Scalable Automatic Performance Analysis on IBM BlueGene/P Systems*

Yury Oleynik and Michael Gerndt

Technische Universität München
Fakultät für Informatik I10, Boltzmannstr. 3, 85748 Garching, Germany

Abstract. Nowadays scientific endeavor becomes more and more hungry for computational power of the state-of-the-art supercomputers. However the current trend in the performance increase comes along with tremendous increase in power consumption. One of the approaches allowing to overcome the issue is tight coupling of the simplified low-frequency cores into massively parallel system, such as IBM BlueGene/P (BG/P) combining hundreds of thousands cores. In addition to revolutionary system design this scale requires new approaches in application development and performance tuning. In this paper we present a new scalable BG/P tailored design for an automatic performance analysis tool - Periscope. In this work we have elicited and implemented a new design for porting Periscope to BG/P which features optimal system utilization, minimal monitoring intrusion and high scalability.

Keywords: Performance analysis, Scalability of Applications & Tools, Supercomputers.

1 Introduction

Traditional supercomputer design which relies on the high single core performance delivered by high frequency has a natural scalability limit coming from unaffordable power consumption and cooling requirements. The BlueGene [1] developers addressed this challenge from two aspects: by utilizing moderate-frequency cores and by tightly coupling them at unprecedented scales, which allows power consumption to grow linearly with the number of cores. This leads to a high density, low-power, massively parallel system design.

Unfortunately the peak performance offered by modern supercomputers can not be achieved by straight forward application porting, one would have to invest significant efforts in achieving reasonable execution efficiency. In order to make this efforts affordable, new instruments supporting application development have to be developed. This is specially true for performance analysis tools. On one hand, the performance analysis results of small runs often can not be extrapolated to the desired number of cores due to new performance phenomena

* This work is partially funded by BMBF under the ISAR project, grant 01IH08005A and the SILC project, grant 6.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part II, LNCS 7156, pp. 146–155, 2012.

manifesting itself only at large scales. On the other hand the amount of raw performance data which has to be recorded for large real-world applications running on hundreds of thousands cores is simply too big for the commodity evaluation approaches. The way performance analysis is done has to be rethought as well as other aspects of extremely parallel computing. Among the challenges to be overcome are efficient recording, storing, analysis and visualization of the discovered results.

Periscope [4], being an automatic distributed on-line performance analysis tool, addresses the challenges of large scale performance analysis from multiple aspects. The distributed architecture of Periscope allows it to scale together with the application relying on multiple agents. On the other hand, on-line analysis of the profile based raw performance data significantly reduces memory requirements. However even then the amount of performance data collected for a large scale run is big enough to overwhelm the user with too much information. Periscope addresses the issue in two ways, first, automatic search for performance inefficiencies dramatically decreases the amount of presented results by reporting only important potential tuning opportunities. Second, performing scalable reduction, based on clustering algorithms, allows to keep the amount of reported results constant independently from the growing number of cores.

Historically the development of Periscope was carried out based on the architecture of commodity clusters, where the maximum scalability levels were considered to be in order of tens of thousands of cores running standard Unix-like kernels. Therefore porting Periscope to a new cluster was a matter of minor adjustments, whereas the overall architecture was being preserved. However with the introduction of BlueGene/P systems it was realized that the straightforward porting approaches would not work. The two main reasons for that were an order of magnitude increase in number of cores and limited operating system functionality. In order to adapt Periscope to the challenges posted by BlueGene/P, significant improvements to the tool's architecture were developed.

The rest of the paper is composed as follows: first we describe the architecture specifics of BG/P as well as the analysis model and architecture of Periscope. From the cross-analysis we derive three promising approaches, from which one was implemented and is discussed in more details. Alternative tools are discussed in related work. In the evaluation section we apply Periscope to the NAS Parallel Benchmark running with 64k processors to demonstrate achieved scalability levels.

2 BlueGene/P Architecture

The BlueGene/P [1] base component is a PowerPC 450 quad core 32 bit microprocessor with a frequency of 850 MHz. One quad-core together with 2 or 4 GB of shared memory forms a next building block of BlueGene - a compute node ASIC. The compute nodes run under the IBM proprietary light-weight Compute Node Kernel (CNK) and are dedicated to run exclusively MPI/hybrid applications. CNK is, on one hand, striped down in order to minimize the system overheads when executing an application and, on the other hand, appears

to the application programmers as a Linux-like operating system, supporting the majority of the system calls. However not all the system calls are executed by CNK, instead this functionality is forwarded to the dedicated I/O nodes. Due to the low OS jitter and a single application process per core, applications show excellent performance reproducibility.

