Performance analysis and tuning of the XNS CFD solver on Blue Gene/L

Brian J. N. Wylie[1], Markus Geimer[1], Mike Nicolai[2], and Markus Probst[2]

[1] John von Neumann Institute for Computing (NIC),
Forschungszentrum Jülich GmbH, D-52425 Jülich, Germany
{b.wylie,m.geimer}@fz-juelich.de
[2] Chair for Computational Analysis of Technical Systems (CATS),
Centre for Computational Engineering Science (CCES),
RWTH Aachen University, D-52074 Aachen, Germany
{nicolai,probst}@cats.rwth-aachen.de

Abstract. The XNS computational fluid dynamics code was successfully running on Blue Gene/L, however, its scalability was unsatisfactory until the first Jülich Blue Gene/L Scaling Workshop provided an opportunity for the application developers and performance analysts to start working together. Investigation of solver performance pin-pointed a communication bottleneck that appeared with approximately 900 processes, and subsequent remediation allowed the application to continue scaling with a four-fold simulation performance improvement at 4,096 processes. This experience also validated the SCALASCA performance analysis toolset, when working with a complex application at large scale, and helped direct the development of more comprehensive analyses. Performance properties have now been incorporated to automatically quantify point-to-point synchronisation time and wait states in scan operations, both of which were significant for XNS on Blue Gene/L.
Keywords: performance analyses, scalability, application tuning.

1 Introduction

XNS is an academic computational fluid dynamics (CFD) code for effective simulations of unsteady fluid flows, including micro-structured liquids, in situations involving significant deformations of the computational domain. Simulations are based on finite-element techniques using stabilised formulations, unstructured three-dimensional meshes and iterative solution strategies [1]. Main and novel areas of XNS are: simulation of flows in the presence of rapidly translating or rotating boundaries, using the shear-slip mesh update method (SS-MUM); simulation of flows of micro-structured (in particular viscoelastic) liquids; and simulation of free-surface flows, using a space-time discretisation and staggered elevation-deformation-flow (EDF) approach. The parallel implementation is based on message-passing communication libraries, exploits mesh-partitioning techniques, and is portable across a wide range of computer architectures.

The XNS code, consisting of more than 32,000 lines of Fortran90 in 66 files, uses the EWD substrate library which fully encapsulates the use of BLAS and communication libraries, which is another 12,000 lines of mixed Fortran and C within 39 files. Although the MPI version of XNS was already ported and running on Blue Gene/L, scalability at that point was only acceptable up to 900 processes.

During early December of 2006, John von Neumann Institute for Computing (NIC) hosted the first Jülich Blue Gene/L Scaling Workshop [2], providing selected applicants an opportunity to scale their codes on the full Jülicher Blue Gene/L system (JUBL) with local NIC, IBM and Blue Gene Consortium support. JUBL is configured with 8,192 dual-core 700MHz PowerPC 440 compute nodes (each with 512MB of memory), 288 I/O nodes, and additional service and login nodes. A pair of the XNS application developers were thereby teamed with local performance analysts to investigate and resolve the application's scalability bottlenecks using the analysis tools available on the system.

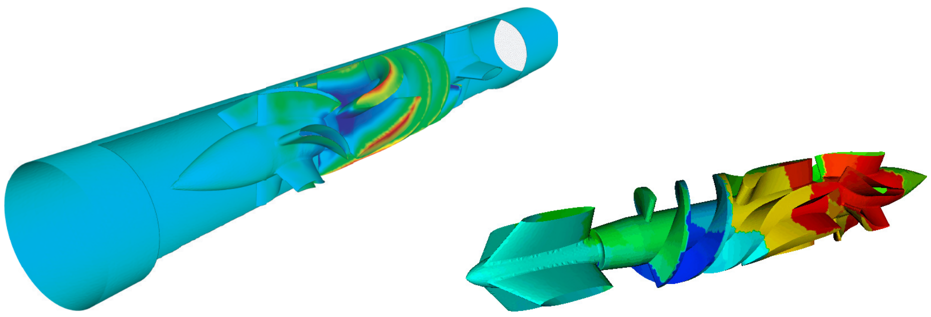
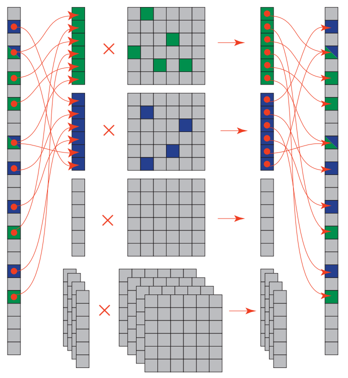

(a) Haemodynamic flow pressure distribution. (b) Partitioned finite-element mesh.

