Enhancing Brainware Productivity through a Performance Tuning Workflow

Christian Iwainsky[1], Ralph Altenfeld[2,⋆], Dieter an Mey[1], and Christian Bischof[1]

[1] Center for Computing and Communication, RWTH Aachen University
{iwainsky,anmey,bischof}@rz.rwth-aachen.de
[2] Access e.V. RWTH Aachen University
r.altenfeld@access-technology.de

Abstract. Operation costs of high performance computers, like cooling and energy, drive HPC centers towards improving the efficient usage of their resources. Performance tuning through experts here is an indispensable ingredient to ensure efficient HPC operation. This "brainware" component, in addition to software and hardware, is in fact crucial to ensure continued performance of codes in light of diversifying and changing hardware platforms. However, as tuning experts are a scarce and costly resource themselves, processes should be developed that ensure the quality of the performance tuning process. This is not to dampen human ingenuity, but to ensure that tuning effort time is limited to achieve a realistic substantial gain, and that code changes are accepted by users and made part of their code distribution. In this paper, we therefore formalize a service-based *Performance Tuning Workflow* to standardize the tuning process and to improve usage of tuning-expert time.

1 Introduction

Researchers from many scientific backgrounds (e.g. biology, engineering, medicine) develop software to satisfy their simulation needs. We observe that software development projects pressed ahead by domain specialists typically lack the necessary knowledge, experience and manpower to effectively use modern manycore-clusters, resulting in insufficient or lacking parallelization, little or no tuning and missing adaption to the local HPC-system and as a consequence results in limited scalability, less than optimal performance and bad hardware utilization.

Besides the ability to solve previously unsolvable problems, the operational cost in terms of power and cooling becomes an additional driving force for performance tuning and parallelization (from now on summarized as "tuning"). It is therefore in the interest of users, i.e. the users of a projects software, and HPC-centers to improve scalability and efficiency.

⋆ Formerly employed at the Center for Computing and Communication.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part II, LNCS 7156, pp. 198–207, 2012.

For this reason many HPC-sites employ tuning specialists to provide the necessary expertise. This can be most prominently seen in the Tier-0 and Tier-1 HPC-sites, like the UK National Supercomputing Service (HECToR) with the Distributed Computations Science and Engineering support[1] and the HPC-Simulation Labs in Germany[2]. These projects provide this essential performance tuning expertise to enable "users", i.e. code developers and program users, to scale to ten-thousands or more cores. Also the smaller Tier-2 sites (like ours) who focus more on productivity and throughput of compute-jobs especially benefit from such support.

Whilst funding for such experts is typically difficult for Tier-2 sites, we argued in previous work (Brainware for Green HPC [1]), summarized in section 2, that savings in the *Total Cost of Ownership* (TCO) obtained by the tuning activity could be used to fund these experts. This claim holds true, as long as these experts achieve sufficient improvement for a project in a definite time.

Under this assumption, we propose a *Tuning Workflow*, described in detail in section 3, to guide and monitor the tuning service effort of a performance tuning expert. This workflow aims to maintain the necessary balance between tuning investment and the obtained savings. To our knowledge such a process has not been published, though tuning-experts may already follow this process.

We discuss our work and, as this workflow is our first implementation of this workflow, further improvement possibilities in section 4.

2 Brainware for Green HPC

To acquire an HPC-system on a Tier-2 level considerable funds are required. At the RWTH-Aachen University the TCO of about 5.5 million € per year covers of costs for the building, compute hardware, maintenance, power, software and staff. Figure 1 shows the percentage of a typical annual load of our system, broken down to the number of users causing the load. Using this figure and the TCO we can derive that the top 10 users consume roughly 40 % of our system amounting to 2.2 million €.

