D5.1.5 – Pre-processing: final tools for exascale mesh partitioning and mesh analysis

WP5: User tools

Project Acronym	CRESTA
Project Title	Collaborative Research Into Exascale Systemware, Tools and Applications
Project Number	287703
Instrument	Collaborative project
Thematic Priority	ICT-2011.9.13 Exascale computing, software and simulation

Due date:	M30
Submission date:	31/03/2014
Project start date:	01/10/2011
Project duration:	39 months
Deliverable lead organization	DLR
Version:	1.0
Status	Final
Author(s):	Gregor Matura (DLR), Derek Groen (UCL), Sebastian Schmieschek (UCL), Adam Peplinski (KTH)
Reviewer(s)	Rupert Nash (UEDIN), Jan Astrom (CSC)

Dissemination level	
PU	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	14/02/2014	First version of the deliverable	Gregor Matura (DLR)
0.2	20/02/2014	Added section 4.1	Derek Groen (UCL), Sebastian Schmieschek (UCL)
0.3	21/02/2014	Completed section 4	Gregor Matura (DLR)
0.4	25/02/2014	Added section 5.1	Adam Peplinski (KTH)
0.5	28/02/2014	Added section 3, 2, 1; completed section 5, internal review version	Gregor Matura (DLR)
0.6	21/03/2014	Revision based on internal reviews	Gregor Matura (DLR)
1.0	25/03/2014	Final version of the deliverable	Gregor Matura (DLR), Derek Groen (UCL), Sebastian Schmieschek (UCL), Adam Peplinski (KTH)

Table of Contents

Index of Figures

Index of Tables

1 Executive Summary

CRESTA deliverable D5.1.3 [3] introduced PPStee in a first prototype version. The pre-processing interface PPStee is designed to balance the load of the overall simulation. It specifically includes all simulation parts, thus extends load balance from simulation core to pre-processing and post-processing tasks, and visualisation. Well-known third-party partitioning libraries are used to calculate the data distribution that is required for the load balance. Deliverable 5.1.4 [4] applied a first analysis of PPStee's features and performance results and marked points where further investigation would be crucial. The most important points were test runs on significantly higher core counts and the integration of PPStee into another CRESTA co-design code. Both are addressed in the document.

This software deliverable D5.1.5 provides a feature-finalised version of the software PPStee. PPStee source code and its documentation can be downloaded from the CRESTA Subversion repository (/wp5/preprocessing). We have eliminated a number of bugs and included further software tests to expand compliance with CRESTA's software standards. Based on our observation of HemeLB using PPStee, we spotted parts that needed improvement; hence we implemented Zoltan query functions and fully revised the weights management. However, we focussed on both investigation hints provided by the analysis in D5.1.4. First, we performed test runs of HemeLB with PPStee on higher core counts and larger geometries, giving better insight into the scalability. Second, we integrated PPStee into the development version of CRESTA's co-design vehicle NEK5000 that implements p4est [10], i.e. a tool for mesh analysis and mesh manipulation. We present here the first performance results.

2 Introduction

The analysis of exascale systems, data formats and algorithms as well as the pre-processing mechanisms in CRESTA's co-design applications (cf. deliverable 5.1.1 [1] and 5.1.2 [2]) led to the design of the pre-processing interface PPStee. PPStee was introduced in deliverable 5.1.3 [3], where PPStee's properties were depicted and basic usage examples were given. In deliverable 5.1.4 [4], we highlighted PPStee's features and its integration into the co-design application HemeLB. This showed the ease with which PPStee can be integrated into a simulation code. We also demonstrated that partitioning libraries can be swapped as easily as it was intended. The first HemeLB test runs, which we reported therein showed no general overhead in runtime or memory.

This first, prototype version of PPStee showed that the main functionality was present, but required additional work. Only the use of software brings hidden bugs to the surface and good software engineering practice demands more elaborate software tests. A prototype version must be studied in practically relevant test cases to spot missing features and situations where performance can be improved. Finally, and in particular regarding PPStee and its application area in exascale simulation codes, the scalability of the software to very large core counts and large geometry data is an open question.

The purpose of this document is to introduce a feature-finalised version of the pre-processing tool PPStee. Section 3 gives a short overview of PPStee and PPStee's design concepts. Recent changes as compared to the prototype version in deliverable 5.1.3 are listed. Section 4 describes the progress in the work on HemeLB using PPStee. Especially, further test runs with higher core counts and larger geometries are presented. Finally, section 5 focusses on the integration of PPStee into CRESTA's development version of the simulation code NEK5000. First runtime results are given.

3 PPStee

3.1 Overview

PPStee was designed with a clear main target in mind: to balance overall simulation load. This leading aspect of an exascale pre-processing strategy resulted directly from the analysis in CRESTA Deliverable 5.1.1 [1], which calls for a tighter integration of pre-processing into the simulation cycle. Communication and computation costs of all simulation parts such as the scientific kernel or the visualisation methods, should be provided to and included in the load balance calculation. Naturally, compatibility should be ensured between the data format of the costs and the data format of the underlying graph, as well as between both data formats and the partitioning libraries used. Furthermore, both data formats should be kept as minimal as possible to keep the memory footprint low (cf. CRESTA deliverable 5.1.2 [2]). Using these costs and the graph data, partitioning tools that implement a state-of-the-art method for graph partitioning should perform the load balance calculation so that improved simulation efficiency may be achieved.

The first prototype version of PPStee, delivered in the context of CRESTA deliverable 5.1.3 [3], already satisfies most of the required properties. PPStee focuses on three well-established third-party partitioning libraries, namely ParMETIS [5], PTScotch [6] and Zoltan [7]. The user can easily set or even swap the partitioning library that is to be used. Accordingly, the data format of the input to PPStee is kept flexible enough to maintain the compatibility among all partitioning libraries. Nevertheless, the data format is designed to be minimal to keep memory requirements as low as possible. Furthermore, PPStee allows submission of costs of several distinct simulation stages. The costs are treated as weights to the submitted graph and allow for a steerable fine-grained load balance of the complete simulation cycle. Moreover, PPStee can be integrated quite easily into an existing simulation, especially, if a partitioner is already used for at least a part of the simulation. In summary, PPStee is a thin additional software layer computing a well-balanced data distribution for a multi-part simulation via various interchangeable partitioning libraries.

3.2 Software status

3.2.1 Zoltan support

The current PPStee version now fully supports Zoltan's partitioning method as well as Zoltan's graph data mechanism, including every possible combination of graph data type and partitioning library. In contrast to the other two partitioning libraries, Zoltan [7] has a different concept for graph data retrieval. ParMETIS [5] uses a minimal set of graph data provided in data arrays. PTScotch's approach is basically the same, but also provides some extensions for non-contiguous arrays and ghost vertices (i.e. vertices not residing on the owning core) [6]. Zoltan, on the other hand, requests various query functions that must be submitted. Therefore, additional conversion functions were needed in PPStee. Firstly, query functions that are passed to Zoltan if normal graph data were passed to PPStee, e.g. if ParMETIS graph data is provided and Zoltan's partitioning method should be used. Secondly, if Zoltan-type query functions were passed to PPStee, functions calling the provided query functions thereby retrieving the required normal graph data. This happens, for example, if the simulation implements its graph data with Zoltan-type query functions but requests a partitioning calculated by PTScotch.

