Exascale potholes for HPC: Execution performance and variability analysis of the flagship application code HemeLB

Brian J. N. Wylie
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
Jülich, Germany
b.wylie@fz-juelich.de
0000-0003-2770-2443

***Abstract*—Performance measurement and analysis of parallel applications is often challenging, despite many excellent commercial and open-source tools being available. Currently envisaged exascale computer systems exacerbate matters by requiring extremely high scalability to effectively exploit millions of processor cores. Unfortunately, significant application execution performance variability arising from increasingly complex interactions between hardware and system software makes this situation much more difficult for application developers and performance analysts alike.**

This work considers the performance assessment of the HemeLB exascale flagship application code from the EU HPC Centre of Excellence (CoE) for Computational Biomedicine (CompBioMed) running on the SuperMUC-NG Tier-0 leadership system, using the methodology of the Performance Optimisation and Productivity (POP) CoE. Although 80% scaling efficiency is maintained to over 100,000 MPI processes, disappointing initial performance with more processes and corresponding poor strong scaling was identified to originate from the same few compute nodes in multiple runs, which later system diagnostic checks found had faulty DIMMs and lacklustre performance. Excluding these compute nodes from subsequent runs improved performance of executions with over 300,000 MPI processes by a factor of five, resulting in 190× speed-up compared to 864 MPI processes. While communication efficiency remains very good up to the largest scale, parallel efficiency is primarily limited by load balance found to be largely due to core-to-core and run-to-run variability from excessive stalls for memory accesses, that affect many HPC systems with Intel Xeon Scalable processors. The POP methodology for this performance diagnosis is demonstrated via a detailed exposition with widely deployed 'standard' measurement and analysis tools.

***Index Terms*—execution performance measurement/analysis; variability detection; parallel applications; POP; HPC; exascale**

I. Introduction

Current HPC computer systems have many compute nodes comprising multi-socket multi-core hyperthreaded vectorised turbo-boosted power-throttled CPU processors often with attached accelerators (GPUs and sometimes FPGAs), with multiple levels of caches for accesses to local and NUMA memory of limited size, connected via hierarchical networks with multiple switches. Whereas homogeneous systems were once common, as heterogeneity is incorporated in all aspects to improve energy efficiency, they are now the exception for the largest (and exascale-candidate) systems [1]. Highly-scalable HPC applications running on these systems, however, are mostly still programmed with explicit message-passing using MPI between compute nodes (or sockets) and multi-threading using OpenMP within shared-memory domains [2], while exploitation of accelerators covers a wide gamut from standardised OpenMP target offload down to GPU-specific languages such as CUDA and ROCm [3].

Application file I/O is a common scalability bottleneck, while shared parallel filesystems such as GPFS and Lustre typically introduce substantial execution time variability between and within compute jobs. Other major sources of variability arise from system daemon processes that periodically take CPU resources [4], as well as turbo-boost and power throttling that dynamically adjust processing speed of CPUs. All of these sources of run-to-run, node-to-node, core-to-core and iteration-to-iteration execution variability typically don't affect correctness[1], but can have substantial impact on performance. Since the variability is so complex, AI data analytics techniques are proposed to facilitate their analysis [5].

The challenges for application developers presented by this relentlessly increasing complexity of HPC computer systems (and the applications themselves) are troublesome, as efficient scalable codes are predicated on a clear understanding of current and future application requirements (including hardware and software roadmaps) and often take large teams many years. Many nations have recognised this and provide targetted funding to assist: for application developers in Europe, in addition to PRACE/SHAPE[2] (specifically supporting SMEs to parallelise their existing application codes) and related programmes, Horizon2020/EuroHPC funds a variety of HPC Centres of Excellence (CoEs)[3].

Most of the HPC CoEs address specific application domains, however, the transversal Performance Optimisation and

[1] otherwise expensive fault-detection and recovery beyond occassional checkpoint/restart is required

[2] http://www.prace-ri.eu/hpc-access/shape-programme/

[3] https://www.focus-coe.eu/index.php/centres-of-excellence-in-hpc-applications/

Productivity (POP) CoE[4] supports all fields of HPC, with emphasis on supporting the HPC CoEs as well as European industry (and particularly SMEs since they tend not to have the necessary resources and skills in house). POP provides training in parallel performance analysis, using the partners' own open-source tools as well as third-party tools, in addition to parallel performance assessment and proof-of-concept prototype services which are free-of-charge for application developers from European institutions.

This paper examines a POP performance assessment of an HPC CoE exascale flagship application code on one of the European Tier-0 leadership HPC systems at its full scale. The basic POP analysis methodology is demonstrated with the "pretty standard" Scalasca toolset, which identifies compute nodes with severely degraded performance that need to be avoided and then investigates performance variability in widely-deployed commodity Intel Xeon Scalable processors. This variability is subsequently found to substantially impact the achieved performance of this code and several others on many top-tier computer systems, motivating the need for widespread adoption of such methodology and tools to identify and ultimately circumvent these issues.

II. Methodology

A. Subject application HemeLB

The open-source HemeLB software[5] developed by University College London and others within the EU HPC CoE for Computational Biomedicine (CompBioMed[6]) is their flagship solver for high-performance parallel lattice-Boltzmann simulations of large-scale three-dimensional hæmodynamic flow in vascular geometries. It supports a range of collision kernels and boundary conditions, and is optimised for sparse, patient-specific geometries. HemeLB has traditionally been used to model cerebral bloodflow, and is now being applied to simulating the fully-coupled human arterial and venous trees with high fidelity [6].

HemeLB has been previously demonstrated to scale exceptionally well up to 100,000 cores [7]. Furthermore, the code was recently found to scale with 80% efficiency on 288,000 AMD 6276 Bulldozer-based Interlagos processor cores of 18,000 Cray XE nodes of NCSA Blue Waters[7]. More recent development focusses on SuperMUC-NG which is based on newer processors with more cores.

B. Execution environment

The Lenovo ThinkSystem SD650 supercomputer SuperMUC-NG[8] at Leibniz-Rechenzentrum (LRZ, Germany) comprises 6480 compute nodes with dual 24-core Intel Xeon Platinum 8174 @ 3.10 GHz ('Skylake') processors [8]. 144 'fat' compute nodes each have 768 GB memory, compared to only 96 GB memory for the remaining 6336 'thin' compute nodes bundled into eight domains (known as 'islands'). The internal interconnect is an Intel OmniPath network, with a fat-tree topology within islands and 1:4 pruned connection between islands. A high-performance 50 TB parallel filesystem is provided by IBM Spectrum Scale (GPFS), with SUSE Linux Enterprise Server (SLES) 12 SP3 operating system.

HemeLB was built with Intel 19.0.4.243 compilers and MPI library. It was configured to use MPI-3 shared-memory windows within each compute node to reduce memory requirements when loading the initial lattice data. For scalability testing, a cerebral arterial circle of Willis geometry dataset of 21.15 GiB was used (corresponding to a lattice spacing of approximately 6.4 microns) [9]. This comprises 1,138,236,832 blocks ($1376\times1087\times761$), of which 20,740,240 are non-empty and there are a total of 10,154,448,502 lattice sites. After reading and distributing this dataset, the time to simulate blood flow for 5000 lattice time steps (without writing intermediate or final state) was recorded for this strong scaling benchmark on SuperMUC-NG. 48 MPI processes were executed on each compute node (i.e., one per core, not using additional hardware threads per core) with processes bound to cores and socket-local memory.[9]

Memory requirements as reported by the HemeLB code are shown in Figure 1(a). With 768 GiB 'fat' compute nodes, executions ran with 864 to 6,144 MPI processes (on 18 to 128 compute nodes), whereas executions with 12,288 and more MPI processes ran on regular 'thin' compute nodes (with 96 GB memory). Requiring more then 2 GiB per process for executions with 9,216 MPI processes on 192 compute nodes, this configuration was not possible on SuperMUC-NG.[10]

Generally only a single execution was done at each scale, during regular operation of SuperMUC-NG, apart from the very largest runs requiring more than half of the total compute nodes which could only be done in a special dedicated session after system maintenance. Since the initial runs requiring more than 3,072 compute nodes repeatedly performed poorly, measurements were taken and analysed to identify the cause, such that runs could be done which delivered performance in line with expectations.