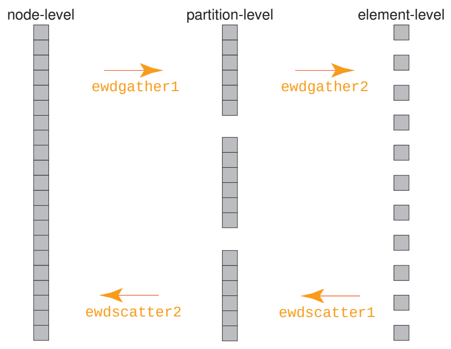
Fig. 1. DeBakey axial ventricular assist blood pump simulated with XNS.

Performance on Blue Gene/L was studied with a test-case consisting of a 3-dimensional space-time simulation of the MicroMed DeBakey axial ventricular assist blood pump (shown in Figure 1). Very high resolution simulation is required to accurately predict shear-stress levels and flow stagnation areas in an unsteady flow in such a complex geometry. The mesh for the pump consisted of 3,714,611 elements (connecting 1,261,386 nodes) which were divided by the METIS graph partitioner into element sets which form contiguous subdomains that are assigned to processes.

With each set of elements assigned to a single process, the nodes are then distributed in such a way that most nodes which are interior to a subdomain are assigned to the process which holds elements of the same subdomain. Nodes at a subdomain boundary are assigned to all processes sharing that boundary. The formation of element-level components of the system of equations proceeds fully in parallel, with all data related to a given element residing in the same process. Solution of that system of equations takes place within a GMRES iterative solver, and it is here that the bulk of inter-process communication occurs, with the element-based structures (stiffness matrices and local residuals) interacting



(a) Distributed vector-matrix multiplication.



(b) Interface between different levels.

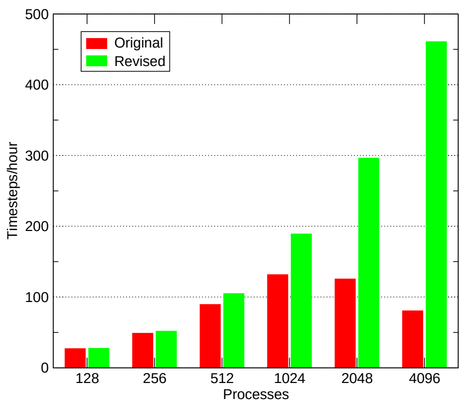
Fig. 2. Distributed sparse vector-matrix multiplications of global state vectors and partitioned matrices require data transfers taking the form of *scatter* and *gather* between node, partition and element levels.

with node-based structures (global residuals and increments). Figure 2(a) shows the required movement of data from element-level to node-level taking the form of a *scatter* and the reverse movement from node-level to element-level taking the form of a *gather*. These operations have two stages (Figure 2(b)): one local to the subdomain (and free of communication) and another at the surface of the subdomains (where communication is required). Four Newton-Raphson iterations are typically carried out in the solver per simulation timestep.

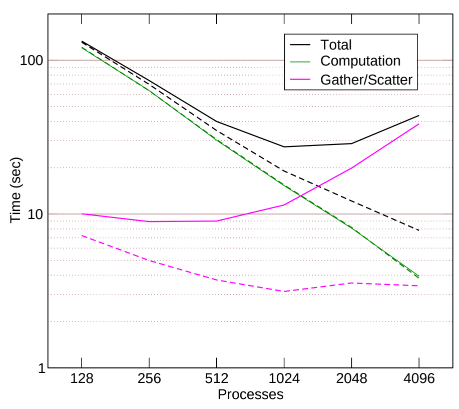
Time-consuming initialisation and data checkpointing (which are also highly variable due to file I/O) are excluded from measured performance reported by XNS in simulation time-steps per hour, originally peaking at around 130 timesteps/hour (Figure 3(a)). From comparison of simulation rates (and later analyses) for various numbers of timesteps, it could be determined that the first timestep's performance was representative of that of larger numbers of timesteps, allowing analysis to concentrate on simulations consisting of a single timestep.