3.2.2 Weight management

Not all stages (i.e. simulation phases registered to PPStee) need both, vertex and edge weight types. Some stages require only vertex or edge weights, and some stages do not require any weights at all. In contrast to the former version (cf. D5.1.3 [3]), PPStee now supports default, i.e. uniformly distributed weights. This way, memory used to save weights data is not consumed unnecessarily. Still, separate stages can be registered independently. Also, PPStee supports detached weights, meaning that vertex and edge

weights can be provided separately. Naturally, default weights and detached weights can be used concurrently in each stage.

In addition to the type of weights, the actual implementation of PPStee handles the problem of multiple weights. Currently only ParMETIS supports multiple “phases” directly by accepting multidimensional vertex and edge weight arrays. Zoltan, on the other hand, was designed to support multiple phases, however it does not implement this feature at the moment. Thus, functions for weight coalescence are needed. There are two approaches to merge all stages into one. The coalesced one-stage weights can be the sum or the maximum of the appropriate multi-stage weights. Since the maximum does disregard some of the weights, PPStee implements a summation to merge multiple stage weights.

3.2.3 Test suite

The prototype version of PPStee provided only a generic system test. Now, full integration into the CMake build system using CTest is available. A tool for automated generation of test graph data is provided and used to create distinctive test for individual usage sequences of PPStee. These tests cover scenarios where a specific partitioning library is called with the appropriate partitioner-native form of graph data input, or with a different graph data layout after an intermediate conversion. Tests of the weight management with various stages are included, some of them dropping certain weight types or weights completely.

4 PPStee and HemeLB

4.1 HemeLB geometries

In this section we describe the geometries used for our performance tests. The scalability and decomposition performance of HemeLB is in part geometry-dependent, and indeed we have observed different trends in load balancing between simplified geometries (e.g. cylinders), and realistic geometries (e.g. vessel networks). As such, we have chosen to restrict ourselves to realistic geometries for the purpose of this performance study.

Within this work we use four data sets, based on three geometries. These are:

- *Bifurcation_50um*, which is a 3D model of an arterial bifurcation, discretised with a voxel size of 50 micrometres.
- *Aneurysm_0.05mm*, which is a highly sparse model of an aneurysm geometry, also discretised with a voxel size of 50 micrometres.
- *Aneurysm_0.025mm*, which uses the same geometry as above, but discretised with a voxel size of 25 micrometres.
- *Smooth_5*, which is a recently acquired data set of a middle cerebral artery. It has been discretised at a voxel size of approximately 28 micrometres.

Below we provide a summary of key characteristics for each of the geometries:

Name	# of lattice sites	Voxel size [10^{-6} m]	% fluid sites in bounding box	Sites per core for 512 cores
Bifurcation_50um	650,492	50	10	1270
Aneurysm_0.05mm	708,472	50	1.5	1383
Aneurysm_0.025mm	5,667,778	25	1.5	11070
Smooth_5	3,907,822	28	11	7632

Table 1: Key characteristics of the four HemeLB geometries.

4.2 Simulation runs on ARCHER

4.2.1 ARCHER

ARCHER is a Cray XC30 supercomputer providing 3008 nodes accompanied by a number of additional components like high-performance parallel filesystem, pre- and post-processing facilities, external login nodes and a large resilient, long-term data facility. Each node contains two 2.7 GHz, 12-core E5-2697 v2 (Ivy Bridge) series processors connected via two QuickPath Interconnect (QPI) links. Standard ARCHER compute nodes have 64GB of memory shared between the two processors and arranged in a non-uniform access (NUMA) form with a region size of 32GB.

4.2.2 Measurements

All runtime measurements were performed on ARCHER using fully-populated nodes adding up to the indicated core count. In terms of ARCHER's 24-core nodes, this means values of 48, 96, and multiples up to 12,288 cores for our experiments. We used a modified HemeLB that incorporates PPStee v0.3.0c; a detailed description of the integration of PPStee into HemeLB can be found in section 4 of CRESTA Deliverable 5.1.4, [4]. We used a default value of 10,000 simulation steps for all geometries apart from "bifurcation_50um", where only 1,000 simulation steps were used.

Figure 1, Figure 2 andFigure 3 depict total runtimes of the simulation, i.e. these measurements include all simulation parts starting with initial read-in of the geometry, followed by partitioning, calculations in the scientific kernel and ending with some visualisation methods. To achieve a more detailed picture, we provide two additional figure series: Figure 4, Figure 5 and Figure 6 show time spent for partitioning only, i.e.

how long the call to the partitioning library lasted. On the other hand, Figure 7, Figure 8 and Figure 9 show the calculation time spent on the scientific kernel which includes computation and communication that is needed to find the solution by the lattice Boltzmann solver.

