A Performance Tool for Earth System Models Development

Hang Li, Zhongzhi Luan
Sino-German Joint Software Institute
Beihang University
Beijing, China
lihangatmail@gmail.com

***Abstract*—We present a performance tool for Earth system models development to aid in analyzing the performance of the climate modeling applications. It is difficult for existing tools to handle with the complex, coupled structure and the long execution time of models. Our performance tool implements rapid analysis based on statistical sampling and grouping aggregation the calling relationship and the actual computing resource consumption excluding waiting losses. Using this tool, we study an ocean model POP in short-term sampling and analyze its scaling bottleneck and acceleration trend. The measuring results of its entire execution prove our predictions on the scaling efficiencies.**

Keywords-Earth system models; performance analysis; parallel computing

I. INTRODUCTION

Over the past decades, climate researchers have developed large-scale Earth science applications, such as ocean, atmosphere and land surface models, under monolithic software-development pattern. Recently, an emphasis on integrated multicomponent modeling has emerged—reflecting an increase in scientific capabilities and computing capacity, and resulting in several ambitious, coupled, Earth system modeling efforts[1, 10-13]. In these coupled efforts, Earth scientists request new technology and perspective to analyze the performance of climate parallel programs. Existed performance analysis method based the old program patterns cannot meet the demand of current continued innovation in complex, highly integrated simulations.

In the project of Earth System Model Oriented Integrated Developing Environment, we built an integrated development supporting environment for Earth system models using high performance computing technologies. The core part of the program is to provide a plugin platform which contains modules, templates and other tools required by earth scientists to build up component models they need and the entire Earth system model, and debugging and performance analysis tools for the models. Besides, performance adjustment is also a crucial part of the debugging work because model execution may last for several months. Unfortunately, it is extremely difficult for applications to use computing resources efficiently at large scale because inefficiencies emerge as a bottleneck on a large number of processors. Understanding which part of the application does not accelerate can be quite difficult. Climate experts need performance analysis tool to figure out the computing capability demand of each component so as to adjust the parameters. However, existing performance analysis techniques are not suitable to handle these structure and coupling characteristics of parallel climate programs [8, 9, 11]. The optimization of execution time or efficiency of computing resource allocation is badly needed, but it usually takes months to accomplish the optimization. In particular the Earth scientists find these techniques not user-friendly.

Our work is to provide useful tool for Earth scientists and application developers attempting to harness the power of parallel systems effectively. Originally, the tool is developed for the use of physicists. In this project, we aim to make it simpler for Earth scientists and yet provide coarse-grained detail about application performance bottlenecks. Our performance tool is not only used by programmers who write undying functions of component models, but also by the Earth scientists who tune and couple the component models into Earth system modeling applications. When using models, Climate experts usually tune the parameter and coupling structure instead of the procedure code in the component models. The way they tune the models differs from programmers. The professional background of Earth scientists concentrates on physics rather than parallel computing and programming. They are more willing to understand the climate programs from their professional view. In their perspective, the models consist of processes of physics computing and data conversion structure (called "coupler"). The performance analysis cannot be comprehended and used only if the results are translated into experts view.

Our performance tool promotes the efficiency of computing resource allocation to each component model. Waiting (e.g. MPI_WAITALL) loss is separated from the consumption of computing capability, so that measurement result is able to reflect the actual resource cost of each component. This is a runtime characteristic of parallel program which can hardly be obtained from code analysis. Using such information, climate experts will be able to make optimal adjustments to parameters (including parallel parameters) and other physics structures, such as couplers.

The rest of this paper is organized as follows. Section 2 present details of our approach to measurement and analysis of how much computing capability it cost to be attributed to each component. In Section 3, we use our tool to analyze the performance of a climate model in use and illustrate the result. Section 4 compares our approach with related work. Section 5 summarizes our conclusions.

Published by Atlantis Press, Paris, France.

II. PERFORMANCE MEASUREMENT

In this section we describe our measurement and analysis methods of consumption, separation and attribution.

A. Measuring Computing Resource Consumption

Computing resource refers to the cores on which a routine or a model executes and the occupancy time it lasts. MPI tasks usually implement on specific resource manager (such as PBS[15], SLURM[16], etc.), which ensures that enough computing capability is provided so that every process is able to own one core. Therefore measurement of tool includes functions that processes are executing and their occupancy related to duration and amount of processes. Additionally, there are three problems that need to be noted.

