Augmenting HPC Profilers with Analysis Capabilities

1st Abhishek Patil
CDAC
Bangalore, India
abhishekpatil@cdac.in

2nd Shamjith K V
CDAC
Bangalore, India
shamjithkv@cdac.in

3rd Senthil Kumar R K
CDAC
Bangalore, India
senthil@cdac.in

4th Dr. S D Sudarsan
CDAC
Bangalore, India
sds@cdac.in

***Abstract*—This paper focuses on framework designed and developed by integrating multiple profiler modules to profile HPC applications. There is a constant demand for application performance on High Performance Computing (HPC) systems. Multiple software tools are currently available in the performance profiling realm to help application developers to profile and identify performance bottlenecks. Application developers strive to achieve the optimum execution of applications on available hardware resources using the profiling tools of their choice. They have to comprehend all the information provided by the tools and identify the bottlenecks to arrive at possible modifications. To assimilate the bottlenecks information from the profiling tool's outputs and representations, one requires experience and expertise in the application algorithm, programming methodologies, domain knowledge, and hardware resources. Many a time it is challenging for the developers to make changes in the application program based on the profiler output. We have developed a profile and analysis framework to identify the bottlenecks in an HPC application and help the developers improve application performance. The framework is developed by augmenting the existing opensource profilers with analysing capabilities for bottleneck identification and potential performance suggestions. This paper describes the architecture of the software framework and the benefits of the adopted mechanism.**

***Index Terms*—hpc profiling, performance analysis, hotspots identification, meaningful analysis, performance suggestions**

I. Introduction

High Performance Computing (HPC) systems are essential for scientific research areas involving computer simulations, like weather forecasting, material analysis, CFD, machine learning, and other applications to solve the grand challenges faced in their respective domains. In order to harness the compute power of the HPC system, utmost care must be taken to tune the performance[1] of the application under consideration.

HPC Systems are composed of multiple computing units, which are capable of carrying out computing operations much faster rate than the traditional computing units. These computing units are supported with high bandwidth and fast memory systems, with varying size and speed depending on how close they are available with the computing units. The computing units can even be of different architecture and working in coordinated way to carry out different tasks, and can also have associated accelerator devices to speed up specialized computations and algorithms. Many of these HPC systems are meant for processing large amount of data, hence a very sophisticated and high through put storage system is present on the nodes. In order carry out multiple tasks at the same time there will be many such computing nodes put together to form a computing cluster system, essentially following a master slave model.

Hardware units in HPC systems require many software drivers, system software and runtime systems to ensure that these components are geared up for performing their respective tasks. Once these components are ready for their operations, they require different set of programs to exploit the capabilities offered by each computing units, their memory subsystems, network systems and storage systems. Also, there will be specialized communication libraries and runtime systems to facilitate parallel communication among participating processes and parallel executing environments. So, in order to spin off an application that needs to be run on such a sophisticated HPC system requires amalgamation of multiple programming paradigms, specialized computation and communication libraries, along with necessary runtime systems.

Hence, the whole process of harnessing the supercomputing power offered by the HPC systems requires handling a great challenge in overcoming the complexities innate to the specialized components of those systems. In order to exploit the optimum performance offered by the HPC system and to impart the same in an application code, one requires in-depth understanding of the system, application domain and the programming models being employed. In order to overcome the challenges in performance optimization, HPC systems vendors have come up with many tools that can capture the performance profile and traces of application execution. These profile and trace information can be used to visualize the application behaviour at the run-time. These visually represented performance behaviour, through timelines, graphs and charts can help an application developer to understand the performance of the application on a given HPC system.

In order to catch up with the performance demand in the rapidly changing HPC systems architectures, the application must often be adapted [30] to execute efficiently [39] on systems with advanced architectures. HPC application performance analysis [3][40] is essential to harness the optimum performance [5] out of the computing resources available for application run or simulation.

To achieve performance improvements a developer needs to

carefully examine the application implementation and how it uses the underlying hardware resources [6]. There are a variety of methods to understand [28] how a scientific application uses computing hardware. The most straightforward and simple one is to get the timing of each section of the program; and the most extensive way is [7] to collect various information about the performance [8] events occurring during application execution by using profiling tools. Ultimately, the developer must apply his experience and expertise to identify the areas of bottlenecks [9] by assimilating all this information with the knowledge and expertise gained on algorithms and HPC systems. Typically, the application will be iteratively improved with successive executions and performance data collection on the target computing hardware [10].