2 XNS execution analysis

In addition to internal timing and reporting of the simulation timestep rate, as charted in Figure 3(a), the XNS code includes a breakdown of the performance of its primary components, namely formation of matrix left and right hand sides, GMRES solver, matrix-vector product, gather and scatter operations, etc. From a graph of these reported component costs, summarised in Figure 3(b), it was clear that the primarily computational components scaled well to larger numbers of processes, however, the gather and scatter operations used to transfer values between the node, partition and element levels became increasingly expensive. This behaviour is common to distributed-memory parallelisations of fixed-size problems, where the computational work per partition diminishes while the cost



(a) Overall simulation performance.



(b) Breakdown by solver component.

Fig. 3. Comparison of original and subsequently revised XNS solver performance with DeBakey axial pump on various partition sizes of JUBL Blue Gene/L. (a) Originally unacceptable large-scale performance is improved significantly to perform over 460 timesteps/hour. (b) Breakdown of the solver component costs/timestep shows good scalability of the primarily computational components and significant improvement to Gather/Scatter scalability (original: *solid lines*, revised: *dashed lines*).

of exchanging data across partition boundaries grows with increasing numbers of processors. Understanding and optimising the communication, particularly at large processor counts, is therefore essential for effective scaling.

2.1 Profile generation and analysis

Several MPI profiling tools were available on JUBL, working in similar fashion and providing broadly equivalent analyses. [3, 4] For example, after re-linking the XNS code with an instrumented library implementing the standard PMPI profiling interface, MPI operation characteristics were accumulated for each process during a run of the instrumented XNS executable, and these profiles were collated and presented in an analysis report upon execution completion.

From such analyses of XNS, the total time in MPI communication could be seen growing to ultimately dominate simulations, and the bulk of this time was due to rapidly growing numbers of `MPI_Sendrecv` operations within the core simulation timestep/iteration loop `ewdgather1i` and `ewdscatter2i` routines. (`MPI_Sendrecv` provides an optimised combination of `MPI_Send` and `MPI_Recv`.)

Closer examination of the profile summaries showed that each process rank makes the same number of `MPI_Sendrecv` calls in these functions, and the associated times are also very similar. Furthermore, message sizes vary considerably, with some messages of zero bytes: i.e., without message data communication. Since these operations are employed to exchange boundary elements between partitions, they could be expected to vary from process to process, however, it appeared that an exchange was done for every possible combination.

Although a zero-sized point-to-point message transfer can be useful for loose pairwise synchronisation (or coordination), it can also indicate unnecessary message traffic when there is no actual data to transfer. Unfortunately, the available profiling tools were unable to determine what proportion of `MPI_Sendrecv` operations consisted of zero-sized messages and their associated cost. This investigation could be pursued, however, via traces of the communication routines.

In addition, the profiles showed that a considerable amount of time was spent in calls to `MPI_Scan`, however, to determine whether this prefix reduction operation incurs any wait states would also require more elaborate trace analysis.

2.2 Trace collection and analysis

Several trace collection libraries and analysis tools were also available on JUBL, generally exclusively for tracing MPI operations. An early release (v0.5) of the SCALASCA toolset [5] had been installed for the workshop, offering tracing of MPI operations, application functions and user-specified annotations. Of particular note, execution traces from each process are unified and analysed in parallel, following the measurement and using the same computer partition. Although it had already demonstrated scalable trace collection and analysis of short benchmarks, this was an opportunity to apply it to a complex application code.