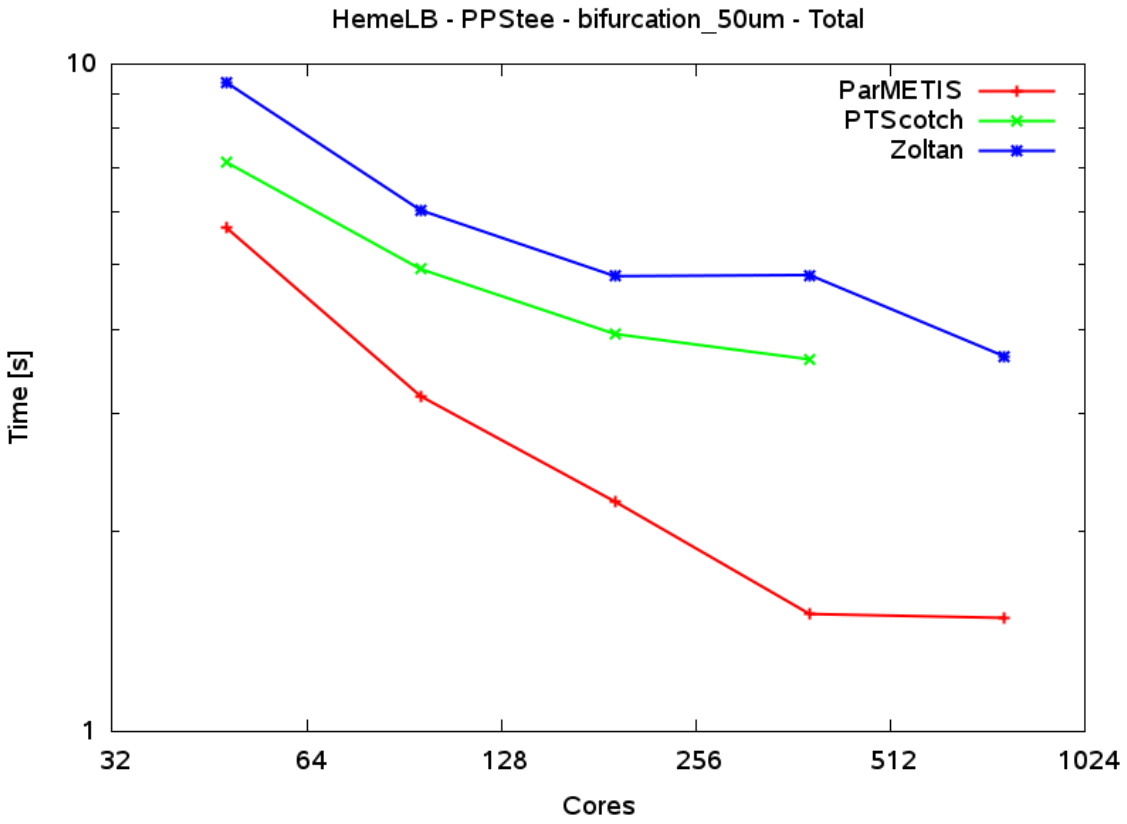

Figure 1: Total runtime of HemeLB on ARCHER for geometry bifurcation_50um with PPStee using one of three partitioning libraries.

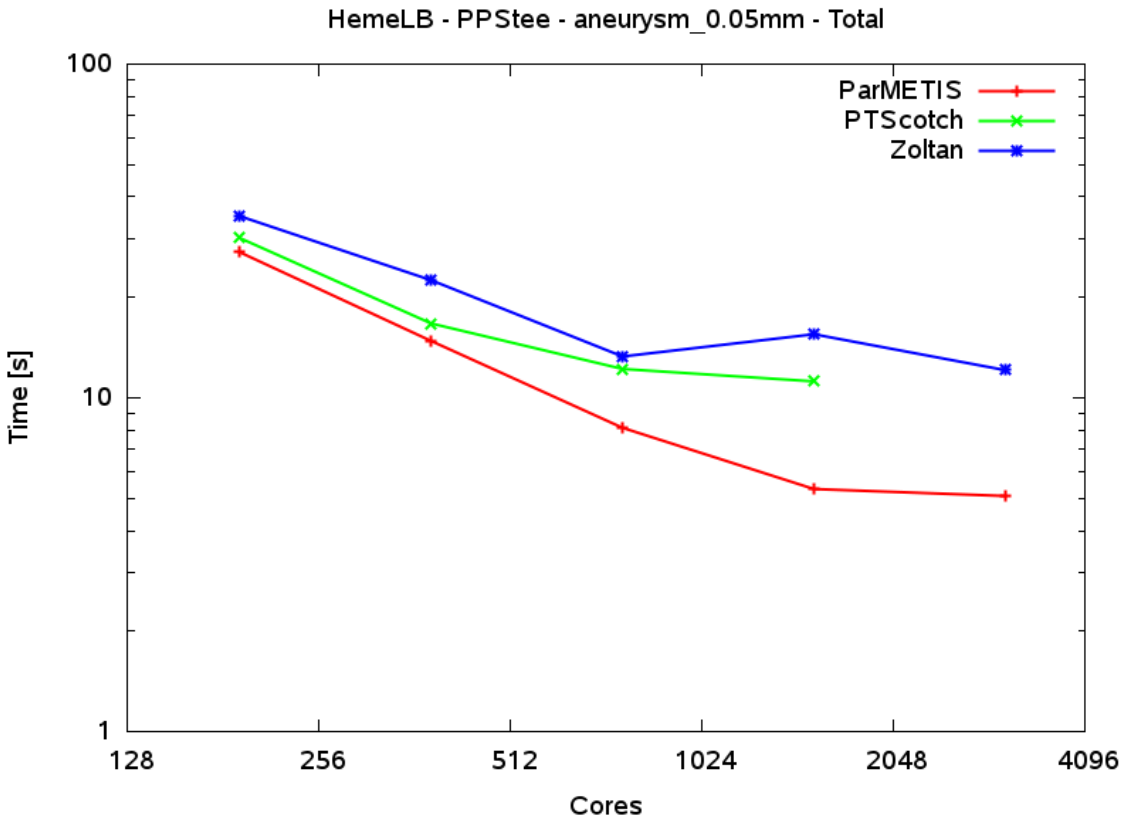

Figure 2: Total runtime of HemeLB on ARCHER for geometry aneurysm_0.05mm with PPStee using one of three partitioning libraries.

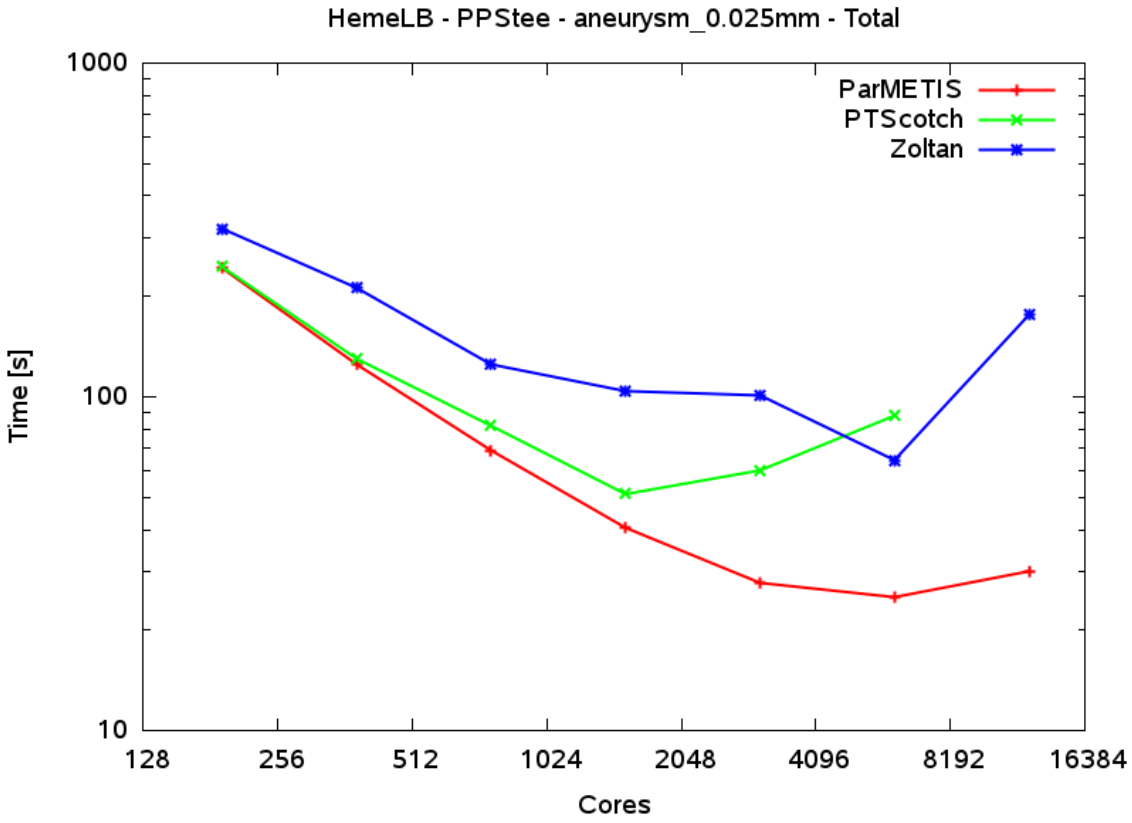

Figure 3: Total runtime of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries.