Application developers have to invest time and effort to identify performance gaps in the application with respect to algorithms and hardware resources. There exist several limitations with respect to underlying hardware and software which won't allow a sub-optimal code to run at its best. Research efforts by Leiserson, C. E. et al.[33], presented the performance improvement in a matrix multiplication that can be obtained with optimization at software level on a target hardware Hence, the best way is to look at performance metrics which could provide application developers something insightful to comprehend.

This paper presents a modular guided approach with static and dynamic analysis mechanisms for performance measurement and bottleneck identification. The profiling framework categorizes the application based on percentage of total execution time in terms of communication, input-output, computation etc., with a direction that can be explored by the user to improve performance. Based on the most dominant category, application is executed again and performance metrics relevant to that category are presented for better understanding of underlying performance bottlenecks.

This paper is organized as follows. The Section II captures the details of various existing profilers currently available for users, both opensource and commercial ones. This is followed by Section III describing about the framework we have developed, where the architecture, design, implementation and experiments are explained. The Section IV describes the various use cases of the framework. The paper concludes with the features of the HPC profiler and analyser framework, and the future activities planned in the same direction.

II. EXISTING PROFILERS

There are plenty of tools for analysing performance behaviour of the HPC application on a computing hardware cluster. Each of the tools require specific understanding and expertise to derive conclusive action points from results for yielding performance benefits. In a multi-node parallel application making use of both distributed and shared memory parallel applications [13], the performance aspects need to be considered at multiple facets. One at the process and thread levels that is limited to a specific core or node, and the other level is the processes that spans across nodes that constitute the application. Most widely used tools [14] in the performance analysis domain are as follows.

ARM Map is the commercial application profiler for HPC applications from ARM [15]. Arm Map provides an easy-to-understand visual representation of the bottlenecks in an application. It also shows various performance metrics related to communication, input-output, vectorization etc. Performance Reports is a tool , where it provides a single page report on the application performance containing the input-output, communication and vectorization details, and provides exploring tips to further improve performance.

HPC Toolkit is an open-source toolkit for capturing and analysing the application performance on computer systems [16]. HPC Toolkit uses statistical sampling of various performance counters and timers in the hardware to gather accurate details of the application execution, resource consumption and inefficiency. It supports OpenMP, MPI and hybrid applications with both OpenMP and MPI. HPC Toolkit is developed by Rice University.

TAU, Tuning and Analysis Utilities, is a tracing and profiling open-source software toolkit developed by the University of Oregon [17]. It supports performance analysis of parallel applications written in C/C++, Fortran, Python and Java. It also supports the performance analysis of OpenMP, MPI and hybrid MPI OpenMP applications. TAU captures profiling and tracing information of the applications in detail, through instrumentation at variety of granularity levels such as functions, methods, and basic blocks. It also supports event-based sampling with the capability of capturing instruction level profile of the application. TAU presents the information to the user in a visually appealing and easy to understand GUI.

Intel VTune is a performance analysis tool for Intel architectures. It helps to analyse the performance of the parallel applications [18] with multiple profiling techniques like stack sampling, thread profiling and event sampling. It prefers users with prior experience in programming to understand and incorporate troubleshooting of the application issues. VTune can drill down the issues to instruction level. It supports multiple programming languages C/C++, Fortran, Python etc., and it also supports parallel paradigms like MPI, OpenMP and hybrid MPI + OpenMP. Due to the availability of lots of tools [29], each tool has an edge in a specific direction or aspect.

Our integrated approach followed in the framework for profiling and analysis help in exploring the dominant time-consuming aspects in the application execution. We have devised a multi-level profiling and analysis mechanism, the combination of static [21] and dynamic analysis [22][23] of the HPC application.

III. FRAMEWORK DESCRIPTION

A typical profiling software captures the inclusive and exclusive time spent in various regions of code, few advanced profiling software are able to provide deeper insights, like the performance of the application and its hotspot areas [24], with automated performance modelling[25]. Even after understanding the insights and representation provided by the advanced

profilers, developer requires a longer learning phase and expert knowledge to derive proper corrective actions. "Premature optimization is root of all evil" [33], without understanding the locality and actual nature of bottleneck the developer cannot make any code modifications and expect better performance.

We have put together various features offered by the available free and opensource software like Perf, LProf and CQA from MAQAO to get the performance details and added analysis and suggestion modules for respective sections to get insights into the issues and provide verbose suggestions for overcoming the issues.