The sheer number of MPI communication operations employed by XNS each timestep was itself a significant test of SCALASCA, quickly filling trace buffers during measurement and requiring efficient internal event management during analysis/replay. Trace measurement was therefore reduced to a single simulation timestep, and analysis similarly focused to avoid the uninteresting initialisation phase (which includes file I/O that is highly variable from run to run).

Initial SCALASCA tracing simply involved re-linking the XNS code with a measurement tracing library. In this configuration, without additional instrumentation of user functions/regions, only MPI operations were traced and subsequently analysed. Traces that can be completely stored in memory avoid highly perturbative trace buffer flushing to file during measurement, and specification of appropriately-sized trace buffers was facilitated by the memory (maximum heap) usage reported by the profiling tools: fortunately, XNS memory requirements diminish with increasing numbers of processes, allowing most of the available compute node memory to be used for trace buffers.

Automatic function instrumentation is a feature of the IBM XL compilers which SCALASCA can use to track the call-path context of MPI operations and measure the time spent in non-communication functions. Unfortunately, when all functions are instrumented, measurements are often compromised by frequent calls to small functions that have a negligible contribution on overall performance but disproportionate impact on trace size and measurement perturbation. When the entire XNS application (including the EWD library) was instrumented in this fashion, ten such routines were identified that produced more events than `MPI_Sendrecv`. These routines were then specified for exclusion from measurement, resulting in traces where 94% of traced events were MPI operations (and more than 92% were `MPI_Sendrecv`).

The resulting trace analysis revealed a rich call-tree, however, navigation and analysis were encumbered by the complexity of the key nodes, often consisting of more than twenty branches at several depths. (Only subroutine names are used to distinguish successor call-paths, such that calls from different locations within a routine are aggregated.) It was therefore helpful to incorporate user-region annotation instrumentation in the XNS `hypo` routine to distinguish initialisation, simulation timestep and solver iteration loops, and finalisation phases.

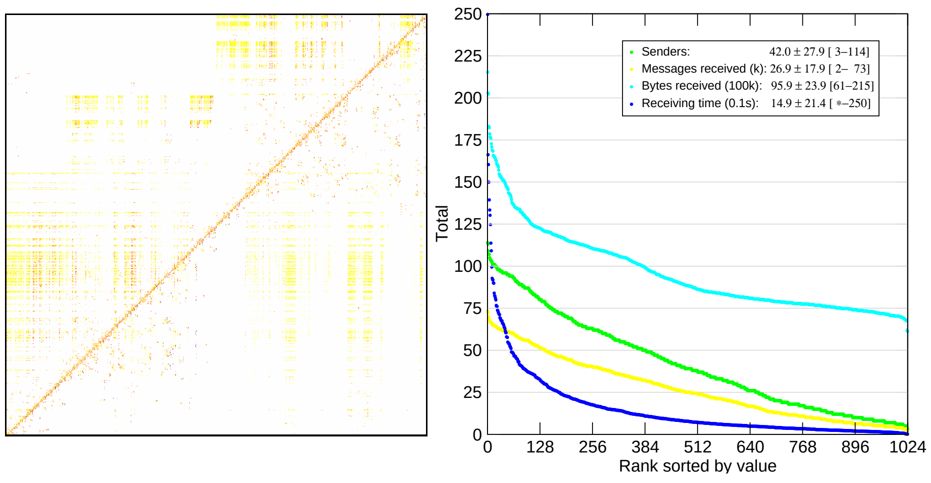
Trace measurement and analysis of the instrumented original XNS code for a single simulation timestep at a range of scales confirmed the analysis provided by the MPI profilers. With 2,048 processes, the main XNS simulation timestep loop was dilated by 15% during trace collection (compared to the uninstrumented version), producing over 23,000 million traced events which were then automatically analysed in 18 minutes.

Figure 5 (back) shows how the SCALASCA analysis report explorer highlit the most time-consuming call-paths to the `MPI_Sendrecv` operations in the `ewdscatter2` and `ewdgather1` routines of the timestep loop and presented the individual process times with the hardware topology of Blue Gene/L: MPI communication times were very balanced across processes, as evident from the 3.4% variation and uniform colouring.