1) Measurement length. Tracking the entire execution of climate models which may last for several hours or months due to the time length of simulation is time-consuming and inefficient. In fact, the difficulty in this kind of applications is able to be resolved from their other characteristics. Earth applications usually consist of numerous processing loops that execute one after another. At each loop iteration, similar physics functions are dealing with data from previous loop and boundary. This kind of structure leads to a certain degree of self-similarity characteristic at particular levels manifested at program executions. Therefore we predict performance allocation of the entire execution from its short-term behaviors, and consider our tool as a monitor tool.

2) Measurement technique. The amount of loop structures limits our measurement approach at statistical sampling. Because of the numerous low-level loops of physics calculation, both source-level and dynamic binary instrumentation suffer from the large overhead and systematic dilation.

3) Focus data. In consideration of the time coverage and measurement method, the importance of performance distribution of component surpasses that of actual resource consumption. Using statistical sampling, we assume that the running content of program lasts from the sampling point for a time interval. Although the behaviors of processes between two sampling points are not measured, the distribution of components is expected to approximate the true distribution of their costs to measure, as long as the number of samples collected during execution is sufficiently large. However, even if the attribution of events is flawed, total time occupancy ratio within loops or procedures will typically be accurate. In most cases, it is the balance within countless loops that matters—for instance, the ratio between ocean model consumption and atmosphere model consumption.

To depict application behavior, performance tool samples stacks of run-time programs over a sampling interval. We leverage the Stack Trace Analysis Tool (STAT) [2], a scalable debugging tool from Lawrence Livermore National Laboratory, to collect stack traces from running applications. A single process exposes a sequence of stack traces representing the functions that is implemented at that time, which depict the caller/callee relationships of the functions being executed by that process. In most cases, these functions start with "main", and utilize the computing capability of current process (core actually) at the same time. Therefore the computing resource is transferred caller to callee. We assume that the functions always “own” the core until the next sample and add current consumption of resource to the records of particular functions (Figure 1).

In our model, we distinguish functions by invocation paths, which means that if the same function is invoked multiple times by different call paths, it occurs multiple times in our performance records. We believe that calling context is essential for understanding layered and coupled applications. The costs incurred for calls to communication primitives (e.g., MPI_WAITALL) vary widely depending upon their calling context. With this distinction, different application semantics such as waiting may be demonstrated by functions invoked via different call paths; in other words, we are more concerned about calling modules because the allocation of computing resource can be separately recorded and analyzed.

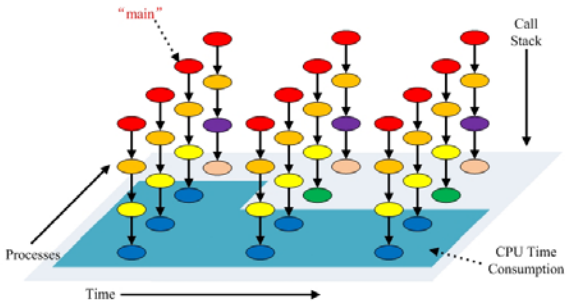


Figure 1. Measuring computing resource consumption of functions.

B. Attribution Functions to Modules

It is unreasonable to require users to wade through mountains of data to hunt for evidence of important problems. To make analysis of complex, coupled models useful to climate experts, performance tool should present measurement data in a hierarchical way, prioritizing what appear to be important problems and supporting a top-down analysis methodology that helps users quickly locate bottlenecks without the need to wade through irrelevant details[6].

From the sampling measurement, the execution information of program is transformed into 3 parts: the function (recorded as call path), the resource consumption, and the calling relationship. All the call paths can be merged into a call graph prefix tree. The presumption, as well as the common reality, is that there will be significant overlap amongst the individual stack traces such that many processes will merge into a relatively small call graph prefix tree [2]. Figure 2 depicts an example of merged graph and the CPU time of each function, and colors indicate the resource consumption.

Published by Atlantis Press, Paris, France.

At the same time, the functions are identified to their modules. The climate models usually use FORTRAN programming language, which is particularly suitable for processing scientific computing. This language writes the source file name of functions to the symbol table which is also pressed into call stack. This feature brings us more convenience to detect the belongings of functions.