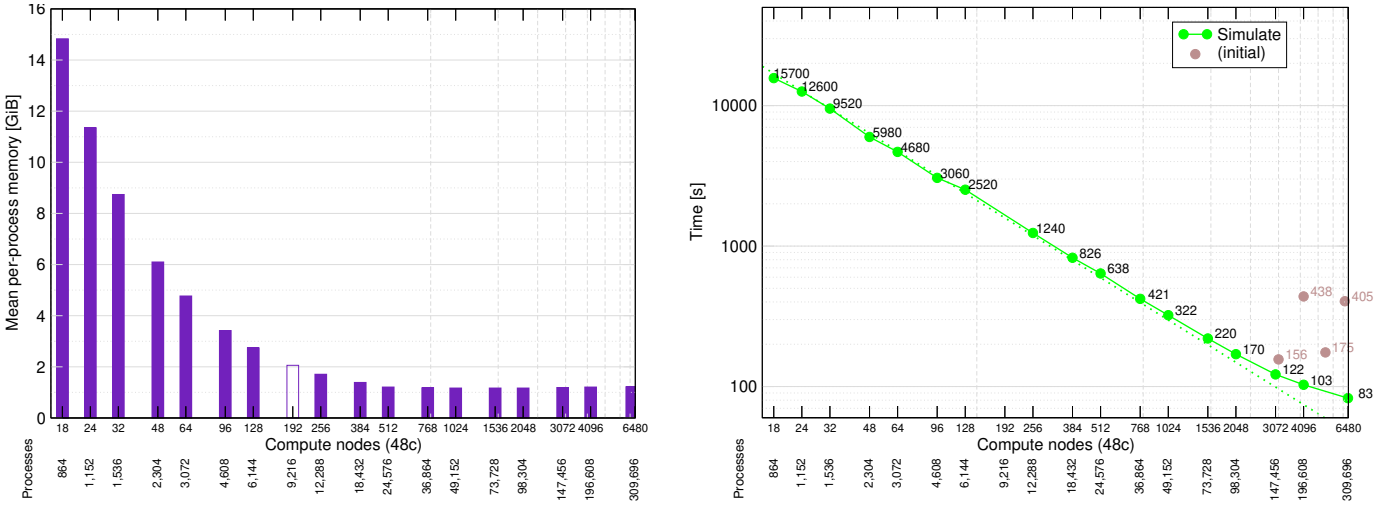
C. Scalability of simulation time and efficiency

Simulation time for different numbers of compute nodes is plotted in Figure 1(b), along with a dotted line representing perfect linear scaling. Simulation time speedup relative to the smallest execution configuration (18 compute nodes) is more than 190-fold obtained for 360-fold increase in compute nodes, with more than 80% scaling efficiency to over 100,000 processes.

With 792 compute nodes bundled within island domains, and islands connected via an additional switch, it is notable that no significant simulation performance advantage was observed when inter-domain switches were avoided or

[4]https://www.pop-coe.eu/
[5]http://www.hemelb.org/
[6]http://www.compbiomed.eu/
[7]https://bluewaters.ncsa.illinois.edu/
[8]https://doku.lrz.de/display/PUBLIC/SuperMUC-NG

[9]SLURM: ntasks-per-node=48, cpus-per-task=1, cpu-bind=cores, mem-bind=local
[10]but done instead on JUWELS Cluster at JSC, as reported later



(a) Memory requirements

(b) Execution time

Fig. 1. Scaling of HemeLB memory requirements and simulation time for 5000 lattice update steps with coW-6.4um.gmy geometry dataset on SuperMUC-NG. Execution of 9216 processes on 192 compute nodes is not possible due to insufficient nodes with adequate memory. Initial executions on the dedicated system performed much less well than expected (brown datapoints compared to dotted line), requiring poorly-performing compute nodes to be explicitly excluded for satisfactory executions.

reduced. Small numbers of failed compute nodes throughout SuperMUC-NG can therefore conveniently be avoided when allowing full flexibility in allocating partitions.

HemeLB executions have been audited on different computer systems by the POP CoE. These performance assessments using a methodology [10] based on measurements taken with the highly scalable open-source Scalasca/Score-P toolset [11] found very good computation scaling and communication efficiencies, while identifying memory consumption and load balance as key issues.

Score-P was used to prepare a HemeLB executable where (by default) application routines are instrumented by the compiler and special measurement libraries are linked interposing on MPI library routines. Initial measurements suffered from excessive overheads due to very frequent executions of `MPI_Comm_size`, `MPI_Comm_rank` and lots of small C++ methods (particularly from the standard libraries). A custom installation of Score-P was therefore made to avoid instrumentation of the uninteresting MPI routines, and the Intel compiler directed to only instrument key application routines specified via a file listing their signatures. While this brought measurement dilation down to an acceptable level of around 3%, execution tracing was still prohibitive due to the huge amount of MPI operations during HemeLB initialisation. The application code was therefore manually annotated with directives to pause recording when executing the initialisation phase, allowing detailed measurement and analysis of the subsequent simulation phase.

Scalasca analysis reports can be interactively explored using the CUBE GUI. Using an execution with 13824 MPI processes as an example, Figure 2 shows efficiency metrics calculated by CUBE for the selected callpath, in this case `RunSimulation` where 32% of execution time is spent: most of the execution time is initialisation, captured here as `MEASUREMENT OFF`.

CUBE presents the detailed measurements and analyses from Scalasca/Score-P in various ways. A three-pane presentation of metrics, callpaths and processes/threads in hierarchical trees is shown in Figure 2 — where selections determine values shown in panels from left to the right, and expanding tree nodes reveals the next level of internal detail — to naturally support exploration of huge amounts of execution performance data.

D. Localisation of degraded performance

For the initial measurement with 300,000 MPI processes the initial CUBE assessment of execution efficiency shown in Figure 3 already highlit very poor load balance of 0.49 (whereas communication efficiency remained a very good 0.97). This was confirmed by examining the computation time per process distribution statistics, whereas the number of instructions completed (from the `PAPI_TOT_INS` counter captured in the measurement) showed very little variation confirming that all processes did roughly the same amount of useful computation.

However, another hardware counter metric for resource stall cycles (`PAPI_RES_STL`) shown in Figure 4 provided insight into the imbalance problem. On one compute node, all 24 of the MPI processes with odd-numbered ranks in the `MPI_COMM_WORLD` global communicator suffered 12 times the mean number of resource stall cycles. Closer examination also revealed a second compute node with elevated resource stall cycles for its even-numbered MPI ranks compared with all of is peer processes. These same two compute nodes were also the culprits in the other measurements with degraded performance, and after explicitly excluding them from sub-