I/O nodes are the service nodes and are not intended to run user applications. The hardware is identical to the one of compute nodes with the only difference of physical placement within the system, which leads to different network connections. I/O nodes use 10GB Ethernet to connect to the BlueGene/P frontend. I/O nodes operate under the "standard"Linux kernel in a four-way SMP mode providing file system access and socket communication. Each I/O node runs one Control and I/O daemon (CIOD) which executes the function-shipped system call requests coming from the back-end compute nodes. Applications are not allowed to run on the I/O node, however upon a request one tool process is allowed to be spawned by the CIOD [2]. This is a typical mechanism employed by debugging tools to control application execution [3]. The tool process normally gets started through additional `-start_tool` argument of the `mpirun`. It is also important to mention that the I/O node and all the associated compute nodes share the same network address.

32 Compute nodes together with 0 or up to 2 I/O nodes form the next building block called compute card. 32 compute cards in their turn form one midplane. One BlueGene/P rack consists of 2 midplanes or 1024 compute nodes. 72 BlueGene/P racks are needed to achieve 1 PFlop of peak performance.

The distinguishing feature of BlueGene/P are multiple high-speed networks coupling numerous compute nodes. The most appreciated one is the torus network that wraps the compute nodes of one midplane into a 3D torus. It allows efficient low-latency nearest neighbor communication between ranks of `MPI_COMM_WORLD`. MPI collective operations are carried out through the dedicated tree-like collective network. In total each compute node has six connections to the 3 dimensional torus network with a bandwidth of 3.4Gbps in each direction, three connections to the global collective network with 6.8 Gbps per link, four connections to the global interrupt network and one connection to the control network JTAG.

As an alternative to the default High Performance Computing (HPC) mode described above, High Throughput Computing (HTC) mode allows to run large number of non-MPI applications simultaneously. Each application could be started independently under different user names, with separate stdin, stdout and stderr. In order to use HTC mode a partition has to be first booted by the system administrator.

3 Periscope Design

Periscope [4] is an automatic distributed on-line performance analysis system developed by Technische Universität München (TUM) at the Chair of Computer

Architecture. In comparison to other tools Periscope relies on profiling rather than on tracing and searches automatically for predefined set of performance inefficiencies during the application's execution. The scalability requirements, being the main design concern, resulted in a distributed tree-like reduction network of agents scaling together with the application. The coordination of distributed components is carried out by Periscope's registry service running in the background.

3.1 Tool Architecture

Periscope consists of multiple agents where the root agent is called the Frontend (FE). The FE is the only process which has to be started by the user. It is responsible for starting the instrumented application processes, computing the optimal agents hierarchy and mapping it to the underlying hardware as well as starting the hierarchy. The FE will coordinate the distributed search by taking global decision on controlling application's execution and when necessary restart application. The FE will also receive the aggregated set of found performance properties and store them into an xml file.

The second layer of the tree-like Periscope network consist of the High-Level (HL) agents. They are responsible for the efficient propagation of analysis decisions from the FE downwards and the scalable aggregation of the results sent in the opposite direction. The results, namely found performance inefficiencies, are clustered on-line within the HL on their way to the FE.

The bottom level of the Periscope's network is represented by Analysis Agents (AA), which are the leaf nodes and are directly connected to the application processes. AAs are responsible for executing the local search for predefined performance inefficiencies on the assigned subset of application processes. An AA instantiates performance hypotheses, creates and submits monitoring requests required to evaluate them. The requests are submitted over sockets to the monitoring library linked in the application processes. After one execution of application phase, which is typically one iteration of the main loop (typically a time loop in scientific simulations), the measured values for the submitted requests are analyzed in order to prove or disprove the candidate hypotheses. The refinement mechanism is then employed to drill the found performance inefficiency down to the specific line of the code as well as down to the specific problem source. An AA relies on a set of search strategies when searching for bottlenecks, which allow efficient evaluation of numerous application code regions against the rich set of available performance hypothesis. All known and measurable performance bottlenecks are formalized following the APART Specification Language [7] and are implemented as Periscope properties.

The application is instrumented and linked against Periscope's monitoring library which measures time, hardware counters and is capable of automatic detection of MPI wait states. In addition, the monitoring library can control the application execution according to the requests received from the AA.

In order to bring multiple distributed components of Periscope together, Periscope's registry service is used. Periscope agents as well as application

processes publish their network address - identity-tag pairs and also look up for the addresses of their communication partners at the registry service.