To enhance the HPC Profiling Framework capabilities, we meticulously integrated various features that empower users to access comprehensive performance details. The framework essentially puts together multiple components of existing profiling software, and make use of various features to extract the bottleneck and performance details. These details are processed by the analysis module in the framework. We incorporated analysis and suggestion modules to look into each bottleneck sections. The suggestion module analyses the respective bottleneck sections and bring out relevant constructive verbose suggestions for performance improvement. Moreover, our framework goes beyond identification of bottlenecks and provides potential suggestions to overcome these issues, ensuring a holistic approach to performance improvement. By combining these functionalities, we have built a framework that helps users to optimize performance and achieve their desired outcomes. The framework supports Intel x86_64 architecture-based computing nodes, but it can also be extended to ARM's aarch64 architecture.

A. Architecture

The HPC Profiler framework is developed by stitching together various components available in the open-source domain based on their key features and augmenting features around that. The framework carries out the profiling and analysing operations mainly over multiple levels. The functional modules involved in the profiling and analysis are depicted in the Fig. 1 and Fig 2.

The initial stage of operation executes the user application and primarily focuses on capturing the intended profiling information. Analysis of the obtained profiling information helps in identifying hotspot regions in user programs. Static analysis module further analyses these hotspot regions of code in user applications to find out the instruction level behaviour of the application with respect to the micro-architecture. Static analysis module compares the binary code with machine model and returns a lower bound on number of cycles required to execute each instruction which helps in quantifying the performance gain that could be obtained if resource usage is optimized. Static analysis can also detect the applied compiler optimization heuristics and analyse them with respect to available hardware resource features to further augment these heuristics for potential performance enhancement.

The initial stage also categorizes the overall distribution of its execution time across different processes, with respect to computation, communication, input-output etc. This serves as a basis for deciding the directions to further explore. Along with profiling information, the initial stage is also capable of collecting raw values from the chosen hardware performance monitoring units.

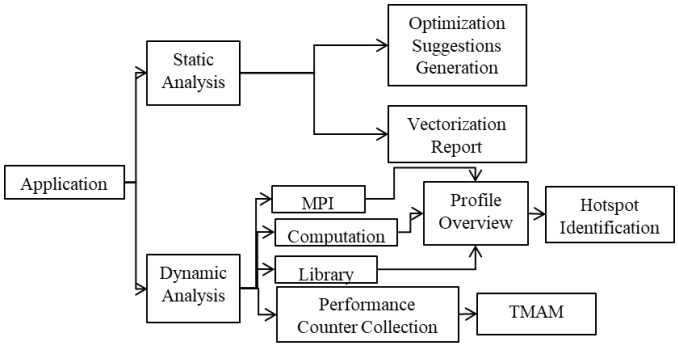

Fig. 1. Profiling stage

Our profiling tool mechanism has an ability to establish a boundary for different blocks of user code. And hence we can capture various performance events distinctly for various blocks of codes. Exploiting this feature and incorporating it with the raw values from the hardware monitoring units we have implemented Top-down Microarchitecture level Analysis Method (TMAM) [26] over the highlighted hotspot regions of user code.

TMAM is a directed approach monitoring the available PMUs [32] which highlights the performance anomalies which were not expected during the development phase of the application. Insights from TMAM helps application developers understand the nature of user applications and helps them in fine tuning the application in order to maximize the performance of the application. TMAM analysis modules identifies hotspot regions of code into frontend bound, backend bound (compute-bound, memory-bound), bad speculation, and retiring. Based on the orientation observed in the TMAM report, the respective suggestions along with directions for deeper analysis with respect to the hardware bottlenecks are formulated. TMAM is used to calculate the efficiency of highlighted hotspot functions and loops. If inefficient, further inefficiencies can be drilled down to highlight a primary cause that accounts for pipeline's resource usage.

Once the initial profiling stage concludes, it provides the application profile information which highlights the areas where further analysis needs to be carried out, and successive analysis stage is initiated. The analysis stage focus on capturing performance parameters based on the information collected from the profiling stage. If the application spends most of the execution time in computation, then the analysis is carried out for inferring how the application is utilizing available hardware resources at the microarchitecture level. This analysis can detect the bottleneck and their cost at the core level. In case with the breakdown of time, the MPI communication is the major contributor, the framework advice

for investigating the MPI communication calls for reducing the intercommunication time.