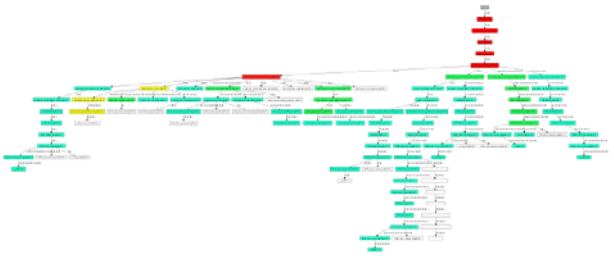


Figure 2. Merged graph of function cost and calling relationship

Then we attribute the function cost in the call graph to their modules. In fact it is a grouping aggregation with two features: keeping caller/callee relationships and separating waiting losses.

Waiting losses interfere with our judgment about the computing resource consumption. Therefore the aggregation should eliminate interference of waiting (MPI_WAITALL) losses which is detected at low-level paths and backtracked to all parent functions. The waiting cost is recorded separately and marked in the final result for further analysis. We use a recursive procedure with detective backtracking to complete grouping aggregation. A depth-first search achieved by the recursion detects the calling relationship of modules and return the cost allocated from the parent module to the sub-module, removing the cost of waiting losses backtracked from low-level invocations.

Figure 3 shows an aggregation instance from a 10000 samples measurement, whose call graph contains more than 150 nodes and 240 edges. The waiting losses are marked in the brackets, and you can pinpoint the source module and allocation of waiting costs.

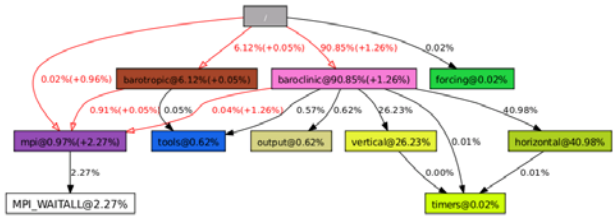


Figure 3. Grouping aggregation and attribution of physics modules

III. CASE STUDIES

Here we study the performance of Parallel Ocean Program (POP) to illustrate the uses of our performance monitor. We record how the computing resources is occupied and consumed by the member modules of the application.

As an ocean circulation model, POP solves the three-dimensional primitive equations for fluid motions on the sphere under hydrostatic and Boussinesq approximations. POP is the ocean component of the Community Climate System Model and has been used extensively in ocean-only mode for eddy-resolving simulations of the global ocean and for ocean-ice coupled simulations with the CICE model. In this study, we examine the performance of POP scaling from 2 to 32 cores. Understanding in detail how impediments to scaling arise in parallel applications, we help scientists identify the scaling bottleneck and select the appropriate scaling parameter to make effective use of computing capability.