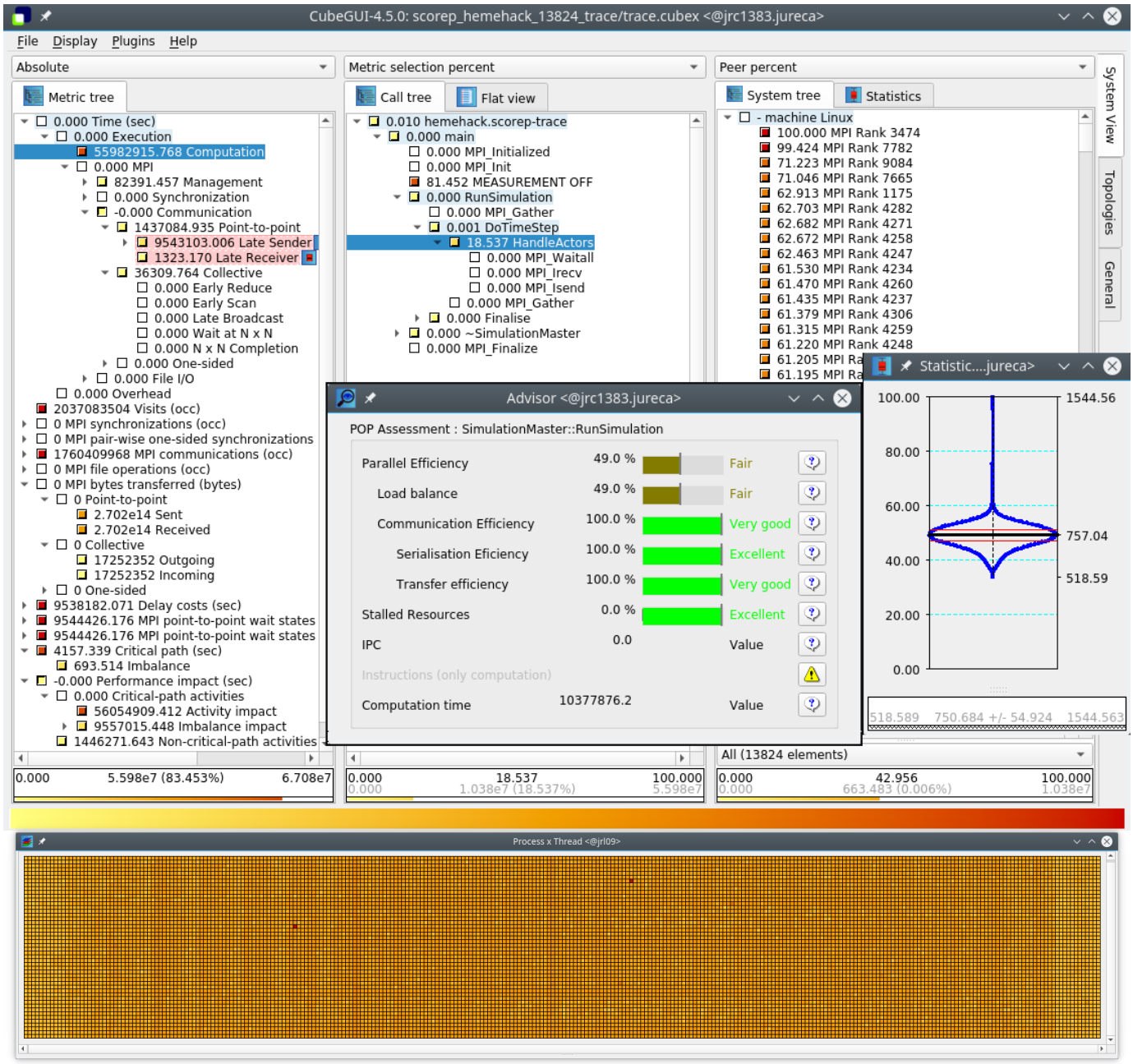


Fig. 2. Scalasca/CUBE analysis report explorer presentation of HemeLB execution on SuperMUC-NG with 13,824 MPI processes. From the hierarchy of metrics in the left panel, which includes metrics determined from analysis of the execution trace, time exclusively for computation has been selected, and this metric shown in panels to the right. From the call-tree hierarchy in the middle panel, the `HandleActors` part of `RunSimulation` has been selected, and computation time exclusively for this callpath is presented in the right panel for each MPI process. The times for each MPI process have been sorted and shown as percentages of that with the largest value, rank 3474 in MPI_COMM_WORLD. Next to each numeric value in the trees is a small box coloured from white through yellow and orange to red according to its percentage of the total metric value to facilitate identification of those which are most significant. Below the main window is an identically coloured topology map, where each column corresponds to the 48 processes (increasing top to bottom) on a compute node (increasing to the right), from which processes 3474 and 7782 stand out from the others with their much higher computation times. Overlaid on the main window are two additionally detached tabs: one showing the distribution of computation times of the processes, and the other an assessment of `RunSimulation` execution identifying load balance efficiency of only 0.49 as the critical factor.

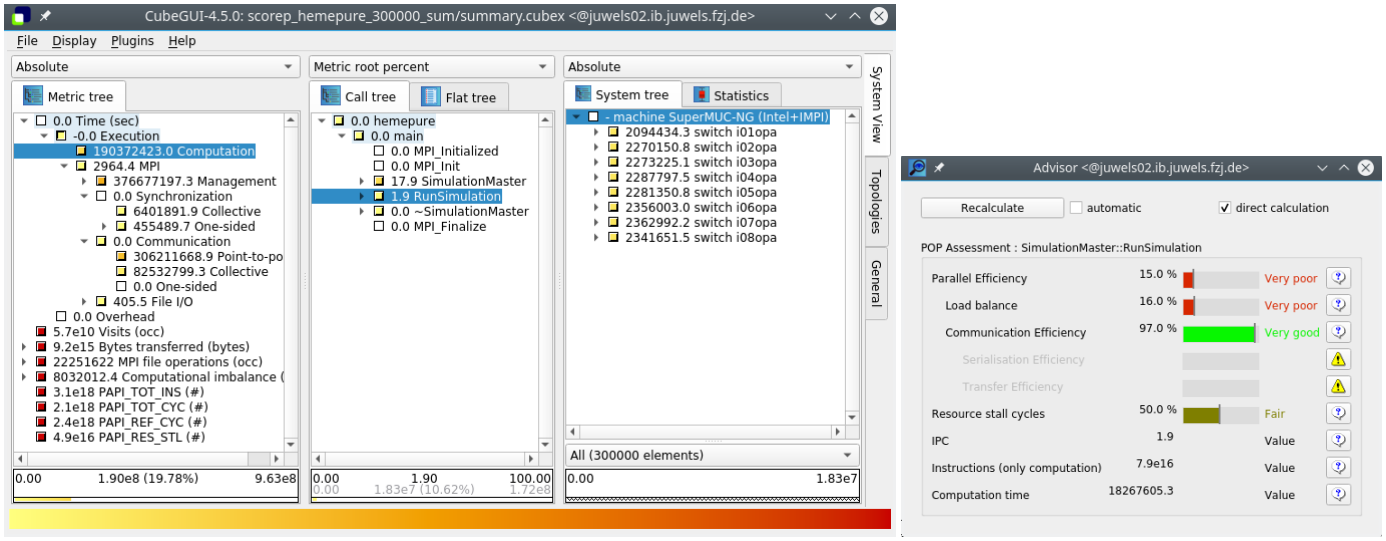


Fig. 3. CUBE presentation of efficiency metrics for `RunSimulation` phase of execution of HemeLB on SuperMUC-NG with 300,000 MPI processes. While communication efficiency of 0.97 is very good, load balance efficiency of 0.16 is very poor.

