How to Accelerate an Application: a Practical Case Study in Combustion Modelling

Benedetto RISIO [a,1], Alexander BERRETH [a], Stéphane ZUCKERMAN [b], Souad KOLIAI [b], Mickaël IVASCOT [b], William JALBY [b], Bettina KRAMMER [b], Bernd MOHR [c], and Thomas WILLIAM [d]

[a] *RECOM Services GmbH, Germany*

[b] *Université de Versailles Saint-Quentin-en-Yvelines, France*

[c] *JSC Jülich, Germany ;* [d] *GWT-TUD GmbH Dresden, Germany*

Abstract. Developing parallel high-performance applications is an error-prone and time-consuming challenge. Performance tuning can be alleviated considerably by using optimisation tools, either by simply applying a stand-alone tool or by applying a tool chain with a number of more or less integrated tools covering different aspects of the optimisation process. In the present paper, we demonstrate the benefits of the latter approach on the industrial combustion modelling software RECOM-AIOLOS. The applied tool chain comprises both low-level and high-level analysis of the application: using the MAQAO tool, the assembly code generated by the compiler can be analysed statically, aiming at possible optimisations on a loop level. Another important aspect is identifying and optimising bottlenecks in memory access and cache utilisation. On a higher level, efficient usage of parallel programming paradigms (MPI, OpenMP) is verified by the VAMPIR and SCALASCA frameworks. Combining the different optimisation strategies leads to a significant overall performance improvement.

Keywords. performance analysis, optimization, application, combustion

1. Introduction

The 3D-combustion modeling software RECOM-AIOLOS is a tailored application for the mathematical modelling of industrial firing systems ranging from several hundred kW to more than 1000 MW. In-depth validation using measurements from industrial power plants, the extension of chemical reaction models and the rapid development of computer technology have made RECOM-AIOLOS a well proven and reliable tool for the prediction of industrial furnace efficiency. The software solves approx. 100 conservation equations (mass, momentum, energy, species concentrations, radiation) on a 10-15 million cells finite volume grid, leading to high computational demands. Originally being designed for high-performance computing on parallel vector-computers and massively parallel

[1] RECOM Services GmbH, Nobelstrasse 15, D-70569 Stuttgart, Germany, Fax: +49-711-6868 9149, E-mail: info@recom-services.de

systems, the software has been ported to low-cost multi-core systems to expand the hardware base. In the present work a suite of optimization tools (MARMOT [1], VAMPIR [2], SCALASCA [3], MAQAO [4]) was used for performance tuning to identify execution bottlenecks and potential improvements in a systematic way.

2. Applied Workflow for Performance Tuning

The applied workflow for performance tuning consists of a combination of high-level and low-level performance analysis. The total workflow for performance tuning was executed in the following sequence:

1.) An optimized massively parallel execution across nodes was achieved by a high-level analysis of the existing MPI-Parallelization. The tool MARMOT was used to check the MPI-Implementation. Furthermore, the MPI-parallel execution was analysed with the tools VAMPIR & SCALASCA.

2.) An optimized execution on individual cores was achieved by a low-level performance analysis for single and multi-core execution within the node. The tools VAMPIR & MAQAO were used to assess single and multi-core execution on a subroutine level using performance counters. Furthermore, the tool MAQAO was also used to identify performance bottlenecks on a loop level within critical subroutines.

3.) An optimized shared memory parallel execution within the multi-core node has been achieved by a high-level performance analysis of hybrid parallelism. The work focused on the investigation of the benefits of using hybrid (OpenMP and MPI) parallel execution within the node.

3. High Level Performance Analysis of MPI-Parallelization

The MPI-performance of the RECOM-AIOLOS code on multi-core systems was analyzed using the tools VAMPIR & SCALASCA. The performance analysis of the original MPI-communication for a portion of the code covering 0.25 s is shown in Fig. 1. The red zones in Fig. 1 indicate that the MPI-communication generates an unexpectedly large number of wait events at the synchronisation points of the application. A deeper analysis of the problem revealed, that the MPI-processes were forced to synchronize in a prescribed sequence by using the MPI_WAIT subroutine for synchronisation. This led to a significant overhead in the synchronisation because some of the MPI-processes were still in computation while others were ready for synchronisation, but because of the prescribed synchronisation sequence had to wait for other MPI-processes to finish their computation. The problem was solved by replacing the MPI_WAIT subroutine

with MPI_WAITALL, thus allowing the MPI-processes to synchronize as they finish their computation.

