HemeLB performance assessment report

Document Information

Reference Number	POP_AR_57 (HemeLB)
Author	Brian Wylie (JSC)
Contributor(s)	Ilya Zhukov & Michael Knobloch (JSC)
Date	July 5, 2017

Notices: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.*

Contents

1 Background

Applicants Name: Alexander Patronis
Institution: University College London, UK (EU H2020 *CompBioMed* Centre of Excellence)
Application Name: HemeLB (internal development version)
Programming Language: C++
Programming Model: MPI
Source Code Available: yes (GPL license: `https://github.com/UCL/hemelb`)
Performance study: check (audit)
User description: Assess the overall performance of HemeLB, including a report on scalability and overall parallel efficiency. We have successfully taken the code to 99,600 cores on ARCHER (at around 7700 sites/rank), but need more information regarding its ability to go beyond. Other performance tools available on the system failed at a fraction of this scale.
Application Description: HemeLB is an open-source lattice-Boltzmann code for simulation of large-scale fluid flow in complex geometries, e.g., intracranial blood vessels near aneurysms.
Input data: `config.xml` + `CoW15.gmy` (15μs resolution Circle of Willis celebral arterial circle)
Testcase Description: 24 MPI processes per compute node.
Machine Description: ARCHER Cray XC30 at EPCC, comprising 4920 standard (64GB) and 374 large-memory (128GB) compute nodes, with dual 12-core Intel Xeon E5-2697v2 (Ivy Bridge) 2.7 GHz processors sharing memory and joined by two QPI links, connected via proprietary Cray Aries interconnect (Dragonfly topology). PrgEnv-gnu using GCC 5.1.0 compilers (plus boost/1.60).
Analysis tools: Scalasca/2.3.1 using Score-P/3.1[1], PAPI/5.4.1 (following hardware counters were collected: PAPI_TOT_INS, PAPI_TOT_CYC, PAPI_REF_CYC, RESOURCE_STALLS). Compiler instrumentation only of HemeLB `main`, `SimulationMaster` and `GeometryReader` (ensuring that other classes and routines are not instrumented via a selective instrumentation file) plus MPI, resulting in $< 2\%$ measurement dilation compared to uninstrumented executions.

2 Setup

The specified testcase uses HemeLB to simulate hæmodynamics (blood flow) in the Circle of Willis celebral arterial circle (3 inlets and 6 outlets). The configuration provided for auditing of strong scaling (fixed overall problem size) specified 10,000 timesteps with status reporting (of wall time and memory usage) every 100 timesteps. Periodic writing of the simulation data was disabled as it introduces uncontrollable variability which was considered undesirable for the initial audit. Example output of executions with 36,000 [1500], 48,000 [2000], 60,000 [2500] and 99,600 [4150] processes [compute nodes] on ARCHER were provided for reference.

Two HemeLB versions were built from the provided sources: one configured using Zoltan and ParMETIS and one without. Although expected to produce an optimised domain decomposition, ParMETIS execution was reported to require substantially more time as well as additional memory when initialising the simulation. Since the provided testcase executed with one MPI rank per core (i.e., fully allocated compute nodes) requires most of the memory available on standard 64GB compute nodes, generally it is not possible to use ParMETIS at large scale.

Measurement executions were done on ARCHER with different numbers of compute nodes, always fully allocated with 24 MPI ranks per node. The full 10,000 timesteps were executed

[1]A custom configuration of Score-P/3.1 was built without measurement wrappers for `MPI_Comm_rank` and `MPI_Comm_size` since these are called so frequently that they cause notable dilation of the *Initialise* phase.

with 3,000 [125], 6,000 [250], 12,000 [500], 24,000 [1000], 48,000 [2000] and 96,000 [4000] MPI processes [compute nodes]. A reduced execution of only 1,000 timesteps was used for preparatory measurements with 1,000 MPI processes on 42 compute nodes. Due to application memory requirements for this large testcase, large-memory compute nodes were necessary for executions with fewer than 500 compute nodes. The version of HemeLB using Zoltan+ParMETIS could also only be run on these large-memory nodes due to its additional memory requirements.

3 Application structure and Focus of Analysis

HemeLB execution consists of an initialisation phase (*Initialise*), where the configuration and geometry files are read and their content distributed, followed by the simulation phase (*Simulate*) which performs the requested number of timesteps. Generally, the latter would be many thousands (perhaps millions) and take the overwhelming majority of the time, therefore constituting the primary focus of analysis (FOA). Since the initialisation is a known limiting factor, it was requested to include this separately in the audit, particularly distinguishing the extra cost of Zoltan+ParMETIS (`GeometryReader::OptimiseDomainDecomposition`). Writing of simulation results was specifically disabled for this initial audit.

The execution timeline of HemeLB with 1000 processes on ARCHER in Figure 1 clearly distinguishes the 687 seconds of initialisation (dominated by distribution of geometry data) from the subsequent 1000 time-steps of *Simulate* (`HandleActors`) containing lots of point-to-point communication.[2]

It is also worth noting that MPI rank 0 has a distinguished role, whereby it doesn't have any of the geometry data and therefore monitors rather than shares in the simulation: it spends almost all of its time in `MPI_Waitall`. Furthermore, only 4 ranks (numbers 1 to 4) read the geometry data and distribute it to the others (excluding rank 0).