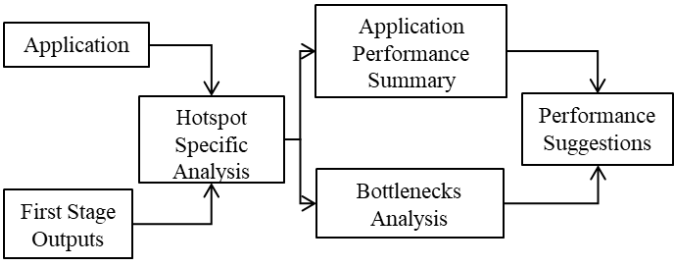

Fig. 2. Analysis stage

The core of our framework is the analysis applied in identifying the various factors that can yield improvement in application performance. This is accomplished through careful and meticulous analysis of the output generated by relevant profiling components, and correlating them with attainable performance improvements.

The framework uses static performance evaluation to spot inefficient code and check the performance of loops. It focuses on the x86 architecture and analyzes the code to provide further insights.

Our framework localizes the bottleneck areas in an application by identifying the sections that contribute the most towards performance anomaly at the application level and at the process level. The application section is localized with respect to the type of bottlenecks, converging under various granularity levels application-level execution, node level, process level, and thread level. The gathered information is further analysed for generating possible performance improvement suggestions. Our framework works with the unmodified executable; hence the measurement mechanism does not alter the application behaviour. This results in reduced overhead for performance profiling [27] and analysis.

B. Key features of the framework

The key features of the profiler framework are listed below.

1) Support for Multithreaded and Multi-process Applications: The current framework supports sequential application and parallel applications implemented using OpenMP for a single node. Additionally, it also supports multi-process parallel applications implemented using MPI paradigms on a compute cluster.

2) Use of unmodified executable: The framework adopts sampling based profiling approach hence it doesn't require modifications in the application binaries for collecting the application profile during the execution of the application. But the application has to be compiled with the "-g" option for profiling and analysis. Hence, the profiling is carried out with minimal execution overhead on an unmodified executable.

3) Multiple granularity levels for bottleneck detection: The profiling framework captures the execution details of the application at various granularity levels, such as at the application level that are innermost loops, vectorization and compute intensive regions, also detects presence of fused multiply add operations, details at node level, process level and thread level. This helps to localize the bottlenecks, at application, process and thread levels.

4) Constructive suggestions: The framework localizes the application bottlenecks and provides constructive performance suggestions to resolve the issues in the bottleneck area.The suggestions include vectorization, detetection of FMA, loop unrolling and slow data structure. Also appropriate compilation flags are suggested. This helps the application developer to potentially improve the performance of the application.The following are the figures of respective suggestions i.e. fig.3 ,fig.4 and fig.5.

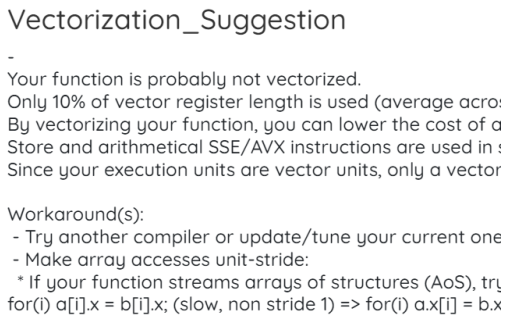

Fig. 3. Vectorization suggestion

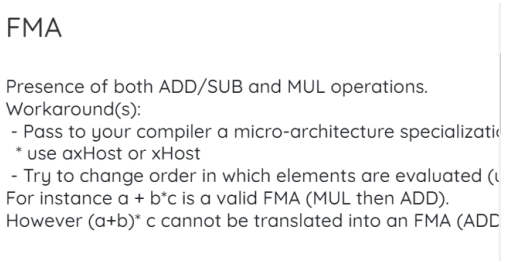

Fig. 4. FMA suggestion

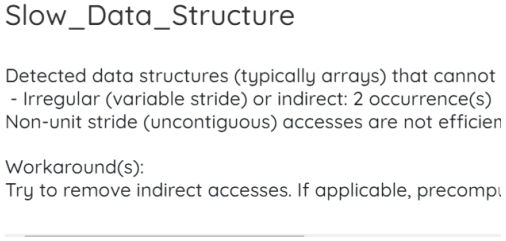

Fig. 5. Slow-Data structure

5) Detecting core-level bottlenecks: The framework is capable to drill down to the source of performance inefficiencies at the microarchitecture level. The pipeline port utilization, memory access pattern and branch prediction anomalies, percentage of successful retiring instructions are the few things

considered. This gives insights on how the application is behaving at the CPU core-level.