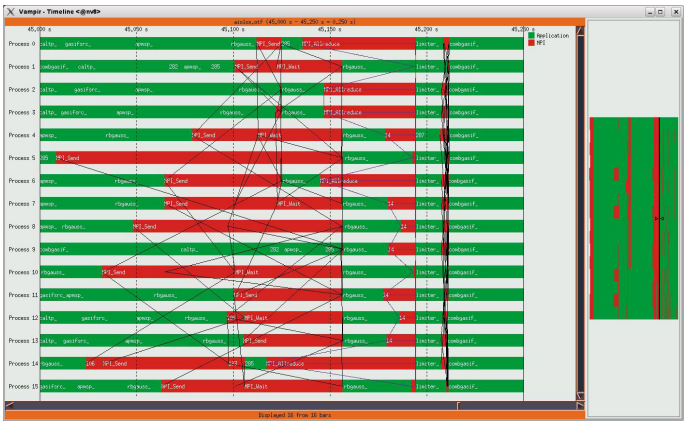


Figure 1: Performance analysis of original MPI-communication

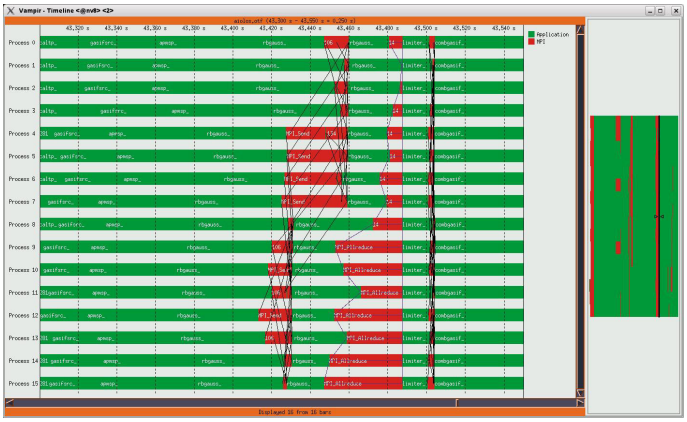


Figure 2. Performance analysis of modified MPI-communication

The modified MPI-communication for the same portion of the code covering 0.25 s is shown in Fig. 2. Fig. 2 shows a significant lower amount of red zones indicating that the MPI-communication now generates a significant lower number of wait events at the synchronization points of the application.

The savings in execution time achieved with the modified MPI-communication (MPI_WAIT versus MPI_WAITALL) are summarized in the Fig. 3. The results indicate, that the performance improvements increase with increasing number of cores. Execution time savings in the order of 10 % were measured for 128 cores.

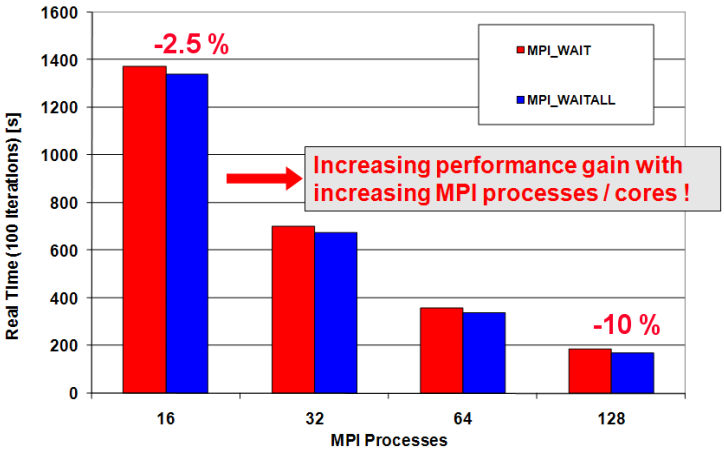


Figure 3. Performance improvements achieved with modified MPI-communication

An MPI-performance analysis with the optimized MPI-code was performed on a Xeon-Cluster (NOVA) using up to 128 cores. The result of this analysis is shown in Fig. 4.