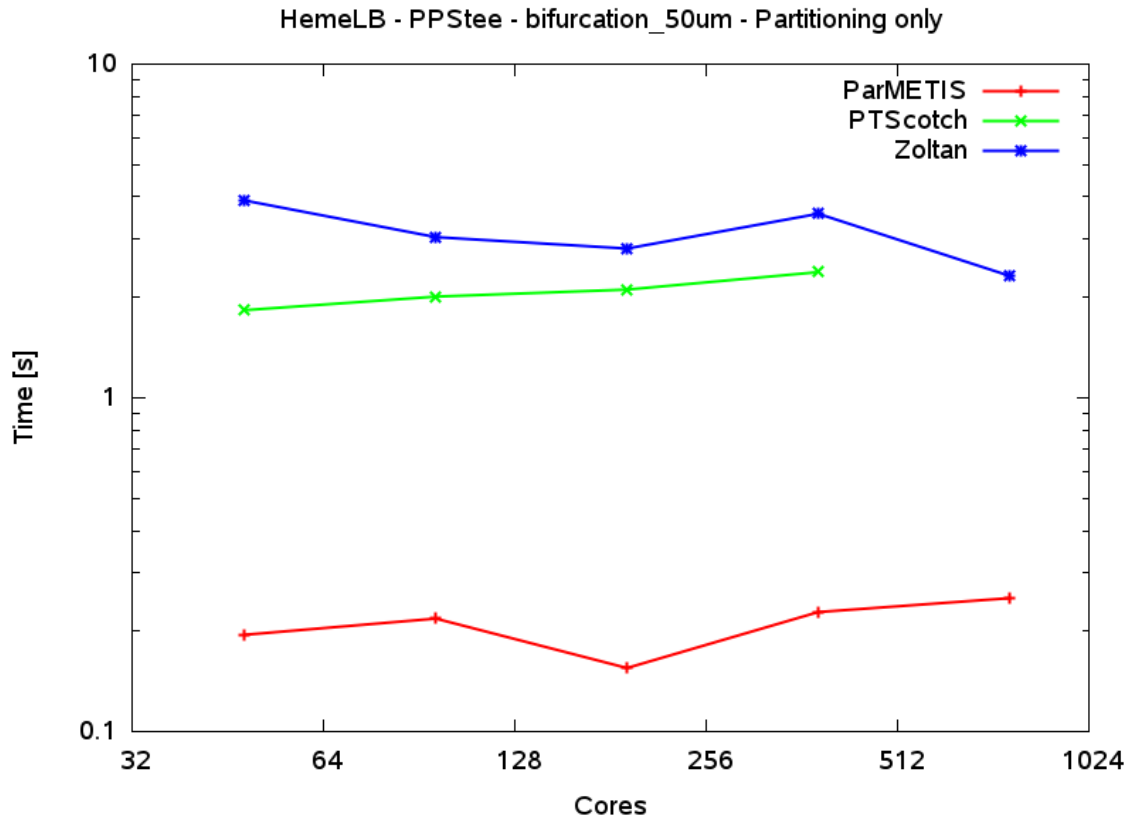

Figure 4: Partitioning time of HemeLB on ARCHER for geometry bifurcation_50um with PPStee using one of three partitioning libraries.

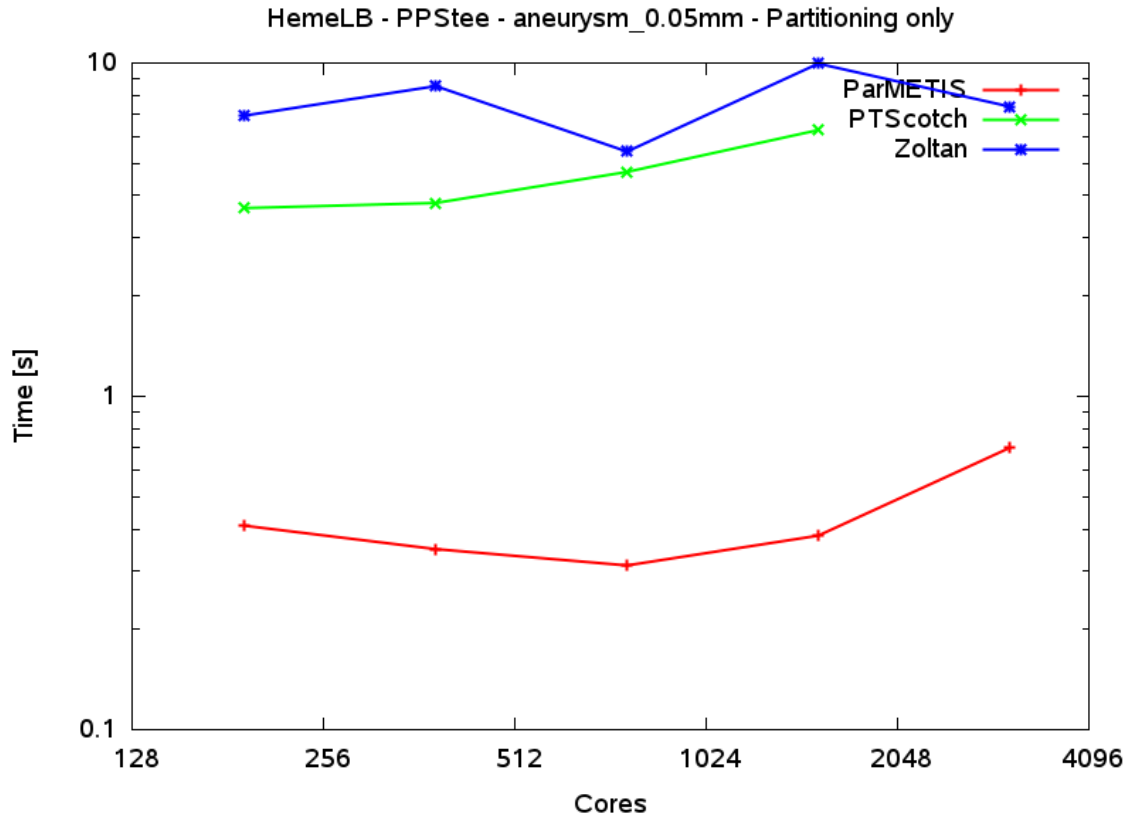

Figure 5: Partitioning time of HemeLB on ARCHER for geometry aneurysm_0.05mm with PPStee using one of three partitioning libraries.