Figure 2 shows a call-tree profile of *Visit* counts from a HemeLB execution measurement of the CoW15 testcase with Zoltan+ParMETIS on ARCHER.[3] Figure 2(a) shows `main` and the *Simulate* phase expanded showing `RunSimulation`, `DoTimeStep` and `HandleActors`. The latter call-paths are visited by each of the 3000 MPI ranks each of the 10000 timesteps. Figure 2(b) is an extract of the call-tree showing the *Initialise* phase with `OptimiseDomainDecomposition` and `RereadBlocks` characteristic of when Zoltan+ParMETIS is used. (Master rank 0 is the only one which doesn't execute `RereadBlocks`.)

4 Scalability

Figure 3(a) shows the strong scaling of HemeLB CoW15 from 3000 to 96000 MPI ranks (125 to 4000 ARCHER compute nodes). *Initialise* time (red) is generally 300 to 400 seconds without Zoltan+ParMETIS [XZ], and double this with Zoltan+ParMETIS [WZ].[4] There seems to be a marked transition from constant time up to 12000 ranks, followed by steadily increasing time from 24000 ranks (with an unexplained discontinuity). *Simulate* time (green) shows good scaling, and no significant benefit when Zoltan+ParMETIS `OptimiseDomainDecomposition` is used.

[2]Vampir display quick reference:
https://pop-coe.eu/sites/default/files/pop_files/vampir_display_quickref.pdf

[3]Cube display quick reference:
https://pop-coe.eu/sites/default/files/pop_files/cube_display_quickref.pdf

[4]Executions of HemeLB with Zoltan+ParMETIS fail during initialisation with 6000 ranks apparently exceeding the available memory of 128 GB per compute node.

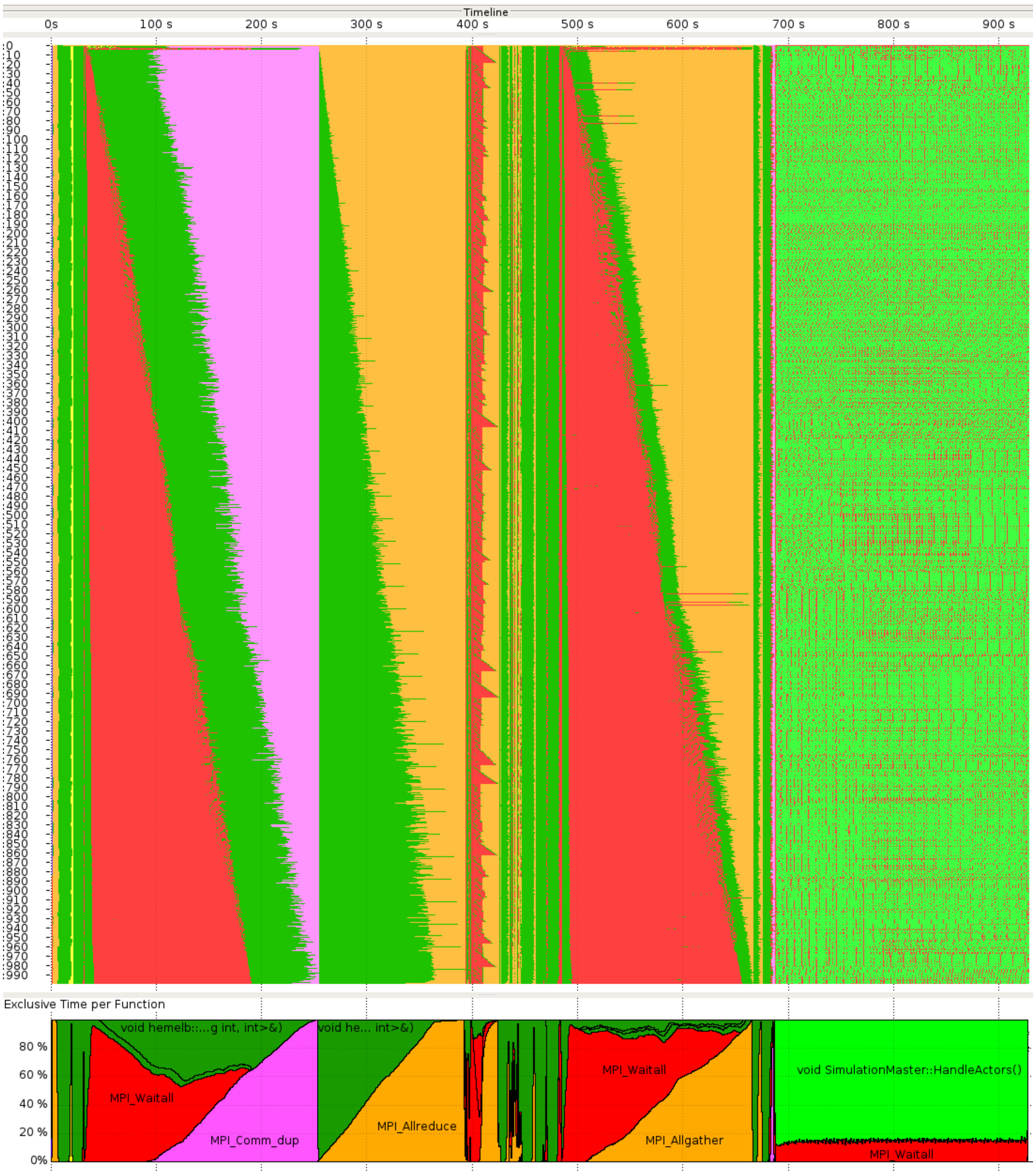
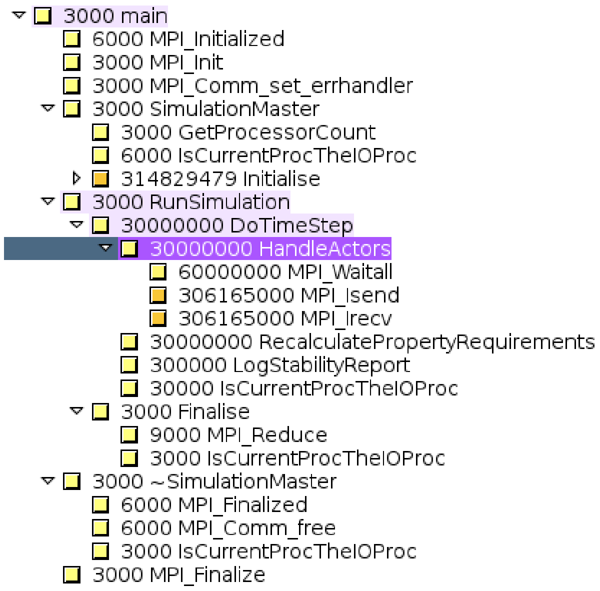
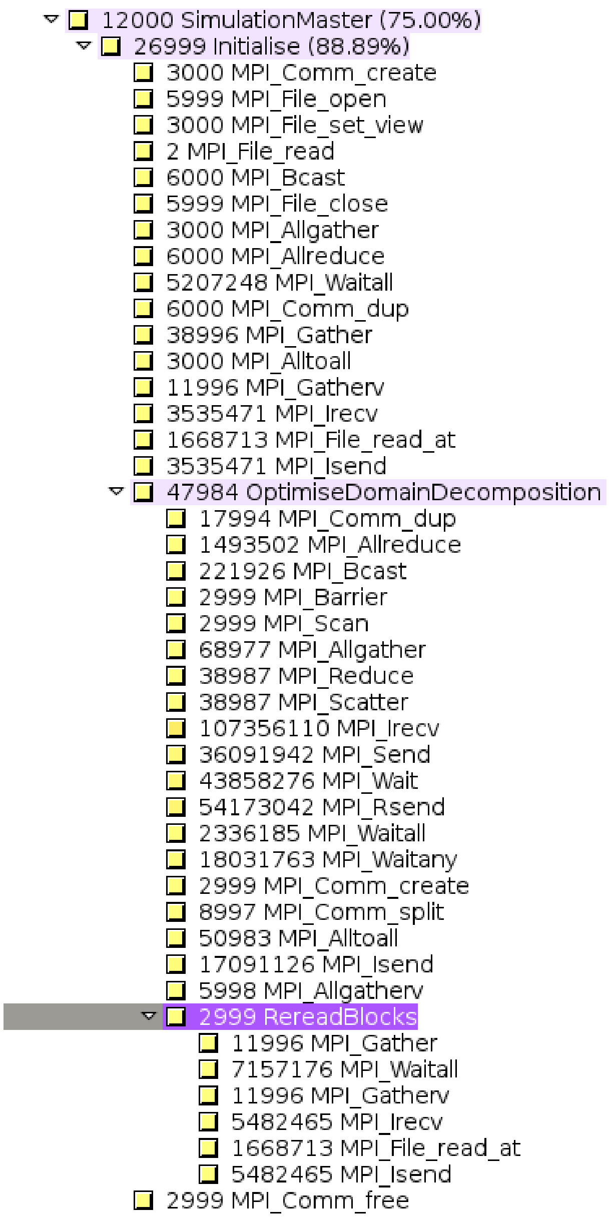