As message size is logged as an attribute with each message in the trace, the number of zero-sized messages could also be determined and was found to grow rapidly with the number of processes employed (where partitions are correspondingly smaller and have fewer connections).

SCALASCA trace analysis was therefore customised to generate a report of the number of bytes received and receiving times for each sender/receiver combination and communication distribution maps were produced (e.g., Figure 4(a)). This analysis for 1,024 processes revealed that 96% of pairs had no data to exchange, and the trend makes this progressively worse for larger process configurations. Statistical analysis of the transfer data for non-zero-sized messages (Figure 4(b)) determined that *on average* each receiver rank takes 1.49 seconds to receive 9.6MB in 27,000 messages from 42 separate senders, however, there is a huge variation with maximal values typically three times the mean and 25 seconds receiving time for the rank that takes the longest.

Initial SCALASCA analyses didn't distinguish communication and synchronisation times for point-to-point messages, reporting only *Point-to-point communication time*. Incorporating a new metric for (pure) *Point-to-point synchronisation time*, for sends and receives of zero-sized messages, quantifies the very significant cost of these potentially redundant operations (Figure 5 (back)). For individual sends and receives, this was straightforward, however, the dual-nature of `MPI_Sendrecv` provides cases where only one of its send and receive parts are zero-sized and it is not possible to separate the respective costs of each part without MPI internal events [6]. After experimentation with various alternatives, it was found that only situations where the bytes sent and received are both zero could be reliably accounted as *Point-to-point synchronisation time*. Even though this underestimates the actual synchronisation cost, it ensures that the

