D5.1.6 – Pre-processing: tool evaluation and investigation with application data

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 Exa-scale computing, software and simulation

Due date:	M39
Submission date:	31/12/2014
Project start date:	01/10/2011
Project duration:	39 months
Deliverable lead organization	German Aerospace Center (DLR)
Version:	1.0
Status	Final
Author(s):	Gregor Matura (DLR), Derek Groen (UCL), Adam Peplinski (KTH)
Reviewer(s)	Jan Åström (CSC), Jing Gong (KTH)

Dissemination level	
PU	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	22/10/2014	First version of the deliverable	Gregor Matura (DLR)
0.2	23/10/2014	Added Sections 4 and 5.	Gregor Matura (DLR)
0.3	24/10/2014	Added Section 3.	Gregor Matura (DLR)
0.4	24/11/2014	Nek5000 input	Adam Peplinski (KTH)
0.5	24/11/2014	HemeLB input	Derek Groen (UCL)
0.6	26/11/2014	Added Sections 1 and 2; version for internal review completed.	Gregor Matura (DLR)
0.7	27/11/2014	Some minor corrections. Added short conclusion section.	Gregor Matura (DLR)
0.8	08/12/2014	Final changes according to the reviewers´ comments.	Achim Basermann (DLR)
0.9	10/12/2014	A few, minor additional changes suggested by Adam Peplinski	Achim Basermann (DLR)
1.0	11/12/2014	Final version for submission	Catherine Inglis (UEDIN)

Table of Contents

Index of Figures

Index of Tables

1 Executive Summary

Following the analysis and system definition of pre-processing for exascale systems in [1] and a study of data formats and algorithms especially tackling CRESTA's co-design applications in [2], the pre-processing interface PPStee was developed and introduced in [3]. It is designed to balance the load of the overall simulation specifically including communication and computation costs of all simulation parts. [4] and [5] provided first implementations into HemeLB [9] and Nek5000 [14] accompanied by simulation runs on up to 12k cores with HemeLB on HECToR [10] and ARCHER [11].

We continued our effort with HemeLB and investigated the results of weighted decomposition. These results of simulation runs on up to 24k cores of ARCHER with geometries containing up to 5.6 million lattice sites are presented and analysed. Additionally, we provide a new example for PPStee application. Together with two small scripts and the HemeLB pre-processing tool *protopart*, this example forms a tool chain that enables a thorough *a priori* analysis of the partitioning and, ultimately, ensemble simulation runs of HemeLB.

Furthermore, we performed simulation runs of the CRESTA-modified version of Nek5000 that includes adaptive mesh refinement and PPStee partitioning. The results of runs on up to 48k cores on ARCHER confirm the simple integration and general applicability of PPStee and its usefulness regarding a comparison of the partitioning quality of the supported partitioning libraries.

2 Introduction

When working with today's simulations, the focus is mostly on the numerical computation of the simulation. This simulation part is, of course, only the main part and a significant amount of work is needed to prepare and process necessary simulation data. The pre-processing phase often contains a domain decomposition. Various mesh manipulation tasks such as creation, generation or refinement can be performed. Following the numerical computation, the post-processing part is constituted of result analysis, visualisation methods together with an appropriate remote rendering technique.

These individual steps that ultimately produce the desired simulation result are usually treated separately and partially even off the high performance system. Thus several bottlenecks are present, e.g. the transmission of possibly huge initial data to the cluster, or the visualisation of the computed results, which may require an additional visualisation cluster.

In the exascale regime, this strategy of separately-handled simulation parts becomes unfeasible. There are simply too many data to transfer or to process. Here, new concepts must be implemented so that bottlenecks are eliminated. In-situ visualisation of the computed simulation results is a well-suited example. Executed right after the computation, the in-situ visualisation exploits full cluster performance on the entire simulation data. Preliminary compression or tedious transmission is not necessary anymore and only a reduced visual output is transferred that still satisfies the desired resolution. Another example for a bottleneck-reducing concept is automated mesh refinement. It is responsible for an automated generation of a fine-grained mesh based on an initial coarse mesh. Only the small coarse mesh is transmitted while the simulation still uses the fine representation of the mesh. Both examples show that, for exascale simulations, all parts of the simulation must merge into one contiguous simulation cycle.

As far as pre-processing is concerned, this merged simulation approach does not change the main task of load-balancing. Now, the load-balancing is expanded to the entire simulation cycle. It must include all parts of the cycle, i.e. a posteriori data analysis, in-situ visualisation and other potential simulation phases like mesh generation or mesh refinement. Obviously, two things are required. First, pre-processing must provide an interface to which all simulation parts can pass their individual load information. Second, pre-processing must combine these load data and compute a load balance that is optimal for the simulation cycle.

In the long run, this extended load-balancing together with its necessary inter-phase messaging mechanism enables further techniques on which future exascale simulations will heavily rely. For instance, the use of scalable immersive Virtual Reality technology to interactively improve mesh structures and mesh quality will require communication between visualisation and pre-processing and inherently imply a recalculation of the data distribution and load balance.

In this deliverable we present our approach to an exascale pre-processing. The pre-processing interface PPStee was developed to calculate an optimal load balance for the complete simulation cycle including all parts like computation and visualisation. Section 3 summarises the intention of the development of PPStee and its features. More detailed information on PPStee was already described in [3], [4] and [5].

Sections 4 and 5 tackle the integration of PPStee into the CRESTA co-design codes HemeLB and Nek5000, respectively. There, we continue our efforts for an improved load balance of the applications and present new simulation runs that are larger with respect to both geometry size and number of cores of the simulation. Specifically regarding HemeLB, we show in Section 4.3 an alternative usage of PPStee that helps to establish ensemble simulation runs of HemeLB.

3 PPStee

The main design goal of PPStee was to develop a software package that balances the load for the full simulation run. Specifically, PPStee supports multiple stages that can reflect different stages of each simulation cycle. Concerning the calculation of the load balance, PPStee is based on well-established partitioning libraries, i.e. ParMETIS [6], PTScotch [7] and Zoltan [8].

This section summarises design objectives and features of PPStee. For detailed information see [3], [4] and [5]. PPStee can be downloaded from the CRESTA Subversion repository at wp5/preprocessing. The repository includes the sources of PPStee, build and installation instructions, software tests, usage examples and related tools and scripts.

3.1 Features

PPStee is a thin intermediate software layer providing access to partitioning libraries that compute a distribution of the simulation data and thus ensure a balance of the simulation load. Currently, the partitioning libraries ParMETIS, PTScotch and Zoltan are supported. PPStee uses a flexible and minimal data format that is compatible with the partitioning libraries (cf. Section 3 in [4]). Using this format, the simulation submits its computation and communication costs to PPStee. Additionally, PPStee is designed to support multiple stages of a simulation with each stage or phase containing its own costs. Hence, the load balance that is computed covers the full simulation cycle and ensures good overall performance.

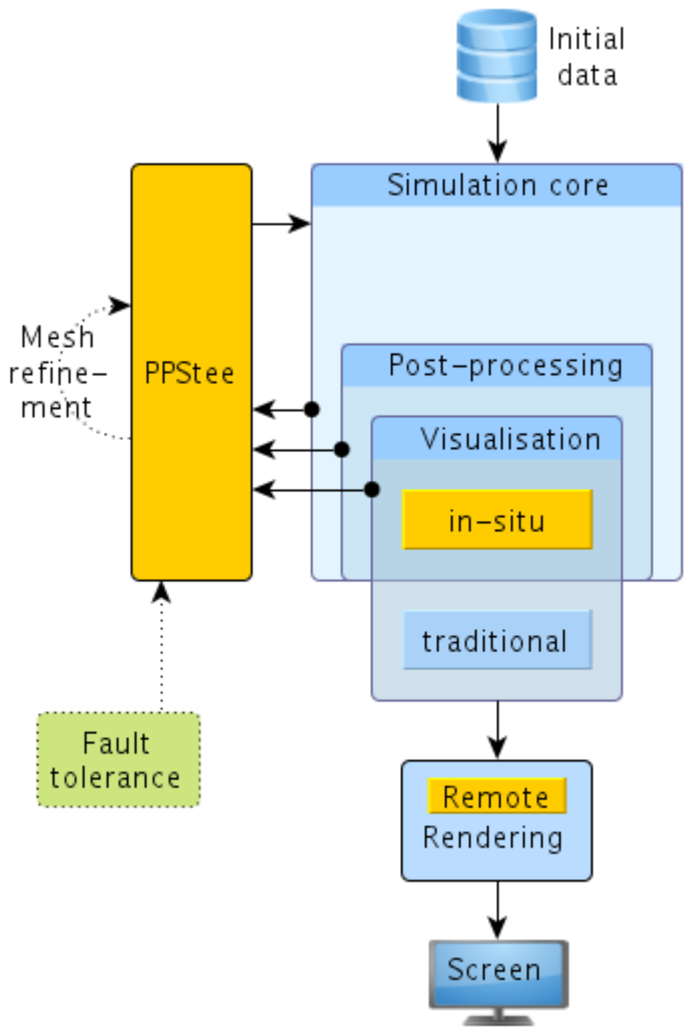

Figure 1: PPStee flow chart

The integration of PPStee into an existing code is simple. It does not interfere with the data flow of the simulation (compare Figure 1) but basically replaces a call to a partitioning library (see Sections 4.1 and 5.1 for a description of the integration of PPStee into HemeLB and Nek5000, respectively). Of course, PPStee imposes only a minimal overhead on the simulation in terms of runtime and memory as runtime measurements show (cf. Section 4 in [4]).

Additional features can be integrated easily into PPStee. For example, any form of additional information on the system might lead to better simulation performance (although not implemented yet). A fault tolerance framework might supply data on failed nodes or cores that must be avoided.