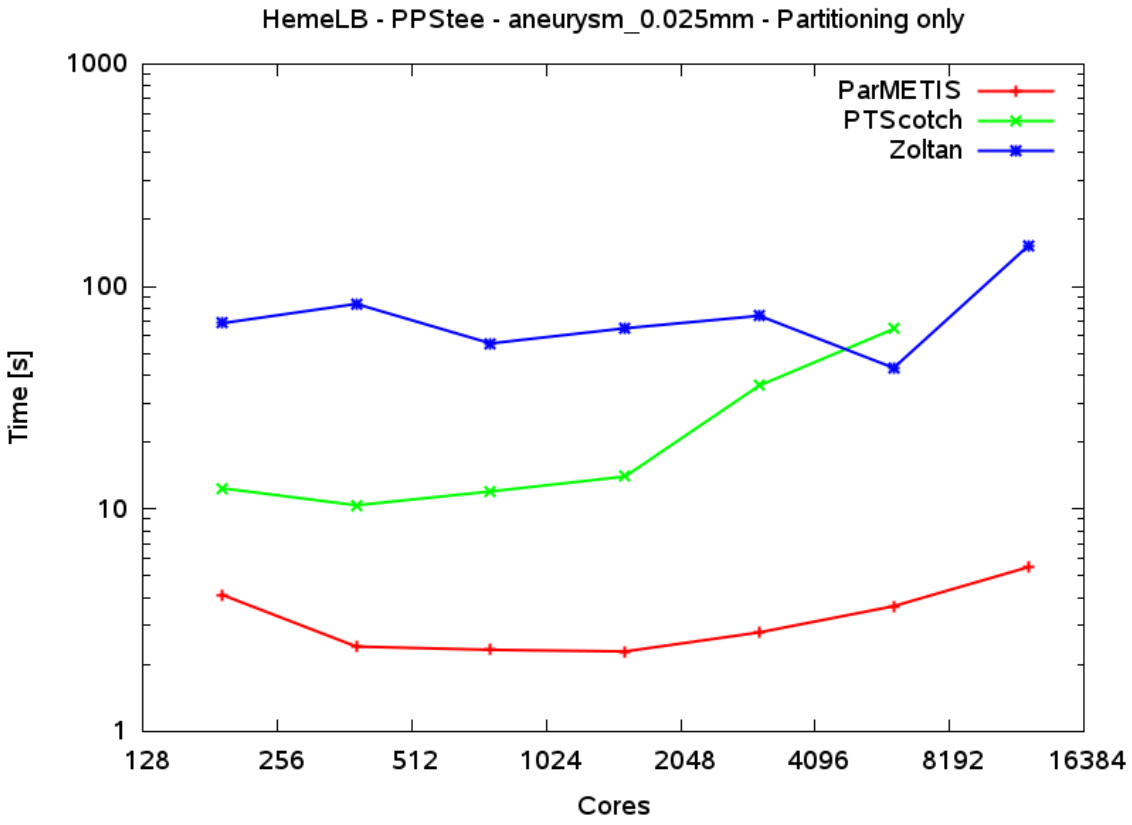

Figure 6: Partitioning time of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries.

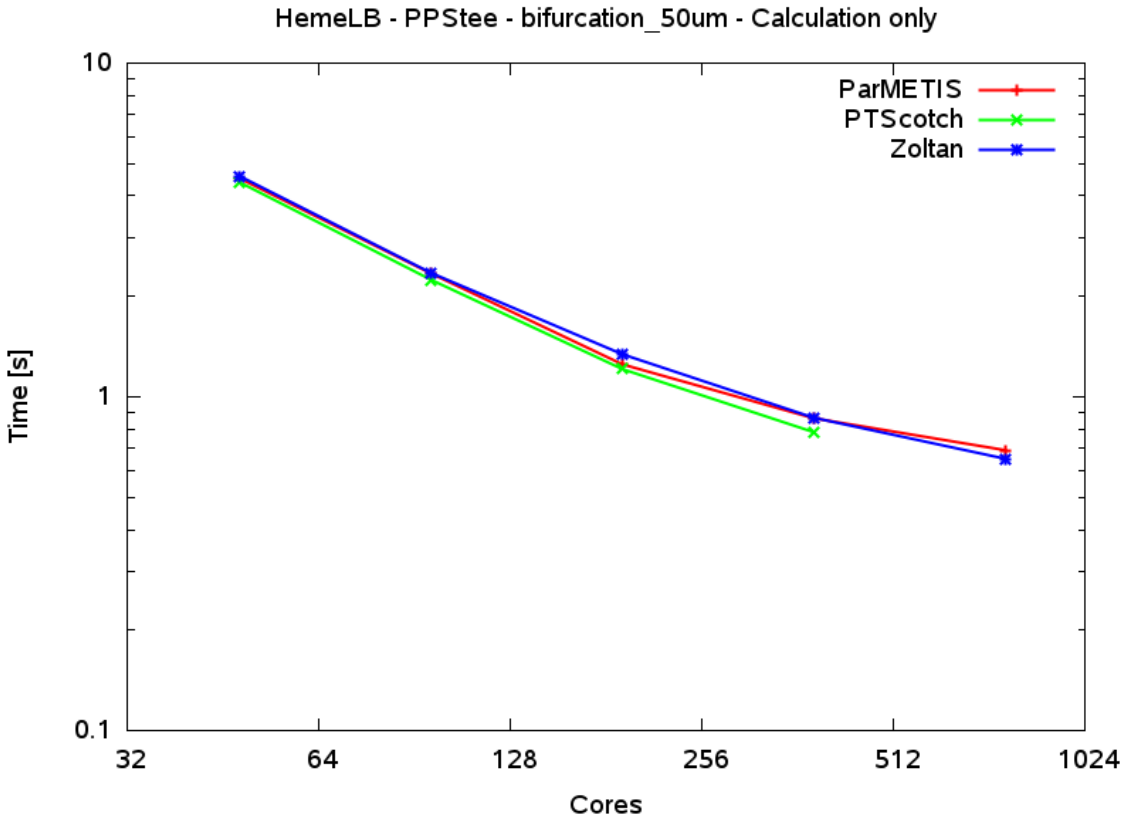

Figure 7: Calculation time of HemeLB on ARCHER for geometry bifurcation_50um with PPStee using one of three partitioning libraries.