Figure 1: Execution timeline of HemeLB (with Zoltan+ParMETIS) for 1000 simulation timesteps of CoW15 testcase execution with 1000 MPI processes on Archer Cray XC30. Timeline chart of 1000 MPI processes at top, and function time summary chart at bottom. The first 687 seconds are the *Initialise* phase in darker green, followed by 240 seconds of the *Simulation* phase in lighter green (primarily `HandleActors`). MPI file operations are yellow, communicator management is magenta, collective communication is orange, and point-to-point communication (such as `Waitall` here) is red.



```
▽ 3000 main
    6000 MPI_Initialized
    3000 MPI_Init
    3000 MPI_Comm_set_errhandler
  ▽ 3000 SimulationMaster
      3000 GetProcessorCount
      6000 IsCurrentProcTheIOProc
    ▷ 314829479 Initialise
  ▽ 3000 RunSimulation
    ▽ 30000000 DoTimeStep
      ▽ 30000000 HandleActors
          60000000 MPI_Waitall
          306165000 MPI_Isend
          306165000 MPI_Irecv
        30000000 RecalculatePropertyRequirements
        300000 LogStabilityReport
        30000 IsCurrentProcTheIOProc
    ▽ 3000 Finalise
        9000 MPI_Reduce
        3000 IsCurrentProcTheIOProc
  ▽ 3000 ~SimulationMaster
      6000 MPI_Finalized
      6000 MPI_Comm_free
      3000 IsCurrentProcTheIOProc
    3000 MPI_Finalize
```

(a) Main with *Simulate* phase expanded.



