Performance engineering of GemsFDTD computational electromagnetics solver

Ulf Andersson[1] and Brian J. N. Wylie[2]

[1] PDC Centre for High Performance Computing, KTH Royal Institute of Technology, Stockholm, Sweden
ulfa@pdc.kth.se
[2] Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany
b.wylie@fz-juelich.de

Abstract. Since modern high-performance computer systems consist of many hardware components and software layers, they present severe challenges for application developers who are primarily domain scientists and not experts with continually evolving hardware and system software. Effective tools for performance analysis are therefore decisive when developing performant scalable parallel applications. Such tools must be convenient to employ in the application development process and analysis must be both clear to interpret and yet comprehensive in the level of detail provided. We describe how the Scalasca toolset was applied in engineering the GemsFDTD computational electromagnetics solver, and the dramatic performance and scalability gains thereby achieved.

Keywords: performance engineering, parallel execution tuning, scalability, MPI, computational electromagnetics

1 Introduction

Parallel applications develop from initial barely-functional implementations, via a series of debugging and performance improvement stages, into efficient and scalable versions via disciplined performance engineering practices. This process remains on-going throughout the productive lifetime of the application, as larger and more complex computational problems are addressed and upgraded hardware and system software become available. Although codes which become accepted benchmarks are more rigorously investigated than typical applications, they are not immune from the need for continual performance evaluation and re-engineering.

This paper considers the GemsFDTD code from the SPEC MPI2007 benchmark suite [6] which was found to perform particularly poorly at larger scales on distributed-memory computer systems. Initial analysis with the Scalasca toolset and then other tools pinpointed aspects in the application's initialization phase that severely limited scalability. Scalasca is an open-source toolset for analysing the execution behaviour of applications based on the MPI and OpenMP parallel

To appear in Proc. PARA 2010 (Reykjavík, Iceland), Part I, LNCS 7133, Springer.

programming interfaces supporting a wide range of current HPC platforms [2,3]. It combines compact runtime summaries, that are particularly suited to obtaining an overview of execution performance, with in-depth analyses of concurrency inefficiencies via event tracing and parallel replay. With its highly scalable design, Scalasca has facilitated performance analysis and tuning of applications consisting of unprecedented numbers of processes [10,11].

Based on these performance analyses, the developers of the GemsFDTD code could rework the initialization and further optimize the time-stepping loop, to realize substantial application execution performance and overall scalability improvements, ultimately leading to an entirely updated version of the benchmark. We review both initial and revised versions of the code, and their performance analysis with the Scalasca toolset which revealed now resolved and still remaining performance optimization opportunities.

2 GemsFDTD code versions

2.1 *113.GemsFDTD* — SPEC MPI2007 v1.1

The SPEC MPI2007 code *113.GemsFDTD* solves the Maxwell equations using the finite-difference time-domain (FDTD) method [8]. The radar cross-section of a perfectly conducting object is computed. *113.GemsFDTD* is written in Fortran 90 and is a parallel version of the SPEC CFP2006 (Floating Point Component of SPEC CPU2006) code *459.GemsFDTD*. *113.GemsFDTD* is a subset of a general purpose time-domain code for the Maxwell equations developed within the General ElectroMagnetic Solvers (GEMS) project at PSCI [5].

The core of the FDTD method is second-order accurate central-difference approximations of Faraday's and Ampere's laws. These central-differences are employed on a staggered Cartesian grid resulting in an explicit finite-difference method. These updates are performed in the module `material_class`. The FDTD method is also referred to as the Yee scheme. It is the standard time-domain method within computational electromagnetics [8].

An incident plane wave is generated using so-called Huygens' surfaces. This means that the computational domain is split into a total-field part and a scattered-field part, where the scattered-field part surrounds the total-field part. It uses the `excite_mod` module to compute the shape of the incident fields.

The computational domain is truncated by an absorbing layer in order to minimize the artificial reflections at the boundary. The uni-axial perfectly matched layer (UPML) by Gedney [1] is used. A time-domain near-to-far-field transformation computes the radar cross-section according to Martin and Pettersson [4], handled by the module `NFT_class`.