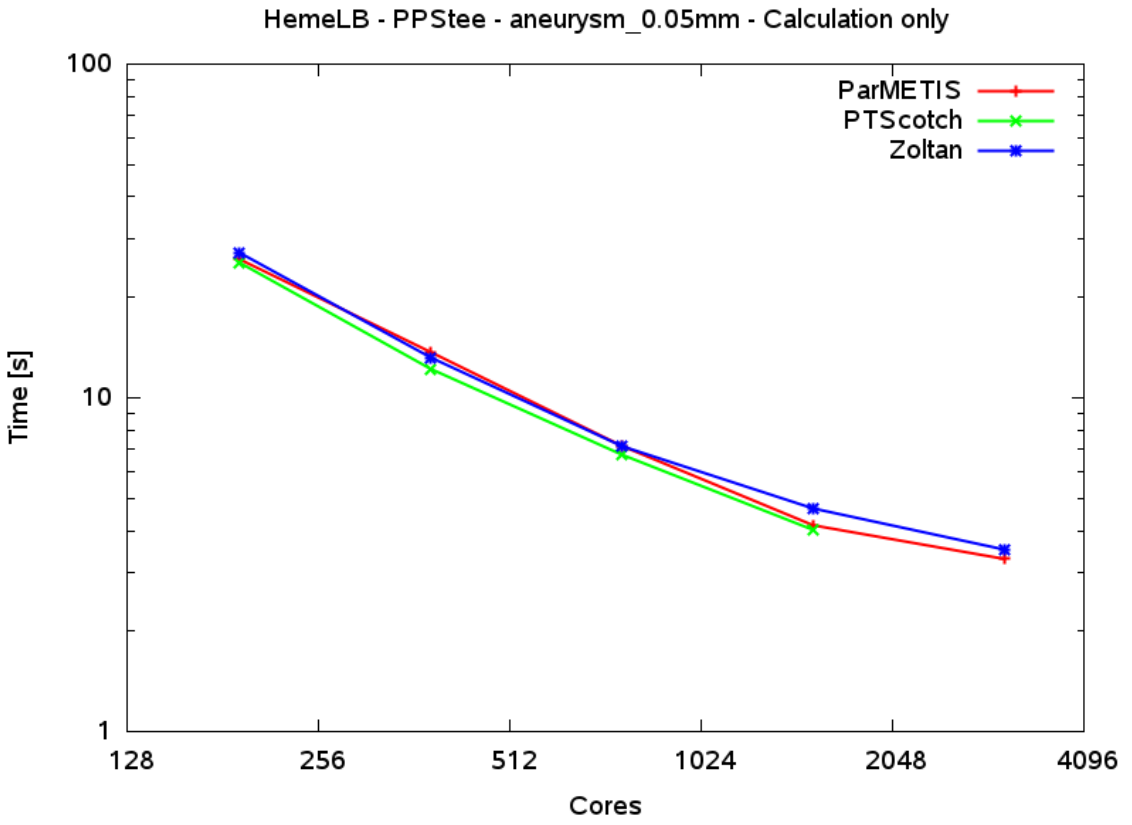

Figure 8: Calculation time of HemeLB on ARCHER for geometry aneurysm_0.05mm with PPStee using one of three partitioning libraries.

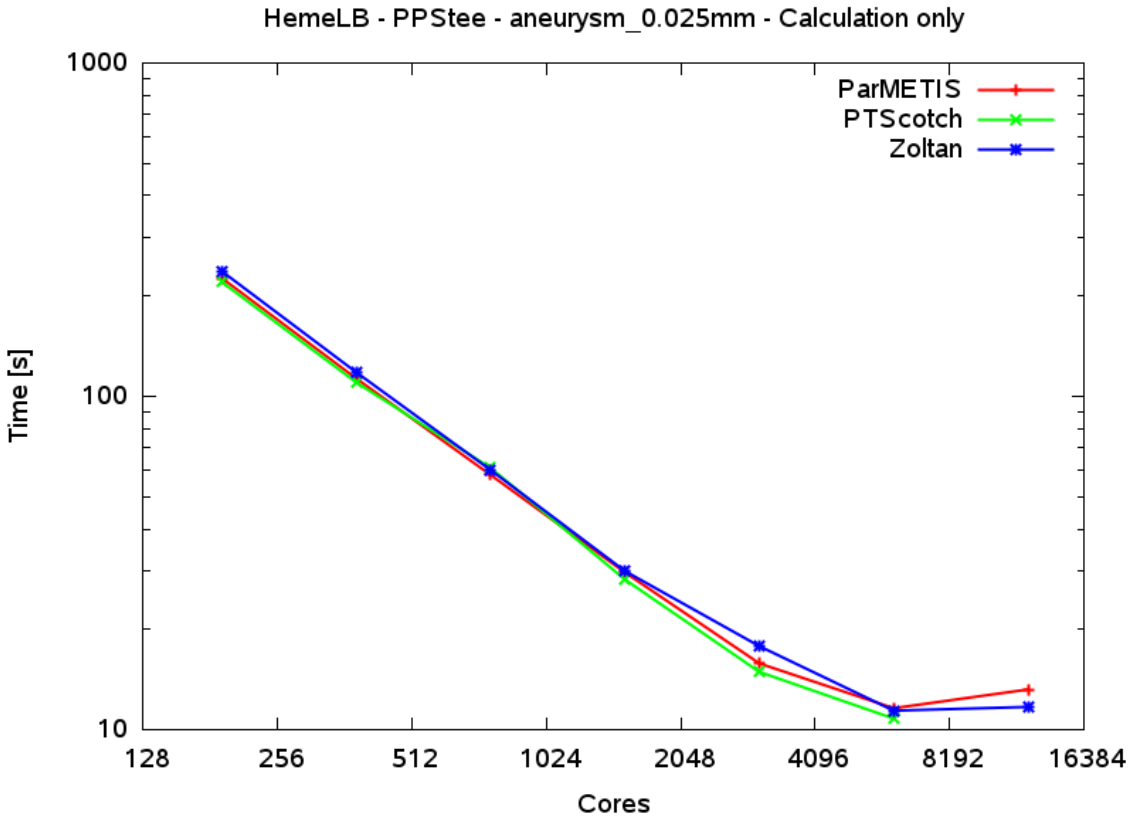

Figure 9: Calculation time of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries.

HemeLB simulation runs using PTScotch fail for the largest-used core counts: for geometry “aneurysm 0.025mm”, the simulation runs out of memory and is terminated

by the system OOM ("Out Of Memory") killer; for geometries "aneurysm 0.05mm" and "bifurcation 50um", PTScotch exits with a floating point exception. The cause has not yet been established.

Cores	Calculation only	Partitioning only	Total
1536	284	127	460
3072	147	184	367
6144	76.3	241	347

Table 2: Runtimes [s] of HemeLB on ARCHER for geometry smooth_5 with PPStee using PTScotch.

Cores	Calculation only	Partitioning only	Total
3072	150	27.3	217
6144	76.2	23.5	135
12288	38.6	17.6	86.4

Table 3: Runtimes [s] of HemeLB on ARCHER for geometry smooth_5 with PPStee using ParMETIS.

Table 2 and Table 3 show runtimes for geometry "smooth_5" of HemeLB with PPStee using PTScotch [6] and ParMETIS, respectively. As with the other three geometries, we list three different timing, i.e. scientific kernel runtime, time spent for partitioning and total simulation runtime.

4.3 Analysis

A first observation concerns the total time measurements in Figure 1, Figure 2 and Figure 3: obviously, ParMETIS [5] performs significantly faster than PTScotch [6] and Zoltan [7] for all geometries inspected and especially for higher core counts. However, this is a very naïve point of view and might lead to misjudgement. The total runtime is influenced by many parameters and thorough separation is needed. Regarding partitioning, two times are of particular interest that are part of the total simulation runtime: the time spent in the partitioning library; and the time spent in calculation in the scientific kernel.