4 PPStee and HemeLB

4.1 Integration and first simulation runs

In CRESTA Deliverable D5.1.4 [4], we described the integration of PPStee into the hemodynamic lattice-Boltzmann simulation HemeLB [9]. We provided a proof of concept showing that the usage of PPStee did not introduce any penalty in runtime over the partitioning that was used before. Additionally, we made runtime measurements with two cylindrical test geometries and a bifurcation dataset on HECToR [10] using up to 2,048 cores. Thus, we proved the applicability of PPStee for HemeLB in general and collected hints for further investigation of the results.

Based on this first analysis, we performed more simulation runs on ARCHER [11]. This time, we scaled up to 12,288 cores (cf. [5]). We used the bifurcation dataset known from before to facilitate a comparison to the former results. Additionally, we used two other datasets that model an aneurysm at two resolutions. To gain more insight into the composition of the total runtime, we split up the timings into time for calculation and partitioning. The former represents the time spent on the scientific kernel that includes computation and all kinds of communication operations that are needed to find the solution by the lattice Boltzmann solver. The latter describes how long the call to the partitioning library lasted.

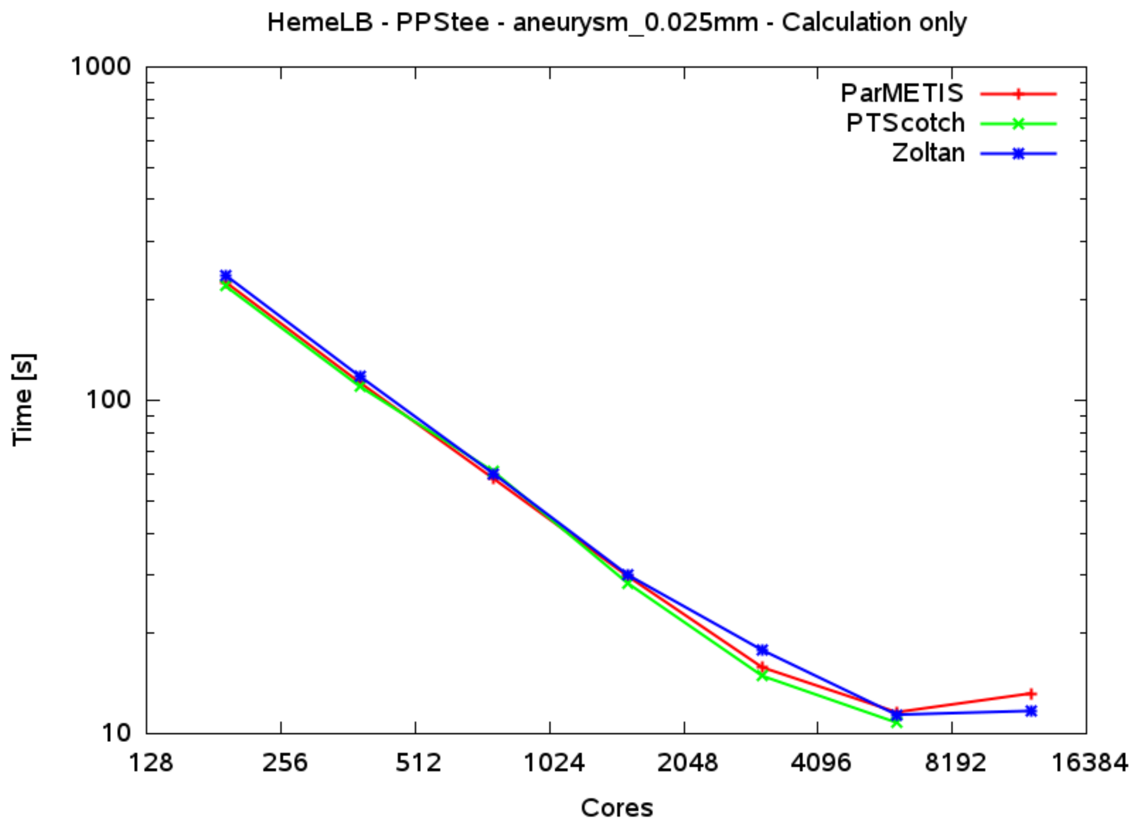

Figure 2: Calculation time of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries. [5]

Figure 2 and Figure 3 are picked from [5] to recapitulate our main findings. In general, we see a calculation time that is almost equal for all three partitioning libraries on all core numbers. This fact points out that all three produce a similarly good quality of partitioning. Yet, there are some small differences that might get bigger for other geometries or on higher core numbers.

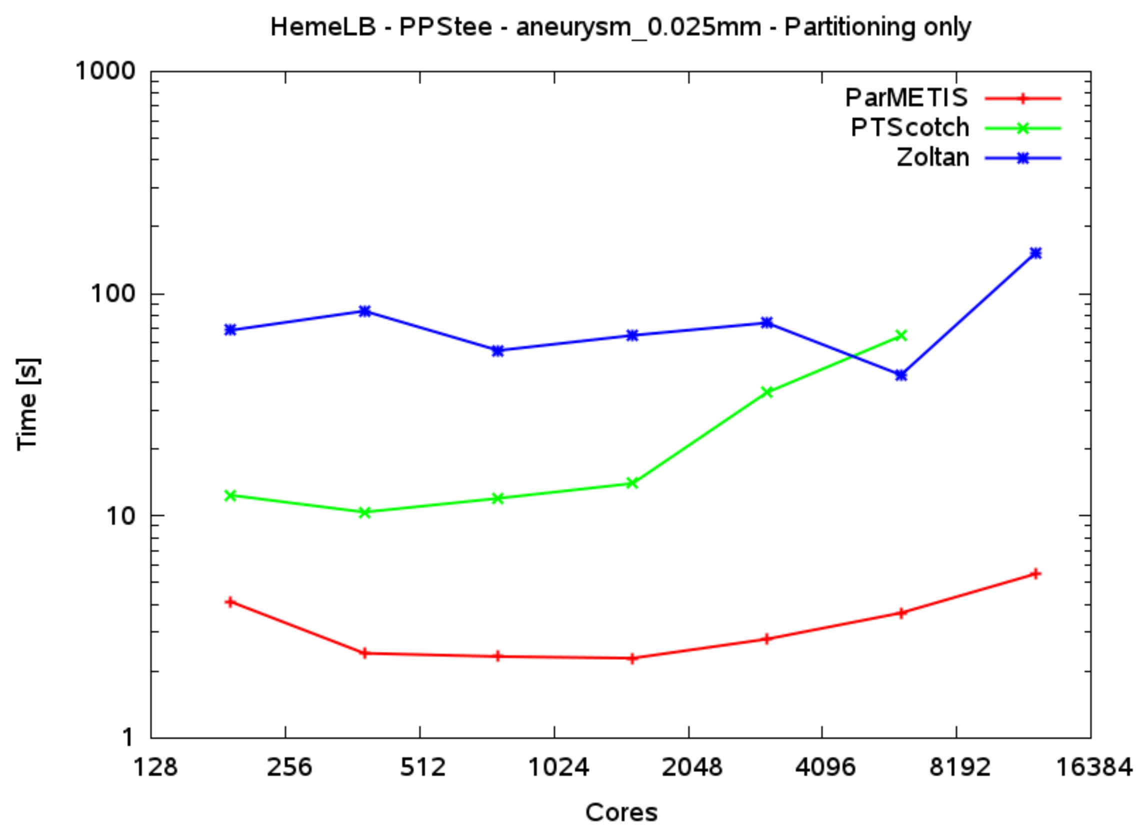

Figure 3: Partitioning time of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries. [5]

On the other hand, the partitioning time depicted in Figure 3 shows significant differences between the three partitioning libraries. ParMETIS is one order of magnitude faster than Zoltan while PTScotch loses ground for higher core counts. As for the calculation time and the correlated quality of the partitioning, this picture might change substantially for other geometries. Nevertheless, the absolute values of the partitioning time give rise to the following conclusion. Especially when dealing with simulation setups that calculate a partitioning after each couple of time steps, i.e. for frequent repartitioning, the partitioning time has to be kept in mind.

4.2 Weighted decomposition

Obtaining a good load balance is a significant challenge in scaling up lattice-Boltzmann simulations of realistic sparse problems to the exascale. Here we analyse the effect of weighted decomposition on the performance of the HemeLB lattice-Boltzmann simulation environment, when applied to sparse domains. Prior to domain decomposition, we assign wall and in/outlet sites with increased weights which reflect their increased computational cost. We have tested our weighted decomposition approach, in conjunction with the different partitioners provided by PPStee, on a sparse bifurcation and very sparse aneurysm geometry. Some of the results presented here are part of an EASC 2014 conference contribution [18].

4.2.1 Description

Within sparse geometries, lattice-Boltzmann codes generally adopt a range of lattice site types to encapsulate all the functionalities required to treat flow in bulk, near walls and near in- and outlets. We provide a simple example of a geometry containing these lattice site types in Figure 4. By default, all types of lattice sites were weighted equally in HemeLB, which means that graph partitioners such as ParMETIS treat all site types with equal importance when creating a domain decomposition. However, we find that both sites adjacent to walls and sites adjacent to in- and outlets require more computational time to be updated. To optimize the load balance of the code, we therefore assign heavier weights to sites which reside adjacent to wall or in/outlet boundaries.

We are currently developing an automated tuning implementation to obtain these computational costs at run-time. However, as a first proof of concept, we have deduced approximate weighting values by running six simulations of cylinders with different aspect ratios. The shorter and wider cylinders have a relatively high ratio of in- and outlet sites, while the longer and more narrow cylinders have a relatively high ratio of wall sites. In addition, the cylinders with an aspect ratio near 1:1 have a relatively high ratio of bulk flow sites.

Based on these runs we have obtained estimated values for the computational cost for each type of lattice site, by using a least-square fitting function. We present the values of these fits, as well as rounded values we use in the respective partitioners, in Table 1. ParMETIS supports the use of weights in graphs, provided that these weights are given as integers. As we found that using large numbers for these weights has a negative effect on the stability of ParMETIS, we chose to normalize and round the weightings such that bulk sites are given a weight of 4, and the other site types are given by values relative to that base value. Because the test runs contained only a very small number of wall + in/outlet sites, we choose to adopt the weighting for in/outlet sites also for the in/outlet sites which are adjacent to a wall boundary.