The execution time during the timestepping is concentrated in five subroutines, two update routines, two UPML routines, and the routine `NFT_store`.

The problem size in *113.GemsFDTD* is $580 \times 580 \times 580$ FDTD cells ($N_x = N_y = N_z = 580$) surrounded by a twelve cell UPML layer.

2.2 *145.lGemsFDTD* — SPEC MPI2007 v2.0

113.GemsFDTD was designed to be scalable up to 256 processes according to the aim of SPEC MPI 2007. Version 2.0 of SPEC MPI demanded that the codes were scalable up to 2048 processes. *113.GemsFDTD* failed miserably at this [7], thus an extensive rewrite was needed. The end result of this rewrite was *145.lGemsFDTD* ('l' signifies large) which was accepted into version 2.0 of SPEC MPI. (The original *113.GemsFDTD* is retained in SPEC MPI 2007 v2.0 medium-sized benchmark suite for compatibility with earlier releases.)

The problem size in *145.lGemsFDTD* is $960 \times 960 \times 960$ FDTD cells ($N_x = N_y = N_z = 960$) surrounded by a twelve cell UPML layer. This was selected in order to meet the SPEC request that the memory footprint should be slightly less than 64 GiB.

Initial performance analysis with Scalasca showed clearly that *113.GemsFDTD* performed a lot of 4-byte broadcasts during initialization. Time measurements inside the code itself showed that the `multiblock_partition` routine on the master rank was a serial bottleneck. In fact, the execution time increased with the number of blocks, which increases linearly with the number of MPI processes. This performance analysis made it clear where the hotspots in the code were and was very useful to the programmer.

2.3 Domain decomposition of GemsFDTD

The original Gems code (*MBfrida*) lets the user select the number of processes (p) and the number of FDTD blocks in all three dimensions (Nbx, Nby, Nbz), where the total number of FDTD blocks are $NbF = Nbx \times Nby \times Nbz$. If the user selects $Nbx = Nby = Nbz = 0$, then the code sets $NbF = p$ and uses `MPI_Dims_create` to set (Nbx, Nby, Nbz). The size of the FDTD blocks are ($Nbx/N_x, Nby/N_y, Nbz/N_z$) (or slightly smaller). If a layer of UPML is added, then the total number of blocks become $NbT = (Nbx+2) \times (Nby+2) \times (Nbz+2)$.

In *113.GemsFDTD* it was chosen to always use `MPI_Dims_create`, but setting NbF to $p-2$ or $p-3$ instead of p so that NbF is an even number. This was done in order to get at least two processes that only have UPML blocks. NbF is then sometimes further decreased in order to avoid getting elongated FDTD-blocks which is undesirable since that will lead to more UPML-blocks and more data to communicate between the blocks. A precalculated table decides which NbF values are accepted. In the case of $p = 256$, 254 and 253 are not accepted values, while 252 ($=6\times6\times7$) is. Due to a bug in the code only 252+2=254 processes are used when computing a distribution of the 576 blocks (252 FDTD blocks and 324 UPML blocks).

When the number of blocks and their sizes are decided, the master computes the workload of each block and then, in the routine `multiblock_partition`, computes a distribution of the blocks onto the MPI ranks.

For *145.lGemsFDTD*, NbF is the largest approved value that is less than or equal to the number of processes p. For $p <= 256$, the same precalculated table as for *113.GemsFDTD* is used to decide whether a NbF value is approved,

whereas for $p > 256$ we demand that $(NbT/NbF)/(1 + 6/\sqrt[3]{NbF}) < 1.3$ in order to approve NbF. This makes sure that NbT/NbF is bounded.

After a block distribution is computed, the master loops through the blocks and broadcasts information about each block. In *113.GemsFDTD* this part had $65 \times NbT + 25$ four-byte broadcasts. In *145.lGemsFDTD* this has been reduced to $7 \times NbT + 11$ small broadcasts by merging adjacent broadcasts. Further reductions were possible but would have meant extensive rewrites of the code.