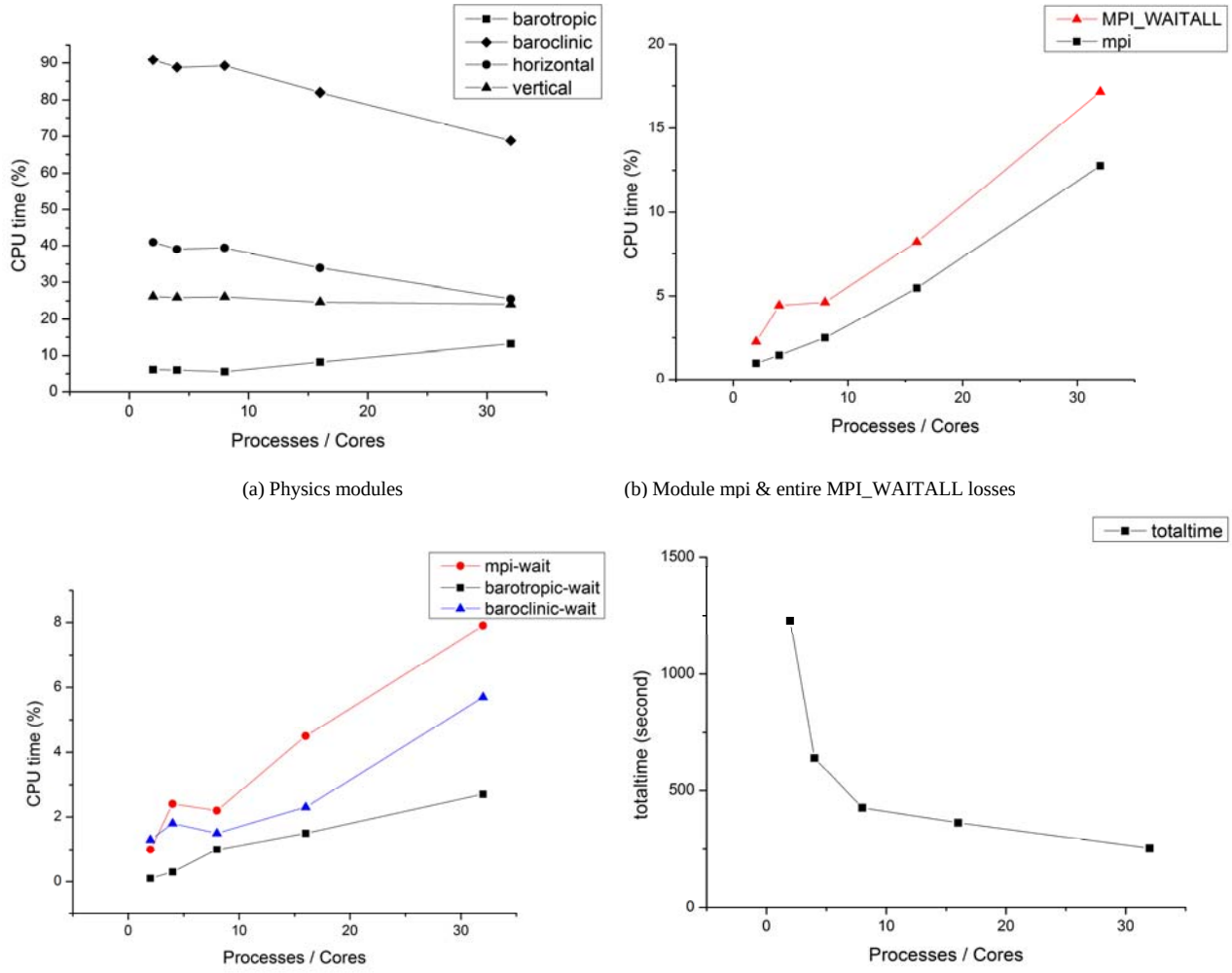
The main physics procedures of POP are baroclinic and barotropic. Barotropic also contains horizontal and vertical calculations. These four procedures spend most of the computing resource. Mpi module represents a number of mpi-related functions including boundary calculation, mpi communication, etc. There are also other non-physics

modules such as output and tools.

Figure 4. Performance monitoring of POP execution

Figure 4 shows a screen snapshot from performance monitor user interface displaying a top-down calling context view of how POP spends its time on 32 cores.

The view has three main components. The process monitor pane (upper pane) shows the sequencing view of module execution of each process. The modules relationship pane (lower left sub-pane) shows a top-down view of the calling relationship of execution modules. One can see the resource distribution along the call paths in the allocation graph. As for the performance that monitor tool measured, on 32 cores the allocation shows that POP spends 19.46% of its computing resource inside MPI_WAITALL. Although all MPI_WAITALL invocations are directly called in module mpi, excluding main (marked with slash), three modules suffering from waiting caused by MPI_WAITALL; in other words, these modules contribute to the waiting behaviors. The highlighted paths represent the invocations effected by waiting time, and the amounts of wasted resource are marked



(a) Physics modules

(b) Module mpi & entire MPI_WAITALL losses

(c) The contribution of high-level modules to waiting losses

(d) Actual execution time of the entire program on different cores

Figure 5. The contribution of high-level modules to waiting losses

in brackets. Low-level modules such as horizontal and vertical are not impacted. Without the interference of waiting (mostly for data from other processes), one can see the actual computing resource consumption for physics simulation.

A. Allocation of CPU Time

When executed by different process scales, test application is sampled by our performance monitor. We record the performance data of each execution which represent the runtime characteristic of the application in specific process parameter.

Figure 5(a) shows the actual computing resource consumption of four main physics procedures without waiting losses. The figure shows that the resource consumption of physics module load declines slowly comparing to the total resources. Figure 5(b) shows the consumption of module mpi and the total waiting losses of the application. The actual work cost of mpi module happens to grow along with the scaling of total cores. It can be interpreted as the boundary and communication work added by more data segmentation adapting to scaling cores. In contrast, the total waiting losses in MPI_WAITALL increase stably at the beginning, and have a sudden growth over 8 cores. The results revealed that even if work cost grows linearly with scaling, waiting cost shows the unstable, non-deterministic growth that may depend on the internal features of application. However, the poor scalability is clear: the 32 core execution spends more time waiting in MPI_WAITALL than in the 2-core execution.