3.2 Current Design Shortcomings

Although the scalability of the tool was an important concern, the overall design was targeting commodity clusters [6] with tens of thousands of cores. Even though, Periscope was showing good scalability at this scale [5], several weak points, which could become a severe bottleneck at higher levels, were revealed.

One of the most severe potential bottlenecks was considered to be the registry service used by all the distributed components (agents and instrumented application processes) to register and find their corresponding communication partners. Since every instrumented application process had to be registered and then is queried by the responsible analysis agent, this can severely hit the analysis time, which will quadratically grow together with the number of processes.

Another drawback of the current network startup implementation was that the hierarchy of agents, the application process distribution, and the sequential spawning of the agents was performed by the frontend agent. It will become a severe startup bottleneck, when executed on a system like BlueGene/P.

Another issue for porting Periscope to the BG/P comes from the limited OS support of the compute nodes, which is restricting compute nodes from running anything else but an MPI application (except for the case of HTC mode or MPMD programs, which will be discussed later), therefore excluding Periscope agents from running on the same partition with application processes.

4 BlueGene/P Tailored Design Alternatives

Taking into account the requirements of Periscope and the architectural limitations of BlueGene/P several porting approaches were elicited. The main question to be answered here is Periscope agents placement, allowing the tool to overcome the scalability bottlenecks considered before as well as achieve best utilization of BlueGene/P avoiding additional overhead perturbations.

4.1 MPMD Approach

With the introduction of MPMD parallelization in the P generation of the BlueGene series, the possibility to run multiple MPI programs within the same communication domain became available. This allows to overcome the limitation of only application processes being allowed to run on the compute nodes of the same partition.

In this configuration the user starts the FE via `mpirun` on the allocated partition. The number of processes, which would have to account for both application processes and periscope agents are specified using `-universe_size` argument for `mpirun`. After being started, the FE opens a port using `MPI_Open_port` and publishes the pair of port and service description. This mechanism, also employed by

other agents and application processes, would allow us to drop the commodity registry service. However this would not remove the bottleneck associated with the thousands of application processes ports being published and then queried at the same time.

After successful registration, the FE computes the agents hierarchy, however this is much simplified since all agents are started at once with a single `MPI_Spawn` command. The hierarchy, in this case, is determined according to the fan-out of the HL agents and the number of leaf agents, which is proportional to the number of application processes.

After setting up the agent hierarchy, the application would be started by the FE and the agents connect to the application processes and execute the analysis as before. This design would also support the restart of the application which is done by Periscope if the application terminated but additional search steps are required.

However, several severe drawbacks of this design were considered. First, the collective network of the BlueGene can not be properly utilized when the application is running in the sub communicator of the `MPI_COMM_WORLD`, which is the case when the application is started by Periscope in MPMD mode. Second, as it was mentioned before, the bottleneck of publishing and querying of every application process and the agents still significantly impacts the efficiency of Periscope's analysis at large scale. Finally, the additionally required, complete reimplementation of the communication substrate of Periscope with MPI as well as the need to port the AA to run under CNK would require significant amount of programming efforts.

4.2 Implementation Based on the HTC Mode

Another design approach, that would allow to place the AAs on the compute nodes, relies on the High-Throughput Computing mode of BlueGene. In this mode multiple non-MPI programs are allowed to be independently started and run simultaneously within the provided partition under the full Linux kernel. Based on this mode, the current approach of starting agents need not be changed.

The overall startup and analysis flow is done as follows. The user starts the FE on the frontend node of BG/P. Then either the user or the FE submits the standard job to run the instrumented application in the HPC mode. While the application is waiting for the dispatch, the FE first computes the tree hierarchy with the explicit assignment of the children-parent relations and then starts the agents within the previously booted in HTC mode partition. The rest of the analysis follows the current procedure including the registration of all the distributed components, which would again be a bottleneck. Additional complications come from the fact, that on the majority of BG/P installations privileged access rights are required to boot the HTC partition, which seriously limits the tool's usability. On the other hand, this design can be realized with little porting effort.

4.3 I/O Nodes Agents Placement

Following this approach, the tool distribution would utilize compute nodes exclusively for application processes linked with the monitoring library, I/O nodes for running AAs and the frontend node for Periscope's FEs. There is one important advantage emerging from such a setup, the agents would neither waste computational resources by occupying additional cores, nor disturb the application processes during execution like in the case of the MPMD or HTC approaches.

The placement of HLs is, however, a more complicated issue, since it is not allowed to run more than one process on the I/O nodes. The two solutions here are either to run HLs on the frontend node or to merge the reduction functionality of the HL into the AAs and reuse the AA to form the tree hierarchy.