2.4 Summary of the main changes

The major improvements in *145.lGemsFDTD* compared to *113.GemsFDTD* for the phases where they apply:

Initialization

1. Develop a new `multiblock_partition` routine designed for larger numbers of processes, which produces a completely different domain decomposition.
2. Use of fewer `MPI_Bcast` operations within `multiblock_distribute` and associated routines.
3. Separation of broadcasts and allocations in the block distribution phase, such that the rank that owns the block delays allocations for the block until after block information has been broadcasted for all blocks.

Time-stepping iterations

1. Removal of expensive recomputation of the communication pattern used to exchange blocks in `multiblock_communicate`, since the same communication pattern is used in each timestep.
2. Replacement of `MPI_Sendrecv` with non-blocking `MPI_Isend` and `MPI_Irecv` communication.
3. Interchanging loops in the near-to-far-field transformation computations. (This improvement was found analyzing the serial code *459.GemsFDTD*.)

3 Performance measurements & analyses

During the course of development of GemsFDTD, performance measurements of each version were done on a variety of platforms and with different compilers to verify the portability and effectiveness of each of the modifications. Although benefits depend on the respective processor, network and compiler/optimizer capabilities, execution performance analysis with Scalasca and the improvement achieved for a representative example system are studied in detail.

3.1 Execution scalability

The original version of the GemsFDTD benchmark code (*113.GemsFDTD* SPEC in MPI2007 v1.1) and revised version (*145.lGemsFDTD* in SPEC MPI2007 v2.0) were executed on the *HECToR* Cray XT4 [9] with varying numbers of processes. The XT4 has quad-core 2.3 GHz AMD Opteron processors in four-socket compute blades sharing 8 GiB of memory, allowing the benchmark code to run with 32 or more MPI processes. The codes were built with PGI 9.0.4 compilers using typical optimization flags "-fastsse -O3 -Mipa=fast,inline" as well as processor-specific optimization. Although *113.GemsFDTD* would normally be run with a 'medium-sized' input dataset (`sphere.pec`), to allow comparison of the two versions we ran both with the 'large-sized' training (i.e., `ltrain`) dataset using `RAk_funnel.pec`. It was convenient to use only 50 timesteps rather than the `lref` benchmark reference number of 1500 timesteps, since previous analysis of GemsFDTD [7] determined that there was no significant variation in execution performance for each timestep. Full benchmark execution time can be estimated by multiplying the time for 50 iterations by a factor of 30 and adding the initialization/finalization time.

Execution times reported for the entire code and only for the 50 timestep loop iterations of both versions are shown in Figure 1. While the iterations are seen to scale well in both versions, the initialization phases only scale to 128 and 512 processes, respectively. For the original code, the initialization phase becomes particularly dominant, even with only 256 processes, and makes it prohibitive to run at larger scales.

3.2 Scalasca performance analyses

Both versions of GemsFDTD were prepared for measurement with the Scalasca instrumenter (1.3 release), which configured the Cray/PGI compiler to automatically instrument each user-level source routine entry and exits, and linked with its measurement library which include instrumented wrappers for MPI library routines. The instrumented executables were then run under the control of the Scalasca measurement and analysis nexus within single batch jobs (to avoid impact of acquiring different partitions of compute nodes in separate jobs).

By default, a Scalasca runtime summarization experiment consisting of a full call-path execution profile with auxilliary MPI statistics for each process is collected and stored in a unique archive directory. With all of the user-level source routines instrumented, there can be undesirable measurement overheads for small frequently-executed routines. An initial summary experiment is therefore scored to identify which routines should be filtered: with GemsFDTD, the names of nine routines were placed in a text file and specified to be filtered from subsequent measurements in Scalasca summary and trace experiments.