Table 1: Weight values as obtained from fitting against the runtimes of six test simulations on two compute architectures (Intel SandyBridge and AMD Interlagos). The site type is given, followed by the weight obtained from fitting the performance data of the six runs, followed by the simplified integer value we adopted in ParMETIS. In this work we use Bouzidi-Firdaouss-Lallemand (BFL) wall conditions and in- and customized outlet conditions described in Nash et al, 2014 ([19]). We observed rather erratic fits for the weightings of in/outlet sites that are adjacent to walls, as these made up only a very marginal fraction of the overall site counts in our benchmark runs (less than 1% in most cases).

Site type	Obtained weight		Rounded weight
	Intel	AMD	
Bulk	10	10	4
Wall	18.708	20.226	8
In/outlet	40.037	37.398	16
Wall and In/outlet	22.700	34.577	16

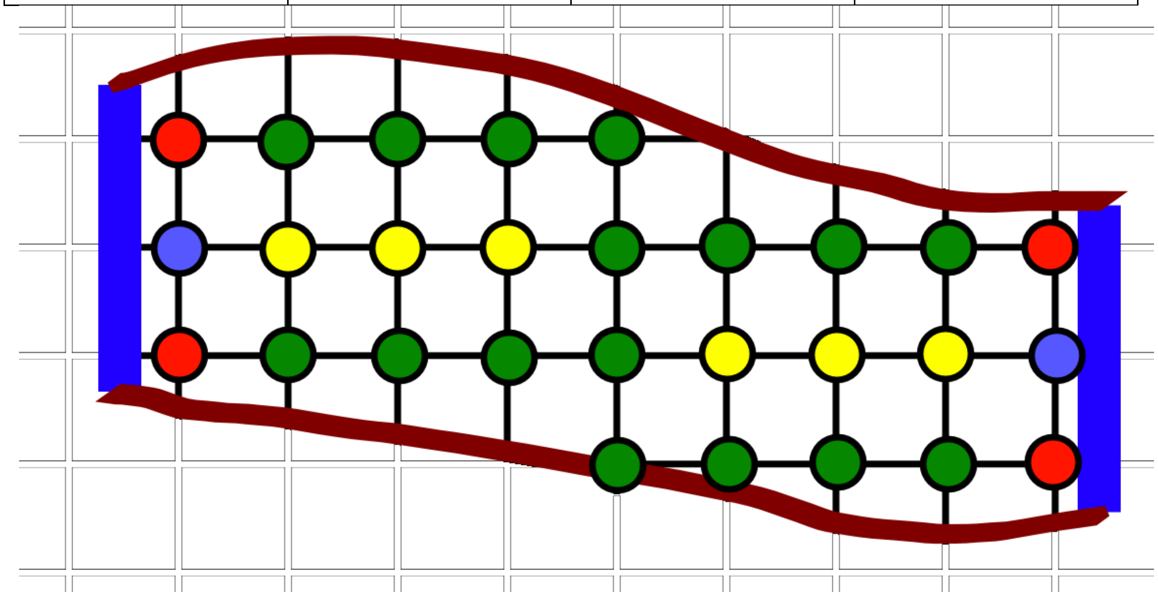

Figure 4: 2D example of a sparse domain with the different types of lattice sites. In/outlets are given by the blue bars and vessel walls by the red curves. Bulk sites are shown by yellow dots, wall sites by green dots, wall in/outlet sites by red dots, and in/outlet sites by blue dots.

4.2.2 Measurements on ARCHER

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 96, 192 and multiples up to 24,576 cores for our experiments. We used a modified HemeLB that incorporates PPStee v0.3.0d; 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 50,000 simulation steps for all geometries.

For our runtime measurements, we used two geometries. These include a smaller *bifurcation* geometry and a larger *aneurysm* geometry (see Figure 5 and Figure 6 for both). The bifurcation simulation domain consists of 650,492 lattice sites, which occupy about 10% of the bounding box of the geometry. The aneurysm simulation domain consists of 5,667,778 lattice sites, which occupy about 1.5% of the bounding box of the geometry. We run our simulations using pressure in- and outlets as described in [19], the LBGK collision operator, the D3Q19 advection model and Bouzidi-Firdaouss-Lallemand wall conditions.

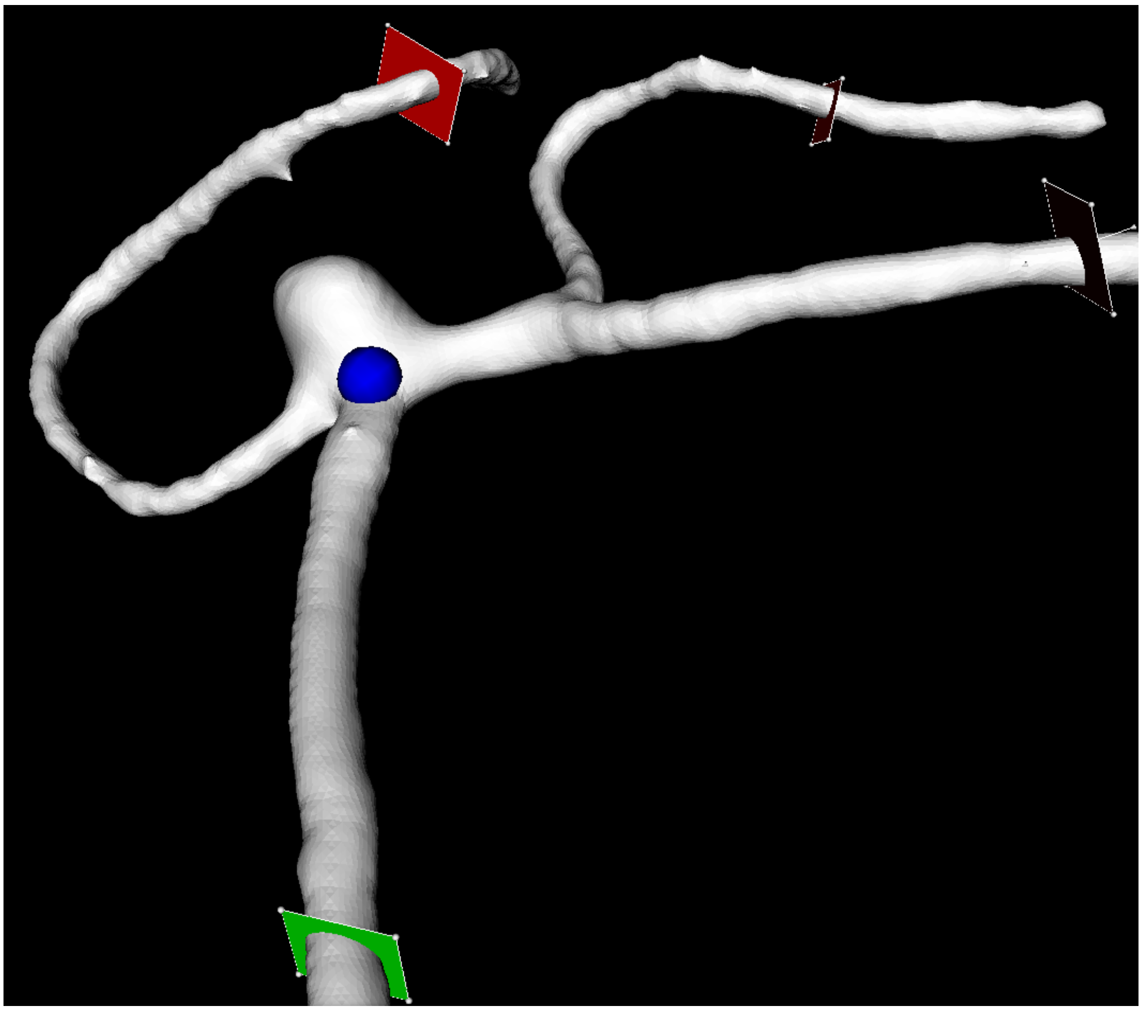

Figure 5: Aneurysm geometry.