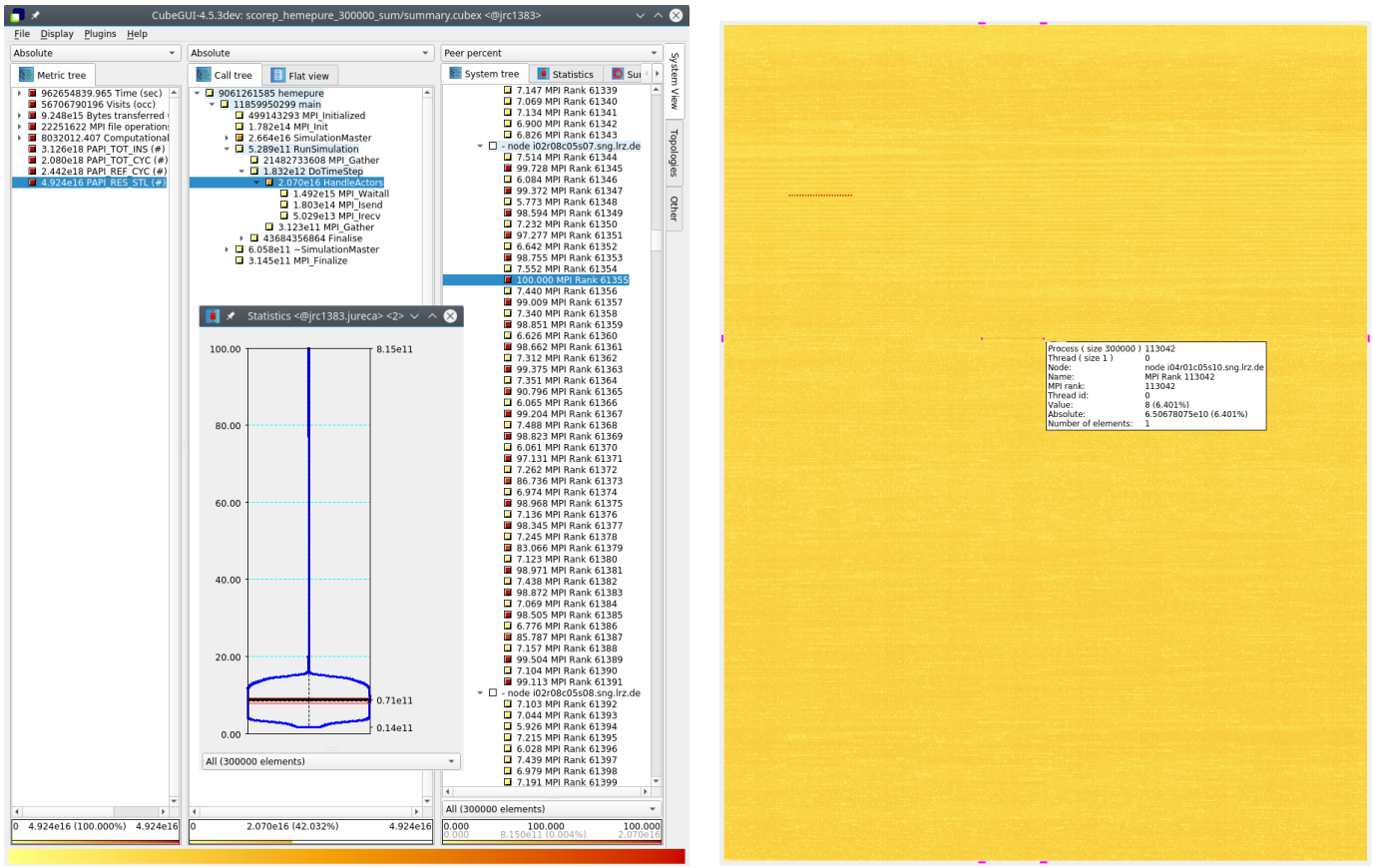


Fig. 4. Scalasca/CUBE analysis report explorer presentation of HemeLB execution on SuperMUC-NG with 300,000 MPI processes. The main window has `PAPI_RES_STL` selected in the *Metric tree* and the `HandleActors` callpath within `RunSimulation/DoTimeStep` selected in the *Call tree* panels. For this combination, the metric values for all processes, as percentages of the largest peer value, are shown in the *System tree* panel and separately in the (detached) *Statistics* chart and *Process topology* panel (folded such that each row has the processes of ten compute nodes). It is immediately clear that all of the 24 processes with odd MPI ranks in `MPI_COMM_WORLD` on node `i02r08c05s07` have more than 14x the resource stall cycles of any others, and closer examination further reveals node `i04r01c05s10` also has values for its even MPI ranks considerably elevated beyond all others.

TABLE I
HEMELB EXECUTION EFFICIENCY AND SCALING RELATIVE TO 1,152 PROCESSES ON 24 COMPUTE NODES OF SUPERMUC-NG.

Compute nodes	24	32	48	64	96	128	192	256	384	512	768	1024	1536	2048	3072	4096	6452
Processes	**1152**	**1536**	**2304**	**3072**	**4608**	**6144**	**9216**	**12288**	**18432**	**24576**	**36864**	**49152**	**73728**	**98304**	**147456**	**196608**	**309696**
Global scaling efficiency	**0.79**	**0.79**	**0.84**	**0.80**	**0.82**	**0.75**		**0.73**	**0.72**	**0.73**	**0.74**	**0.68**	**0.68**	**0.65**	**0.62**	**0.57**	**0.45**
- Parallel efficiency	0.79	0.80	0.87	0.83	0.86	0.80		0.75	0.74	0.74	0.77	0.71	0.72	0.70	0.72	0.70	0.73
- - Load balance efficiency	0.79	0.80	0.88	0.84	0.86	0.80		0.75	0.74	0.75	0.78	0.72	0.74	0.72	0.74	0.73	0.80
- - Communication efficiency	1.00	1.00	1.00	1.00	1.00	1.00		1.00	1.00	0.99	0.99	0.99	0.98	0.98	0.97	0.96	0.92
- Computation scaling	*1.00*	0.99	0.96	0.96	0.95	0.93		0.98	0.98	0.98	0.96	0.96	0.94	0.93	0.87	0.81	0.61
- - Instructions scaling	*1.00*	1.00	1.00	1.00	1.00	1.00		1.00	1.00	1.00	0.99	0.97	0.94	0.89	0.79	0.67	0.45
- - IPC scaling	*1.00*	0.99	0.96	0.96	0.95	0.93		0.98	0.98	0.99	0.98	0.99	1.00	1.04	1.11	1.21	1.36
IPC	1.411	1.395	1.353	1.355	1.342	1.316		1.377	1.387	1.396	1.383	1.390	1.417	1.473	1.566	1.704	1.919
											Key:	<0.65	<0.75	<0.85	<0.95	<1.00	>1.00

Global scaling efficiency is the product of parallel efficiency and computation scaling.
Parallel efficiency is the ratio of mean computation time to total runtime of all processes.
Load balance efficiency is the mean/maximum ratio of computation time outside of MPI.
Communication efficiency is the ratio of maximum computation time to total runtime.
Serialisation efficiency is estimated from idle time within communications where no data is transferred.
Transfer efficiency relates to essential time spent in data transfers.
Computation scaling is the relative total time in computation (outside of MPI).
Instructions scaling is the relative total number of instructions executed (outside of MPI).
IPC scaling is the relative value of instructions executed (outside of MPI) per CPU cycle.
(Scaling efficiencies are relative to a serial execution or the smallest parallel execution configuration.)

sequent executions 5 times faster simulation performance was obtained.

Diagnostic checks run on those nodes by the system administrators after they had been taken out of production identified that they had faulty DIMMs and lacklustre performance, needing the DIMMs to be replaced.

E. Efficiency analysis

For the updated Scalasca/Score-P measurements of HemeLB executions on SuperMUC-NG with up to 6,452 compute nodes (309,696 MPI processes), Table I summarises their performance assessment. Computational instructions retired per clock cycle (IPC) was a reasonable 1.9, compared to 1.4 for the smaller execution configurations, suggesting better cache efficiency as the lattice partitions get smaller. Perfect instruction scaling up to 768 compute nodes thereafter deteriorates as there is more processing of lattice block boundaries compared to their interiors. Since these two effects counteract each other, very good computation scaling above 0.87 is sustained. Efficient non-blocking communication to exchange fluid particles between neighbouring lattice blocks maintains excellent communication efficiency above 0.97. The most significant inefficiency at all scales tested is load balance, generally around 0.80 but dropping to 0.72 in some larger execution configurations. While this is still fairly good, it presents the largest opportunity for performance improvement and warrants more in-depth investigation.

F. Investigation of load balance

Investigation of the breakdown of (MPI) communication versus computation time of each process showed that one or a few processes spend essentially no time waiting for the non-blocking point-to-point MPI communication to complete during the simulation phase. Since these processes are fully occupied with computation for longer than the others, by the time they reach the `MPI_Wait` the incoming messages for them have already been received into the corresponding buffers. This is characteristic of computational load imbalance.

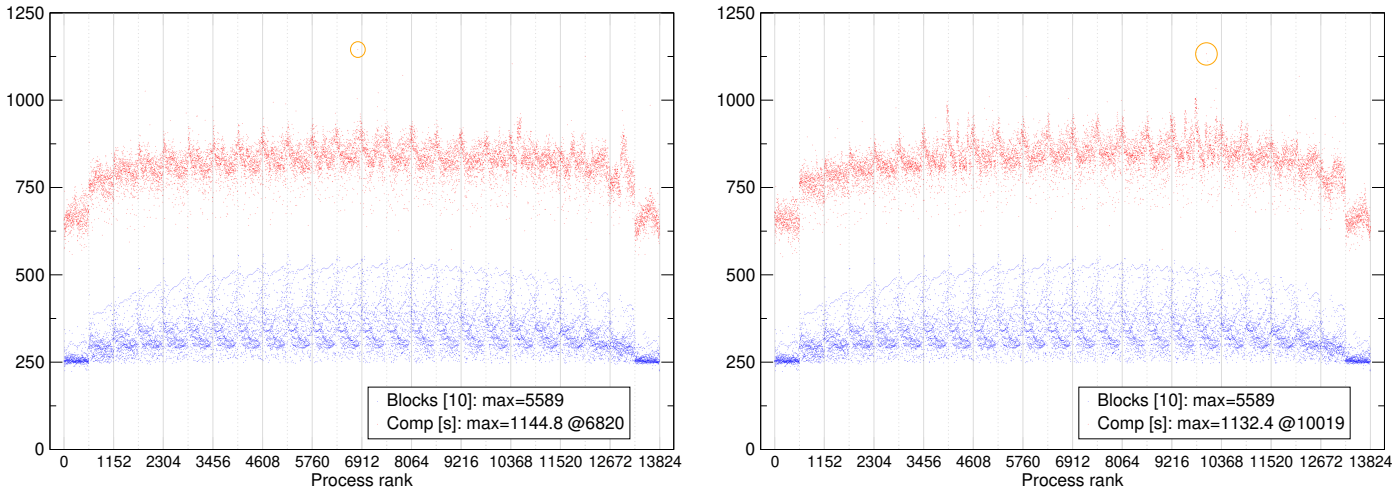
The HemeLB 'basic decomposition' method for splitting the lattice grid into parts for each process was already known to result in a notable imbalance, with many of the higher-ranked processes given disconnected sections that result in additional computation for them. The decomposition is expected to be deterministic, however, allowing comparison between runs. This was verified by comparing the numbers of lattice blocks and sites allocated to each process, which matched exactly. Similarly, the number of messages and the amount of data sent and received by each process also matched exactly, verifying that the pattern of communication was also identical.