Scalasca summary analysis reports contain a breakdown of the total run time into pure (local) computation time and MPI time, the latter split into time for collective synchronization (i.e., barriers), collective communication and point-to-point communication, as detailed in Figure 2. Both code versions show good

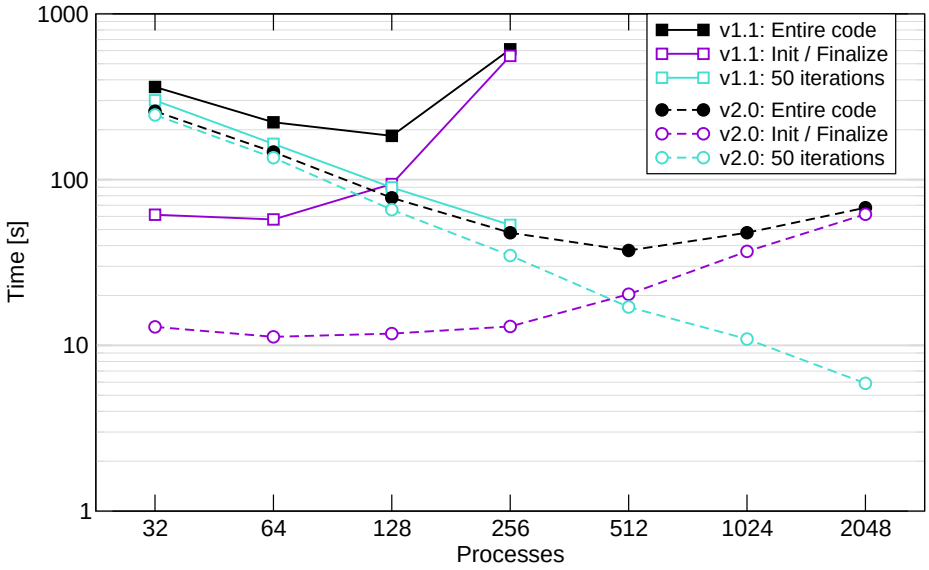

Fig. 1. Execution times of original and revised GemsFDTD versions with 'ltrain' dataset for a range of configuration sizes on Cray XT4.

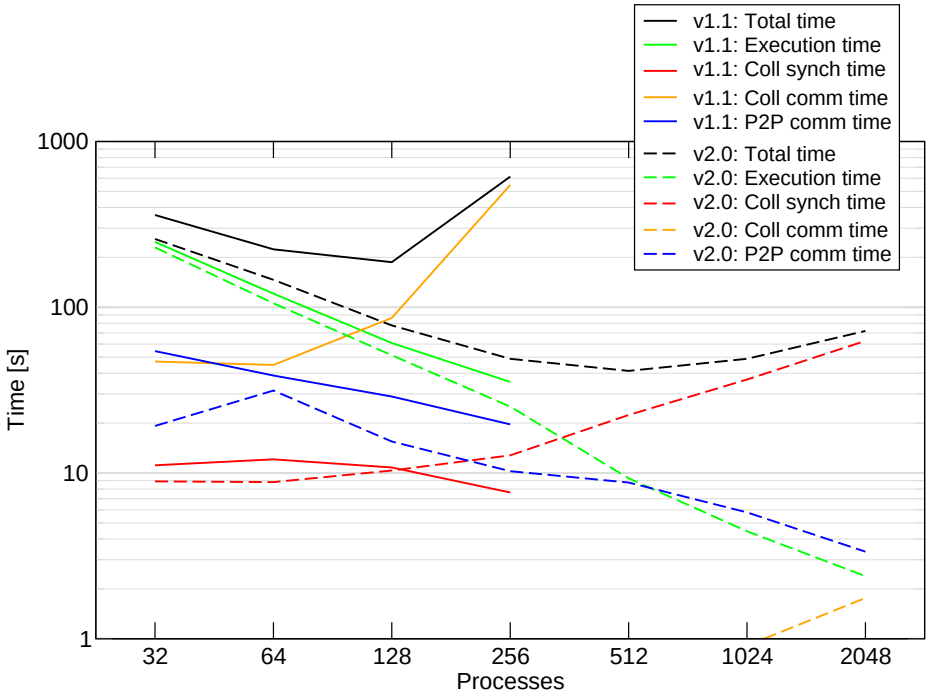

Fig. 2. Scalasca summary analysis breakdown of executions of original and revised GemsFDTD versions with 'ltrain' dataset on Cray XT4 (averages of all processes).

scaling of the computation time, with the new version demonstrating better absolute performance (at any particular scale) and better scalability overall. The extremely poor scalability of the original code version is due to the rapidly increasing time for collective communication, isolated to the numerous `MPI_Bcast` calls during initialization. Collective communication in the revised version is seen growing significantly at the largest scale, however, the primary scalability impediment is the increasing time for explicit barrier synchronization (which is not a factor in the original version). Point-to-point communication time is notably reduced in the revised version, but also scales less well than the local computation, which it exceeds for 1024 or more processes.