6) Pluggable and modular architecture: The framework follows a pluggable and modular architecture. This helps in extending the capabilities of the framework by incorporating various new profiling tools, based on the demands of the application. Further, this will also help to support profiling and analysis of applications on a newer compute architecture.

7) Integration with batch systems: The Local Resource Manager is the component responsible for getting the application run on a compute cluster. The framework integrates well with popular batch systems, like SLURM, to spawn the profiling jobs. This helps the user to seamlessly submit jobs on to the HPC clusters for profiling and analysis.

8) Dedicated web interface: The framework receives input parameters from web interface so overall it functions as web application from where users can submit jobs and navigate, visualize application performance details.Framework is designed such as architecture details,topology of process/thread id and data visualization aspects helps developer to analyze details in better way.

C. Usage and Experiment

The HPC profiler and analyser framework has been used to identify bottlenecks and fixes on a compute cluster with 8 nodes. The nodes have dual Intel(R) Xeon(R) Gold 6126 CPUs with 12 cores on each, along with 96GB of RAM and 1TB of storage. We are also planning to deploy and run the HPC profiler and analyser HPC cluster with higher number of nodes.

We have experimented with k-means clustering and n-body MPI applications. Initially we submitted the k-means clustering MPI application with input data of 10k number of sites per process, and 100 clusters in 100 dimensions. The compilation was carried out with "–O3" optimization option. We could get an average execution time of 27.8 seconds from 5 submissions with 8 processes.

Preliminary application categorization detected most of the time spent by user application is disseminated in application code and MPI library calls. This directed the user to explore potential issues in both user code and MPI region. The software highlighted the hotspots in user code at multiple granularity levels and code sections of these hotspots are further analysed statically. The analysis of these hotspots revealed very low vectorization ratio. The software further suggested the options to maximize the utilization of vector units with respect to the current architecture.

It also suggested the changes required in the data structures to make it more cache-memory friendly, like changing the array of structures to a structure of array. Initial TMAM analysis highlighted that most of the hotspot's areas of the application are inclined towards core bound operations.

After accommodating the above suggestions, we observed that the execution took 20.47 seconds with the same input data, giving a 26% reduction in average execution time compared to the runs without the suggested optimizations, and it was observed that the performance of application has improved.

TABLE I
APPLICATION EXPERIMENTS

Application	Exec time(unopt)	Exec time(opt)	Reduction(%)
K-means	27.8	20.47	26
N-Body	5.9	4.22	28
Matrix Multiplication	150	132	12
2D Heat Equation	35.55	19.06	46
2D Navier Stokes	25.84	20.80	19
NPB-IS	74.6	69.7	6

In the case of N-body MPI application, we ran the simulation with 10k particles and 1k iterations as inputs and for 24 processes. It took 5.9 seconds. We let the n-body application run through our software for performance bottlenecks identification and suggestions.

Preliminary application categorization detected most of the time spent is in application code. This guided the user to explore potential optimizations in user code only. The static analysis highlighted that the application is not efficiently making use of the capabilities available in the processor. The analysis indicated the traces of partial vectorization in the hotspot sections of the application. The analysis also detected the data structures that are not compliant for efficient read and write operations. The framework gave suggestions to improve the vectorization ratio and to avoid mixing different data types in the loops. The software also brought out the instruction level conversions that are costly. Initial TMAM analysis brought out that the application is inclined towards backend bound.

After incorporating the suggestions, we observed the execution time is reduced to 4.22 seconds, a reduction of 29% in execution time.

Similarly, we have used the framework to analyze other applications, like Matrix Multiplication, 2D Heat Equations, 2D Navier Stokes Equation and NPB-IS benchmark. The details of the execution with and without bottleneck improvement suggestions, along with reduction in execution time are given in the Table 1.

IV. USECASE DESCRIPTION

The framework we developed is trying to address the challenges faced by the application developers in improving the performance of the application. The challenges are addressed at multiple facets. First, we put together multiple existing free and open-source profiling tools, like Perf, LProf and CQA from MAQAO to work for identifying bottlenecks and generating performance improvement suggestions. The steps of execution, analysis and decision making followed in the framework are governed by the behaviour of the application at the runtime. The framework orients the direction of profiling based on the sections which make a larger impact with respect to the performance footprint. The execution of application is carried out in a way to ensure that the performance data collection and performance analysis is oriented towards the sections where there is a greater potential of performance

impact. The information gathered during the run is analysed through our own analysis sections to decide the further course of action in the application performance improvement.

