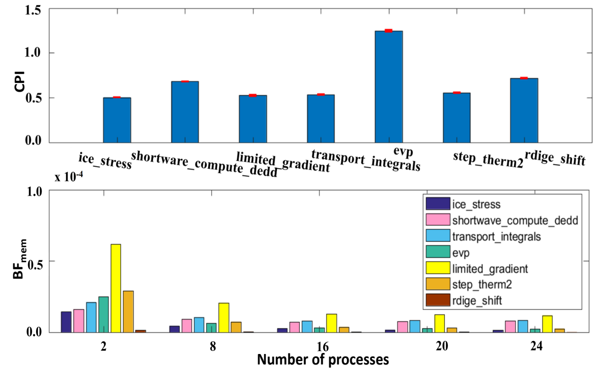

Figure 1: Distinguish similar CPIs from lower memory traffic/higher instruction efficiency and higher memory traffic/lower instruction efficiency in CICE [19].

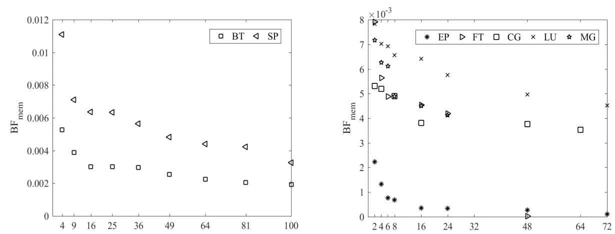

Figure 2: Characterize the memory impact of the NPB [18]. The x-axis is the number of processes.

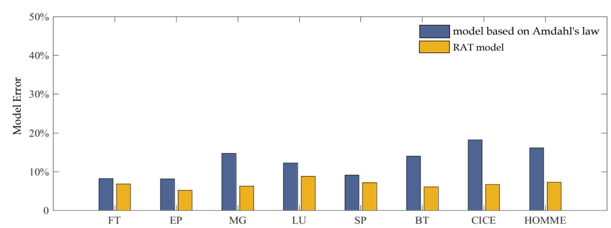

Figure 3: Compared to well known model [20], RAT model is able to lower the model error rate whilst capture the key performance characteristics.

ACKNOWLEDGMENTS

Authors from Lawrence Berkeley National Laboratory were supported by the U.S. Department of Energy's Advanced Scientific Computing Research Program under contract DEAC02-05CH11231. Authors from Tsinghua University are partially supported by the National Key R&D Program of China (Grant No. 2016YFA0602100 and 2017YFA0604500), National Natural Science Foundation of China (Grant No. 91530323 and 41776010).

REFERENCES

[1] Aniruddha Marathe, Rushil Anirudh, Nikhil Jain, Abhinav Bhatele, Jayaraman Thiagarajan, Bhavya Kailkhura, Jae-Seung Yeom, Barry Rountree, and Todd Gamblin. Performance modeling under resource constraints using deep transfer learning. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Denver, Colorado, USA, 2017.

[2] Prasanna Balaprakash, Ananta Tiwari, Stefan M. Wild, Laura Carrington, and Paul D. Hovland. Automomml: Automatic multi-objective modeling with machine learning. In *International Conference on High Performance Computing*, pages 219–239, 2016.

[3] Xingfu Wu, Charles Lively, Valerie Taylor, Hung Ching Chang, Chun Yi Su, Katherine Cameron, Steven Moore, Dan Terpstra, and Vince Weaver. *MuMMI: Multiple Metrics Modeling Infrastructure*. Springer International Publishing, 2014.

[4] Anthony P Craig, Sheri A Mickelson, Elizabeth C Hunke, and David A Bailey. Improved parallel performance of the cice model in cesm1. *The International Journal of High Performance Computing Applications*, 29(2):154–165, 2015.

[5] Arnamoy Bhattacharyya and Torsten Hoefler. Pemogen: Automatic adaptive performance modeling during program runtime. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, pages 393–404. ACM, 2014.

[6] Vivek Pallipuram, Melissa Smith, Nilim Sarma, Ranajeet Anand, Edwin Weill, and Karan Sapra. Subjective versus objective: classifying analytical models for productive heterogeneous performance prediction. *Journal of Supercomputing*, 71(1), 2015.

[7] Vivek K Pallipuram, Melissa C Smith, Nimisha Raut, and Xiaoyu Ren. A regression-based performance prediction framework for synchronous iterative algorithms on general purpose graphical processing unit clusters. *Concurrency and Computation: Practice and Experience*, 26(2):532–560, 2014.

[8] Alexandru Calotoiu, Torsten Hoefler, Marius Poke, and Felix Wolf. Using automated performance modeling to find scalability bugs in complex codes. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 45. ACM, 2013.

[9] Nan Ding, Shiming Xu, Zhenya Song, Baoquan Zhang, Jingmei Li, and Zhigao Zheng. Using hardware counter-based performance model to diagnose scaling issues of hpc applications. *Neural Computing and Applications*, pages 1–13, 2018.

[10] Greg Bauer, Steven Gottlieb, and Torsten Hoefler. Performance modeling and comparative analysis of the milc lattice qcd application su3_rmd. In *Cluster, Cloud and Grid Computing (CCGrid)*, pages 652–659. IEEE, 2012.

[11] Ding Nan, Xue Wei, Ji Xu, Xu Haoyu, and Song Zhenya. Cesmtuner: An auto-tuning framework for the community earth system model. In *High Performance Computing and Communications (HPCC)*, pages 282–289, Washington, DC, USA, 2014.

[12] Philip W Jones, Patrick H Worley, Yoshikatsu Yoshida, JB White, and John Levesque. Practical performance portability in the parallel ocean program (pop). *Concurrency and Computation: Practice and Experience*, 17(10):1317–1327, 2005.

[13] Sunpyo Hong and Hyesoon Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 152–163. ACM, 2009.

[14] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[15] Rainer Keller, Edgar Gabriel, Bettina Krammer, Matthias S Mueller, and Michael M Resch. Towards efficient execution of mpi applications on the grid: porting and optimization issues. *Journal of Grid Computing*, 1(2):133–149, 2003.

[16] Chau-Yi Chou, Hsi-Ya Chang, Shuen-Tai Wang, Kuo-Chan Huang, and Cherng-Yeu Shen. An improved model for predicting hpl performance. In *International Conference on Grid and Pervasive Computing*, pages 158–168. Springer, 2007.

[17] Arnamoy Bhattacharyya, Grzegorz Kwasniewski, and Torsten Hoefler. Using compiler techniques to improve automatic performance modeling. In *Proceedings of the 24th International Conference on Parallel Architectures and Compilation*. ACM, 2015.

[18] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The nas parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[19] Elizabeth C Hunke, William H Lipscomb, Adrian K Turner, et al. Cice: the los alamos sea ice model documentation and software userâĂŹs manual version 4.1 la-cc-06-012. *T-3 Fluid Dynamics Group, Los Alamos National Laboratory*, 675, 2010.

[20] Patrick H Worley, Anthony P Craig, John M Dennis, Arthur A Mirin, Mark A Taylor, and Mariana Vertenstein. Performance of the community earth system model. In *High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11. IEEE, 2011.

[21] Georgios Chatzopoulos, Aleksandar Dragojević, and Rachid Guerraoui. Estima: Extrapolating scalability of in-memory applications. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, page 27. ACM, 2016.