Scalasca automatic trace analysis profiles are similar to those produced by runtime summarization, however, they include assessments of waiting times inherent in MPI collective and point-to-point operations, such as *Late Sender* time when an early receiver process must block until the associated message transfer is initiated. Trace analysis profiles from 256-process experiments as presented by the Scalasca analysis report explorer GUI are shown in Figures 3 and 4 comparing the original and improved GemsFDTD execution performance. In Figure 3, with the metric for total time selected from the metric trees in the leftmost panes, and the routines that constitute the initialization phase of GemsFDTD selected from the central call-tree panes, the distribution of times per process is shown with the automatically acquired Cray XT4 machine topology in the right panes. The MPI process with rank 0 is selected in the v2.0 display, and the two processes in the uppermost row of compute nodes that idled throughout the v1.1 execution can be distinguished. (Only the subset of the HECToR Cray XT4 associated with the measured execution is shown, with non-allocated compute nodes grayed or dashed.) In contrast, Figure 4 features *Point-to-point communication time* selected from the metric tree and the associated MPI routines within `multiblock_communicate` of the solver iteration phase.

The Scalasca analysis reports from GemsFDTD v1.1 and v2.0 trace experiments with 256 processes on the Cray XT4 are compared in Table 1 to examine the 12-fold speedup in total execution time of the revised version. Dilation of the application execution time with instrumentation and measurement processing was under 2% compared to the uninstrumented reference version.

Initialization is more than 40 times faster through the combination of reworking the `multiblock_partition` calculation and using almost 9 times fewer broadcasts in `multiblock_distribute` (even though 3% more bytes are broadcast). Note that in v1.1 the majority of broadcast time (measured in `MPI_Bcast`) is actually *Late Broadcast* time on the processes waiting for rank 0 to complete `multiblock_partition`, however, the non-waiting time for broadcasts is also reduced 20-fold for v2.0. Improvement in the solver iterations is a more modest 33%, however, still with speedup in both calculation and communication times. Significant gains were therefore realized through use of non-blocking communication and improved load balance (including exploiting the two previously unused processes), despite 5% more data being transfered in `multiblock_communicate`.