The user submits the compiled application with "-g" to the framework. Our framework does not require any modification or insertion of hooks/probes in the source code. The only configuration required is related to the job submission for the cluster.

The key use cases of the HPC profiler framework are automated performance analysis, bottleneck and granularity-based performance analysis, and constructive suggestion generation.

A. Automated performance analysis

HPC applications are inherently complex in nature owing to the magnitude of the problems they address, and the programming methodologies used to exploit the HPC hardware resources. And the performance analysis of the HPC applications require specialized domain expertise and in-depth understanding of the hardware resources. This makes the performance analysis of the HPC application complex to an average application developer. Hence, most of the application developers heavily rely on tools on performance analysis. Tools also require a great level of user interference and actions to achieve the required performance improvements to the applications. This necessitates need of an automated performance analysis that can take away the hardships involved in HPC performance analysis.

The HPC Profiler framework embarks on the automated performance analysis to help the application developers in carrying out the performance analysis and to constructively improve the performance. The framework collects the required inputs from the users and frames the batch scripts to submit on to the HPC cluster and analyse the application execution. The analysed data is processed to identify the potential areas of issues in performance. The framework uses heuristics and analyse the application based on its behaviour at various granularity levels like application computation, node level, process level and thread levels. This automated way of analysis by the HPC profiler framework helps the user to simplify the procedures involved in improving the HPC applications performance.

B. Bottleneck and granularity-based performance analysis

The HPC profiler framework helps the application developer to focus on specific areas of interest in performance analysis. The developer can select various granularity of bottlenecks where the framework must focus for analysis to bring out the hotspots and the suggestions for overcoming the identified issues at different granularity levels. The area of analysis will be focused on the specific granularity levels spanning from application level, node level, process, and thread level. If an application developer wishes to profile the application to cater to computation aspects at process level, the HPC profiler framework can be made to run through the required phases of execution, analysis and decision making to bring out the bottlenecks and suggestions catering to the computational aspects of the application. This helps in finetuning the HPC application for a specific system.

C. Constructive suggestion generation

The most challenging part of application profiling and analysis is identifying the hotspots and bringing out the ways and means to overcome the issues prevailing in the hotspots. It requires a considerable effort and expertise to arrive at a constructive solution, and to bring out modifications in and around the application that can accomplish performance improvement. After the analysis, the framework shows the constructive suggestions, potential gain, and workarounds. Suggestions could be source restructuring, configuration changes, compiler optimizations, hardware changes, fine-tuning the environment, load balancing the execution, memory optimization, input-output optimization, optimizing the communication between the processes, use of better data-structure etc. In this direction, we have brought in various heuristics to provide verbose suggestions in respective bottlenecks that can bring in positive improvement in performance of the application.

V. Future Work

The current HPC Profiler Analysis Framework supports performance profiling and bottleneck detection of the OpenMP and MPI applications on an HPC cluster. We will be carrying out the deployment of the framework on a HPC cluster with higher number of nodes and cores. Additionally, we are also working on incorporating performance analysis of the HPC applications, which includes accelerator programs like CUDA, OpenACC and SYCL. Subsequently, we also plan to include bottleneck detections and refined performance improvement suggestions in multiple dimensions by using Large Language Models. Further, we will incorporate mechanisms to repair and fix the bottlenecks based on the user's selection choices for the performance.

VI. Conclusion

In this paper, we have explained the software framework developed to effectively identify and analyse performance bottlenecks and generation of possible suggestions through static and dynamic approaches. The software can highlight various critical bottlenecks of an application inside its hotspot areas under consideration, which are identified for the currently available computing resources. The framework modules carry out analysis at multiple granularity levels to localize the area of a bottleneck. The meaningful suggestions generated by the framework assist the developers to achieve performance improvements in their applications. We have implemented the profiling framework by making use of the existing free and open-source components. These framework will allow the users and application developers to make their applications perform better , by identifying the bottleneck areas and incorporating potential suggestions for performance improvement.