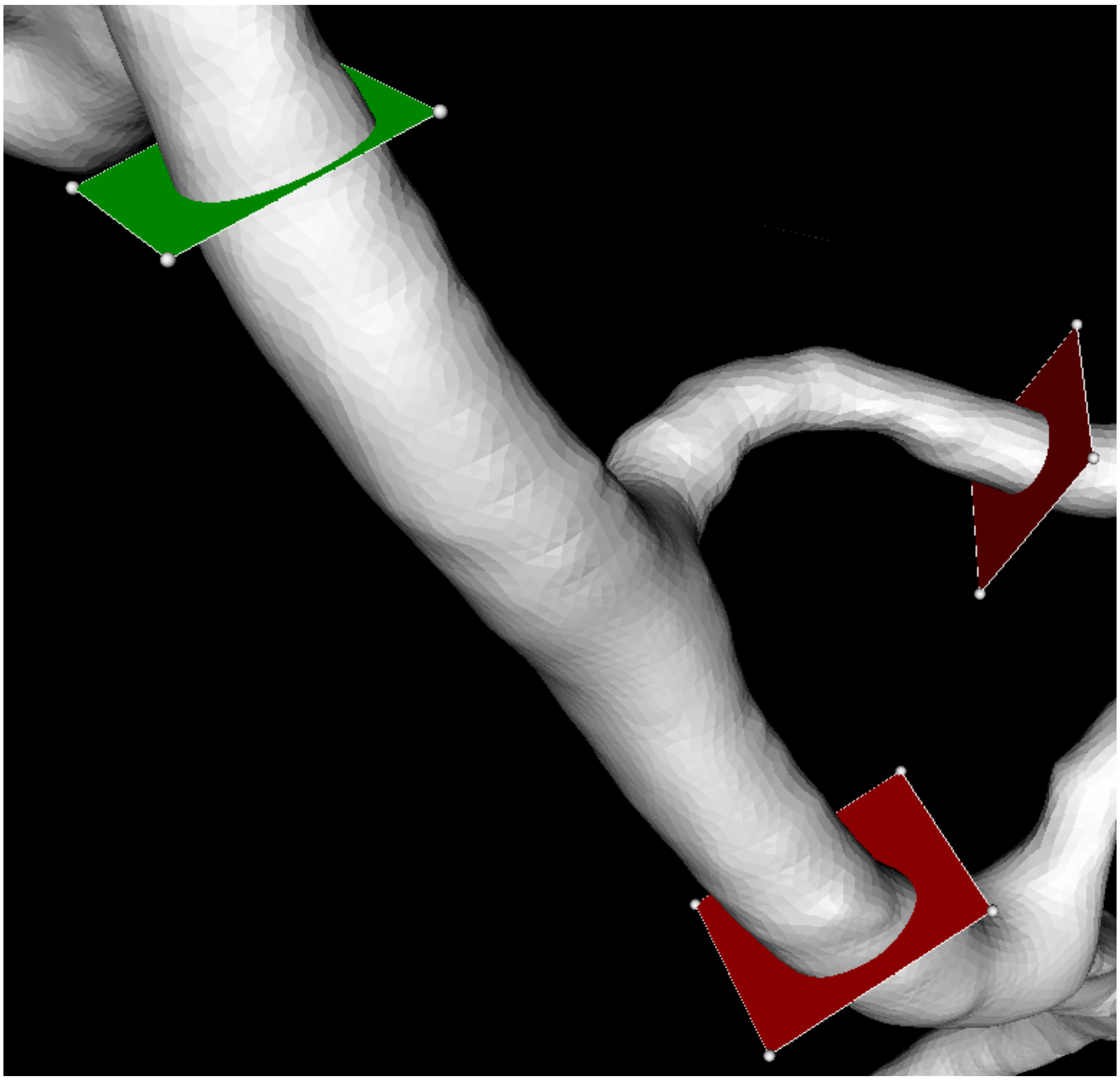

Figure 6: Bifurcation geometry.

Figure 7 and Figure 8 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 9 and Figure 10 show time spent for partitioning only, i.e. how long the call to the partitioning library lasted. On the other hand, Figure 11 and Figure 12 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.

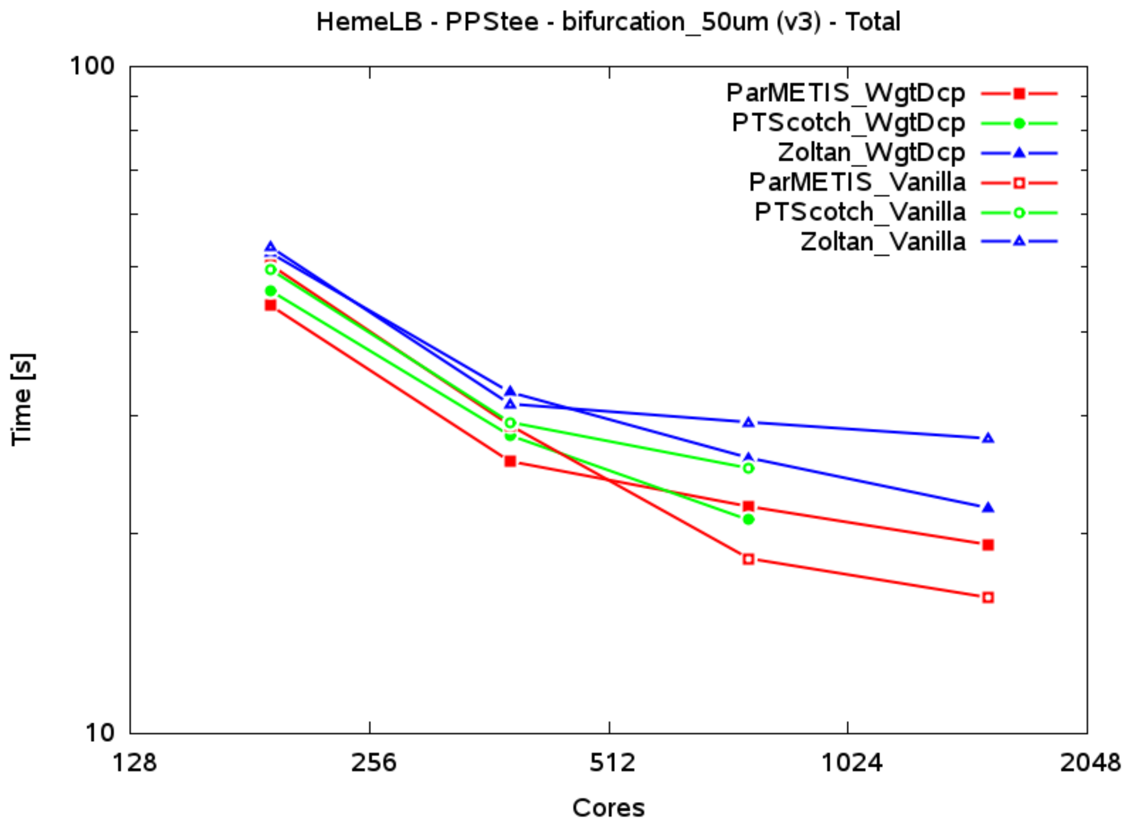

Figure 7: Total runtime of HemeLB on ARCHER for geometry bifurcation_50um (v3) with PPStee using one of three partitioning libraries comparing uniform versus weighted decomposition.

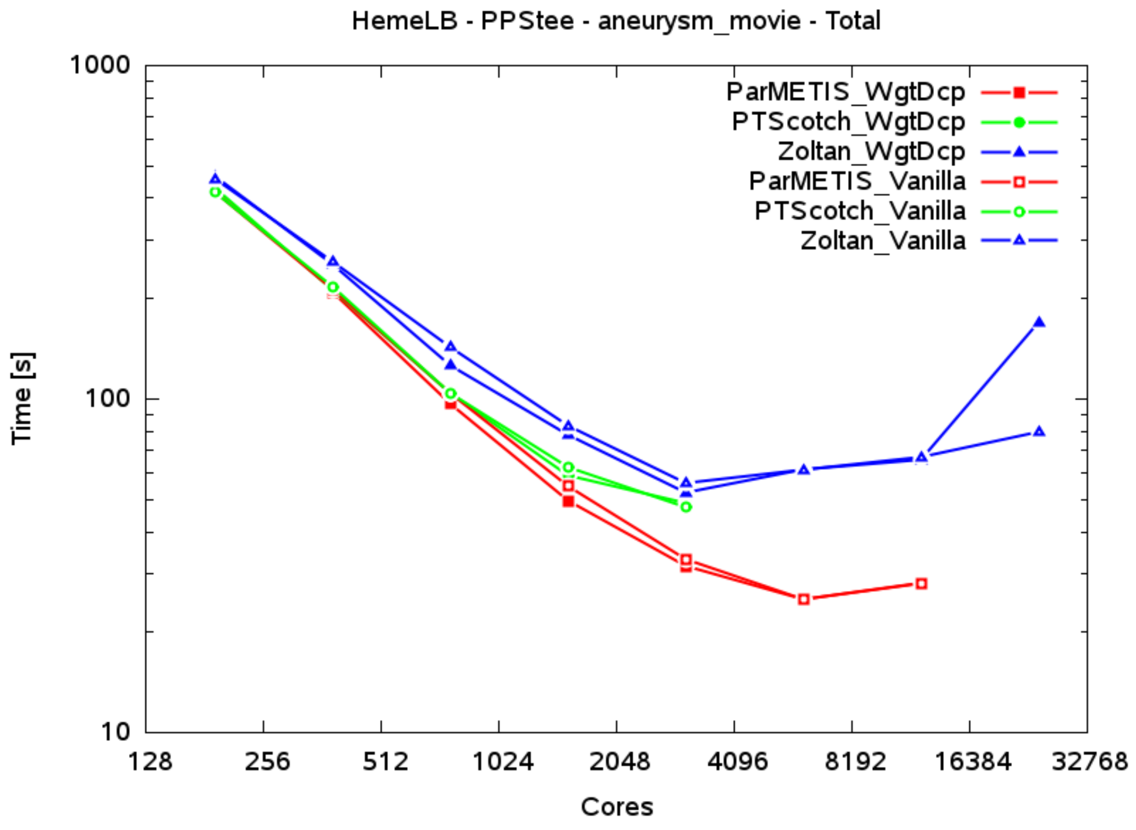

Figure 8: Total runtime of HemeLB on ARCHER for geometry an_mov with PPStee using one of three partitioning libraries comparing uniform versus weighted decomposition.