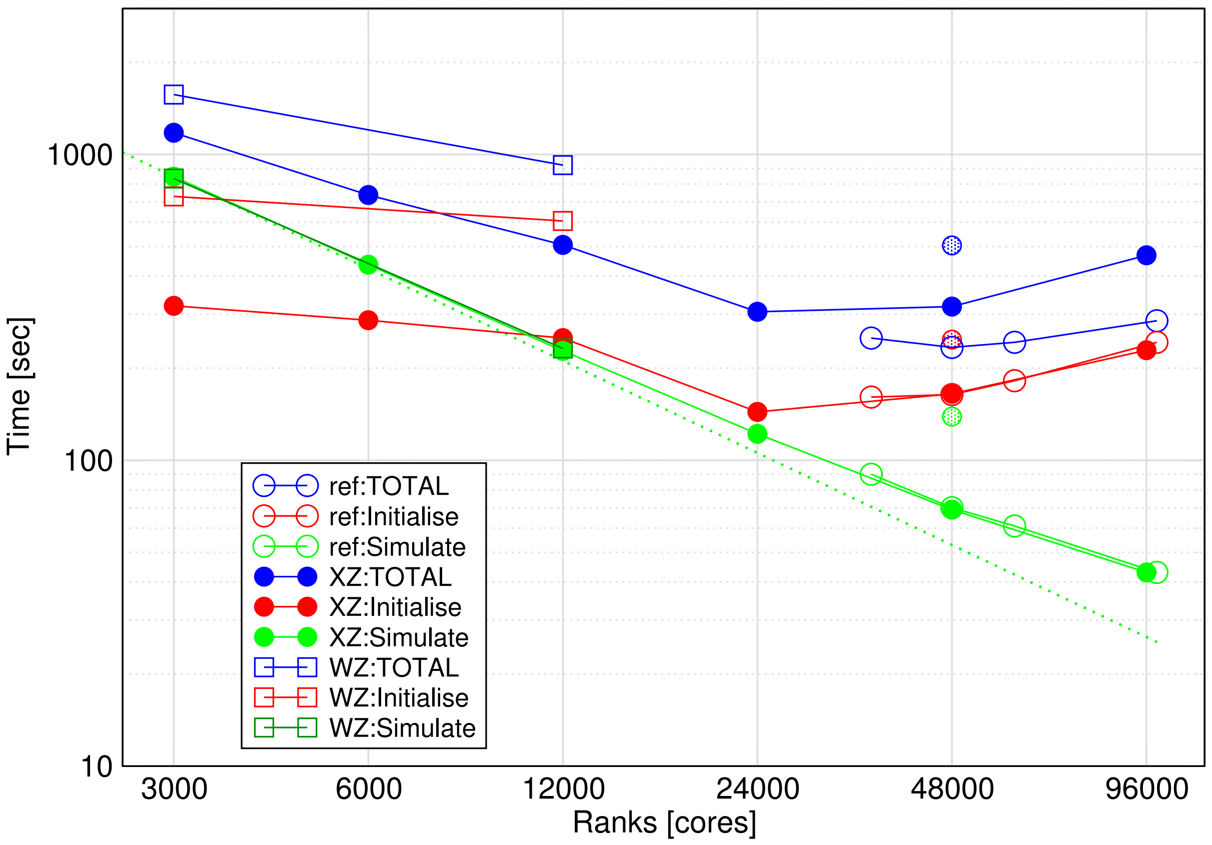
```
▽ 12000 SimulationMaster (75.00%)
  ▽ 26999 Initialise (88.89%)
      3000 MPI_Comm_create
      5999 MPI_File_open
      3000 MPI_File_set_view
      2 MPI_File_read
      6000 MPI_Bcast
      5999 MPI_File_close
      3000 MPI_Allgather
      6000 MPI_Allreduce
      5207248 MPI_Waitall
      6000 MPI_Comm_dup
      38996 MPI_Gather
      3000 MPI_Alltoall
      11996 MPI_Gatherv
      3535471 MPI_Irecv
      1668713 MPI_File_read_at
      3535471 MPI_Isend
    ▽ 47984 OptimiseDomainDecomposition
        17994 MPI_Comm_dup
        1493502 MPI_Allreduce
        221926 MPI_Bcast
        2999 MPI_Barrier
        2999 MPI_Scan
        68977 MPI_Allgather
        38987 MPI_Reduce
        38987 MPI_Scatter
        107356110 MPI_Irecv
        36091942 MPI_Send
        43858276 MPI_Wait
        54173042 MPI_Rsend
        2336185 MPI_Waitall
        18031763 MPI_Waitany
        2999 MPI_Comm_create
        8997 MPI_Comm_split
        50983 MPI_Alltoall
        17091126 MPI_Isend
        5998 MPI_Allgatherv
      ▽ 2999 RereadBlocks
          11996 MPI_Gather
          7157176 MPI_Waitall
          11996 MPI_Gatherv
          5482465 MPI_Irecv
          1668713 MPI_File_read_at
          5482465 MPI_Isend
      2999 MPI_Comm_free
```

(b) *Initialise* phase extract.