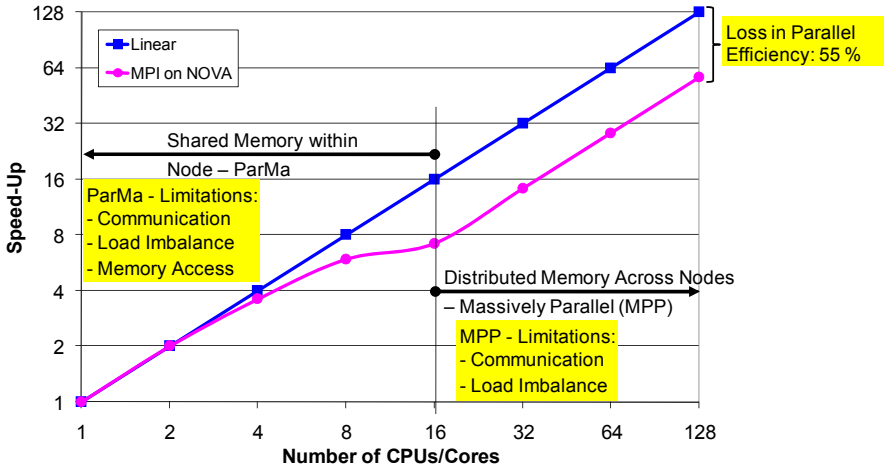


Figure 4. MPI-performance analysis on a Xeon-Cluster (NOVA)

The results shown in Fig. 4 indicate an almost linear scaling for the massively parallel (MPP) execution across the nodes. However, a performance breakdown can be observed within the node. A closer look at this so-called Parallel Multi-Core (ParMa) region reveals good scaling up to four cores and a continuous breakdown for 8 and 16 cores. The total loss in parallel efficiency when using the entire node is 55%. The performance loss within the node is hereby the critical factor for the entire multi-core system.

4. Low Level Performance Analysis for Single and Multi-Core

In order to analyze the performance breakdown within the node more deeply the tools VAMPIR & MAQAO were used to assess the general sustained performance in terms of MFlops achieved. The results of the performance measurements for single core and full node utilisation are summarised in Figs. 5 and 6. The analysis shown in Figs. 5 and 6 reveals, that the subroutines Combgasif and Combedc (calculation of the homogeneous gas-phase and the heterogeneous particle chemistry) that perform a large number of floating point operations in relation to the data that is required from the memory achieve almost 930-970 MFlops of computational speed on a single core, and the computational speed drops only slightly to 820-910 MFlops when using 16 cores simultaneously. On the other hand the subroutines Rbgauss and Apmsp that do a smaller number of floating point operations in relation to the required data from the memory achieve only a reduced computational speed of 400-410 MFlops on a single core, and the computational speed severely drops to 100-110 MFlops when using 16 cores simultaneously.

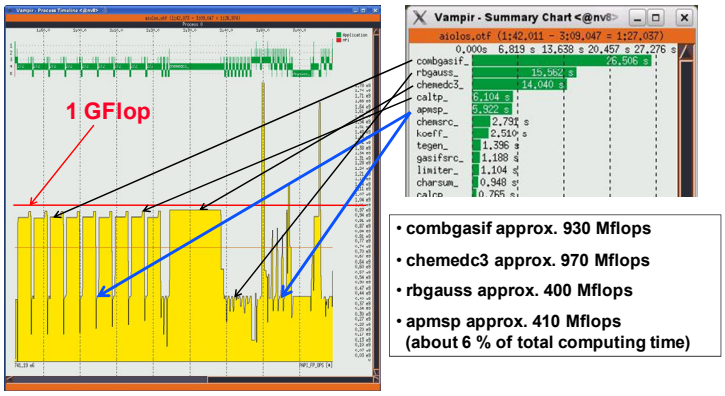


Figure 5. Analysis of single core performance

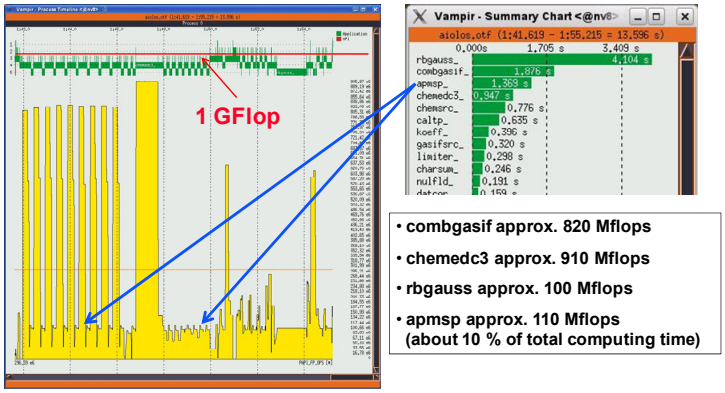


Figure 6. Analysis of full node performance (16 cores)

The analysis of the parallel performance on subroutine level summarized in Fig. 7 shows, that the memory-intense routines Rbgauss and Apmsp also exhibit a severe breakdown in parallel performance when using more than two cores simultaneously, while scaling is almost linear for the compute-intense subroutines Combgasif and Combedc. The function powL (power law system function) scales ideally with increasing number of CPUs/Cores because it only performs floating point operations without any memory access. This is somehow misleading, because this function is called mostly within the Combgasif and Combedc subroutine that contains the required memory access time. Taking this into account these both routines would even scale better.