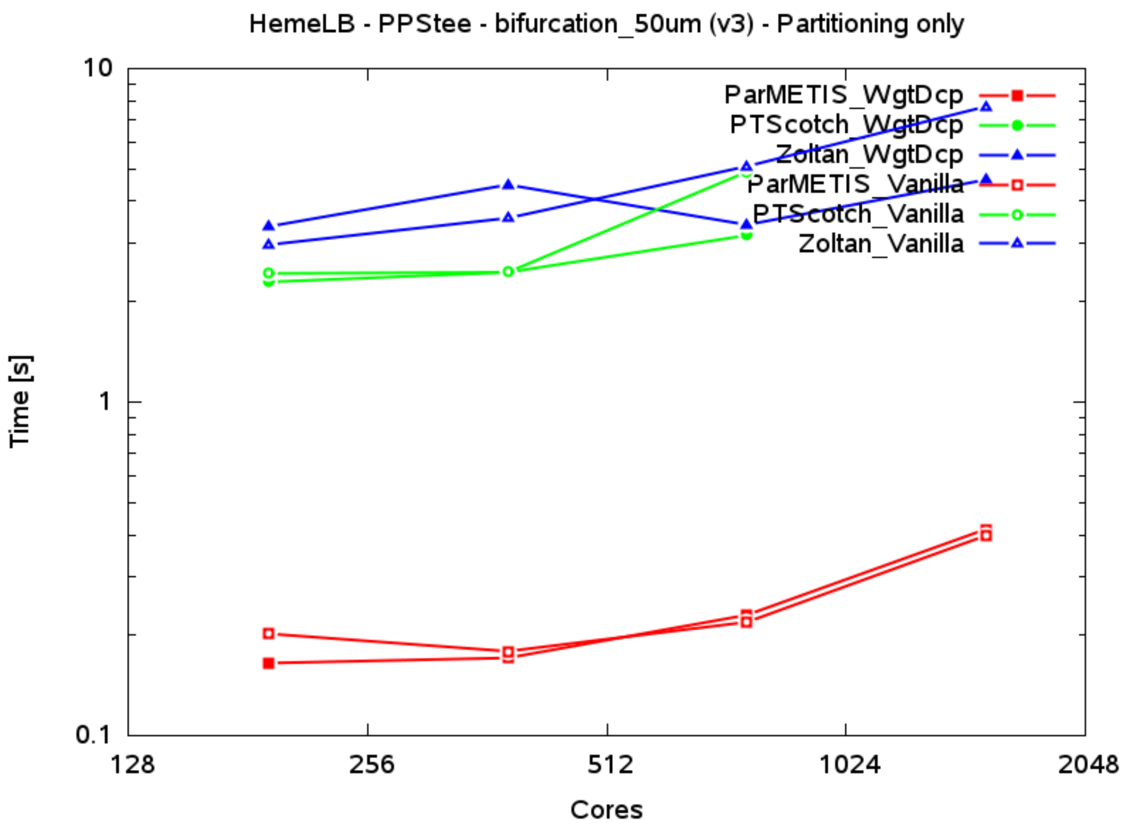

Figure 9: Partitioning time of HemeLB on ARCHER for geometry bifurcation_50um (v3) with PPStee using one of three partitioning libraries comparing uniform versus weighted decomposition.

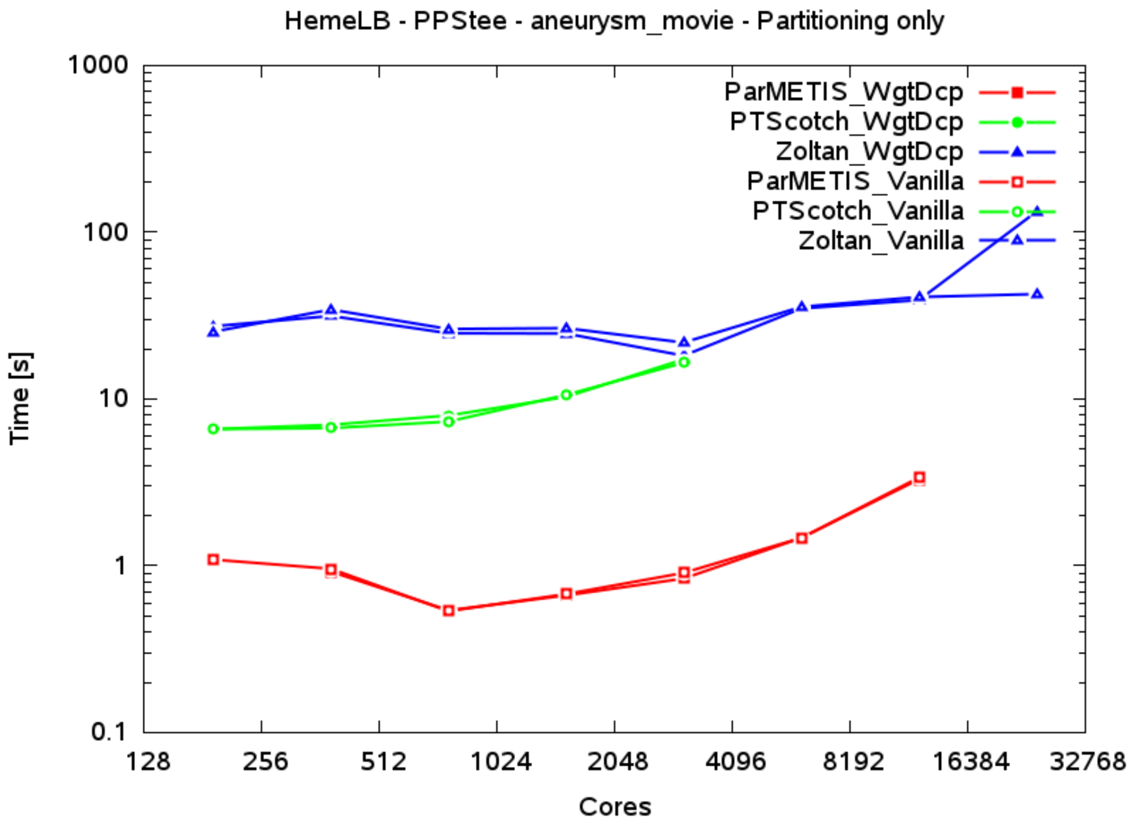

Figure 10: Partitioning time of HemeLB on ARCHER for geometry an_mov with PPStee using one of three partitioning libraries comparing uniform versus weighted decomposition.

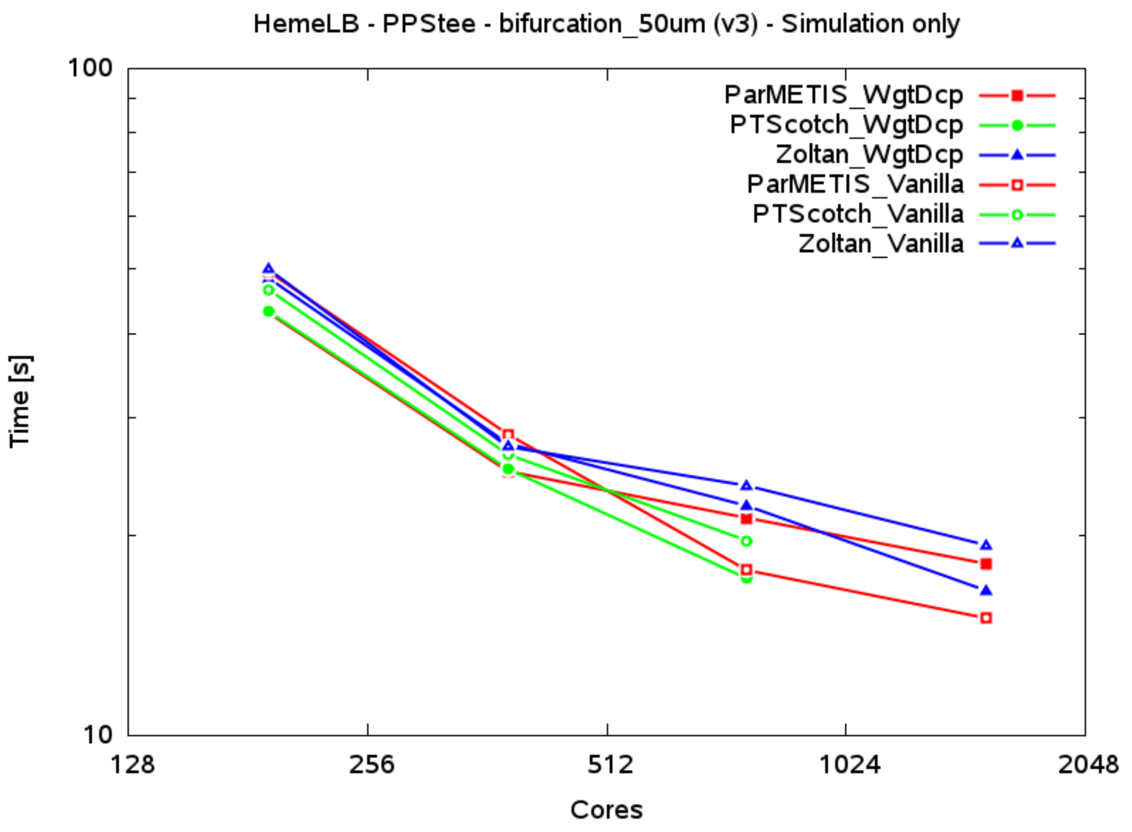

Figure 11: Calculation time of HemeLB on ARCHER for geometry bifurcation_50um (v3) with PPStee using one of three partitioning libraries comparing uniform versus weighted decomposition.

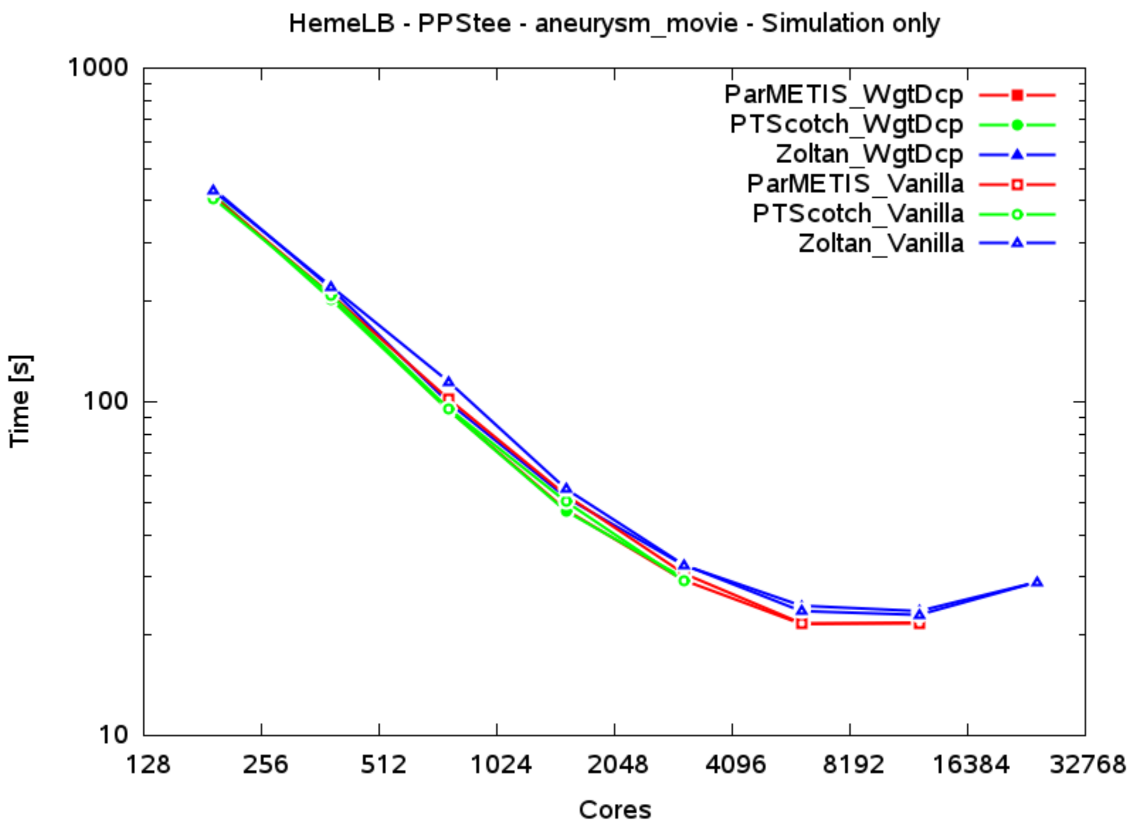

Figure 12: Calculation time of HemeLB on ARCHER for geometry an_mov with PPStee using one of three partitioning libraries comparing uniform versus weighted decomposition.