From our past experience and consistent with findings of the Computational Science and Engineering support of the HECToR UK National Supercomputing Service [3], we claim that on average a tuning expert can achieve at least a performance improvement of 10 % to 20 %. In this case this means that the project, i.e. code, dataset and software environment, executed faster on the same number of cores or used the available nodes more efficiently by scaling to further cores. For several projects in the past, we even experienced performance improvements of a factor of 2 to 5, with similar findings reported in the HECToR reports. The amount of work necessary for these improvements depends on the specific project. If one for example augments an MPI-only project with OpenMP (multi-threading), very little time is needed to identify the hotspot, parallelize it with OpenMP and to

[1] http://www.hector.ac.uk/cse/distributedcse/

[2] http://www.jara.org/de/research/jara-hpc/forschung/

[3] http://www.hector.ac.uk/cse/distributedcse/

verify the results. However, if one performs a change in algorithms or other extensive code modifications, more time is necessary for these changes but the potential outcome is larger. In our experience a tuning expert invests on average about 2 months worth of work to achieve the aforementioned conservative 10 % to 20 % performance improvements.

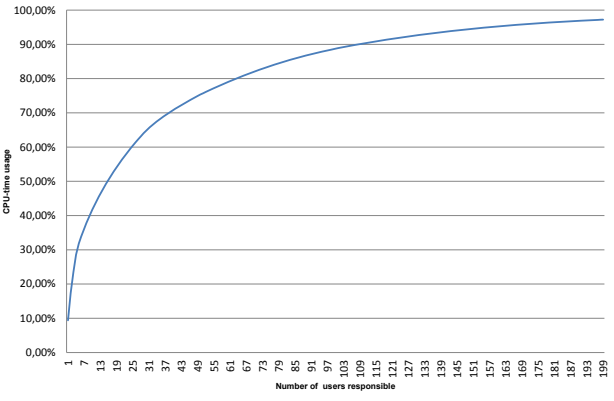

Fig. 1. Accumulated Cluster Usage

Assuming a constant load, these funds are initially freed up by shutting down the respective surplus hardware starting with older, less power efficient systems thus saving energy. In the long run a portion money can be diverted from procurements towards tuning as the new machinery could be considered 10 % more productive. For a non-constant load, i.e. the users allways use all available ressources, the systems behaves as if one had obtained 10 % more hardware, which would cost more in terms of running costs and procurement. So in either way the tuning would conserve funds in the long run.

With this in mind and looking at the top users who consume 10 % of our system (approx. 550k€), it is easy to see, that by just improving such a users project by 10 % a full time employee at 60k€ per year could be funded. Looking at the total usage, a mere average improvement of 5% on all projects would be sufficient to continuously support 3 HPC experts, if those experts tune one of the top projects every 2 months. For further detail and information please refer to "Brainware for Green HPC" [1].

3 A Performance Tuning Workflow

Performance experts themselves need to be concerned about the productivity of their efforts, i.e. they should try to obtain maximum gains in minimal time. Based on our experience in tuning several user-codes, we determined two issues the most common, which we will focus on in this work: First without outside stimulus a tuning process can continue for an indefinite time, as always some more tuning or improvement is possible. The second issue is the lack of user-acceptance of code changes preventing the tuning to take effect.

To address these issues, we developed a systematic approach by defining a *Tuning Workflow* (see Fig. 2)[4]. In our experience, involving the user in the

[4] In this workflow, we only incorporated those performance analysis tools in use at the RWTH Aachen University.

tuning process the acceptance of tuning-recommendations and code modifications at the end of tuning effort increases significantly. It should also be noted at this point that we consider a program or code with a specific data set and runtime environment as a single project. Therefore multiple datasets or different execution sites call for multiple, though potentially abbreviated tuning projects.

From our experience with past and current projects (e.g. XNS [2], CDPLit [3] and MICRESS [4]) we identified five general stages for a tuning / parallelization project:

1. Initial Meeting
2. Baseline
3. Pathology
4. Tuning Cycle
5. Finalization / Handover.

These stages are explained in the subsections to follow.

3.1 Initial Meeting

The tuning process should be initiated by the user, however it is also possible for the expert to approach top users. We recommend an initial face-to-face meeting with all parties involved. From our experience this improves the personal involvement of everyone and the acceptance of recommendations at the end of the tuning-project. This meeting should establish *clear expectations* by users and *clear definitions of goals* by the tuning expert. This practice helps to prevent overstated improvement expectations and unnecessary prolonged expert-involvement. The expert and user should also agree on the time frame and schedule a date for a later review, already preventing the expert-time allotted for the project to get out of hand.