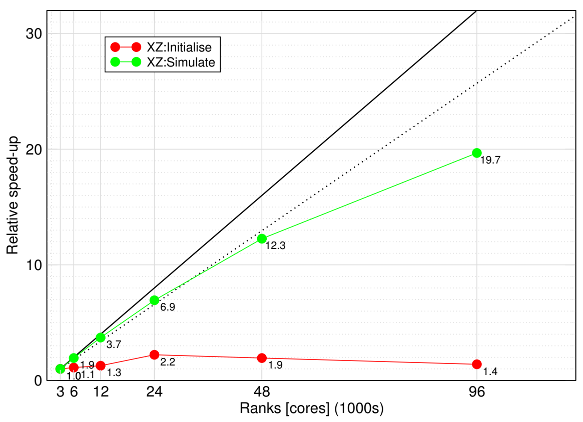
Figure 2: Call-trees with visit counts of HemeLB CoW15 testcase execution using Zoltan+ParMETIS with 3000 MPI processes on ARCHER Cray XC30.

The measurement execution times for both *Initialise* and *Simulate* very closely match those of the uninstrumented reference executions provided, indicating little run-to-run variability and mesaurement dilation $< 2\%$. The only notable exception is the first measurement with 48000 ranks (shaded circles in Figure 3(a)), where *Simulate* took twice as long and *Initialise* 45% longer than the identical measurement done subsequently.

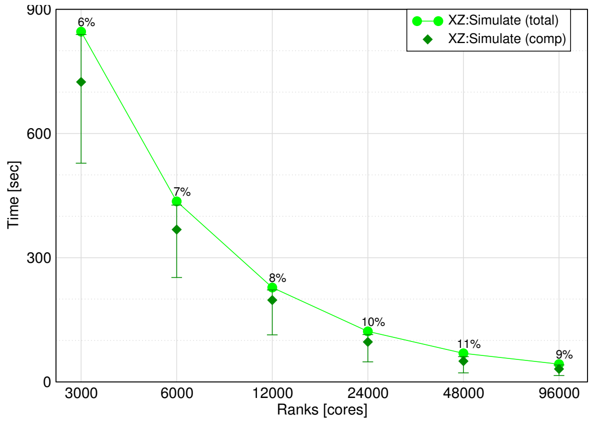
Figure 3(b) shows speed-up relative to 3000 MPI ranks is about a factor of 20 for *Simulate* with 96000 ranks, however, *Initialise* speed-up is at best 2.2 with 24000 ranks.



(a) Wallclock execution time of different HemeLB configurations with CoW15 testcase. Runs using Zoltan+ParMETIS are shown as squares, those without as circles. Uninstrumented reference executions are open circles, whereas measurement executions are filled circles. The shaded circles for 48000 are for identical measurement that ran twice as slowly. Dotted line represents ideal scaling of *Simulate* phase.



(b) Speed-up compared to 3000 processes. Solid line is perfect scaling, dotted line 80% of ideal.



(c) Range of computation time by process in the *Simulate* phase showing extent of imbalance.

Figure 3: HemeLB CoW15 testcase execution time scalability on different numbers of ARCHER Cray XC30 compute nodes (always with 24 MPI ranks per node), distinguishing the *Initialise* and *Simulate* phases.

5 Parallel efficiency metrics

Basic parallel efficiency metrics[5] are defined where the higher the value (closer to 1.00) then the better is the efficiency. *Load balance* is the ratio of average computation to maximal computation time. *Communication* efficiency is the ratio of maximal computation to maximal executing time. *Parallel efficiency* is the ratio of the average computation time to the maximal executing time which is also the product of *Load balance* and *Communication*.

Table 1: Efficiency metrics for HemeLB CoW15 testcase

	without Zoltan+ParMETIS						with Z
Compute nodes	125	250	500	1000	2000	4000	500
Processes	3000	6000	12000	24000	48000	96000	12000
Initialise							
- Parallel efficiency	**0.32**	**0.34**	**0.37**	**0.61**	**0.56**	**0.38**	**0.26**
- - Load balance	**0.92**	**0.94**	**0.94**	**0.93**	**0.93**	**0.91**	**0.62**
- - Communication efficiency	**0.35**	**0.36**	**0.39**	**0.66**	**0.60**	**0.42**	**0.42**
Instruction scaling	*1.00*		**0.30**		**0.08**		
IPC scaling	*1.00*		**0.92**		**0.89**		
IPC	**0.38**		**0.35**		**0.34**		
Simulate							
- Parallel efficiency	**0.85**	**0.84**	**0.82**	**0.79**	**0.73**	**0.72**	**0.77**
- - Load balance	**0.86**	**0.86**	**0.84**	**0.85**	**0.82**	**0.76**	**0.80**
- - Communication efficiency	**0.99**	**0.98**	**0.97**	**0.93**	**0.89**	**0.94**	**0.97**
Instruction scaling	*1.00*		**0.99**		**0.81**		
IPC scaling	*1.00*		**0.99**		**1.09**		
IPC	**1.41**		**1.40**		**1.53**		
Resource stalls	**0.55**		**0.55**		**0.51**		

Table 1 gives an overview of the parallel efficiency of the HemeLB CoW15 testcase executions (with and without Zoltan+ParMETIS for 12000 processes, otherwise only without).

For the *Initialise* phase, parallel efficiency is generally poor due to poor communication efficiency, as load balance remains very good at over 90%. With Zoltan+ParMETIS, communication efficiency is similarly poor and load balance reduced to 62%, such that parallel efficiency for *Initialise* drops to a very poor 26% with 12000 processes.

For the *Simulate* phase, parallel efficiency of 80% or more remains good for up to 24,000 processes, as communication efficiency is very good and load balance also remains good. With the optimised domain decomposition of Zoltan+ParMETIS, load balance is somewhat lower and parallel efficiency drops to 77%.