4.2.3 Analysis

First, we point out general observations that validate our former findings (cf. Section 4.1). The plots of the partitioning time clearly show that not one of the partitioning libraries scales at all. The time spent for the calculation of the partitioning rises for a higher number of cores. On the other hand, the quality of the partitioning is encoded in the calculation time depicted in Figure 11 and Figure 12 and is almost equal for all three libraries. Additionally, we mention the absolute values of the partitioning time as they might amount to a considerable part of the runtime if frequent repartitioning is applied.

Comparing the simulation series with respect to weighted decomposition and normal decomposition, we observe almost no differences between the decomposition types. Only for Zoltan and on the highest number of cores, i.e. 24k cores for geometry "aneurysm_movie", we notice a clear gap.

In summary, we conclude that we need a systematic approach to assess, compare and investigate domain decomposition techniques, as none of the ones we have used so far appears to produce results that ensure smooth scaling to Exascale resources. Based on this outcome, we have proceeded to develop a separate domain decomposition tool, which allows us to investigate this problem in isolation, without requiring the organizational and computational overhead of running HemeLB simulations for each decomposition.

4.3 Ensemble partitioning

4.3.1 Motivation

As shown in Section 4.2, partitioning simulation domains within HemeLB can be a time-consuming task. When we run ensemble simulations, we reuse the same geometry, subjecting it to a different type of flow regime in each instantiation. Using our existing approach where the domain decomposition is integrated in the HemeLB compute environment, we need to perform the domain decomposition repeatedly for each instance of the ensemble.

Here we report on *preliminary* developments to make a stand-alone partitioning tool, which will eventually allow us to do domain decomposition outside of the main simulation, and to reuse previously decomposed domains for multiple HemeLB simulations.

4.3.2 Approach

We have proceeded to develop an independent partitioning tool by combining PPStee with a newly-written Python environment (*Protopart)* for analysing and converting graph partitioning data. Protopart allows us to assess the quality of a decomposition without launching HemeLB, and is able to export partitioning information both in plain text and HemeLB .gmy format. The plain text format is particularly useful because it allows us to trivially read it in with a visualization tool, allowing us to visually inspect the quality of partitioned simulation domains.

Our current workflow for partitioning is as follows:

Protopart -> binary graph data of HemeLB geometry including coordinates -> PPStee (wrapper) -> partitioned HemeLB geometry data

As we started this new effort very late in the CRESTA project, we will not have the opportunity to adapt HemeLB to read in pre-partitioned simulation domain data before CRESTA has completed. Instead, we plan to perform this activity after the end of the CRESTA project, as part of a separately funded ARCHER eCSE project. The proposal for this, written by Rupert Nash and Derek Groen, is currently under review by the UK Engineering and Physics Research Council.

4.3.3 Analysis and visualisation of the partitioning results

The extraction of the partitioning from the simulation cycle into a separate simulation-preceding pre-processing step leads to a beneficial side effect. Even before the simulation is run, we have access to a data set that describes the HemeLB geometry in detail. The data contain not only the graph connections but coordinates and partitioning information too. Now, we can use these data either to apply some sort of partitioning analysis, or to visualise the geometry and its distribution according to the partitioning.

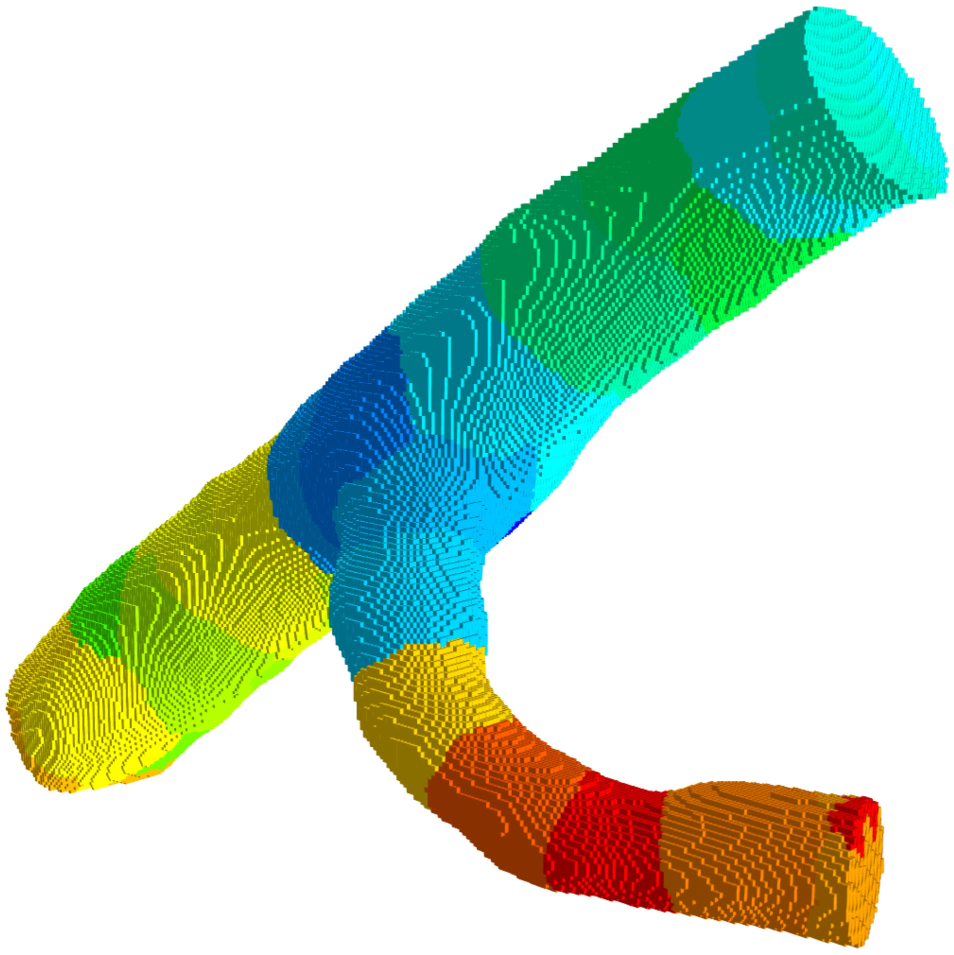

Figure 13: Visualisation of the HemeLB geometry dataset “bifurcation_50um” based on the partitioning result of PPStee using ParMETIS and a target number of 32 partitions.

To facilitate this opportunity, we added two scripts to the tools section of the PPStee repository. The first script converts the output of the PPStee wrapper described in Section 4.3.2, i.e. the partitioned HemeLB graph data, into a VTK file format [12]. Then the second script reads the VTK file and renders the geometry using Mayavi [13]. Figure 13 shows the rendered HemeLB geometry dataset “bifurcation_50um” partitioned by PPStee using ParMETIS with a target number of 32 partitions. Figure 14 shows the dataset “aneurysm_movie” with 4.6 million lattice sites and 128 partitions, again partitioned by PPStee using ParMETIS.

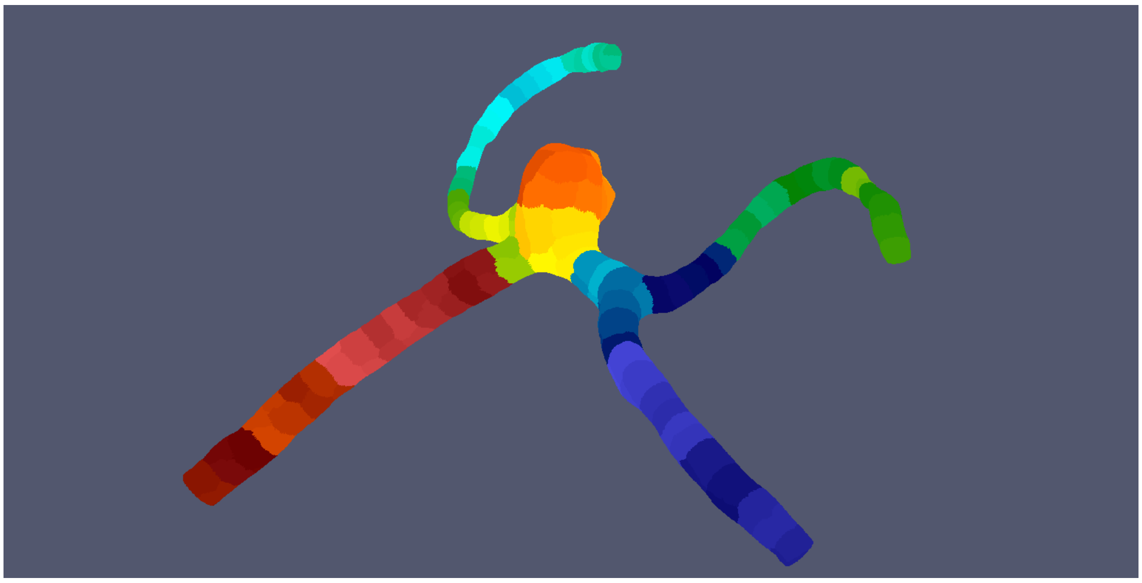

Figure 14: Visualisation of the HemeLB geometry dataset "aneurysm_movie" with 4.6 million lattice sites based on the partitioning result of PPStee using ParMETIS and a target number of 128 partitions.