REFERENCES

[1] Williams, J.J., Tskhakaya, D., Costea, S., Peng, I.B., Garcia-Gasulla, M., Markidis, S. (2024). Leveraging HPC Profiling and Tracing Tools to Understand the Performance of Particle-in-Cell Monte Carlo Simulations. In: Zeinalipour, D., et al. Euro-Par 2023: Parallel Processing Workshops. Euro-Par 2023. Lecture Notes in Computer Science, vol 14351. Springer, Cham.

[2] Tan, S., Jiang, Q., Cao, Z. et al. Uncovering the performance bottleneck of modern HPC processor with static code analyzer: a case study on Kunpeng 920. CCF Trans. HPC 6, 343–364 (2024).

[3] Gravelle, B. (2019, March). Understanding the Performance of HPC Applications. https://ix.cs.uoregon.edu/ gravelle/Brian_Area_Paper.pdf.

[4] Rocha, Rodrigo CO, et al. "Vectorization-aware loop unrolling with seed forwarding." Proceedings of the 29th International Conference on Compiler Construction. 2020.

[5] Chen, Yishen, Charith Mendis, and Saman Amarasinghe. "All you need is superword-level parallelism: systematic control-flow vectorization with SLP." Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. 2022.

[6] O. H. Mondragon, P. G. Bridges, S. Levy, K. B. Ferreira and P. Widener, "Understanding Performance Interference in Next-Generation HPC Systems," SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2016, pp. 384-395, doi: 10.1109/SC.2016.32.

[7] Malony, A. D., Ramesh, S., Huck, K., Chaimov, N., & Shende, S. (2019, August). A plugin architecture for the tau performance system. In Proceedings of the 48th International Conference on Parallel Processing (pp. 1-11).

[8] Madsen, Jonathan R., et al. "TiMemory: modular performance analysis for HPC." International Conference on High Performance Computing. Springer, Cham, 2020.

[9] Deakin, T.,& McIntosh-Smith, S. (2020, April). Evaluating the performance of HPC-style SYCL applications. In Proceedings of the International Workshop on OpenCL (pp. 1-11).

[10] Knobloch, M., & Mohr, B. (2020). Tools for gpu computing–debugging and performance analysis of heterogenous hpc applications. Supercomputing Frontiers and Innovations, 7(1), 91-111.

[11] Tuncer, O., Ates, E., Zhang, Y., Turk, A., Brandt, J., Leung, V. J., ... & Coskun, A. K. (2017, June). Diagnosing performance variations in HPC applications using machine learning. In International Supercomputing Conference (pp. 355-373). Springer, Cham.

[12] Lehr, J. P. (2016). Counting performance: hardware performance counter and compiler instrumentation. Informatik 2016.

[13] Machado, R. S., Almeida, R. B., Jardim, A. D., Pernas, A. M., Yamin, A. C., & Cavalheiro, G. G. H. (2017, October). Comparing performance of C compilers optimizations on different multicore architectures. In 2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW) (pp. 25-30). IEEE.

[14] Palomares, V., Wong, D. C., Kuck, D. J., & Jalby, W. (2016). Evaluating out-of-order engine limitations using uop flow simulation. In Tools for High Performance Computing 2015 (pp. 161-181). Springer, Cham.

[15] Pedretti, K. (2019). Experiences Scaling a Production Arm Supercomputer (No. SAND2019-11171C). Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[16] Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J.,& Tallent, N. R. (2010). HPCToolkit: Tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience, 22(6), 685-701.

[17] Shende, S. S., & Malony, A. D. (2006). The TAU parallel performance system. The International Journal of High Performance Computing Applications, 20(2), 287-311.

[18] Marowka, A. (2011, September). On performance analysis of a multithreaded application parallelized by different programming models using intel vtune. In International Conference on Parallel Computing Technologies (pp. 317-331). Springer, Berlin, Heidelberg.

[19] Marques, D., Duarte, H., Ilic, A., Sousa, L., Belenov, R., Thierry, P., & Matveev, Z. A. (2017, July). Performance analysis with cache-aware roofline model in intel advisor. In 2017 International Conference on High Performance Computing & Simulation (HPCS) (pp. 898-907). IEEE.

[20] Zhukov, I., Feld, C., Geimer, M., Knobloch, M., Mohr, B., & Saviankou, P. (2015). Scalasca v2: Back to the future. In Tools for high performance computing 2014 (pp. 1-24). Springer, Cham.

[21] Meng, K., & Norris, B. (2017, September). Mira: A framework for static performance analysis. In 2017 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 103-113). IEEE.