G. Computation time variation

In different runs with the same number of processes the communication time can be expected to vary, both due to the mapping of processes to different compute nodes distributed throughout SuperMUC-NG (in some cases requiring additional communication inter-island hops via top-level switches) and also contention on those communication paths shared with other applications concurrently executing on the system [12].[11] While the first aspect can be addressed by forcing execution on identical machine partitions, e.g. doing multiple runs consecutively within a single job (Figure 5), avoiding possible communication interference from other applications can only be achieved via dedicated use of islands or the entire computer system (i.e., single user execution).

Figure 5 shows the amount of HemeLB work per MPI process, represented by the number of blocks of lattice sites they were assigned (which is identical in both runs), and the corresponding measured computation time (which varies

[11] There is no isolation of job partitions as on IBM Blue Gene systems.



(a) 1st run in job (b) 2nd run in job

Fig. 5. Distribution of the number of blocks of lattice sites (in tens, blue points) and resulting computation time (in seconds, red points) per MPI process rank for consecutive executions of HemeLB within a single job on same partition, showing locations of outliers moving from run to run and substantially degrading execution performance.

only slightly apart from outliers, the worst of which are circled in each run). While the amount of computation and communication are not perfectly balanced, they do not change between runs. The mean computation time in both cases is 810 seconds, however, it is more relevant to compare the outliers to the usual maximum of approximately 1000 seconds (i.e. from processes with no significant memory stall cycles): the outliers of 1144 and 1132 seconds therefore result in a degradation of around 14%.

Of course, MPI waiting time of each process also depends on when the corresponding partner processes are ready to initiate point-to-point communication, which can be delayed when they require longer for their computational tasks.

Comparison of the number of instructions executed by each process (from the `PAPI_TOT_INS=INSTRUCTIONS_RETIRED` hardware counter)[12] showed little variation between the processes that took longer and the others, suggesting that each process did very similar amounts of work.

Comparison of the number of cycles executed by each process (`PAPI_TOT_CYC=CPU_CLK_THREAD_UNHALTED`) to the number of reference cycles executed (`PAPI_REF_CYC=UNHALTED_REFERENCE_CYCLES`) showed that they correlated perfectly, indicating that (dynamic) variation in clock frequency — due to turbo-boost, throttling due to thermal constraints, and/or the use of AVX512 vector instructions — did not occur. This was to be expected since AVX512 vector instructions were not used[13], and the compute nodes were specifically configured for reproducible performance with turbo-boost and energy-conserving optimisation[14] disabled, but naturally needed to be verified.

Computation can also be disrupted by system noise and jitter from daemon processes running on compute nodes, however, this could impact different compute nodes in every execution. This was eliminated as being unlikely, since the processes which were slow in any execution were verified to be slow throughout the Simulate phase. Figure 6 shows the HemeLB execution trace collected by Scalasca/Score-P in a timeline visualisation by Vampir[15] showing the same process(es) taking longer for each and every timestep, as necessary for them to have essentially no MPI waiting time. And when those processes perform as expected in a consecutive execution within the same job, there is nothing to indicate that the processor (or indeed individual core) is deficient.

H. Core performance

So far it had been verified that the computation and communication were completely deterministic, yet the time required for them varied somewhat from run to run due to very pronounced variations between processes which change in each run. A few processes require considerably longer for the same amount of computation, such that they had no need to wait for communication with the others which had correspondingly large amounts of waiting time.

To investigate deeper into the processor (core) execution behaviour, further measurements were done incorporating additional hardware counters in a top-down fashion as recommended by Intel [13]. It was immediately clear that slow processes correlated with larger numbers of resource stall cycles (`PAPI_RES_STL=RESOURCE_STALLS:ANY`), which cover both front-end and back-end pipelines,

[12]not counting those when within MPI routines, which are not useful computation

[13]this is planned as future optimisation work, following the current performance assessment

[14]Energy Aware Runtime (EAR) on SuperMUC-NG

[15]https://vampir.eu/

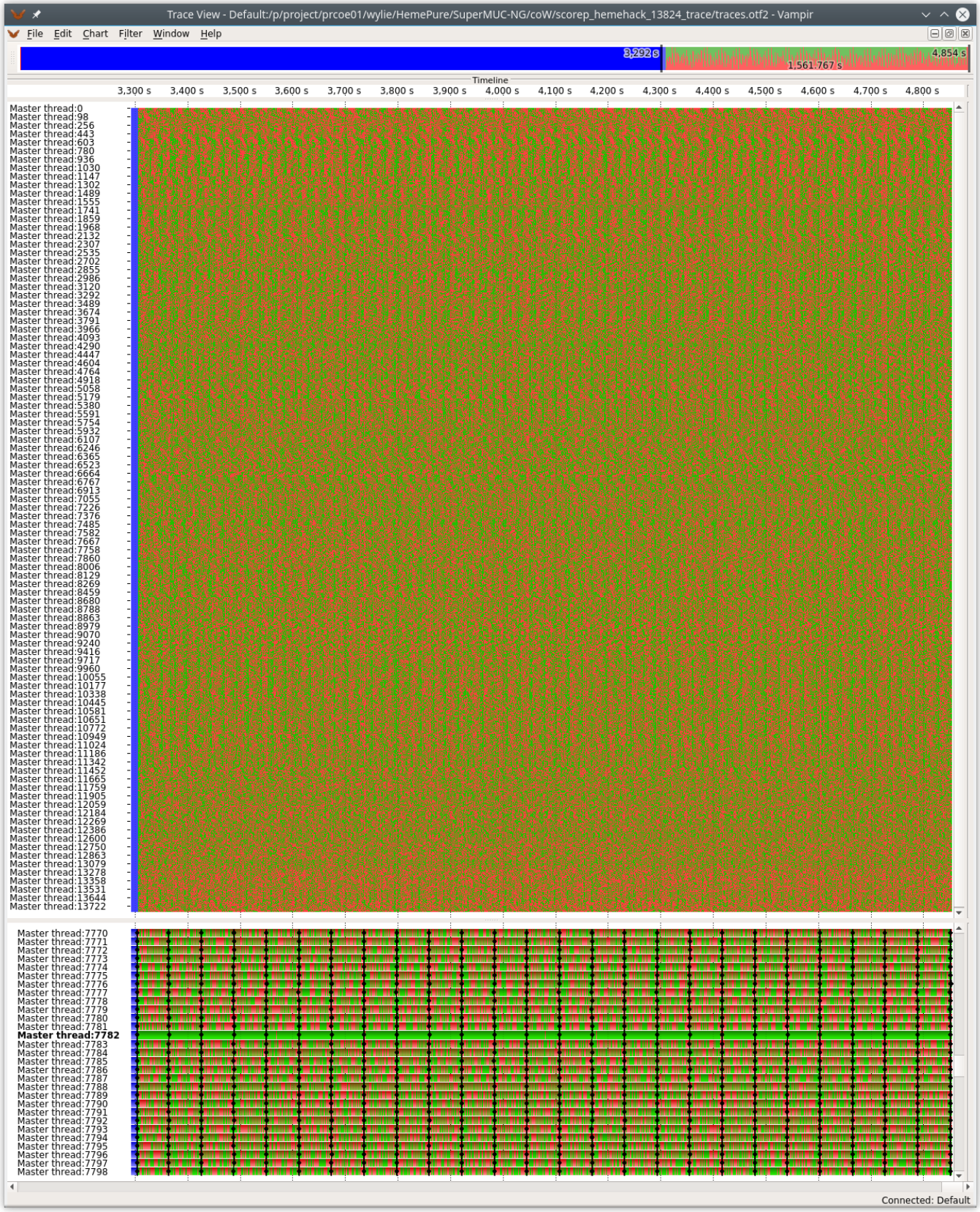


Fig. 6. Vampir execution timeline of 13,824 HemeLB MPI processes (on 288 compute nodes) showing all processes and zoomed to show individual processes in 5,000 time-steps of RunSimulation phase (duration 1,500 seconds), with MPI process rank 7782 clearly distinguished by its lack of time in `MPI_Waitall` [red] although it does sends and receives with its neighbours just like the others. Equally spaced collective communication evident every 200 steps.

which was followed by back-end stall cycles (CYCLE_ACTIVITY:STALLS_TOTAL), and ultimately most strongly correlated with stall cycles waiting for memory (CYCLE_ACTIVITY:STALLS_MEM_ANY). Notably, counters for the various levels of cache (L1D/L2/L3), DRAM and NUMA accesses were all very low and not strongly correlated with the slow processes.