The former is responsible for the computation of the geometry distribution that is used throughout the simulation and this is performed only once. The latter is strongly related to the number of "simulation steps", i.e. the number of iterations in the main loop of propagation of HemeLB. Our test runs were performed with 10,000 steps (and with 1,000 steps for geometry "bifurcation_50um"). On the other hand, current productive simulation runs tend to have 2-3 million time steps. This difference in simulation steps used manifests, approximately, in a factor between 200 and 3,000 for the core computations. Compensating for time lost in the partitioning call, which is done only once, is very possible and should be investigated.

We therefore need to take a closer look at Figure 7, Figure 8 and Figure 9 showing the calculation times of the lattice Boltzmann solver (this includes communication, too). In contrast to the unfavourable results for the total simulation time, PTScotch and Zoltan are on a par with ParMETIS when it comes to calculation time only. Also, small variations are clearly visible, depending on geometry and core counts used. They become notably larger beyond a core count of one thousand. Here, PTScotch does a little better than the other two partitioning libraries. For Zoltan and ParMETIS, it depends on the geometry and core number which one performs better.

On the other side, Figure 4, Figure 5 and Figure 6 show that PTScotch and, particularly, Zoltan perform almost an order of magnitude worse in actually computing the partitioning. Thus it is questionable if PTScotch's and Zoltan's slightly better partition quality can achieve a better performance for the total simulation run. Some preliminary comparisons of time gain through quality versus partitioning time could be

obtained using above figures. However, further investigation is needed to illustrate a detailed picture of these two opposing effects.

Finally, we want to point out that all the figures shown above demonstrate good scalability of all partitioners when used up to 6,000 cores. A decrease in performance for 12k cores for “aneurysm_0.025mm” is evident, but this may be correlated with a geometry that is too small and for which the number of lattice site per core drops below a significant value. Further test runs with bigger geometries could help validate this thesis.

5 PPStee and NEK5000

5.1 Pre-processing in NEK5000

Nek5000 [9] is an open-source code developed at Argonne National Laboratory for the simulation of incompressible flow in complex geometries. It is written in mixed Fortran77/C and uses MPI to employ fully large-scale parallelism, scaling up to a million processes on ALCF BG/Q Mira with parallel efficiency equal to 0.6 (cf. [9], Features > Scaling). The discretisation is based on the spectral-element method (SEM) that combines the higher-order accuracy from spectral methods with the geometric flexibility of finite element methods. In SEM the computational domain is decomposed into a set of disjoint subdomains (elements), which can be transformed to quadrilaterals (2D) or hexahedrals (3D) by general coordinate mapping. The simulated variable space is spanned by Nth-order Lagrange polynomial interpolants, based on tensor-product arrays of Gauss–Lobatto–Legendre quadrature points in every element. This domain decomposition is the main source of parallelism, as loosely coupled elements can be easily distributed among set of processors. It also provides flexibility in grid generation that is used in adaptive mesh refinement algorithm (AMR) implemented in NEK5000 within the CRESTA project.

The current version of the Nek5000 code uses a conformal grid with uniform order of the spatial interpolations throughout the domain. The static grid partitioning based on the dual graph bisection is applied in a pre-processing step to create global element ordering. It is later used during the initialization of the simulation to redistribute the elements among processors and limit the communication volume. For given set of elements the grid resolution can be modified by adjusting the approximation order globally.

There are two basic methods of introducing adaptive mesh refinement: adaptive h-refinement, i.e. the splitting of cells into smaller ones; and adaptive p-refinement, i.e. increasing the polynomial order of a given element. Within the CRESTA project we work on a framework of h-type AMR, which dynamically changes element numbers and their connectivity, and requires dynamic mesh partitioning. Proper load-balancing is crucial for Nek5000 to obtain full scaling for exascale computations. It is especially important for the communication-dominated coarse grid pressure solver [11], where every element is, effectively, represented by a single grid point.

In our implementation the grid refinement and de-refinement is managed by the p4est [10] library, which enables the dynamic management of a collection of adaptive octrees. This library is designed to work in parallel and scale to hundreds of thousands of processor cores. It provides element connectivity information for the dual graph, which is later manipulated by partitioning software, giving a new element-to-processor mapping. We performed initial tests with the ParMETIS library [5] as the partitioner, which showed good parallel performance and a partitioning quality similar to the native NEK5000 static partitioning. However, the element number imbalance was greater in case of ParMETIS than the native NEK5000 partitioning.

5.2 Integration of PPStee

As with HemeLB (cf. section 4.1 in CRESTA deliverable 5.1.4, [4]), the integration of PPStee into the recent version of NEK5000, which was developed and is being used within CRESTA, is straightforward. It does not require deep insights into the simulation code and the necessary code changes are small. In this CRESTA version of NEK5000, there is a call to the partitioning library ParMETIS which has to be located and replaced by the necessary calls to the PPStee routines. A basic example can be found in section 3.1 of D5.1.3, [3].

5.3 Proof of concept

As proof of concept, we integrated PPStee into NEK5000 (in its current CRESTA development version), built it and performed a range of tests. Our test system is a

desktop machine providing an Intel Xeon E5520 with 8 real and 16 virtual cores. We used different thread counts between 1 and 16; we used the minimum time from ten runs for each data set. Figure 10 shows runtime spent in the solver in a simulation run of NEK5000. We used a cylindrical test geometry with 1472 quadrants and PPStee with each one of three partitioning libraries. Figure 11 shows runtime spent only in the partitioning library for the same simulation run. Both figures do not explicitly show the timings of the non-PPStee CRESTA-version code that uses ParMETIS because they are equal to the timings of NEK5000 with PPStee using ParMETIS.