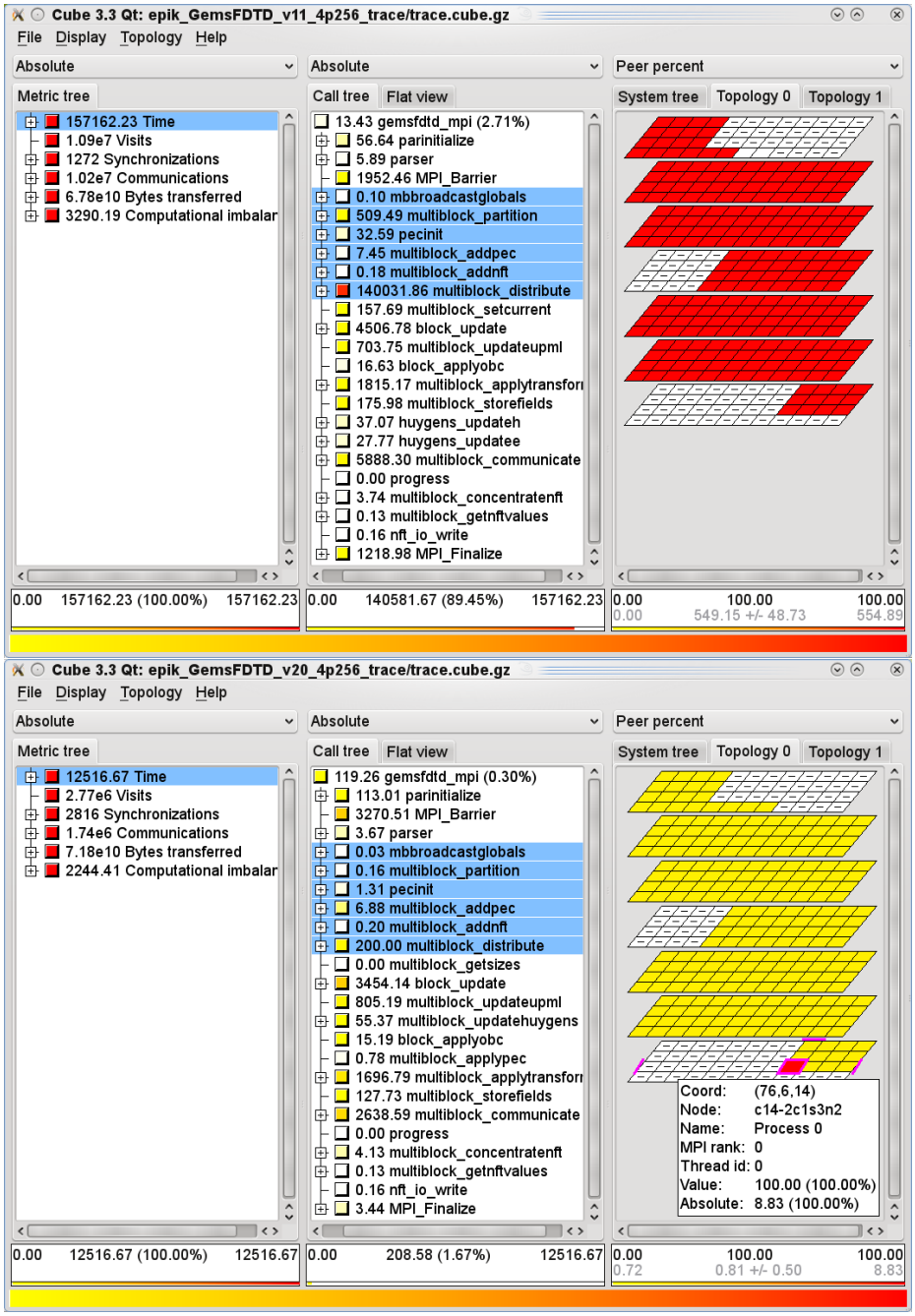

Fig. 3. Scalasca analysis report explorer presentations of GemsFDTD trace experiments with 'ltrain' dataset for 256 processes on Cray XT4 (v1.1 above and v2.0 below) highlighting 675-fold improvement of total time for the initialization phase.