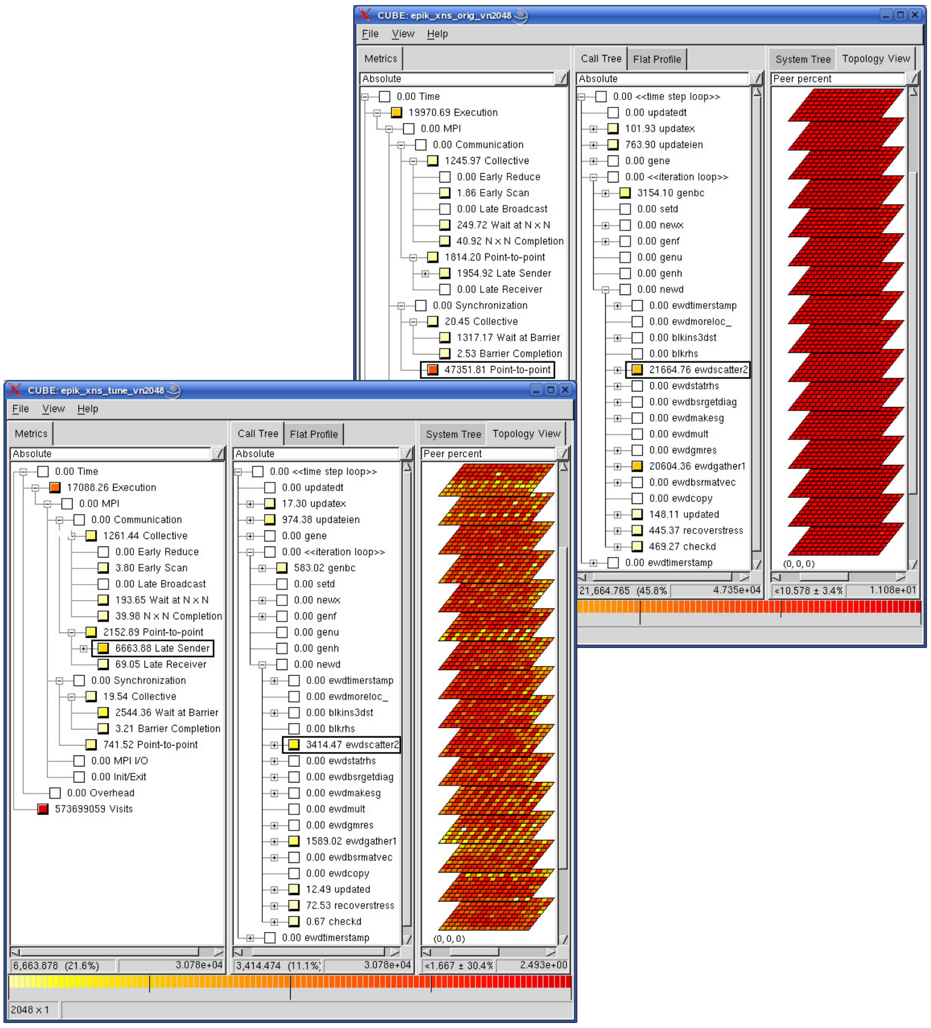


(a) Communication time map (b) Receiver statistics

Fig. 4. Message analysis by sender/receiver for `ewdscatter` with 1,024 MPI processes. (a) The map is a matrix of message transfer times for each sender/receiver combination, coloured with a logarithmic scale: white implies no message transfer. (b) Message statistics aggregated per receiver for number of senders, messages, bytes transfered and receiving times (and then sorted by total value) highlight the considerable variation.

associated communication cost remains consistent when redundant zero-sized operations are eliminated.

Furthermore, the solver time-step loop requires around 1,500 seconds of *Collective communication time*, 53% of which is due to 11 global `MPI_Scan` operations, and almost all of it (789s) is isolated to a single `MPI_Scan` in `updateien`. Quantifying the time a scan operation on process rank n had to wait before all of its communication partners (ranks $0, \ldots, n-1$) also entered the `MPI_Scan`, another extension to the trace analysis was the implementation of a new *Early Scan* pattern. As seen in Figure 5, the aggregate *Early Scan* time is negligible for XNS, indicating that each `MPI_Scan` is called when the processes are well balanced. Further investigation of the traces determined that no rank ever exited the `MPI_Scan` before all had entered, and this generally results in longer waits for lower process ranks. Such a collective synchronisation on exit appears to be an unnecessary artifact of the MPI implementation on Blue Gene/L. While it would be desirable to define a *Scan Completion* pattern just for the cost of delayed exits from `MPI_Scan`, this requires a measure of the local scan processing time, which could only be estimated in the absence of explicit MPI internal events (e.g., via extensions to [6]). For impacted applications various remedies could be pursued: the entire MPI library or simply the implementation of `MPI_Scan` could be exchanged, or the application could try to adapt to the behaviour of the library `MPI_Scan` by redistributing or rescheduling parts of its preceding computation accordingly.


```
CUBE: epik_xns_orig_vn2048
File  View  Help
Metrics | Call Tree | Flat Profile | System Tree | Topology View
Absolute | Absolute | Peer percent

0.00 Time                              0.00 <<time step loop>>
  19970.69 Execution                     0.00 updatedt
    0.00 MPI                             101.93 updatex
      0.00 Communication                 763.90 updateien
        1245.97 Collective               0.00 gene
          0.00 Early Reduce              0.00 <<iteration loop>>
          1.86 Early Scan                  3154.10 genbc
          0.00 Late Broadcast              0.00 setd
          249.72 Wait at N x N             0.00 newx
          40.92 N x N Completion           0.00 genf
        1814.20 Point-to-point             0.00 genu
          1954.92 Late Sender              0.00 genh
          0.00 Late Receiver               0.00 newd
      0.00 Synchronization                   0.00 ewdtimerstamp
        20.45 Collective                     0.00 ewdmoreloc_
          1317.17 Wait at Barrier            0.00 blkins3dst
          2.53 Barrier Completion            0.00 blkrhs
        47351.81 Point-to-point              21664.76 ewdscatter2
                                             0.00 ewdstatrhs
                                             0.00 ewdbsrgetdiag
                                             0.00 ewdmakesg
                                             0.00 ewdmult
                                             0.00 ewdgmres
                                             20604.36 ewdgather1
                                             0.00 ewdbsrmatvec
                                             0.00 ewdcopy
                                             148.11 updated
                                             445.37 recoverstress
                                             469.27 checkd
                                         0.00 ewdtimerstamp
                                                                   (0, 0, 0)
21,664.765  (45.8%)   4.735e+04          ≤10.578 ± 3.4%   1.108e+01

CUBE: epik_xns_tune_vn2048
File  View  Help
Metrics | Call Tree | Flat Profile | System Tree | Topology View
Absolute | Absolute | Peer percent

0.00 Time                              0.00 <<time step loop>>
  17088.26 Execution                     0.00 updatedt
    0.00 MPI                             17.30 updatex
      0.00 Communication                 974.38 updateien
        1261.44 Collective               0.00 gene
          0.00 Early Reduce              0.00 <<iteration loop>>
          3.80 Early Scan                  583.02 genbc
          0.00 Late Broadcast              0.00 setd
          193.65 Wait at N x N             0.00 newx
          39.98 N x N Completion           0.00 genf
        2152.89 Point-to-point             0.00 genu
          6663.88 Late Sender              0.00 genh
          69.05 Late Receiver              0.00 newd
      0.00 Synchronization                   0.00 ewdtimerstamp
        19.54 Collective                     0.00 ewdmoreloc_
          2544.36 Wait at Barrier            0.00 blkins3dst
          3.21 Barrier Completion            0.00 blkrhs
        741.52 Point-to-point                3414.47 ewdscatter2
      0.00 MPI I/O                           0.00 ewdstatrhs
      0.00 Init/Exit                         0.00 ewdbsrgetdiag
  0.00 Overhead                              0.00 ewdmakesg
573699059 Visits                             0.00 ewdmult
                                             0.00 ewdgmres
                                             1589.02 ewdgather1
                                             0.00 ewdbsrmatvec
                                             0.00 ewdcopy
                                             12.49 updated
                                             72.53 recoverstress
                                             0.67 checkd
                                         0.00 ewdtimerstamp
                                                                   (0, 0, 0)
6,663.878  (21.6%)  3.078e+04   3,414.474  (11.1%)  3.078e+04   ≤1.667 ± 30.4%  2.493e+00
2048 x 1
```