[22] Mantovani, F., & Calore, E. (2018). Performance and power analysis of HPC workloads on heterogeneous multi-node clusters. Journal of Low Power Electronics and Applications, 8(2), 13.

[23] Molka, D., Schöne, R., Hackenberg, D., & Nagel, W. E. (2017, April). Detecting memory-boundedness with hardware performance counters. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (pp. 27-38).

[24] Luecke, G. R., Groth, B. M., Weeks, N. T., & Kraeva, M. (2017, April). Comparing Allinea's and Intel's performance tools for HPC. In Proceedings of the 25th High Performance Computing Symposium (pp. 1-12).

[25] Wolf, F., Bischof, C., Calotoiu, A., Hoefler, T., Iwainsky, C., Kwasniewski, G., ... & Wittum, G. (2016). Automatic performance modeling of hpc applications. In Software for Exascale Computing-SPPEXA 2013-2015 (pp. 445-465). Springer, Cham.

[26] Yasin, A. (2015). Software Optimizations Become Simple with Top-Down Analysis Methodology on Intel Microarchitecture, Code Name Skylake. In Intel Developer Forum.

[27] Yoo, W., Koo, M., Cao, Y., Sim, A., Nugent, P., & Wu, K. (2015, December). Patha: Performance analysis tool for hpc applications. In 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC) (pp. 1-8). IEEE.

[28] Agelastos, A., Allan, B., Brandt, J., Gentile, A., Lefantzi, S., Monk, S., ... & Stevenson, J. (2015, September). Toward rapid understanding of production HPC applications and systems. In 2015 IEEE International Conference on Cluster Computing (pp. 464-473). IEEE.

[29] Collins, W., Martinez, D. T., Monaghan, M., Munishkin, A. A., Blenkhorn, A. R., Graf, J. S., ... & Linford, J. C. (2015). Comparison of Performance Analysis Tools for Parallel Programs Applied to CombBLAS. UMBC Student Collection.

[30] Shamjith, K.V., Mangala, N., Deepika, H.V, & Pandey, P. (2019). Dynamic Adaptation of Application Execution on Heterogeneous Architectures. 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 1-8.

[31] Lebras, Y. (2019). Code optimization based on source to source transformations using profile guided metrics (Doctoral dissertation, Université Paris-Saclay (ComUE)).

[32] Doweck, J., Kao, W. F., Lu, A. K. Y., Mandelblat, J., Rahatekar, A., Rappoport, L., ... & Yoaz, A. (2017). Inside 6th-generation intel core: New microarchitecture code-named skylake. IEEE Micro, 37(2), 52-62.

[33] Leiserson, C. E., Thompson, N. C., Emer, J. S., Kuszmaul, B. C., Lampson, B. W., Sanchez, D., & Schardl, T. B. (2020). There's plenty of room at the Top: What will drive computer performance after Moore's law?. Science, 368(6495).

[34] Randall Hyde. 2009. The Fallacy of Premature Optimization. Ubiquity 2009, February, Article 1 (February 2009), 5 pages. DOI:https://doi.org/10.1145/1569886.1513451

[35] X. Tian et al., "Practical SIMD Vectorization Techniques for Intel® Xeon Phi Coprocessors," 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, 2013, pp. 1149-1158, doi: 10.1109/IPDPSW.2013.245.

[36] Cebrián, J.M., Natvig, L. & Meyer, J.C. Performance and energy impact of parallelization and vectorization techniques in modern microprocessors. Computing 96, 1179–1193 (2014)

[37] Dursun, H., Kunaseth, M., Nomura, Ki. et al. Hierarchical parallelization and optimization of high-order stencil computations on multicore clusters. J Supercomput 62, 946–966 (2012).

[38] Nikitenko, D. A., Shvets, P. A., & Voevodin, V. V. (2020). Why do Users Need to Take Care of Their HPC Applications Efficiency?. Lobachevskii Journal of Mathematics, 41(8), 1521-1532..

[39] Klôh, V., Yokoyama, D., Yokoyama, A., Silva, G., Ferro, M., & Schulze, B. (2018, October). Performance and energy efficiency evaluation for HPC applications in heterogeneous architectures. In 2018 Symposium on High Performance Computing Systems (WSCAD) (pp. 162-169). IEEE

[40] Abraham, E., Bekas, C., Brandic, I., Genaim, S., Johnsen, E. B., Kondov, I., ... Streit, A. (2015, September). Preparing HPC applications for