It is also worth mentioning that the number of compute nodes affiliated with one I/O node is installation-defined and fixed and, what is even more important, they share the same network address. This simplifies agent placement and removes the need for the registry service, since addresses are by definition known. Thus one of the most severe scalability issues of the current design is eliminated when the AAs run on the I/O nodes.

Usually I/O nodes are not accessible to an unprivileged user. The only process allowed to run by a regular user is a debug process of CIOD and is started normally by the `-start_tool` argument of the `mpirun` command. `mpirun` will spawn the tool process on all of the I/O nodes affiliated with the allocated partition. The startup will no longer be done sequentially by the FE, which removes another scaling issue. However there is an important drawback, since the AAs will be automatically killed when `mpirun` exits. Thus Periscope's application restart capability becomes not possible.

4.4 Comparison of Design Alternatives

Considering the current architecture of Periscope, its scalability issues and BlueGene/P system specifics, a set of selection criteria were recognized, i.e., amount of changes, system utilization during analysis, application intrusion and tool usability. The amount of changes, associated with porting, is an important factor influencing both the porting effort as well as the further maintenance of an additional branch of Periscope. In addition, system utilization becomes very important when large scale experiments are considered. Even one additional process per few application processes might severely hit the computational budget of the user. The easy metric allowing to estimate the system utilization would be the amount of additional cores needed to run Periscope's agents. Overheads and application execution intrusion is another critical factor which can potentially corrupt the measurements, therefore the design should not introduce any additional overheads. The last factor is the tool's usability. It is severely limited if privileged rights are required to run Periscope. The possibility of application restart within one Periscope run is also a valuable functionality which is important to preserve.

The summary comparison of the derived design alternatives against the selection criteria is presented in Table 1.

Table 1. Design alternatives comparison

Selection criteria	MPMD	HTC mode	I/O node placement
Amount of changes	high	low	medium
Administrative issues	no	yes	no
Application restart	possible	possible	not possible
Additional user input	no	yes	no
Additional cores	#agents	#agents	0
Additional overheads	yes	no	no

The comparison table shows that the design relying on the MPMD functionality of MPI 2.1 standard is the less preferable failing to meet the majority of the selection criteria.

The design approach utilizing the HTC partition to run Periscope agents features low efforts to be invested in porting and maintenance, however suffers from the fact that booting a HTC partition is a privileged operation. In addition, the system utilization is worse since additional cores are required to run agents.

The best match with the selection criteria is the I/O node agent placement design and therefore was chosen for implementation. This approach features best system utilization, since it doesn't require any additional compute nodes to run Periscope's agents. Instead it runs them on the I/O nodes which are not intended for computation by design. However the efforts to port Periscope following the described design are considered to be moderate. In oder to prove the selected concept fast and minimize associated porting risks, it was decided to split the porting efforts in two phases. Within the first phase the idea of running the AAs on the I/O nodes and the application processes on the affiliated compute nodes was evaluated and considered to be a low-effort task. The other agents are intended to run on the frontend node of BlueGene/P in this phase. The majority of the efforts, though, come from the task to merge the functionality of the AA and the HL in order to run them within the single user process allowed to run on the I/O nodes. Therefore this task was assigned to be implemented in phase two, which will deliver optimal tool distribution and capability to operate at the full-scale 72-rack BlueGene/P.

5 Evaluation

The phase one porting task was implemented and Periscope was installed on the IBM BlueGene/P supercomputer operated by King Abdullah University of Science and Technology (KAUST). The machine consists of 16 racks containing in total 65536 IBM Power450 cores delivering 222 TFlops of peak performance.

In order to prove the scalability of the new Periscope design a large scale performance analysis run was carried out on a standard BT benchmark from NAS Parallel Benchmark suite [9]. The benchmark is Block Tridiagonal solver of a synthetic system of nonlinear PDE's. The benchmark was built to solve the E problem size which corresponds to 1020x1020x1020 grid size. The MPI call sites

were instrumented by Periscope's instrumenter and then the application compiled using the IBM mpxlf Fortran compiler with -O3 optimization flag and the BlueGene/P MPI library. The analyzed application was running on all the available 65536 cores and was analyzed with Periscope's MPI strategy automatically detecting MPI wait states.