Fig. 5. SCALASCA performance analysis reports for original (*back*) and revised (*front*) versions of the XNS simulation on JUBL Blue Gene/L with 2,048 processes. A metric chosen from the metrics hierarchy (*left panes*) is shown for call-paths of the solver time-step loop (*central panes*) and its distribution per process for the selected **`ewdscatter2`** bottleneck (*right panes*): navigation to the most significant metric values is aided by boxes colour-coded according to each pane's specified mode and range scale (*bottom*). In each tree, collapsed nodes present inclusive metrics while expanded nodes present exclusive metrics. 47,352s of *Point-to-point synchronization time* (45.8% in **`ewdscatter2`** and 43.5% in **`ewdgather1`**) is reduced to only 742s (in routines which were not modified) by switching from **`MPI_Sendrecv`** to separate **`MPI_Send`** and **`MPI_Recv`** operations where zero-sized transfers are eliminated. *Point-to-point communication time*, particularly *Late Sender* situations, and *Wait at Barrier* are both significantly increased due to resulting communication load imbalance, manifest in the variation by process rank, however, overall communication time is substantially improved in the solver.

3 Modification of ewdgather and ewdscatter

The insights provided by the preceding analyses of XNS suggested splitting the MPI_Sendrecv operations used within the ewdgather1 and ewdscatter2 routines into separate MPI_Send and MPI_Recv operations which are only called when actual message data needs to be transfered. Since a static partitioning of the mesh is employed, the number of elements linking each partition is also known in advance (by potential senders and receivers), and when there are no links there is no data to transfer.

The graph comparing original and revised XNS simulation timestep rates in Figure 3(a) shows that below 1,024 processes, elimination of these zero-sized messages had little effect on the performance, which could be expected since the communication matrix remains relatively dense at this scale. Performance improved dramatically for larger configurations though, resulting in a more than four-fold overall performance improvement with 4,096 processes to over 460 timesteps/hour. Further scalability is also promising, however, lack of suitably partitioned datasets for larger numbers of processes has unfortunately prevented pursuing this investigation to date.