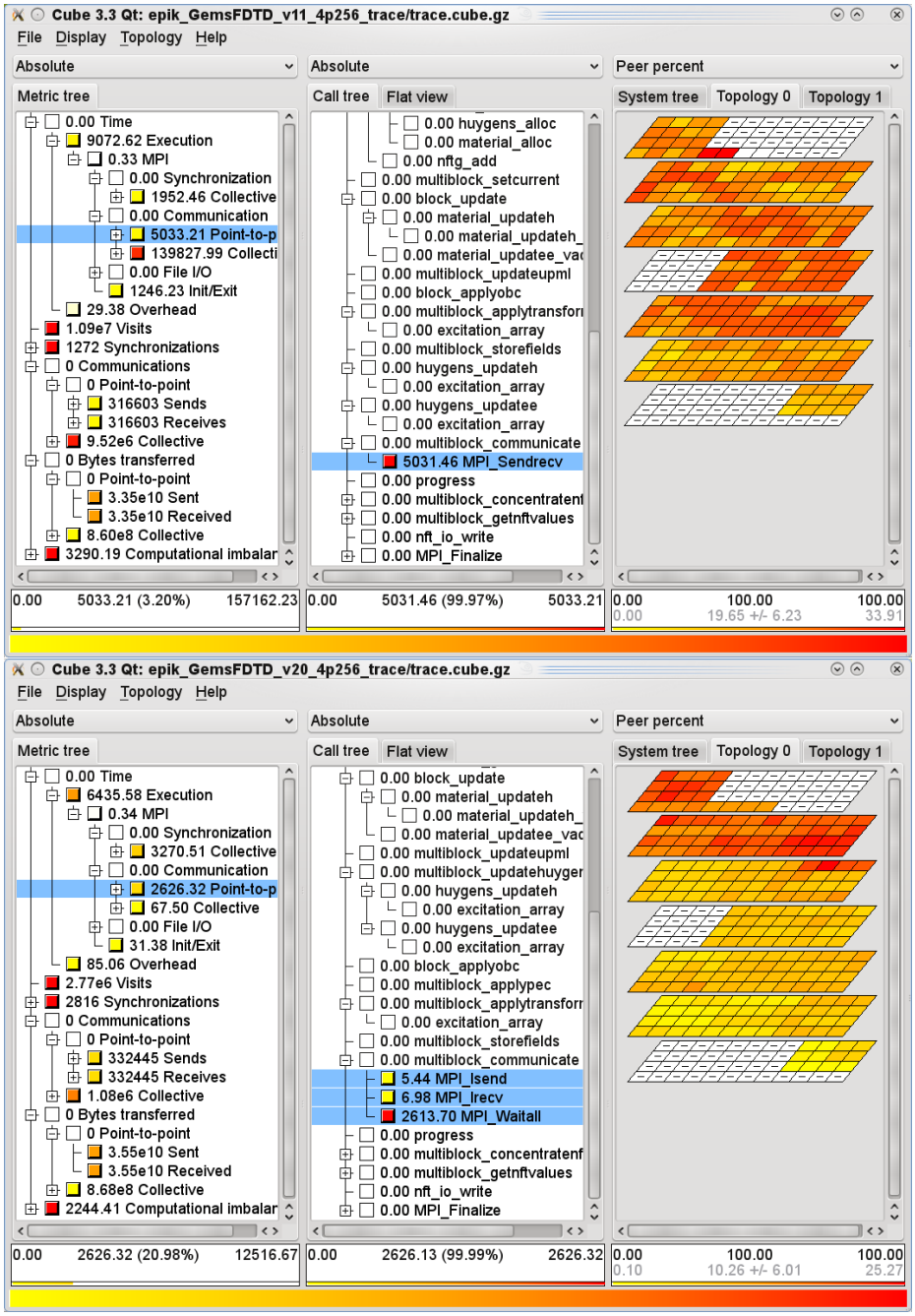

Fig. 4. Scalasca analysis report explorer presentations of GemsFDTD trace experiments with 'ltrain' dataset for 256 processes on Cray XT4 (v1.1 above and v2.0 below) highlighting 2-fold improvement of point-to-point communication time in solver phase.

Table 1. Selected performance metrics and statistics for Scalasca trace experiments for GemsFDTD versions with 'ltrain' dataset for 256 MPI processes on Cray XT4. (Maximum of any individual measured process where qualified.)

GemsFDTD		v1.1	v2.0
Reference execution time [s]		611.3	47.8
Measured execution time [s]		613.8	48.6
Initialization		561.1	13.1
– `multiblock_partition` time [s]	(max)	509.5	0.2
– `multiblock_distribute` time [s]	(max)	554.9	0.9
– broadcast time [s]	(max)	554.1	0.3
– *Late Broadcast* time [s]	(max)	552.0	0.2
– number of broadcasts	(max)	37465	4214
– bytes incoming by broadcast [kiB]	(max)	146.3	150.1
Time-stepping iterations		52.6	35.0
– Calculation time [s]	(max)	47.6	34.4
– Communication time [s]	(max)	33.9	25.3
– number of sends/receives	(max)	2900	2200
– bytes sent [MiB]	(max)	134.5	140.7
– *Late Sender* time [s]	(max)	33.7	25.2
– number of *Late Senders*	(max)	1455	50
Scalasca tracing			
– Trace total size [MiB]		410	96
– Trace collection time [s]		0.5	0.4
– Trace analysis time [s]		227.6	2.3
– event replay analysis [s]		2.2	0.5
– event timestamp correction [s]		223.8	1.3
– number of violations corrected		2529	762