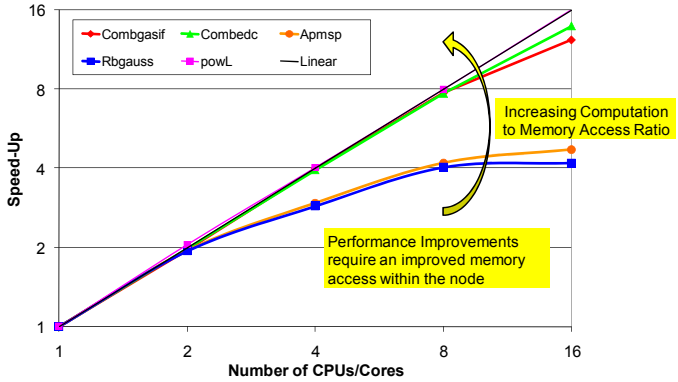


Figure 7. Parallel performance on subroutine level

The performance breakdown of the Rbgauss subroutine, which is a Gauss-Seidl Solver with Red-Black-Ordering for resolving the data dependency within the loop, was further analysed on the loop-level with the MAQAO tool.

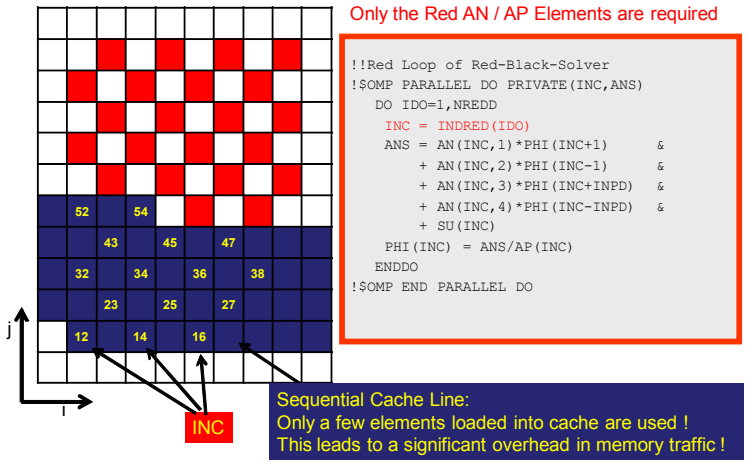
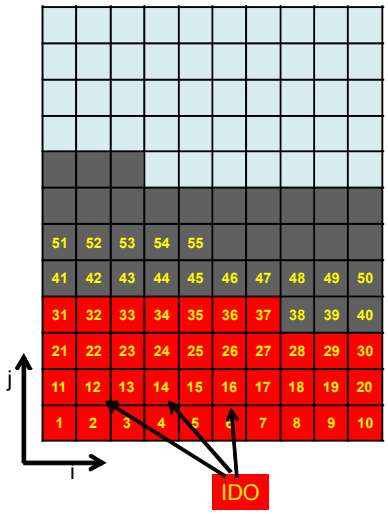


Figure 8. Original implementation in Rbgauss subroutine

The outcome of this analysis showed, that due to the internal data structure within the code, only a few elements of the sequential cache line were used for calculation (see Fig. 8). This led to a significant overhead in memory traffic. The majority of the memory traffic was hereby generated through the loading of the array AN and AP (see Fig. 8) using an indirection array. The remedy was a reorientation of the load intensive data stream. The arrays AN and AP were stored linearly in the memory and accessed via a direct access (see Fig. 9), leading to a full utilisation of the data that was loaded into the cache. The reoriented arrangement is shown in Fig. 9. The resulting performance improvement due to the restructuring of the Rbgauss subroutine is 38 % on a subroutine level and around 13.6 % on 16 cores for the entire application.