The elapsed analysis time reported by the FE was 432 seconds, from which the 382 seconds took the tool startup. However it is important to mention that the majority of the startup time is spent on booting the partition, in this case the whole machine. The automatically computed agents hierarchy included 20 HL agents running in the service node of the BG/P and 128 Analysis Agents running on the I/O nodes. Only one iteration of the BT main loop was executed to complete the analysis for MPI wait states. For each process the monitoring library measured 7 relevant MPI inefficiency related metrics resulting in 458752 measurements in total. Out of the received measurements, the Analysis Agents instantiated 393216 candidate properties out of which only 196608 properties were evaluated true. The found properties then were clustered while being propagated to the FE and the final analysis report contained only 3 properties. The found properties were "Excessive MPI communication time"reported for one MPI_Waitall and two MPI_Wait call sites with a maximum severity of 5.6%. This property identifies MPI communication overhead which is 5.6% of one main loop iteration time. Synchronization properties were also checked but appeared to be below threshold and thus not reported.

6 Related Works

There are only a few performance analysis tools available on BlueGene/P, and even less of them are specially designed for large scales. SCALASCA [8], being one of them, is an open-source performance analysis toolset specifically designed for an evaluation of codes running on hundreds of thousands of processors. The tool performs parallel trace analysis searching for MPI bottlenecks, which allows it to scalably handle the trace size linearly increasing with the number of cores. However it was found that the time spent for the analysis as well as the report size were growing linearly with the employed parallelism scale. In contrast, Periscope performs on-line profile based search, thus omitting tracing. Also online reduction allows it to keep the report size independent from the number of processes.

7 Conclusion and Outlook

The new scale of supercomputing is being currently rapidly developed with more and more petaflop capable machines being installed world-wide. However achieving reasonable levels of sustained performance becomes a tremendously complicated task for the application developers. New instruments are needed to support programmers in particular in the area of performance analysis. However tool development faces challenges induced by the growing number of cores such

as proportional growth in analysis times, measurements and final report sizes. In this paper we have presented a tailored adaptation of the Periscope toolkit for the IBM BlueGene/P supercomputer. According to the Periscope requirements, revealed scalability issues, and the limitations of the BG/P architecture several porting approaches were elicited. The design, following the idea of placing Analysis Agents on the I/O nodes, was considered to be superior to the other alternatives due to optimal system utilization, no additional overheads, high usability and therefore was implemented.

The large scale performance analysis experiment with the NPB BT benchmark has shown very promising scalability of the Periscope analysis. Due to optimal placement of multiple Periscope agents and optimized analysis flow the analysis time was kept constant and independent of the number of cores. The size of the report was shown also to be independent from the employed parallelism scale due to on-line results reduction.

References

1. IBM BlueGene team: Overview of the IBM BlueGene/P project. IBM Jopurnal of Research and Development 52(1/2), 199–220 (2008)
2. Sosa, C., Knudson, B.: IBM System Blue Gene Solution: Blue Gene/P Application Development. International Technical Support Organization, 4th edn. (August 2009)
3. DelSignore, J.: TotalView on Blue Gene/L, "Presented at" Blue Gene/L: Applications, Architecture and Software Workshop, `http://www.llnl.gov/asci/platforms/bluegene/papers/26delsignore.pdf`
4. Gerndt, M., Fürlinger, K., Kereku, E.: Advanced techniques for performance analysis. NIC, vol. 33, pp. 15–26 (2006)
5. Benedict, S., Brehm, M., Gerndt, M., Guillen, C., Hesse, W., Petkov, V.: Automatic Performance Analysis of Large Scale Simulations. In: Lin, H.-X., Alexander, M., Forsell, M., Knüpfer, A., Prodan, R., Sousa, L., Streit, A. (eds.) Euro-Par 2009. LNCS, vol. 6043, pp. 199–207. Springer, Heidelberg (2010)
6. Gerndt, M., Strohhäcker, S.: Distribution of Periscope analysis agents on ALTIX 4700. In: Proceedings of the International Conference on the Parallel Computing (ParCo 2007). Advances in Parallel Computing, vol. 15, pp. 113–120. IOS Press (2007)
7. Fahringer, T., Gerndt, M., Riley, G., Träff, J.: Knowledge specification for automatic performance analysis. APART Technical Report (2001), `http://www.fz-juelich.de/apart`
8. Wylie, B.J.N., Bohme, D., Mohr, B., Szebenyi, Z., Wolf, F.: Performance analysis of Sweep3D on Blue Gene/P with the Scalasca toolset. In: IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW 2010). IEEE (2010) - 978-1-4244-6533-0. - S. 1 - 8
9. NPB. NAS Parallel Benchmark, `http://www.nas.nasa.gov/resources/software/newlinenpb.html`