I. Comparison with other HPC computer systems

Lacking further ideas for progress on SuperMUC-NG, it was decided to see whether other similar computer systems would be able to reproduce this variability issue. The JUWELS *regular* cluster nodes have almost identical dual 24-core Intel Xeon Platinum 8168 'Skylake' processors, but a Connect-X4 EDR Infiniband interconnect from Mellanox (opposed to Intel OmniPath), and are running a CentOS 7 Linux kernel. JUWELS *accelerated* cluster nodes only differ in that they have dual 20-core Intel Xeon Gold 6148 'Skylake' processors plus four Nvidia V100 'Volta' GPUs. 240 large-memory compute nodes have 192 GiB, with the others having only 96 GiB (as on SuperMUC-NG). Although the same Intel compiler and MPI were available, HemeLB executions failed to run fully to completion due to a bug handling MPI windows, therefore ParaStationMPI was used instead.[16]

While the HemeLB communication on JUWELS standard nodes was somewhat slower than on SuperMUC-NG, it showed the very same characteristic pattern with one or a few processes having very low MPI waiting time, yet varying from run to run. Similarly, the computation itself was generally slightly faster than on SuperMUC-NG but taking notably longer for those processes with little waiting time and correlating with the corresponding hardware counters, particularly that for stall cycles waiting for memory.

On the JUWELS accelerated nodes, running 20 MPI processes per socket (instead of 24)[17], and on the DEEP-EST prototype Data Analytics Module (DAM) with dual 24-core Intel Xeon Platinum 8260M 'CascadeLake' processors and Extoll Tourmalet interconnect, the same pattern of performance variation between cores was also observed.

III. Discussion

A. Origin of performance variability

Discussion with the system administrators at this point brought to light the fact that significant run-to-run performance variability had been recently reported with the xPic particle-in-cell application [14] on JURECA-Booster (Intel Xeon Phi 'Knights Landing (KNL)') which seemed to match symptoms that had been reported when benchmarking HPL [15]. TACC Stampede2 comprises both Intel Xeon 'Skylake' and Intel Xeon Phi 'KNL' processors which both showed significant HPL and DGEMM performance variations from run to run, that are due to elevated snoop filter conflicts arising from the mapping of physical memory pages into the L3 cache based on a proprietary hashing mechanism, as also documented by Intel [17]. The off-core memory controller hardware counters which can be used to confirm this are priviledged and therefore not accessible to user codes, as well as not being directly related to cores or application processes.

Further investigation and analysis [16] also determined that, while the 24-core Skylake and 68-core KNL are apparently the most seriously impacted, Intel Xeon Scalable processors with non-power-of-two cores are all apparently affected to varying extents by this issue. Huge memory pages[18] of 1 GiB, which are Intel's workaround to optimise HPL performance, were determined to be effective, however, these are not yet available on Stampede2, SuperMUC-NG, JUWELS, or other production systems with similar processors. On HPC clusters where other proposed workarounds such as disabled hardware threading or sub-NUMA clustering (SNC)[19] is configured, identical performance variability was encountered.

B. Impact on HPC applications

While the occurrence of this issue can be considered to be rare, with only one in every ten or one hundred thousand cores being affected with a slowdown of more than 10%, the impact on large-scale parallel applications is very significant. Since these are characterised by occassional (and often more frequent) synchronisations, whether via global collective operations or indirectly via interactions with neighbours, all processes will be held back waiting on the slowest. The end result is that an additional 10% or more of the entire computing time is required for busy waiting, with corresponding impact on energy consumption and system throughput. Performance profiles of executions of the SPECFEM3D seismic wave propagation simulation application[20] taken on the Skylake compute nodes of the Irène Joliot-Curie supercomputer also show the same behaviour and impact on performance at large scale, therefore the issue is likely to be widespread but perhaps only identified when explicitly looked for.

A wide variety of commercial/vendor and open-source performance tools are available which could be used to identify such situations: the applicability and capabilities of many are summarised in [18]. The tools used and demonstrated in this paper were found to be particularly convenient, supporting (very) large-scale measurements and analyses, however, the performance variability and variation by process/core were also verified using Intel Parallel Studio XE[21] (notably Application Performance Snapshot and Amplifier/VTune), although limited to only modest scale.

System-level job-reporting and monitoring tools operate on leadership HPC systems, including PerSyst [19] at LRZ and LLview [20] at JSC, however, these currently don't capture and present the RAS data required to identify the performance issues encountered in this work. Since the performance issues are related to application-specific access patterns to

[16] configured to avoid excessive memory requirements for MPI message buffers: PSP_UCP=1, UCX_TLS=sm,self,ud

[17] therefore still one process per physical core, and not using the GPGPUs

[18] https://github.com/libhugetlbfs/libhugetlbfs

[19] https://patents.google.com/patent/US8862828B2/en

[20] https://geodynamics.org/cig/software/specfem3d

[21] https://software.intel.com/en-us/parallel-studio-xe

TABLE II
COMPUTER SYSTEMS AND CODES TESTED

Computer system	Site	CPU	Processor	Cores	HTT	SNC	GHP	Uncore	xPic	Other codes
SuperMUC-NG	LRZ	SKX	Platinum 8174 'Skylake'	2x24	2	-	-	-	7.6%	HemeLB
JUWELS-Cluster	JSC	SKX	Platinum 8168 'Skylake'	2x24	2	-	-	-	6.5%	HemeLB
Irène Joliot-Curie	TGCC	SKX	Platinum 8168 'Skylake'	2x24	2	-	-	-		SPECFEM3D
MareNostrum-IV	BSC	SKX	Platinum 8160 'Skylake'	2x24	-	-	-	-	8.2%	
CLAIX-2018	RWTH	SKX	Platinum 8160 'Skylake'	2x24	2	2	-	-	7.3%	
Barbora (cpu)	IT4I	CLX	Gold 6240 'Cascade Lake'	2x18	2	-	-	-	6.0%	
JUWELS-GPU (V100)	JSC	SKX	Gold 6148 'Skylake'	2x20	2	-	-	-	6.1%	
Barbora (gpu)	IT4I	SKX	Gold 6126 'Skylake'	2x12	2	-	-	-	6.5%	
JURECA-Cluster	JSC	HSW	E5-2680 v3 'Haswell'	2x12	2	-	-	-	0.9%	

Cores = sockets-per-node×cores-per-socket
HTT = Hyper-Threading Technnology enabled: Intel proprietary simultaneous multithreading (SMT) implementation
SNC = Sub-NUMA Clustering enabled (number of NUMA domains per socket)
GHP = Gigabyte Huge Pages enabled providing backing for application address space
Uncore = (priviledged) access available to Intel system agent non-core hardware counters such as memory controller and snoop agent
xPic = worst observed slowdown relative to best (minimum) execution on the same processor
Other codes = multi-node MPI applications that manifest similar non-deterministic slowdown (5%+) for individual processes/cores

large amounts of memory, for processes/threads executing on individual processor sockets/cores and only during particular execution phases, this will require more fine-grained data collection and analysis.

C. Application malleability and reactivity

Avoidance of suspect processors when submitting jobs cannot be done when the affected processors/cores are not determinable in advance, necessitating dynamic load re-balancing after an application has already started execution. Fortunately, once affected processes been identified and have all had their effective load re-balanced (which might require more than one iteration), it is expected that efficient execution will be unimpeded thereafter (unless additional or different memory is subsequently accessed).