Also in this meeting the scope of involvement, e.g. actual development work, consultation, minor code changes, should be discussed as copyright, code- and data-security are often of significance. Full access to the source code is desirable for the best tuning outcome, though tuning may still be possible for binaries by optimizing scheduling and environment parameters, like thread-counts and MPI-parameters. If the expert is granted access to the source code, he should be given a hands-on introduction to the build system and program execution setup, to speed up the following tuning and experimentation phase. It may even be possible, within reason, to request the user to modify the build-system to resemble a site-specific standard, like using standard compilers, GNUmake and a single configuration file. We also recommend a review of the algorithms used in the program, to check later in the *Pathology* phase for known issues of given algorithms.

3.2 Base-Line

The *Base-line* is intended to be quick and easy to obtain information. Great emphasis should be put on disturbing the target application as little as possible

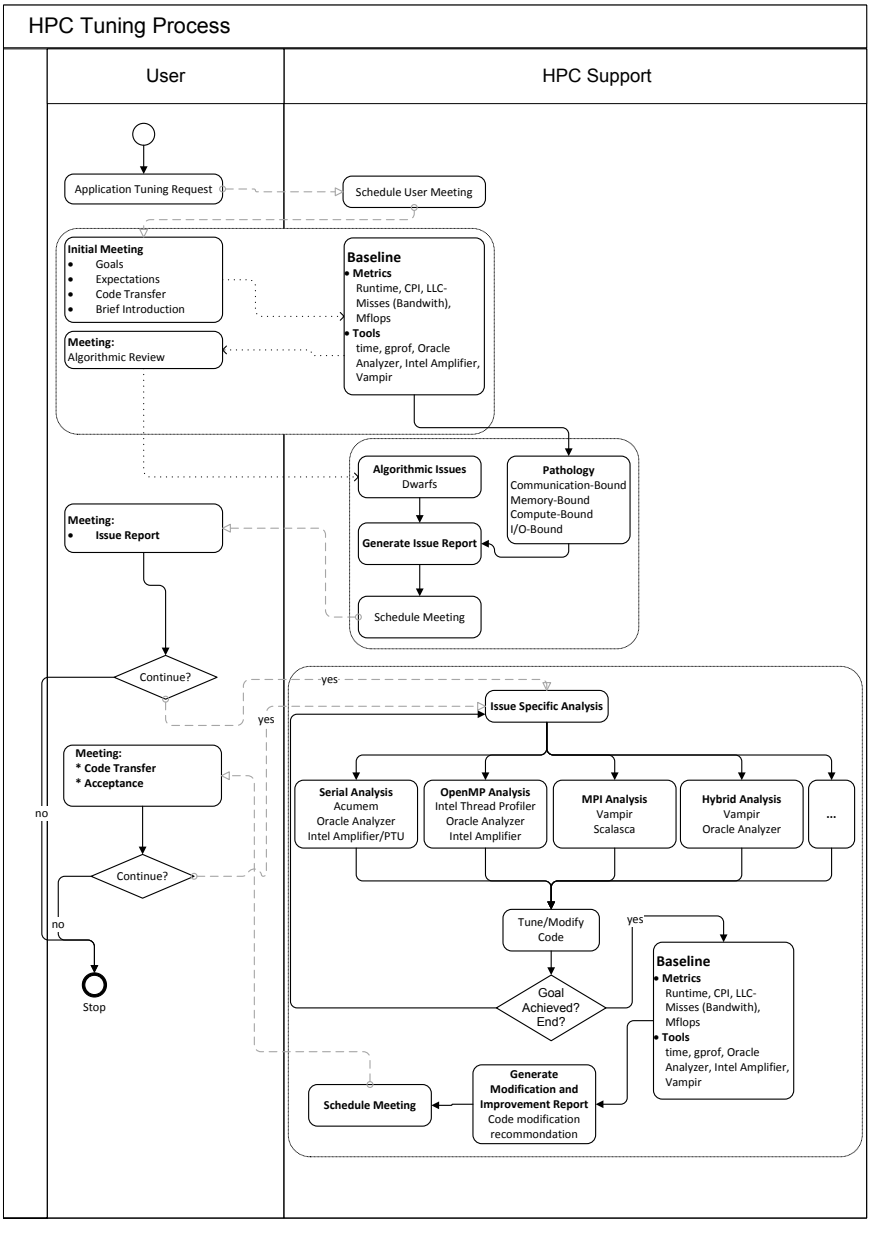

Fig. 2. Proposed Performance Tuning Workflow

in order to use this information to later detect undue instrumentation overhead. It should in no way provide details about deficiencies or tuning opportunities, though it should provide a good overview where the program does its work. This baseline obviously incorporates the runtime, peak memory usage, a simple runtime-profile, some basic hardware-performance counters (e.g. FLOP/s[5], IOP/s[6], Memory Access, Cache-Miss-Rate and CPI[7]) and if the application is parallel, a few scalability samples. If possible, information of solver-convergence rates and similar information should be preserved in order to spot deviations from the expected behavior during the tuning and measurement phases.

The recorded runtime can be used to easily measure the progress of the tuning effort as well as the perturbance induced by later performance measurements. The memory consumption itself is typically not a performance sensitive point, though the available RAM on a machine is typically the limiting factor for problem set size. The runtime-profile provides information about the general program behavior and hot-spots and becomes important during the later measurement and tuning stages, e.g. for adapting the measurement systems of many performance analysis tools, like Scalasca[5], Vampir[6] and Tau[7]. The hardware-performance counters mostly serve as easy-to-monitor progress scale. The scalability sample (serial, two-processes, many processes) serves the expert as an indicator for the programs scalability limits.

The tools for the baseline should be a fixed and predefined tool set to facilitate cross-projects comparisons and prevent unnecessary experimentation during the tuning project. For example one could define gprof[8] for serial applications, Intel Amplifier[9] for shared memory and Scalasca with hardware performance counters through PAPI[10] for message passing applications. In general, alterations to the tool-sets recommended by the workflow should only be altered in extreme situations or through a separate evaluation process between tuning activities.