Elimination of zero-sized messages reduced the size of trace files collected from the new version and similarly improved trace analysis performance. Comparing analyses from the original and modified versions (Figure 5) shows the significant improvement in MPI communication time and the contributions from ewdgather1 and ewdscatter2. *Point-to-point synchronisation time* decreased more than 98%, for a substantial overall performance improvement, however, the new versions of these functions show significant imbalance by process. This manifests in increased time in other parts of the solver, particularly *Wait at Barrier* and *Late Sender* situations for *Point-to-point communication* (i.e., where the receiver was blocked waiting for a sender to initiate a message transfer).

Further modifications of XNS to use asynchronous (non-blocking) message transfers within ewdscatter2 and ewdgather1 were investigated, but showed no additional performance improvement. This may be due to the small amount of computation available for overlap with communication within these routines. Although there is potentially more computation in the rest of the iteration loop, Figure 3(b) shows that it diminishes rapidly as the number of processes increase.

4 Conclusion

The first Jülich Blue Gene/L Scaling Workshop was a catalyst for successful collaborations between application and analysis tools developers. Analysis of the execution performance of the XNS application with more than one thousand processes was crucial in the location of adverse characteristics that developed at scale in some critical communication routines. Straightforward modification of these routines significantly improved XNS simulation performance, and enabled scaling to processor configurations four times larger than previously practical.

Further optimisations with potentially significant performance benefits are currently being evaluated, such as improved mesh partitioning and mapping of mesh partitions onto the Blue Gene/L topology. Communication distribution maps summarising message transfers between sender/receiver combinations will be important for this purpose, and provision of these by the SCALASCA toolset is being investigated.

While the SCALASCA toolset demonstrated that it could automatically quantify and help isolate common performance problems in large-scale complex applications, various aspects could be identified for improvement. Automated trace analysis was subsequently extended to quantify inefficiencies in `MPI_Scan` and `MPI_Sendrecv`, the latter being found to be responsible for costly and unnecessary point-to-point synchronisations. Synchronisation and communication costs are currently based on heuristics that ensure analysis consistency, yet which might be determined more accurately in future.

The profiling tools available on Blue Gene/L were a convenient starting point for performance analysis, however, they provided limited insight into synchronisation costs and imbalance. Message statistics for communication and synchronisation operations can be calculated from trace analysis or accumulated during measurement, and these capabilities are now being incorporated in the SCALASCA toolset. By integrating runtime summarisation and tracing capabilities, convenience of use is being pursued particularly for measurement configuration and selective event tracing.

The open-source SCALASCA toolset is freely available for download [7].

Acknowledgements We would like to thank the sponsors, organisers and co-participants of the Jülich Blue Gene/L Scaling Workshop for the valuable opportunity to benefit from their assistance, advice and productive atmosphere. We also thank the reviewers for their constructive suggestions.

References

1. Behr, M., Arora, D., Coronado, O., Pasquali, M.: Models and finite element techniques for blood flow simulation. Int'l J. Computational Fluid Dynamics 20, (2006) 175–181
2. Frings, W., Hermanns, M.-A., Mohr, B., Orth, B. (eds): Jülich Blue Gene/L Scaling Workshop, December 2006. Forschungszentrum Jülich ZAM-IB-2007-01. `http://www.fz-juelich.de/zam/bgl-sws06/`
3. IBM Advanced Computing Technology Center: High Performance Computing Toolkit. `http://www.research.ibm.com/actc/`
4. Vetter, J., Chambreau, C.: MPIP — lightweight, scalable MPI profiling (2005) `http://www.llnl.gov/CASC/mpip/`
5. Geimer, M., Wolf, F., Wylie, B. J. N., Mohr, B.: Scalable parallel trace-based performance analysis. Proc. 13th European PVM/MPI User's Group Meeting (Bonn, Germany), Lecture Notes in Computer Science 4192, Springer (2006) 303–312
6. Jones, T., *et al*: MPI PERUSE: A performance revealing extensions interface to MPI. `http://www.mpi-peruse.org/`
7. Forschungszentrum Jülich GmbH: SCALASCA: Scalable performance analysis of large-scale parallel applications. `http://www.scalasca.org/`