[5] POP standard metrics for parallel performance analysis: https://pop-coe.eu/node/69

6 Load balance

Despite the heterogeneity of the four reader/distributor ranks, load balance in the *Initialise* phase is considered to be very good.

Rank 0 also provides no contribution to the *Simulate* computation, but has an diminishing impact on the load balance of the entire collection of processes. Although reasonably good, the load balance gets progressively poorer for increasing numbers as it is more challenging to effectively balance the very heterogeneous geometry data. In this regard, use of Zoltan+ParMETIS also shows no consistent benefit.

Figure 3(c) shows that as the minimum time that any process does computation in *Simulate* diverges from the mean, the standard deviation grows from 6% (for 3000) to 11% for (48000).

7 Serial computation

A subset of execution configurations were done including hardware counters for 3000, 12000 and 48000 processes.

For the *Simulate* phase, the number of non-MPI instructions executed increases slightly, however, the corresponding instructions per cycle improves from 1.4 to just over 1.5, which seems like a reasonable instruction-level parallelism. The number of non-MPI instructions increases much faster during *Initialise*, resulting in the IPC dropping from 0.38 to 0.34.

Resource stall cycles (as reported by the native `RESOURCE_STALLS` hardware counter) constitute over 50% of cycles in `HandleActors`, which is very high and needs further investigation. The somewhat lower value for 48,000 processes may be due to data fitting better in caches.

8 Communication

Figure 4 shows a breakdown of time in HemeLB phases. For the *Initialise* phase, there is a constant amount of mean *computation* as more processes are used, however, both *point-to-point communication* and *collective communication* tend to be comparatively expensive. Use of non-blocking sends and receives results in almost all *point-to-point communication* time being for `MPI_Waitall`. While two `MPI_Alltoall` calls are the most expensive *collective communication* at large scale, two `MPI_Bcast` calls (a total of 1GB incoming for each rank) are also costly, with lesser contributions from various `MPI_Allgather`, `MPI_Allreduce`, `MPI_Gather` and `MPI_Gatherv` calls. Only for the largest scales is *file management* time (predominantly `MPI_File_open`) starting to be notable.

In the *Simulate* phase, both mean *computation* time and *point-to-point communication* time are progressively reduced, however, the latter is a slowly growing proportion of total time. Three calls to `MPI_Reduce` when finalising the simulation contribute negligible *collective communication* time. Non-blocking point-to-point communication is used to exchange boundaries with neighbours, such that almost all of the communication time is for 20,000 `MPI_Waitall` calls (two per timestep). The amount of data exchanged increases by more than a factor of three from a total of 36 TB for 3,000 processes to 115 TB for 96,000 processes.

For executions with 1000 processes, the matrix of aggregate message volume for communication partners during the *Simulate* phase when Zoltan+ParMETIS is not employed shown in Figure 5(b) is quite similar to when Zoltan+ParMETIS is employed in Figure 6(b). During the *Initialise* phase, much less communication is done with a different pattern. In Figure 5(a) the four ranks distributing the geometry data are clearly distinguished.