3.3 Pathology

In the *Pathology*-phase the expert uses the information form the initial *baseline* to spot potential problems that the project-code may have. The expert should only survey the program for general common issues, i.e. if the application is bound by *Algorithmic Issues*,*Communication Issues*, *Computation Issues*, *Memory Access Issues* and *I/O Issues*, without spending a lot of time in detail analysis. Also each of these issues may require different tool-sets for observation and measurement. Whilst it is in this phase tempting to identify the cause of a specific problem, we recommend against it, as an early focus on a specific issue may lead to wrong conclusions. It is, in our experience, much more important to get an overall picture. In addition, the expert should gather information about the hardware-capabilities of the target platform, in order to later correlate any observed issues. For example, if an application is spending most of its time

[5] Floating Point Operations per Second.
[6] Integer Operations per Second.
[7] Cycles Per Instruction.

floating-point intensive routines (compare profile) the peak-floating point rate of the architecture may become important.

It is also important in this phase to look for general scalability issues that may derive from the used algorithms in the code potentially leading to a coarse grained performance model for the application. However, as detailed performance models have a different focus, we consider such model a separate project or service. Whilst the performance expert can recognize such algorithmic issues, the precise identification and solution can usually only be found in conjunction with the domain expert in an interdisciplinary effort, that we can not depict in this workflow.

The *Pathology*-phase should be concluded with a detailed report of problems on a coarse level, their impact on the program behavior and a recommendation which single issue to address first in the following tuning step. This report then should be discussed with the user including risk-benefit assessment based on the experience with similar codes and issues in the past. Also a specific time frame should be determined, in which the issue at hand is to be solved. In addition a process or method to verify the correctness of the code during the tuning process has to be defined or established jointly by the user and the tuning expert, as domain experts are very sensitive about numerical deviations of the correctness of their code. This verification method should later be used by the expert during the tuning cycle to prevent work that invalidates the correctness and numeric stability of the program.

If at this point no specific issue is found, the expert must decide if the potential improvement of the code is worth his effort. For computationally demanding codes, i.e. many execution instances at a high cost per execution, it may still be a prudent choice to continue to invest additional time and effort.