Chameleon [21] and DLB [22] are examples of libraries being developed for reactive load balancing. Both rely on applications incorporating hybrid task parallelism via OpenMP to allow dynamic migration of work (tasks) between threads in SMP compute nodes after self-introspection and analysis, which can be more efficient than standard OpenMP dynamic loop scheduling but similarly damages data locality when work is migrated. For persistent load imbalances, as encountered in this study, a customised lightweight migration strategy needs to be developed.

IV. CONCLUSION

Performance analysis of (prospective) exascale applications has long been recognised to present significant challenges, particularly with extreme scale and ever increasing complexity.

Inherent performance variability has always been an issue for HPC parallel applications, from network contention, shared filesystems, intrusive system daemons, energy policies, etc., for which corresponding amelioration strategoes are well documented in best practices. However, application-specific memory and processor core performance variablity exacerbates this, and now (large-scale) executions have neither predictable, repeatable nor reproducible performance, even for immediately following runs on the exact same hardware! The task for performance analysts is thereby made much more difficult.

For robust exa-scalability, applications are likely to need to dynamically adapt to both hardware failures and performance degradations that would otherwise cripple executions. Performance tools will therefore also need to become more dynamic in not only measuring parallel executions, but also providing analyses of execution inefficiencies directly to applications on the fly so that they can be addressed promptly.

ACKNOWLEDGMENTS

This work was done under the auspices of the European Union's Horizon 2020 research and innovation programme HPC Centres of Excellence for Performance Optimisation and Productivity (POP, 824080) and Computational Biomedicine (CompBioMed, 675451) [23].

The Partnership for Advanced Computing in Europe (PRACE) and Gauß Centre for Supercomputing e.V. (www.gauss-centre.eu) are gratefully acknowledged for providing computing time on the GCS supercomputers SuperMUC-NG at Leibniz-Rechenzentrum and JUWELS at Jülich Supercomputing Centre. Additional allocations for testing were also provided by JSC on DEEP-EST DAM, JURECA & JUSUF, and by POP partners Barcelona Supercomputing Center on MareNostrum-IV, IT4Innovations on Barbora, RWTH Aachen University on CLAIX-2018, and Très Grand Centre de calcul du CEA (TGCC) on Irène Joliot-Curie.

Alex Patronis and Jacopo de Amicis guided configuration and building of the HemeLB and xPic codes and running their testcases, support for Intel, Scalasca (including Score-P and CUBE) and Vampir tools came from their developers, while Damian Alvarez, Reinhold Bader and their colleagues gave technical support and key advice.

Particular thanks are also due Dirk Brömmel, Peter Coveney, Stephane Eranian, Wolfgang Frings, René Halver, Jesús Labarta, John McCalpin, Larry Meadows, Heidi Poxon, Godehard Sutmann, Josef Weidendorfer and the anonymous referees for their insightful feedback on earlier versions of this work which greatly improved the analysis and its presentation.

REFERENCES

[1] J. Dongarra, "Report on the Fujitsu Fugaku system," University of Tennessee-Knoxville Innovative Computing Laboratory, Tech. Rep. ICL-UT-20-06, June 2020.
[Online]. Available: https://www.icl.utk.edu/files/publications/2020/icl-utk-1379-2020.pdf

[2] D. Brömmel, W. Frings, B. J. N. Wylie, B. Mohr, P. Gibbon, and T. Lippert, "The High-Q Club: Experience with extreme-scaling application codes," *Supercomputing Frontiers and Innovations* 5(1), 59-78 (2018).
[Online]. Available: http://doi.org/10.14529/jsfi180104

[3] T. J. Deakin, S. N. McIntosh-Smith, J. Price, A. Poenaru, P. R. Atkinson, C. Popa, and J. Salmon, "Performance portability across diverse computer architectures," in *Proc. P3HPC Int'l Workshop on Performance, Portability and Productivity in HPC (Denver, Colorado, USA)*, ACM, Nov. 2019.
[Online]. Available: https://doi.org/10.1109/P3HPC49587.2019.00006

[4] F. Petrini, D. J. Kerbyson, and S. Pakin, "The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q," in *Proc. SC03 Int'l Conf. for High Performance Computing, Networking, Storage, and Analysis (Phoenix, Arizona, USA)*, ACM, Nov. 2003.
[Online]. Available: https://doi.org/10.1145/1048935.1050204

[5] T. Patki, J. J. Thiagarajan, A. Ayala, and T. Z. Islam, "Performance optimality or reproducibility: that is the question," in *Proc. SC19 Int'l Conf. for High Performance Computing, Networking, Storage, and Analysis (Denver, Colorado, USA)*, ACM, Nov. 2019.
[Online]. Available: https://doi.org/10.1145/3295500.3356217

[6] J. W. S. McCullough, R. A. Richardson, A. Patronis, R. Halver, R. Marshall, M. Ruefenacht, B. J. N. Wylie, T. Odaker, M. Wiedmann, B. Lloyd, E. Neufeld, G. Sutmann, A. Skjellum, D. Kranzlmüller, and P. V. Coveney, "Towards blood flow in the Virtual Human: Efficient self-coupling of HemeLB," *J. Royal Society Interface Focus*, to appear.
[Online]. Available: https://arxiv.org/abs/2010.04144

[7] A. Patronis, R. A. Richardson, S. Schmieschek, B. J. N. Wylie, R. W. Nash, and P. V. Coveney, "Modeling patient-specific magnetic drug targeting within the intracranial vasculature," *Frontiers in Physiology* 9:331, Apr. 2018.
[Online]. Available: https://doi.org/10.3389/fphys.2018.00331

[8] Intel Corp., "Intel Xeon Platinum 8174 Processor (33M cache, 3.10 GHz) product specifications,"
[Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/136874/intel-xeon-platinum-8174-processor-33m-cache-3-10-ghz.html

[9] A. C. Figueroa, ".stl file of Circle of Willis Benchmark geometric model for hemodynamic simulation," University of Michigan – Deep Blue data repository, Oct. 2020.
[Dataset]. Available: https://doi.org/10.7302/xx1r-zg70

[10] Performance Optimisation and Productivity Centre of Excellence in HPC, "POP Standard Metrics for Parallel Performance Analysis," 2016.
[Online]. Available: https://pop-coe.eu/node/69

[11] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The Scalasca performance toolset architecture," *Concurrency and Computation: Practice and Experience*, 22(6):702-719, Apr. 2010.
[Online]. Available: https://doi.org/10.1002/cpe.1556

[12] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: performance degradation due to nearby jobs," in *Proc. SC13 Int'l Conf. for High Performance Computing, Networking, Storage, and Analysis (Denver, Colorado, USA)*, IEEE, Nov. 2013.
[Online]. Available: https://doi.org/10.1145/2503210.2503247

[13] A. Yasin, "A top-down method for performance analysis and counters architecture," in *Proc. ISPASS Int'l Symp. on Performance Analysis of Systems and Software (Monterey, California, USA)*, IEEE, Mar. 2014.
[Online]. Available: https://doi.org/10.1109/ISPASS.2014.6844459

[14] A. Kreuzer, N. Eicker, J. Amaya, and E. Suarez, "Application performance on a cluster-booster system," in *Proc. Int'l Parallel and Distributed Processing Symp. Workshops (IPDPSW, Vancouver, Canada)*, pp. 69-78, IEEE, May 2018.
[Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2018.00019

[15] J. D. McCalpin, "HPL and DGEMM performance variability on the Xeon Platinum 8160 processor," in *Proc. SC18 Int'l Conf. for High Performance Computing, Networking, Storage, and Analysis (Dallas, Texas, USA)*, IEEE, Nov. 2018.
[Online]. Available: https://doi.org/10.1109/SC.2018.00021

[16] J. D. McCalpin, "(HPL and DGEMM) performance variability due to snoop filter conflicts on the Xeon Gold and Platinum processors with 18, 20, 22, 24, 26, and 28 cores — and Xeon Phi x200 (KNL)," annotated slides from [15] presentation.
[Online]. Available: https://sites.utexas.edu/jdm4372/2019/01/07/sc18-paper-hpl-and-dgemm-performance-variability-on-intel-xeon-platinum-8160-processors

[17] Intel Corp., "Run-to-run performance variability on 24-core Intel Xeon Scalable processor," white paper, number 576612, revision 0.3, Feb. 2018.