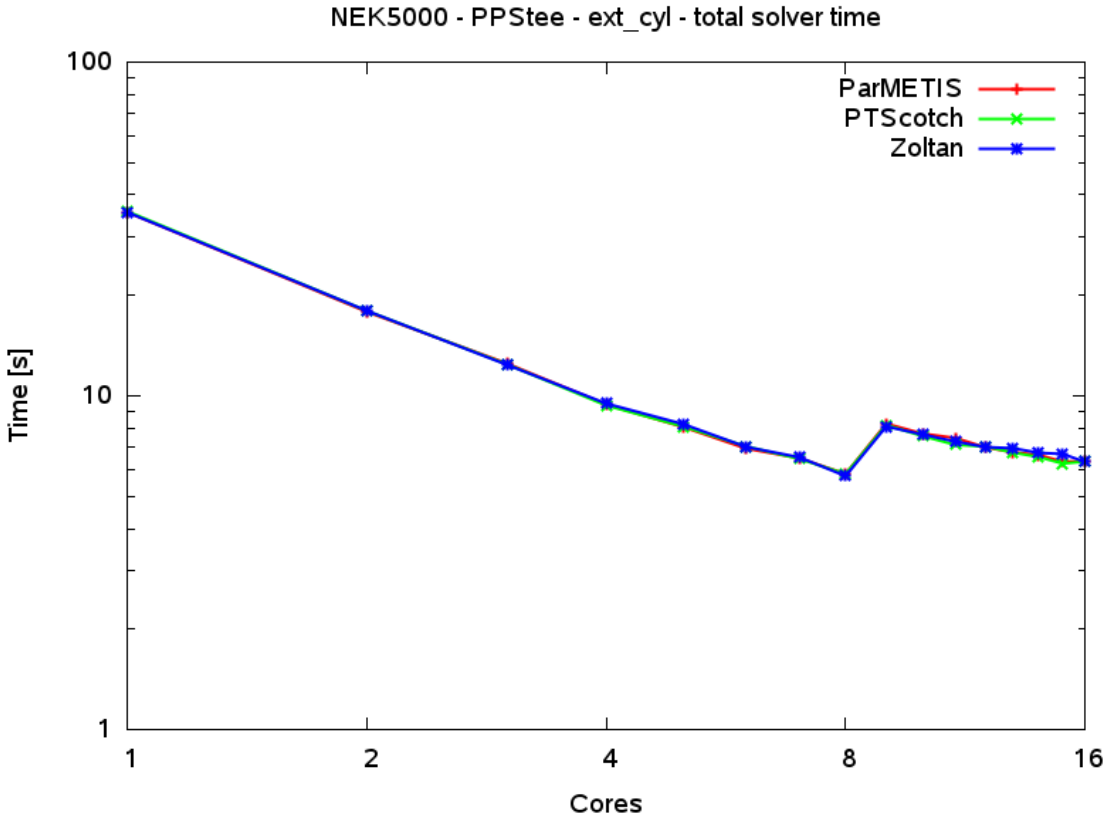

Figure 10: Solver runtime for NEK5000 for a cylindrical geometry on an Intel Xeon E5520 with PPStee using one of three partitioning libraries.

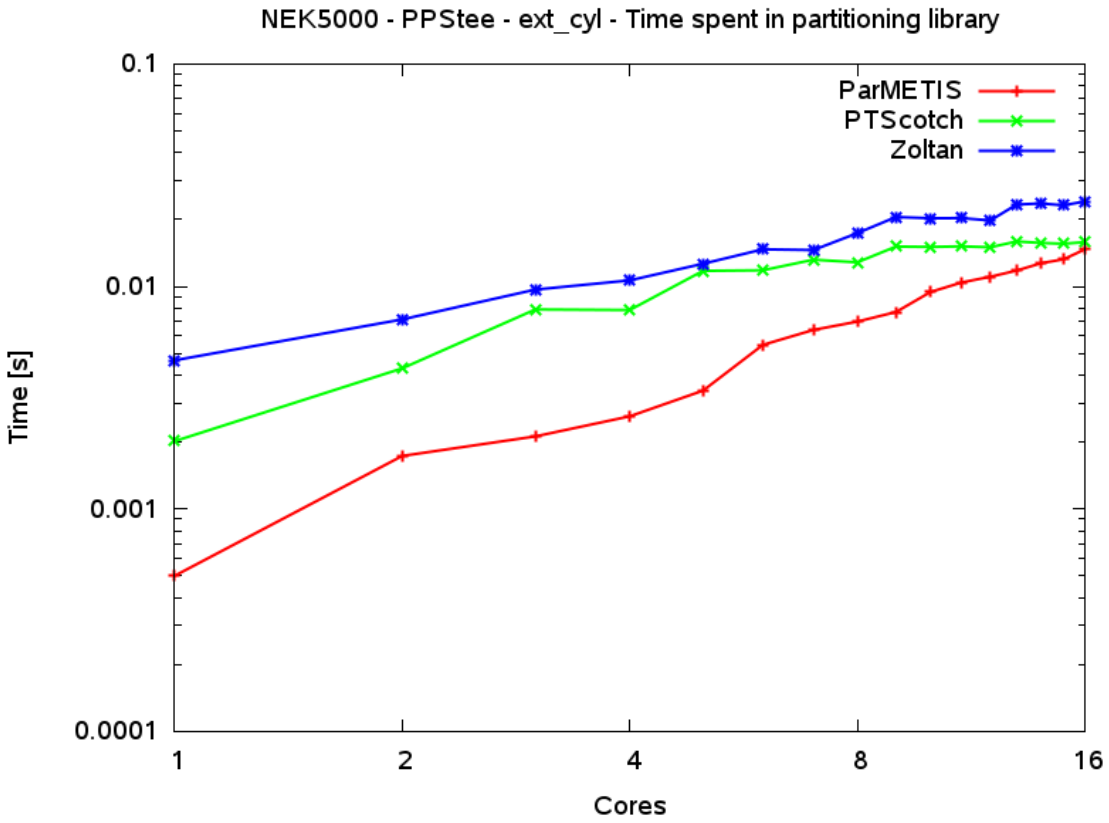

Figure 11: Time spent in partitioning library for a NEK5000 simulation run for a cylindrical geometry on an Intel Xeon E5520 with PPStee using one of three partitioning libraries.

These results show the general applicability of PPStee to a simulation using NEK5000. The integration was as easy, as intended, and introduces the possibility of comparing the partitioning quality of all three libraries with regard to the geometry used. Figure 10 shows that the solver times differ only slightly from each other. Thus, at least for small thread counts, the quality of the calculated data distributions is almost equal. However, Figure 11 points out a possible drawback. Already at these small thread counts, the times measured for the call to the partitioner differ significantly. Thus it may depend strongly on the size of the geometry and the number of executing cores which partitioning library leads to the minimal overall simulation time that includes both, the simulation time and the time spent on calculating the partitioning.

In conclusion, the general behaviour corresponds to the one observed for HemeLB (cf. section 4), i.e. solving times are almost equal yet the time to compute the partitioning vary considerably. Further investigation will show which effect will dominate the NEK5000 simulation for bigger geometries with substantially higher core counts, e.g. acquired on ARCHER.

6 References

[1] CRESTA Deliverable 5.1.1, Pre-processing: analysis and system definition for exascale systems

[2] CRESTA Deliverable 5.1.2, Pre-processing: data format and algorithms

[3] CRESTA Deliverable 5.1.3, Pre-processing: first prototype tools for exascale mesh partitioning and mesh analysis

[4] CRESTA Deliverable 5.1.4, Pre-processing: revision of system, data format and algorithms definition for exascale systems

[5] ParMETIS, *Parallel graph partitioning and fill-reducing matrix ordering,* http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview

[6] PTScotch, *Software package and libraries for sequential and parallel graph partitioning, static mapping, and sparse matrix block ordering, and sequential mesh and hypergraph partitioning,* http://www.labri.fr/perso/pelegrin/scotch/

[7] Zoltan, *Data-Management Services for Parallel Applications,* http://www.cs.sandia.gov/Zoltan/Zoltan_phil.html

[8] ARCHER UK National Supercomputing Service, http://www.archer.ac.uk

[9] Fischer, P., Lottes, J., Kerkemeier, S.: nek5000 Web page (2008). http://nek5000.mcs.anl.gov

[10] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas, p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. Published in SIAM Journal on Scientific Computing 33 no. 3 (2011), pages 1103-1133.

[11] Tufo, H., Fischer, P., "Fast Parallel Direct Solvers For Coarse Grid Problems", J. Par. & Dist. Comput., 61 p. 151-177 (2001).