3.4 Tuning Cycle

Once a specific issue, like reduction of wait-times at a specific barrier or load imbalances, has been chosen as the tuning target, the expert has to decide how to detect the root-cause of the problem. Typically performance analysis tools are employed. For example in Aachen we use tools like the Intel Amplifier, Oracle Analyzer, or community developed tools like Vampir-Trace[6], Scalasca[5] or PAPI[10]. As these tools all focus on different properties and exhibit different measurement methods, the expert initially has gauge a tool's advantages and disadvantages based on experience alone. Our workflow provides valuable information for the selection of the right tool or tool-chain from the beginning, reducing his time spent on experiments.

We also recommend to establish rules and cook-book like approaches for different, typically occurring issues. Besides providing reminders and guidelines for specific checks, these cook-books can be used to monitor tuning-quality and train new experts. In our example workflow, we have indicated this, with no claim to be complete, by providing four place-holders for *Serial Analysis*, *OpenMP Analysis*, *MPI Analysis* and *Hybrid Analysis* along with tool selection for each analysis. We do not provide details on such cook books, but are of the opinion, that

a process standardization would improve average tuning quality, in particular for less experienced tuning staff. An exemplary cook-book could for example, based on a dominant wait-time in MPI routines without network-saturation from the Pathology-phase, recommend an initial MPI-only[8] Vampir based analysis for confirmation followed by a Scalasca based investigation to automatically identify the root-cause. Of course branches for potential different diagnoses and observation would have to be defined as well as detailed configuration steps.

The tuning phase itself contains many inherently iterative steps. Many available performance analysis tools require in the initial step an adaption of the measurement system to reduce the overhead and program perturbation [11] and a focus on the specific problem at hand. Is is beneficiary to use information from the base-line as a "pre-conditioner" for this process. Once a measurement of sufficient quality has been obtained, the expert has to analyze, investigate and hypothesize based on the information to determine the cause for an issue and to develop a solution.

This solution is then implemented or recommended to the user and ideally verified using the performance tool in question. If the modification does not yet deliver the agreed-on goal, then the process iterates again until the expert either solves the problem or runs out of time.

The tuning activity is then concluded by a repetition of the base-line to document the improvement, as well as to check for any side effects that may influence the usability of the change. This may for example be an increased memory footprint or different requirements upon the data-set.

The last crucial, but often forgotten step of this phase is the *Modification and Improvement Report*. It should contain detailed reasons for the code-change, its effects and extent on the programs performance in a user-understandable fashion. It should contain a protocol of the resources used for the tuning phase, be it work-time, tools or compute resources. We consider such a report to be very important, as in our experience the information provided influences the acceptability of the effort to a great extent.

3.5 Results/Handover

The tuning cycle ends in a review meeting to elaborate the modified code in conjunction with the final *Modification and Improvement Report* in detail. This provides the user with an opportunity to raise concerns regarding code-changes or the results of the last tuning cycle. These concerns and the general acceptance or reasons for dismissal should be attached to the report. Lastly, the invested tuning effort should be reflected against the achieved improvement to decide if further tuning is worthwhile or desirable. If further open issues from the pathology exists, a new iteration of the tuning cycle is started with a new specific issue (see second part of section 3.3) - otherwise the tuning projects concludes at this point.

[8] i.e. using only the wrapped MPI routines.

4 Conclusion

In this work we proposed a *Tuning Workflow* to improve the productivity of tuning-experts. We argued, that increasing complexity and diversity of parallelism in clusters requires specialized expertise to efficiently use current hardware. As domain experts typically do not have this expertise, it must be provided, in particular at University installations, where a large diversity of scientific applications typically is supported by a single HPC installation. Previous work showed that such a service pays for itself, if top user's projects are targeted in a systematic fashion by performance tuning experts. We called this the "brainware" component of an HPC operation. However, to maximize gains from brainware, we need to develop standards and processes to govern the performance tuning process to maximize its efficiency. If performance tuning is only left to "gurus" there will just not be sufficient staff available for this task.