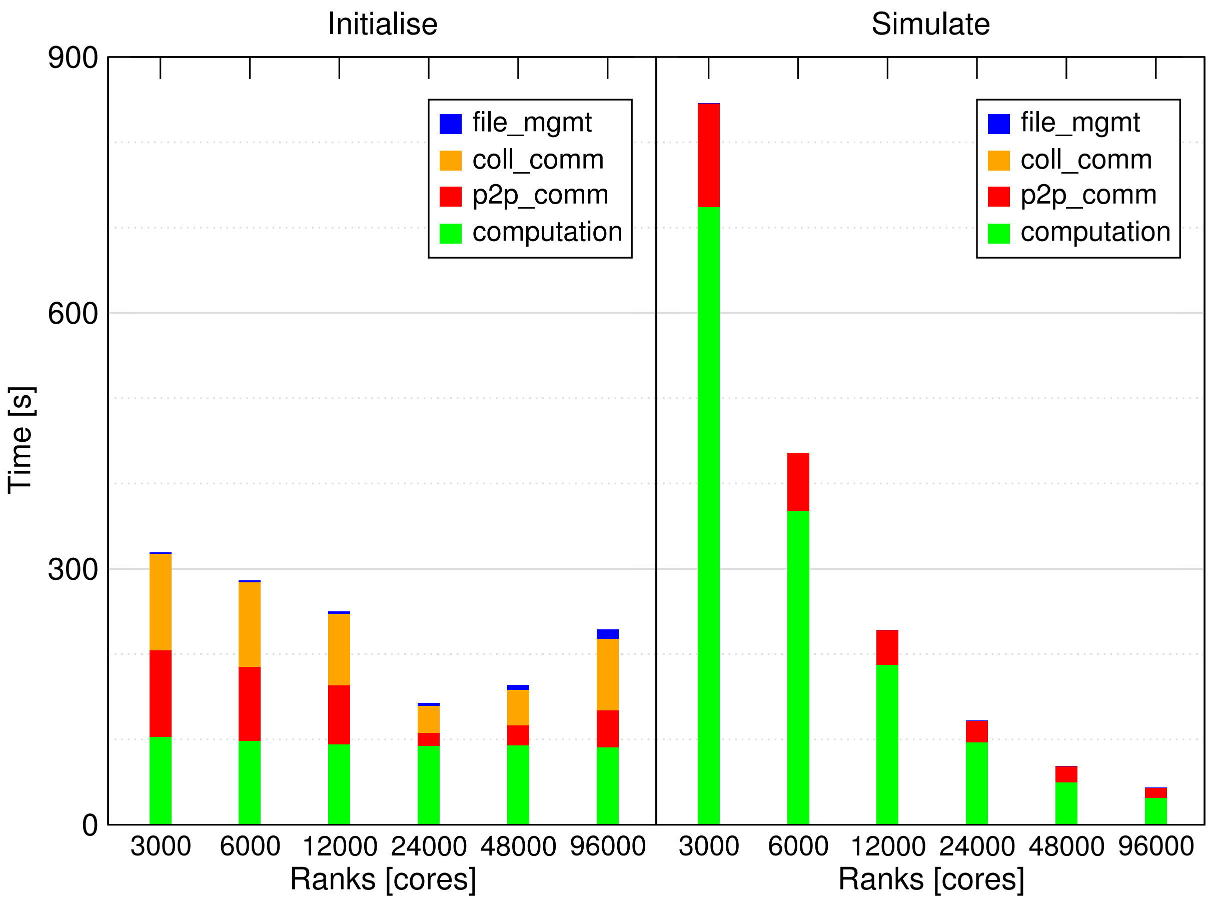


Figure 4: Breakdown of time in *Initialise* and *Simulate* phases of HemeLB CoW15 execution (without Zoltan+ParMETIS).

9 File I/O

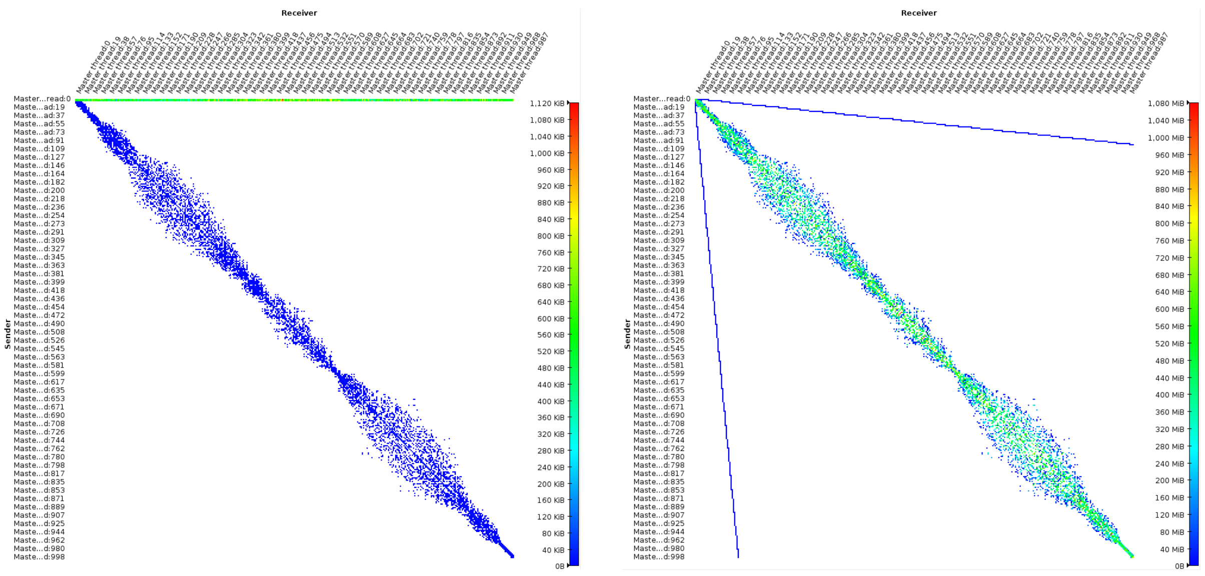
Simulation data writing was disabled for this initial audit, therefore only the file reading during HemeLB initialisation is analysed. MPI File operations are used with individual (rather than collective) reads.

Each rank opens the configuration file (`config.xml`), and rank 0 reads it in 2–3 seconds. All ranks also open the geometry data file `CoW15.gmy`, requiring over 12 seconds when all 96000 processes do so. Ranks 1 to 4 additionally require around 10 seconds to read their share of the 2113 MB geometry data from `CoW15.gmy`.

Although only a relatively small contribution to the increasing inefficiency of *Initialise*, it would seem preferable for either all ranks to read their share of the geometry data directly or only the ranks which actually read the file should open it.