5 PPStee and Nek5000

5.1 Integration and first simulation runs

In CRESTA Deliverable D5.1.5 [5], we described the integration of PPStee into the computational fluid dynamics simulation Nek5000 that is based on the spectral element method [14]. We provided a short description of the modifications of Nek5000 that were developed within CRESTA. On top of these modifications that implement only ParMETIS for the partitioning, PPStee is a natural extension as PPStee enables the additional use of PTScotch and Zoltan. We presented a proof of concept and showed runtime results of Nek5000 with PPStee for a simple cylindrical geometry measured on a small desktop machine. Thus, we proved the applicability of PPStee for Nek5000 in general.

5.2 Strong scaling with three partitioning libraries

5.2.1 Setting and geometry

Nek5000 is an open-source code for the simulation of incompressible flow in complex geometries. It adopts the spectral-element discretisation that combines the higher-order accuracy from spectral methods with the geometric flexibility of finite element methods. Original version of Nek5000 uses 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, which is later used to create element-processor mapping.

Within the CRESTA project, we implemented an adaptive mesh refinement algorithm (AMR) in Nek5000, which gives the possibility of increasing the accuracy of numerical simulations with minimal computational cost. We focused on the h-refinement framework, i.e. the splitting of elements into smaller ones, due to its flexibility and possible improvements in the solver performance. This refinement scheme dynamically changes grid structure modifying element number and connectivity and requires dynamical grid partitioning.

In our implementation the grid modification is managed by p4est [15] library, which provides correct grid structure and element connectivity information. For load balancing we adopted ParMETIS library, which in the initial tests gave the partitioning quality similar to the native Nek5000 static partitioning. We adopted partitioning from scratch strategy, which allows for highest possible quality of the mesh distribution, but does not take into account partitioning and communication costs. We implemented in Nek5000 all the tools necessary for dynamical modification of the mesh structure during the simulation including element creation, destruction and redistribution, and the main solver restart on the new mesh. Another crucial tool required by AMR is an error estimator, which allows for evaluation of the computational error and identification of the regions in the flow requiring refinement or coarsening. Such error estimators based on the expansion of the solution in the basis of Legendre functions was implemented in Nek5000 [20].

This implementation was tested with a model problem based on the convected-cone example introduced by Gottlieb and Orszag [16]. It is the passive scalar transport problem, in which a unit-height cone with a base-radius of 0.1 centred at (x,y)=(0, 0.25) in a square mesh is subjected to plane rotation according to time independent velocity field **v**=(y-0.5,0.5-x). We adopted this example to 3-dimensional simulations evolving a sphere-shape cone according to energy equation in Nek5000. In this case spectral error estimator identifies discontinuities in the initial condition increasing grid resolution at the edge and the centre of the cone (see **Error! Reference source not found.**). We have to mention here, that the global number of elements in some of the strong scaling tests is not constant for different processor numbers due to the fact that p4est performs de-refinement of the 8 children elements into the single parent element only if all the children elements reside on a single process. That is why the global number of elements slowly grows with the number of processors (see [17] for more detailed discussion).

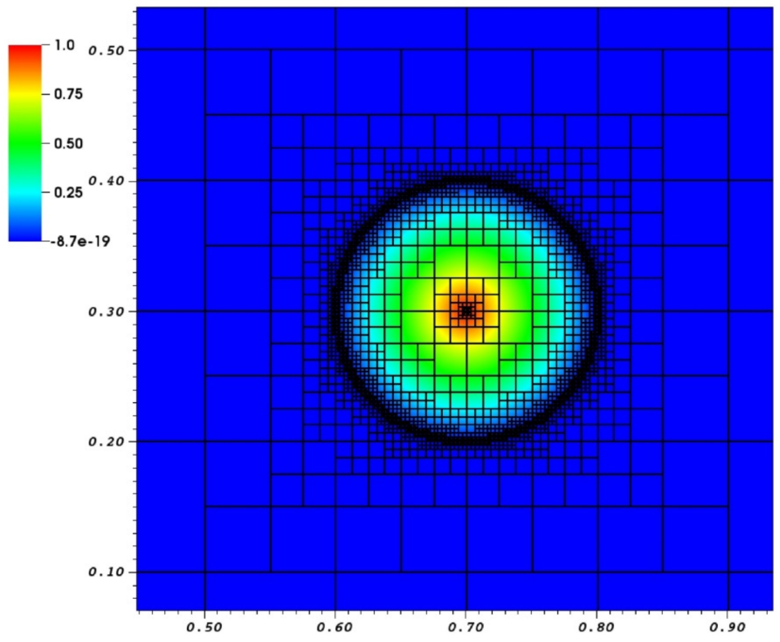

Figure 15: Two–dimensional cut through the domain of the convected-cone problem showing the grid structure (black squares) and the passive scalar profile (colour scale). Each element (3D cube depicted by a square) corresponds to the mesh of 12x12x12 grid points

In all performed tests we follow advected features in the flow (the cone), which requires continuous adjustment of the mesh and does not converge to any time-independent grid structure. Although this strategy is not applicable to stability calculations, where instead of individual flow structures the sensitive regions in the flow have to be identified and resolved, it allows the frequency of grid modification to be increased, and possible limitations of the method to be studied.

Our tests performed with ParMETIS show non-conformal version of Nek5000 to be the best parallelized component of our code. The biggest constraint in the parallel scaling comes from the performance of the grid partitioner, showing partitioning from scratch strategy to be inefficient. The partitioning time grows quickly with the number of processors and becomes dominant in the runs with fewer than 10 elements per core (see [17] for more detailed discussion). That is why in cooperation with WP5 we have been investigating other available partitioning tools implemented in PPStee.

5.2.2 Measurements

All runtime measurements were performed on ARCHER [11] using fully-populated nodes adding up to the indicated core count. In terms of ARCHER's 24-core nodes, this means values of 768, 1,536 and multiples up to 49,152 cores for our experiments. We used the CRESTA version of Nek5000 and modified it to incorporate PPStee v0.3.0d; a detailed description of the integration of PPStee into an existing code can be found in Section 2 of CRESTA Deliverable 5.1.3, [3].

We used a default value of 100 simulation steps for all geometries. Nek5000 solver uses variable time step with the fixed Courant number equal to 0.3, and the mesh was regenerated every 50 Nek5000 steps. The maximum refinement level was set to 5, which for fully-refined initial conditions corresponds to 86,288 3-dimensional elements with polynomial order 11.

Figure 16 depicts total runtimes of the simulation, i.e. these measurements include all simulation parts starting with initial read-in of the geometry, followed by partitioning, and calculations in the scientific kernel. To achieve a more detailed picture, we provide two additional figures. Figure 17 shows time spent for partitioning only, i.e. how long the call to the partitioning library lasted. On the other hand, Figure 18 shows the calculation time spent on the scientific kernel, which includes computation and the communication that is needed for the spectral element solver to find the solution.

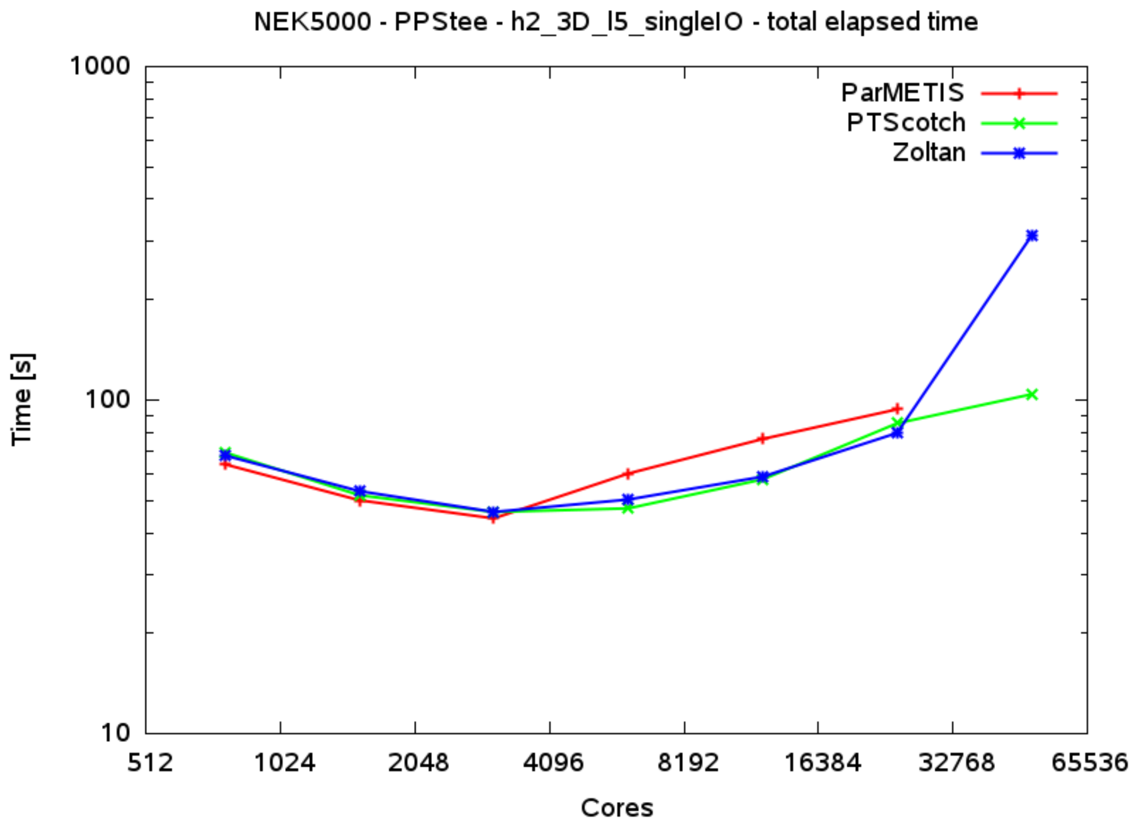

Figure 16: Total runtime of Nek5000 on ARCHER for geometry “h2_3D” on level 5 (single IO) PPStee using one of three partitioning.