Our tuning process itself is based on the notion of a service and distinguishes the roles of users and experts. In reality this role separation is not so clear as both sides may interact in every phase to assist each other with details when tuning and adapting the code. Nevertheless, we still see a formal tuning process description as necessary to ensure the quality of a tuning service: clear goals, deadlines, avoidance of exaggerated expectations, limitation of wasted effort and an indication when to stop. A particularly important fact is the documentation of the tuning effort in a *Modification and Improvement Report*. Such a workflow together with the entailed documentation also can provide better argumentation for funding to the management, as the costs and benefits of tuning become evident. Furthermore, this workflow can be used to train additional experts and even integrate non-scientific staff in the tuning process.

The workflow itself is based so far solely on the collective experience of the HPC-group of the RWTH Aachen University. We recognize the fact, that this workflow has yet to see a throughout study and that additional input from other HPC-sites must still be incorporated. At the time of this work, the workflow was only partially applied to one ongoing tuning effort, using only the *Baseline* and *Reporting* with good acceptance by the users.

In its current form there are some additional conceivable steps. For example the topic of version control, data management and verification remain unanswered. However, we consider the workflow in its current form to be already quite complex, such that we plan to gain further use-case experience and feedback, before revising and adding additional steps.

Whilst we did not cover any performance tools in specific, we would like to raise the question, to what extent tools could generate external documentation of performance issues.

The workflow we described as a guide for tuning processes is of course not meant to be the last word, but rather to serve as a rough guide and incentive for implementation and improvement of such tuning processes. It is also clear that depending on local characteristics these processes may need to be modified. Nonetheless, we believe that defined processes, "cook-books" for specific tasks and the requirement of modification and improvement reports are important

ingredients that should be part of such a structured process. In the long run, we hope that, similar in spirit to ITIL[9] for general IT operations, also for HPC code development and tuning a structured body of best-practice knowledge will develop to structure the increasingly complex task of ensuring good HPC performance.

References

1. Bischof, C., an Mey, D., Iwainsky, C.: Brainware for Green HPC. In: Ludwig, T. (ed.) Proceedings EnA-HPC 2011 (2011) (to appear)
2. Behr, M., Arora, D., Benedict, N.A., O'Neill, J.J.: Intel compilers on linux clusters. Intel Developer Services online publication (October 2002)
3. Zeng, P., Sarholz, S., Iwainsky, C., Binninger, B., Peters, N., Herrmann, M.: Simulation of Primary Breakup for Diesel Spray with Phase Transition. In: Ropo, M., Westerholm, J., Dongarra, J. (eds.) PVM/MPI. LNCS, vol. 5759, pp. 313–320. Springer, Heidelberg (2009)
4. Altenfeld, R., Apel, M., an Mey, D., Böttger, B., Benke, S., Bischof, C.: Parallelising Computational Microstructure Simulations for Metallic Materials with OpenMP. In: Chapman, B.M., Gropp, W.D., Kumaran, K., Müller, M.S. (eds.) IWOMP 2011. LNCS, vol. 6665, pp. 1–11. Springer, Heidelberg (2011)
5. Geimer, M., Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Mohr, B.: The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience 22(6), 702–719 (2010)
6. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The vampir performance analysis tool-set. In: Proceedings of the 2nd HLRS Parallel Tools Workshop, Stuttgart, Germany (July 2008)
7. Shende, S.S., Malony, A.D.: The tau parallel performance system. The International Journal of High Performance Computing Applications 20, 287–331 (2006)
8. GNU: gprof, `http://sourceware.org/binutils/docs/gprof/`
9. Intel: Intel©parallel amplifier (2011) `http://software.intel.com/en-us/articles/intel-parallel-amplifier/`
10. London, K., Moore, S., Mucci, P., Seymour, K., Luczak, R.: The papi cross-platform interface to hardware performance counters. In: Department of Defense Users Group Conference Proceedings, pp. 18–21 (2001)
11. Iwainsky, C., an Mey, D.: Comparing the Usability of Performance Analysis Tools. In: Cèsar, E., Alexander, M., Streit, A., Träff, J., Cèrin, C., Knüpfer, A., Kranzlmüller, D., Jha, S. (eds.) Euro-Par 2008 Workshops. LNCS, vol. 5415, pp. 315–325. Springer, Heidelberg (2009)

[9] `www.itil.org`