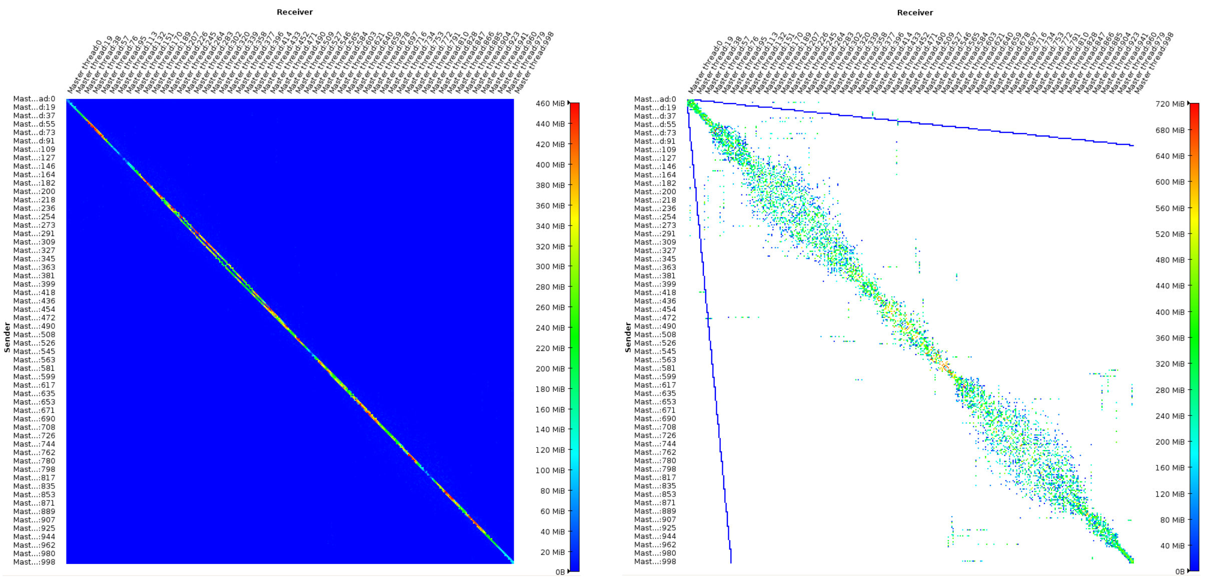
10 Memory utilisation

ARCHER standard compute nodes have 64 GB, therefore at most 2.67 GB available per process with 24 MPI ranks per node. Some of this will be reserved by the OS, and some ranks may have access to more if others on the same compute node require less. Each process' maximum memory usage (RSS) is reported by HemeLB, and would need to be aggregated per compute



(a) *Initialise* phase [1.12 MiB scale]. (b) *Simulate* phase [1080 MiB scale].

Figure 5: Communication matrices of point-to-point aggregate message volume for HemeLB CoW15 testcase execution *without Zoltan+ParMETIS* with 1000 MPI processes on ARCHER.



(a) *Initialise* phase [460 MiB scale]. (b) *Simulate* phase [720 MiB scale].

Figure 6: Communication matrices of point-to-point aggregate message volume for HemeLB CoW15 testcase execution *using Zoltan+ParMETIS* with 1000 MPI processes on ARCHER.

node to determine how close to the limit executions are.

The mean process (maximum) memory usage is around 2.3 GB without Zoltan+ParMETIS [and 2.5 GB or more when Zoltan+ParMETIS is used]. The process with the maximum memory usage requires 2.43 GB [2.7 GB] with 3000 processes and 2.56 GB for 96,000 processes.

The larger memory requirement when using Zoltan+ParMETIS typically require using the 374 large-memory compute nodes of ARCHER which have 128 GB. Unfortunately combining them with standard nodes is not supported. Furthermore, executions with 6000 processes (on 250 large-memory compute nodes) failed reporting insufficient memory available.

11 Summary of observations

Scalability and parallel efficiency of HemeLB on ARCHER Cray XC30 was investigated with the CoW15 dataset up to 96,000 MPI processes (4000 compute nodes).

- Strong scaling is generally very good for the *Simulate* phase, with a speed-up by a factor of 20, whereas the increasingly inefficient *Initialise* phase quickly becomes a bottleneck.
- Parallel efficiency of 80% is maintained by *Simulate* to 24,000 processes. Load imbalance increasingly degrades overall performance as communication efficiency stays high.
- Executions with 48,000 processes (2000 compute nodes, roughly 40% of the entire ARCHER resource) may be prone to significant run-to-run variability. One instance was encountered with a slowdown of MPI communication in both *Simulate* and *Initialise* by more than a factor of two (shaded circles in Figure 3(a)), likely due to network interference from other applications. Very large configurations, such as 96,000 processes (4000 compute nodes, roughly 80% of ARCHER) may also occassionally be impacted, however, this is perhaps less likely since they require most of the available resources themselves.
- While the memory available on standard compute nodes of ARCHER is fully used by HemeLB for the CoW15 testcase, there is generally insufficient additional memory to be able to incorporate the Zoltan+ParMETIS optimised domain decomposition. Although 2.7 GB was sufficient for 3000 processes [125 compute nodes], with 6000 processes [on 500 128 GB compute nodes] the available memory was apparently exceeded.
- For the smaller scale executions which were possible with Zoltan+ParMETIS incorporated in this strong scaling test, *Simulate* showed little performance benefit and even slight degradation with 12,000 processes as *load balance* is worse.

Recommendations

- Investigate computation efficiencies of individual processors with hardware counter measurements, e.g., cache performance, vector instructions, branch instructions, etc., to determine the origin of the high resource stalls.
- Investigate why load imbalance is not improved with Zoltan+ParMETIS optimised domain decomposition, and generally try to improve it.
- Investigate memory allocations, particularly when using Zoltan+ParMETIS, and try to identify how the memory requirements can be reduced to enable larger simulations.
- Investigate file I/O both reading during *Initialise* and writing of simulation output.
- Profile the *Simulate* phase (and/or potentially *Initialise*) in more depth by instrumenting more classes/methods.
- Consider whether incorporation of shared-memory multi-threading using OpenMP within compute nodes would enable more efficient exploitation of available memory by avoiding replication of data.

One or more of these recommendations could be pursued via follow-up POP services, such as a Performance Plan or Proof-of-Concept prototyping.