[18] Virtual Institute – High Productivity Supercomputing, "VI-HPS Tools Guide," Nov. 2019.
[Online]. Available: http://www.vi-hps.org/tools/cms/upload/material/general/ToolsGuide.pdf

[19] C. Guillen, W. Hesse, and M. Brehm, "The PerSyst monitoring tool: A transport system for performance data using quantiles," in *Euro-Par 2014: Parallel Processing Workshops (Porto, Portugal)*, Lecture Notes in Computer Science, vol 8806, pp. 363-374, Springer, Aug. 2014.
[Online]. Available: https://doi.org/10.1007/978-3-319-14313-2_31

[20] S. Lührs and W. Frings, "LLview-based job-reporting: Non-intrusive job performance-monitoring," in *Performance Engineering Workshop (Dresden, Germany)*, Mar. 2019.
[Online]. Available: http://hdl.handle.net/2128/22727

[21] J. Klinkenberg, P. Sanfass, M. Bader, C. Terboven, and M. S. Müller, "CHAMELEON: Reactive load balancing for hybrid MPI+OpenMP task-parallel applications," *J. Parallel Distrib. Computing* 138, pp. 55-64, Apr. 2020.
[Online]. Available: https://doi.org/10.1016/j.jpdc.2019.12.005

[22] M. Garcia, J. Labarta, and J. Corbalan, "Hints to improve automatic load balancing with LeWI for hybrid applications," *J. Parallel Distrib. Computing* 74(9), pp. 2781-2794, Sep. 2014.
[Online]. Available: https://doi.org/10.1016/j.jpdc.2014.05.004

[23] B. J. N. Wylie, "HemeLB on SuperMUC-NG performance assessment report," POP2_AR_041, Feb. 2020.
[Online]. Available: https://doi.org/10.5281/zenodo.4105742

APPENDIX

A. Artefact Description

The HemeLB application was run with a 6.4μs resolution circle of Willis geometry dataset on LRZ's SuperMUC-NG supercomputer, and its execution performance and scalability to over 300,000 MPI processes measured and analysed via run-time summaries and execution event traces from the system's 6,480 dual 24-core compute nodes.

a) Software Artefact Availability: Some author-created software artefacts are NOT maintained in a public repository.

b) Hardware Artefact Availability: There are no author-created hardware artefacts.

c) Data Artefact Availability: Some author-created data artefacts are NOT maintained in a public repository.

d) Proprietary Artefacts: No author-created artefacts are proprietary. There are associated proprietary artefacts that are not created by the author.

e) DOIs where artefacts are available:

- 10.5281/zenodo.4105742: HemeLB on SuperMUC-NG performance assessment report (POP2_AR_041)
- 10.5281/zenodo.4104374: Scalasca summary analysis of HemeLB application execution on SuperMUC-NG (300000 MPI processes)
- 10.5281/zenodo.4104345: Scalasca trace analysis of HemeLB application execution on SuperMUC-NG (13824 MPI processes)

B. Baseline experimental setup

a) Relevant hardware details: Lenovo ThinkSystem SD650 supercomputer SuperMUC-NG at Leibniz-Rechenzentrum (LRZ, Germany) comprising 6,480 compute nodes with dual 24-core Intel Xeon Platinum 8174 @ 3.10 GHz ('Skylake') processors, each node with a minimum of 96 GiB memory. (144 compute nodes with 768 GiB.)

Intel OmniPath network, with a fat-tree topology within islands and 1:4 pruned connection between islands. 50 TB IBM Spectrum Scale (GPFS) parallel filesystem.

b) Operating system:

- SUSE Linux Enterprise Server (SLES) 12 SP3

c) Compiler:

- Intel Studio C++ compiler 19.0.4.243

d) Libraries:

- Boost 1.61.0
- Intel Studio MPI 19.0.4.243
- OTF2 2.1.1
- PAPI 5.6.0
- SIONlib 1.7.2 [http://www.fz-juelich.de/jsc/sionlib]

e) Tools:

- Cmake 3.12.1
- Scalasca 2.5 [10.5281/zenodo.4103923]
- Score-P 5.0 [10.5281/zenodo.2605026]
- CubeGUI 4.5 [10.5281/zenodo.3885304]
- Vampir/VampirServer 9.6.1 [https://www.vampir.eu]

f) Application:

- HemeLB (including development versions) configured to use two reader ranks and MPI-3 shared memory model when distributing geometry data during initialisation [https://github.com/UCL-CCS/HemePure]

g) Input dataset:

- `input.xml` specifying 5000 lattice simulation steps of 0.1μs with writing of intermediate and final state disabled
- `coW-6.4um.gmy` "circle of Willis" lattice geometry at 6.4 micron resolution [10.7302/xx1r-zg70]

C. Modifications made for the paper

a) Score-P configuration: A custom installation specifying `MPI_CPPFLAGS=-DSCOREP_MPI_NO_MINI`, to avoid generating wrappers for `MPI_Comm_rank/size` (required for trace collection/analysis), was configured with the PAPI library and for Intel MPI and compilers.

b) Instrumentation: The HemeLB application dependencies were built without modification. Source code of the HemeLB application `src/main.cc` was annotated with Score-P measurement control API macros (`SCOREP_RECORDING_OFF/ON`) to pause event recording during the initialisation phase, and configured using `CXX=scorep-icpc` with environment settings to direct the Score-P instrumenter

```
SCOREP_WRAPPER_INSTRUMENTER_FLAGS=
    "--user --mpp=mpi --thread=none --nomemory"
SCOREP_WRAPPER_COMPILER_FLAGS=
    "-tcollect-filter=<path_to_file>"
```

with the latter filter file for the Intel compiler containing the signatures of routines to be instrumented or not:

```
.* OFF
main ON
BasicDecomposition::RotateAndAllocate ON
BasicDecomposition::BasicDecomposition ON
BasicDecomposition::DecomposeBlock ON
GeometryReader::LoadAndDecompose ON
GeometryReader::OptimiseDomainDecomposition ON
GeometryReader::ReadOnAllTasks ON
GeometryReader::ReadHeader ON
GeometryReader::ReadInBlocksWithHalo ON
GeometryReader::RereadBlocks ON
SimulationMaster.cc ON
_sti__* OFF
'non-virtual thunk to' OFF
'virtual thunk to' OFF
hemelb::geometry::LatticeData::Get*Count OFF
SimulationMaster::GetProcessorCount OFF
SimulationMaster::IsCurrentProcTheIOProc OFF
SimulationMaster::LogStabilityReport OFF
SimulationMaster::OutputPeriod OFF
SimulationMaster::RecalculatePropertyReq*s OFF
```

c) Measurement: Job execution via SLURM scripts specified `--ear=off` to have access to hardware counters in measurements and set the processor 'performance' profile disabling dynamic frequency changes. `--ntasks-per-node=48` and `--cpus-per-task=1` allocated one MPI process to each physical core for the specified number of compute nodes. `--cpu-bind=cores` and `--mem-bind=local` bound each process to a dedicated core and local (socket) memory.

Exclusion clauses were also added to get allocations avoiding various compute nodes as necessary, e.g., `--exclude=i02r08c05s07,i04r01c05s10`.

Runtime environment variables for Score-P measurement:

```
SCOREP_DEVELOPMENT_MEMORY_STATS=aggregated
SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC,
                   PAPI_REF_CYC,PAPI_RES_STL
SCOREP_TIMER=gettimeofday
SCOREP_TRACING_USE_SION=true
SCOREP_TRACING_MAX_PROCS_PER_SION_FILE=576
```

`SCOREP_TOTAL_MEMORY` was also set as required based on the Score-P memory usage statistics reported, e.g., to 200MB for the runtime summary execution configurations with 300,000 and more processes. Simulation execution dilation (compared to uninstrumented reference execution) was less than 4%.

d) Analysis: For Scalasca automated trace analysis, event timestamp consistency correction was applied via `SCAN_ANALYZE_OPTS="--time-correct"`.

Analysis report exploration with CUBE GUI and execution trace exploration with Vampir was done on local clusters and a notebook computer.

Extraction of metric values for specific call-paths from analysis reports for additional graphs:

```
cube_dump -z incl -m comp
   -c name=/SimulationMaster::DoTimeStep/  ...
cube_dump -z incl -m bytes_sent_p2p -c 64  ...
```