The Scalasca traces are less than one quarter of the size for the v2.0 version due to the reduced number of broadcasts, requiring less than half a second to write the traces to disk and unify the associated definitions. Replay of recorded events requires correspondingly less time, however, the time processing event timestamps to correct logical inconsistencies arising from the unsynchronized clocks on Cray XT compute nodes is reduced by over 170-fold, since correcting timestamps of collective operations is particularly expensive. For the v2.0 version of GemsFDTD at this scale, Scalasca trace collection and automatic analysis require only 6% additional job runtime compared to the usual execution time.

4 Conclusions

The comprehensive analyses provided by the Scalasca toolset were instrumental in directing the developer's performance engineering of a much more efficient and highly scalable GemsFDTD code. Since these analyses show that there are still significant optimization opportunities at larger scales, further engineering improvements can be expected in future.

Acknowledgements John Baron of SGI helped identify some bottlenecks in the original benchmark using the MPInside profiling tool, which led the developers to replace `MPI_Sendrecv` with non-blocking communication. EPCC service staff provided valuable assistance to use their Cray XT4 system for the performance measurements. Part of this work has been performed under the HPC-Europa2 project (number 228398) with the support of the European Commission – Capacities Area – Research Infrastructures. Part of this work has been supported by the computing resources of the PDC, KTH, and the Swedish National Infrastructure for Computing (SNIC).

References

1. Gedney, S.D.: An anisotropic perfectly matched layer-absorbing medium for the truncation of FDTD lattices. IEEE Transactions on Antennas and Propagation 44(12), 1630–1639 (Dec 1996)
2. 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 (Apr 2010)
3. Jülich Supercomputing Centre, Germany: Scalasca toolset for scalable performance analysis of large-scale parallel applications (2010), http://www.scalasca.org/
4. Martin, T., Pettersson, L.: Dispersion compensation for Huygens' sources and far-zone transformation in FDTD. IEEE Transactions on Antennas and Propagation 48(4), 494–501 (Apr 2000)
5. Parallel & Scientific Computing Institute (PSCI), Sweden: GEMS: General ElectroMagnetic Solvers project (2005), http://www.psci.kth.se/Programs/GEMS/
6. Standard Performance Evaluation Corporation (SPEC), USA: SPEC MPI2007 benchmark suite (version 2.0) (2010), http://www.spec.org/mpi2007/
7. Szebenyi, Z., Wylie, B.J.N., Wolf, F.: SCALASCA parallel performance analyses of SPEC MPI2007 applications. In: Proc. 1st SPEC Int'l Performance Evaluation Workshop (SIPEW, Darmstadt, Germany). Lecture Notes in Computer Science, vol. 5119, pp. 99–123. Springer (June 2008)
8. Taflove, A., Hagness, S.C.: Computational Electrodynamics: The Finite-Difference Time-Domain Method. Artech House, 3rd edn. (2005)
9. University of Edinburgh HPCx, UK: HECToR: United Kingdom National Supercomputing Service (2010), http://www.hector.ac.uk/service/
10. Wylie, B.J.N., Böhme, D., Frings, W., Geimer, M., Mohr, B., Szebenyi, Z., Becker, D., Hermanns, M.A., Wolf, F.: Scalable performance analysis of large-scale parallel applications on Cray XT systems with Scalasca. In: Proc. 52nd CUG meeting (Edinburgh, Scotland). Cray User Group (May 2010)
11. Wylie, B.J.N., Böhme, D., Mohr, B., Szebenyi, Z., Wolf, F.: Performance analysis of Sweep3D on Blue Gene/P with the Scalasca toolset. In: Proc. 24th Int'l Parallel & Distributed Processing Symposium, Workshop on Large-Scale Parallel Processing (IPDPS–LSPP, Atlanta, GA, USA). IEEE Computer Society (Apr 2010)