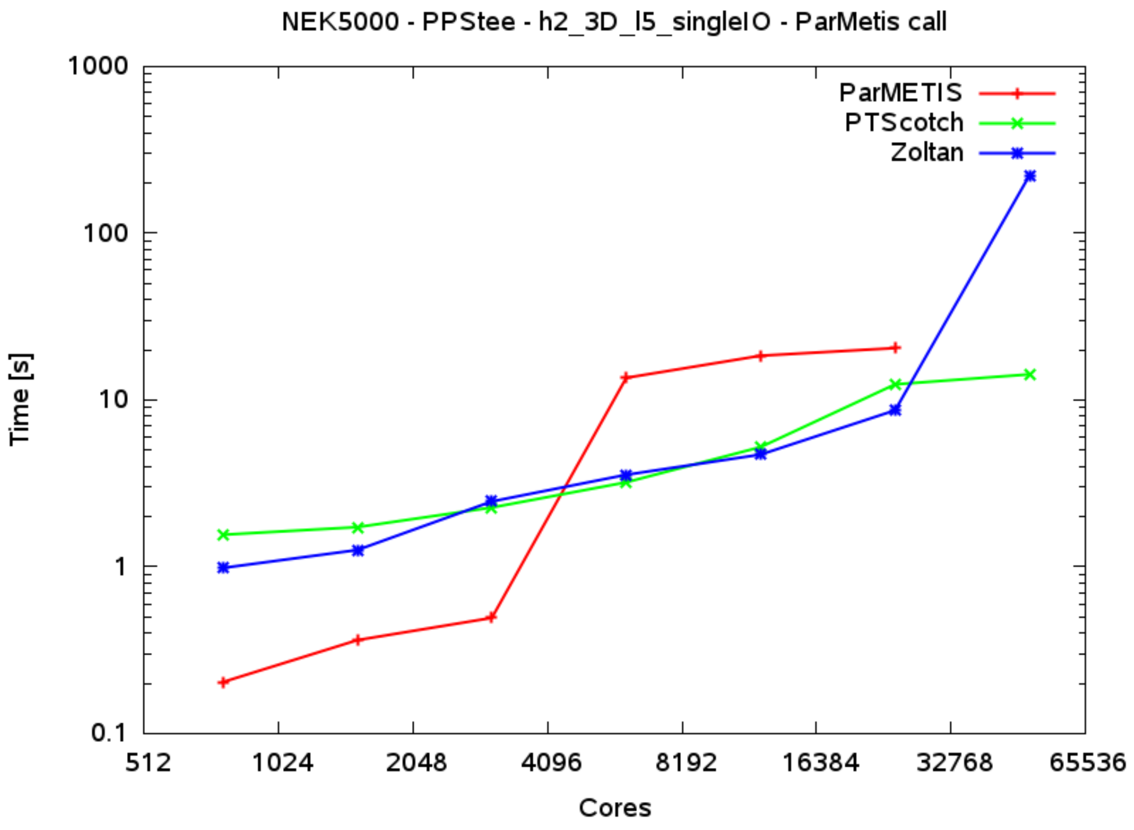

Figure 17: Partitioning time of Nek5000 on ARCHER for geometry "h2_3D" on level 5 (single IO) PPStee using one of three partitioning.

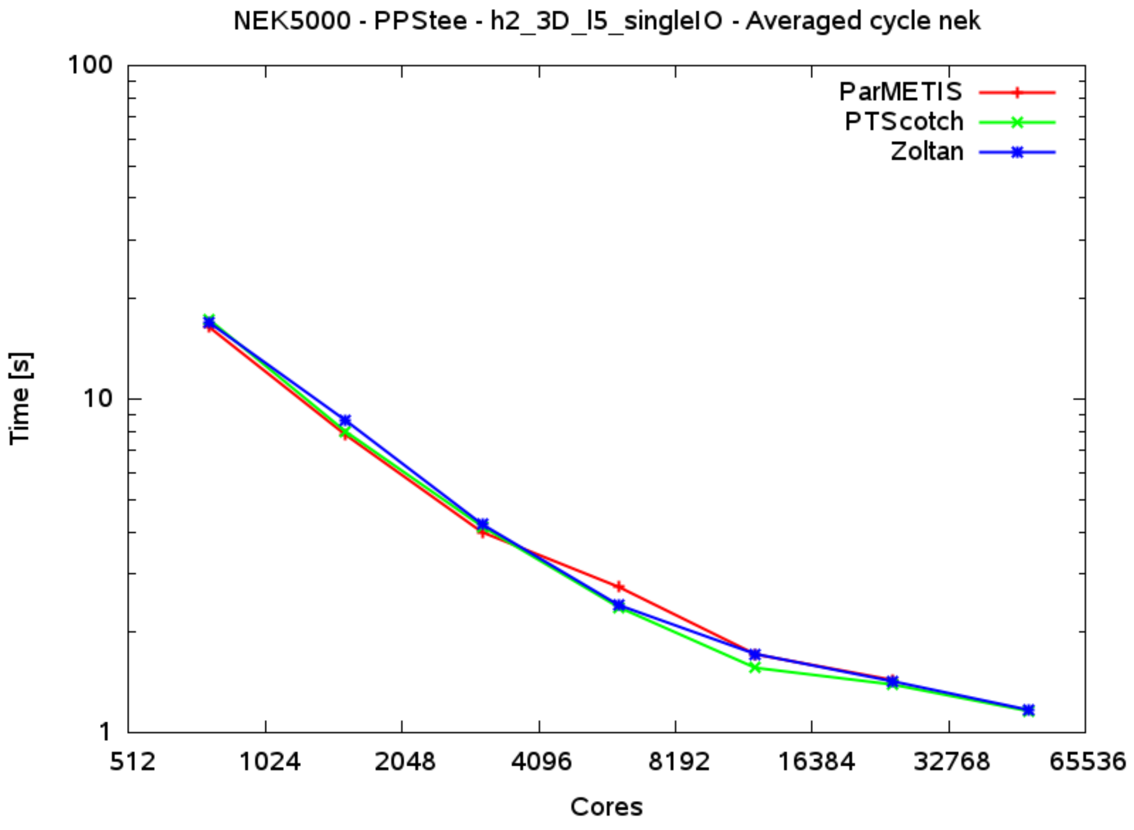

Figure 18: Calculation time of Nek5000 on ARCHER for geometry "h2_3D" on level 5 (single IO) PPStee using one of three partitioning.

5.2.3 Analysis

Plots of the calculation time (see Figure 18) show an almost equal value for all libraries for all core counts. Only ParMETIS is slightly better up to 3,072 cores but gets slightly worse for higher core counts. We deduce that the quality of the calculated partitioning is almost equal for all three partitioning libraries.

The partitioning time depicted in Figure 17 led to a first and disappointing observation. None of the three partitioning libraries scales at all and, even worse, they get slower for higher core counts. We stress that the situation might, of course, change for other geometries. Moreover, there is a possibility to work around this restriction. PTScotch and Zoltan perform almost equally for almost the full range of core counts. ParMETIS, however, is faster on lower numbers of cores and slower on higher numbers of cores. Hence, we can optimally exploit one of PPStee's main features and swap the partitioner depending on the number of cores the simulation is running on. This method gives us, at least, the fastest partitioning library for the full range of core counts.

6 Conclusions

The pre-processing interface PPStee works as intended. It is lightweight and easy to integrate into an existing code as the examples HemeLB and Nek5000 show. PPStee introduces the opportunity to easily test the cooperative quality of simulation data layout and one of three different partitioning libraries. At the same time, PPStee supports load-balancing of multiple stages of a simulation. Thus, it encourages the inclusion of costly simulation phases into the simulation cycle on the high performance system. A simulation with in-situ visualisation and mesh refinement, for instance, will benefit greatly from the renewed load-balancing.

7 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] CRESTA Deliverable 5.1.5, *Pre-processing: final tools for exascale mesh partitioning and mesh analysis available*

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

[7] 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/

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

[9] D. Groen, J. Hetherington, H.B. Carver, R.W. Nash, M.O. Bernabeu, P.V. Coveney, *Analysing and modelling the performance of the HemeLB lattice-Boltzmann simulation environment*, Journal of Computational Science 4 (5), 412-422, 2013, http://ccs.chem.ucl.ac.uk/hemelb

[10] HECToR UK National Supercomputing Service, http://www.hector.ac.uk

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

[12] Will Schroeder, Ken Martin, and Bill Lorensen, *An Object-Oriented Approach To 3D Graphics*, Kitware, Inc. 4th edition (December 2006).

[13] P. Ramachandran and G. Varoquaux, *Mayavi: 3D Visualization of Scientific Data,* IEEE Computing in Science & Engineering, **13** (2), pp. 40-51 (2011)

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

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

[16] D.I. Gottlieb, and S.A. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications*, SIAM-CBMS, Philadelphia, 1977

[17] CRESTA Deliverable 6.1.3, *Roadmap to exascale*

[18] http://arxiv.org/abs/1410.4713

[19] R.W. Nash, H.B. Carver, M.O. Bernabeu, J. Hetherington, D. Groen, T. Krüger, P.V. Coveney, *Choice of boundary condition for lattice-Boltzmann simulation of moderate Reynolds number flow in complex domains*, Physics Review E 89, 023033, 2014.

[20] C. Mavriplis, *Adaptive Mesh Strategies for the Spectral Element Method*, Computer Methods in Applied Mechanics and Engineering, 116 (1994), pp. 77-86.