By looking up the call chain to see how high-level modules caused the program to incur scalability losses in MPI_WAITALL, we discover in Figure 5(c) that from 4 cores on, the majority of waiting losses comes from the use of module mpi. More importantly, it grows sharply from 8 cores to 32 cores. Comparing with the module mpi, stable growth can also be seen in data processing module barotropic and baroclinic over 8 cores.

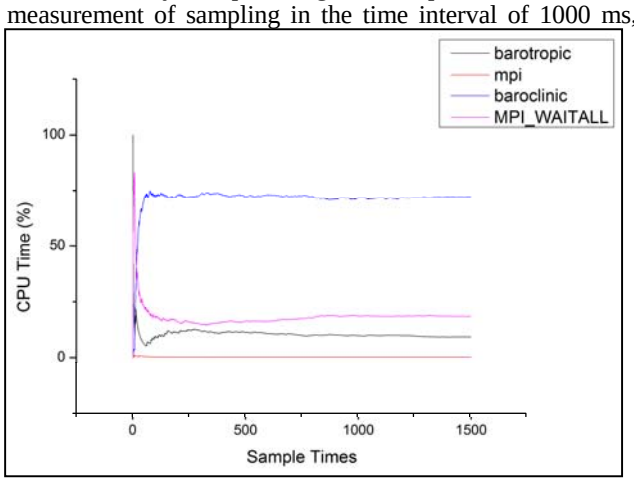
The trend reveals that the work of boundary calculation and mpi communication increases significantly for the scaling arise in parallel applications which bring fine-grained data segmentation and more boundary handling and communication work.

From the analysis above, we can focus the most efficient scaling of POP on 8 cores. Since at that scaling, the program performs better on the characteristic of the unstable, non-deterministic waiting losses, we can give a scaling suggestion considering the effective use of resources and program execution time.

It' should be noted that we obtain this result without measuring the entire execution of program which may be time-consuming. We make this analysis and prediction by the short-term sampling measurements. The actual running time of the program, as shown in Figure 5(d), proves our prediction. The case study demonstrates several facts. Some program scaling problems only show up beyond certain scales and depend on the features of program. Furthermore, the performance may be non-deterministic, and thus difficult to analysis from source code, so that short-term runtime sampling often play a key role in this case.

B. Measurement Speed

Our performance tool is suitable to analyze the execution of parallel programs consisted of countless low-level loops and iterations. The module cost ratios of cpu time are measured and estimated in limited samples. Although the final result after the entire execution is more accurate, an approximated analysis is provided in short-term samples. Apparently users need to make a compromise between accuracy and time consumption. However, our tool does not need too many samples. Figure 8 depicts the execution measurement of sampling in the time interval of 1000 ms,



and the estimated results is almost available after 300 samples.

IV. Related Work

Our work focuses on performance bottleneck diagnosed in complex, coupled parallel Earth system models. Although not designed for characteristics of climate applications, many tools have been developed for performance analysis of parallel computing platforms, with varying levels of perspective. VAMPIR [4] is a commercial post-mortem trace visualization tool. It uses profiling extensions to MPI and permits analysis of message events during parallel execution, helping to identify bottlenecks and inconsistent run time behavior. Performance tools such as Tau [5, 6], VTune [17] use source code instrumentation to insert special profiling code into the source program before compilation. Paradyn [7] is unique among performance analysis tools by using dynamic instrumentation to perform an online performance bottleneck search. Nearly all other tools to identify the root causes of load imbalance use instrumentation-based tracing. The basic disadvantages of these approaches are that instrumentation-based measurement faces an inelastic tension between accuracy and precision. HPCTOOLKIT [3, 14] uses statistical sampling to measure performance, which avoids the systematic overhead of measurement. But its post-mortem profiling and tracing is not fit to handle with the longtime execution and coupled structure of climate applications.

There are several existing techniques that also face the same challenges with respect to enhancing the practicality to Earth scientists who determine the final parameters of coupled programs which largely impact the performance. Performance tools typically attribute performance metrics to calling context. Two widely-used tools that collect call graph profiles are gprof [18] and Intel's VTune [17]. Call path profiling needs fully understand of performance aspects and module codes, but physics experts are difficult to handle with the low-level content of applications wrote by programmers.