Reorientation of AN / AP Field:

```
!!Red Loop of Red-Black-Solver
!$OMP PARALLEL DO PRIVATE(INC,ANS)
   DO IDO=1,NREDD
    INC = INDRED(IDO)
    ANS = AN(IDO,1)*PHI(INC+1)      &
        + AN(IDO,2)*PHI(INC-1)      &
        + AN(IDO,3)*PHI(INC+INPD)   &
        + AN(IDO,4)*PHI(INC-INPD)   &
        + SU(INC)
    PHI(INC) = ANS/AP(IDO)
   ENDDO
!$OMP END PARALLEL DO
```

Figure 9. Reoriented data arrangement in Rbgauss subroutine

5. High Level Performance Analysis of Hybrid Parallelism

Another approach for better using the available memory bandwidth on a multi-core systems was found by adapting the parallel programming model to the given memory architecture. The adaption was done by varying the number of MPI-processes and OpenMP threads in the parallel execution (hybrid approach). A comparison of the performance achieved with hybrid parallel execution on Itanium versus Xeon X7350 (Tigerton) and Nehalem is shown in Fig. 10. The results in Fig. 10 show, that memory degradation is so severe on the Xeon system that a variation of the number of MPI-processes and OpenMP threads hardly effects the parallel performance, while the Itanium and the Nehalem system take advantage from using 4 OpenMP threads on a single logical memory block. Fig. 10 demonstrates the superiority of the memory access in a NUMA architecture compared to a conventional bus system. Performance improvements of roughly 15% were achieved through the usage of a hybrid parallel approach.

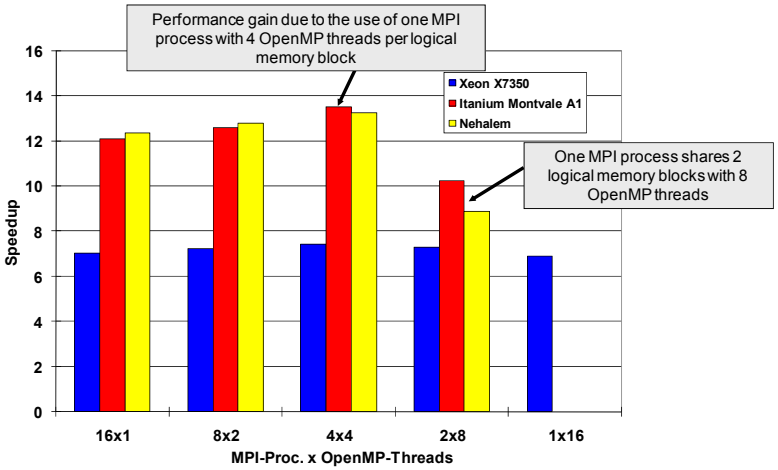


Figure 10. Hybrid parallel performance on multi-core architectures

6. Summary

This article reveals that applying a tool chain of well selected optimization tools allows a thorough analysis of the performance bottlenecks and the identification of the most promising measures for performance improvements on multi-core systems. Even for a high performance computing application like the 3D-combustion modeling software RECOM-AIOLOS a performance gain of roughly 34% for the entire application has been achieved.

Acknowledgements

The research presented in this paper has partially been supported by the German Ministry for Research and Education (BMBF), and the French Ministry for Economy, Industry and Employment (DGCIS) through the ITEA2 project "ParMA" [5] (June 2007 – May 2010).

References

[1] Krammer, B.; Bidmon, K.; Müller, M.S.; Resch, M.M.: MARMOT: An MPI Analysis and Checking Tool. In Joubert, G.R., Nagel, W.E., Peters, F.J., Walter, W.V., eds.: PARCO. Volume 13 of Advances in Parallel Computing., Elsevier (2003) 493–500

[2] Müller, M.S.; Knüpfer, A.; Jurenz, M.; Lieber, M.; Brunst, H.; Mix, H.; Nagel, W.E.: Developing Scalable Applications with Vampir, VampirServer and VampirTrace, Advances in Parallel Computing, Volume 15, ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

[3] Geimer, M.; Wolf, F.; Wylie, B. J. N.; Abraham, E.; Becker, D.; Mohr, B. (2008): The SCALASCA Performance Toolset Architecture, Proceedings of the International Workshop on Scalable Tools for High-End Computing (STHEC), Kos, Griechenland / ed.: M. Gerndt, J. Labarta, B. Miller. - 2008. - pp. 51 – 65

[4] Djoudi, L.; Barthou, D.; Carribault, P.; Lemuet, C.; Acquaviva, J-T.; Jalby, W.: MAQAO: Modular Assembler Quality Analyzer and Optimizer for Itanium 2, Workshop on EPIC architectures and compiler technology, San Jose, 2005.

[5] ParMA: Parallel Programming for Multi-core Architectures - ITEA2 Project (06015). http://www.parma-itea2.org/