V. Conclusions

We have presented the design and implementation of a performance monitor tool oriented to analyze the parallel computing resource consumption in Earth system models. This tool addresses the issue of how to help climate scientists quickly analyze and diagnose the performance bottlenecks for climate applications feature long-term running. Specifically, we handle the result from the perspective of climate experts who treat the application as coupled physics models. Our tool measures the computing resource consumption of each function (as call path) and attributes the cost to the coupling modules excluding the waiting losses.

With the tool, we have presented the case study of a real world ocean model POP and demonstrated that how we pinpoint the bottleneck of module which affects the scaling of the entire application. Besides, we also propose advices about scaling parameter. Even more importantly, the experiments of complete executions have proved our prediction made by short-term sampling measurements.

Acknowledgment

This work was supported by the National High Technology Research and Development Program of China (863 Program No. 2010AA012404). We would like to thank the support of Dr. Li Qingquan from National Climate Center and Dr. Wang Lanning from Earth System Simulation Lab of Beijing Normal University.

References

[1] C. Hill, C. DeLuca, V. Balaji, M. Suarez, and A. Da Silva, "The architecture of the Earth System Modeling Framework," *Computing in Science and Engineering*, 6(1), pp. 18–28, 2004.

[2] D. C. Arnold, D. H. Ahn, B. R. de Supinski, G. L. Lee, B. P. Miller, and M. Schulz, "Stack trace analysis for large scale debugging," *Parallel and Distributed Processing Symposium 2007(IPDPS 2007)*, IEEE International, Madison, pp. 1–10, 2007.

[3] N. R. Tallent, J. M. Mellor-Crummey, M. Franco, R. Landrum, and L. Adhianto, "Scalable fine-grained call path tracing," *Proceedings of the international conference on Supercomputing, ICS '11*, ACM, New York, USA, pp. 63-74, 2011

[4] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach, "VAMPIR: Visualization and analysis of MPI resources," *Supercomputer*, 12(1):69-80, 1996.

[5] S. S. Shende and A. D. Malony, "The TAU parallel performance system," *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[6] A. D. Malony, S. Shende, A. Morris, S. Biersdor, W. Spear, K. Huck, and A. Nataraj, "Evolution of a parallel performance system," *Tools for High Performance Computing*, Springer, Berlin Heidelberg, pp. 169–190, 2008.

[7] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, "The Paradyn parallel performance measurement tool," *Computer*, IEEE Computer Society, 28(11):37–46, 1995.

[8] N. R. Tallent and J. Mellor-Crummey, "Effective performance measurement and analysis of multithreaded applications," Technical report, Rice University, August 2008.

[9] N. R. Tallent, J. M. Mellor-Crummey, L. Adhianto, M. W. Fagan, and M. Krentel, "Diagnosing performance bottlenecks in emerging petascale applications", In *Proceedings of the Conference on High Performance Computing Networking*, ACM, New York, USA, pp. 1–11, 2009.

[10] Earth System Modeling Framework, http://www.earthsystemmodeling.org/, 2011

[11] P. H. Worley, A. P. Craig, J. M. Dennis, A. A. Mirin, M. A. Taylor, and M. Vertenstein, "Performance of the Community Earth System Model," In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, ACM, Seatle, WA, USA, pp. 1-11, 2011.

[12] M. Blackmon, B. Boville, F. Bryan, P. Gent, J. Kiehl, G. Bonan, et al., "The Community Climate System Model," *Bulletin of the American Meteorological Society*, American Meteorological Society, Boston, MA, 82(11): 2357–2376, 2001.

[13] V. Balaji, FMS: The GFDL Flexible Modelling System. http://www.gfdl.noaa.gov/fms, 2004.

[14] N. R. Tallent, L. Adhianto, and J. M. Mellor-Crummey, "Scalable identification of load imbalance in parallel executions using call path profiles," In *Proceedings of 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*, ACM, New Orleans, LA, pp. 1-11, 2010.

[15] R. L. Henderson, "Job scheduling under the portable batch system". In *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, pp. 279–294, 1995.

[16] A.B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," In *Job Scheduling Strategies for Parallel Processing*, SpringerVerlag, pp. 44-60, 2003.

[17] Intel Corporation. Intel VTune performance analyzer. http://www.intel.com/software/products/vtune.

[18] S. L. Graham, P. B. Kessler, and M. K. McKusick, "Gprof: A call graph execution profiler," In *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction (SIGPLAN '82)*, ACM Press, New York, NY, USA, pp. 120–126, 1982.