D5.1.4 – Pre-processing: revision of system, data format and algorithms definition for exascale systems

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:	M24
Submission date:	30/09/2013
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organization	DLR
Version:	1.0
Status	Final
Author(s):	Gregor Matura (DLR)
Reviewer(s)	Berk Hess (KTH), Mats Hamrud (ECMWF)

Dissemination level	
<PU/PP/RE/CO>	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	06/09/2013	First version of the deliverable	Gregor Matura (DLR)
0.2	09/09/2013	Final version for internal submission	Gregor Matura (DLR)
1.0	20/09/2013	Addressed reviews – final version for submission	Gregor Matura (DLR)

Table of Contents

Index of Figures

1 Executive Summary

In CRESTA Deliverable 5.1.1, [1], we analysed the current situation of simulations regarding pre-processing and gave a system definition: main aim is a closer simulation cycle including all simulation parts and an improved overall simulation load balance. CRESTA Deliverable 5.1.2, [2] studied algorithms of partitioning libraries used for pre-processing so far and pointed at basic properties required for the graph data format. These requirements culminated in the development of the prototype pre-processing steering interface PPStee introduced in CRESTA Deliverable 5.1.3, [3].

Here we review the design of PPStee and collect performance data to evaluate this prototype tool. The integration of PPStee into HemeLB was relatively simple, as intended, and allows for performance tests of HemeLB with various geometries and all three by PPStee supported partitioners, ParMETIS [4], PTScotch [5]and Zoltan [6]. Runtime measurements with up to 2048 cores on HECToR are presented as first results. PPStee's runtime overhead vanishes and ensures usage of PPStee without a priori drawbacks. The configuration using PTScotch performs, in general, slightly worse but reveals scalability issues starting at 512 cores. HemeLB with PPStee using Zoltan suffers from a constant loss in runtime, the reason is yet unknown. Further investigations will, in particular, focus on graph data conversion, scalability and usage of partitioner-characteristic routines and parameters to enable a better match to specific simulation data.

Lastly, we address CRESTA's co-design vehicle OpenFOAM. Simulations using OpenFOAM are not a priori suited for application of PPStee due to the nature of OpenFOAM being a box of separated tools of solvers and utilities. However, PPStee may be applicable if each phase of a simulation using the OpenFOAM framework is aggregated into one monolithic program. The OpenFOAM co-design team is currently investigating the feasibility of this monolithic program..

2 Introduction

A simulation can be logically divided into several parts, pre-processing, simulation core, and post-processing are examples. So far, i.e. up to the petascale regime, these parts are often separated strongly, e.g., by large IO operations or even by separation into different programs. Such cuts in the simulation work and data flow are not alone very expensive but become unbearable when core counts rise above hundreds of thousands; they have to be overcome if a simulation wants to perform in an exascale environment.

We identify a delicate point: All simulation parts cannot stay apart and have to grow together or an exascale system will not be exploitable to its full performance. This is the task of CRESTA's work package 5 and its subtasks pre-processing, post-processing and remote rendering. Here, pre-processing focuses on balancing the load of the simulation including all parts and not the simulation core alone.

In previous deliverables (see [1] and [2]) we analysed the current situation. Communication and calculation costs of all simulation parts must be included in the calculation of the overall simulation load balance. Since pre-processing usually is invoked only once and only at the beginning it cannot interact with simulation or post-processing results. Thus the desired inclusion of the costs requires a tighter integration of pre-processing into the simulation cycle. Convenient interfaces for information flow between the simulation parts, pre-processing, simulation core and post-processing, are needed. Steering methods can be integrated there as well to allow for an active influence of one simulation part to the other. A repeatedly passed simulation cycle facilitates additional chances for performance and convenience gains, for example, repartitioning capabilities, automated mesh refinement techniques and other.

The amount of data used must be kept minimal: exascale systems will presumably suffer from a further decrease in the ratio of memory size to computation power per node. Hence a small memory footprint is vital and additionally lowers communication time when data are exchanged between the nodes. Algorithms for partitioning must stay universally applicable and simply comparable as their performance and scalability may depend critically on the specific simulation data structure.

To address these requirements, we introduced the pre-processing steering interface PPStee (see [3]). Its main purpose is the achievement of an optimised overall simulation load balance and its main feature is the exchange of the partitioning library used regardless of the input data (ParMETIS [4], PTScotch [5] and Zoltan [6] are supported). PPStee provides a flexible data format that is both, minimal and compatible to all three partitioners. It comprises different simulation stages and their communication and calculation costs, like computation (equals simulation core or solver) and visualisation. However, PPStee stays easily adjustable to new partitioning tools, different kinds of stages, even fault tolerance or an automated mesh refinement can be integrated as well.

Now, we want to evaluate this prototype tool and, here for, use CRESTA's co-design application HemeLB as a test bed. We describe and carry out the integration of PPStee into HemeLB and start an analysis of PPStee's functionality and performance. Following some small tests, we do runtime measurements on HECToR with up to 2048 cores. These will shows us if PPStee works as intended and provide first information on usability and performance of PPStee for a real-life example.

With these first lessons learned, we try to expand our set of test cases to another CRESTA co-design application, OpenFOAM. In contrast to the self-contained Lattice-Boltzmann code HemeLB, OpenFOAM is only a toolbox for the development of customised numerical solvers and utilities for the solution of continuum mechanics problems. Thus, our primary task concerning OpenFOAM is an examination of how OpenFOAM can benefit from PPStee and how and where PPStee can be integrated into a simulation using OpenFOAM.

We begin this deliverable with short summaries on previous work in section 3. An outline of analysis and system definition and data format and algorithms is given followed by a description of the pre-processing steering interface PPStee. Section 4 describes the integration of PPStee into the lattice-Boltzmann code HemeLB and shows a first proof of concept integration for small core counts. Measurements for large core counts on HECToR are depicted and analysed. Our work with OpenFOAM regarding usage of PPStee is explained in section 5. The last section summarises PPStee properties we were able to confirm and points to issues that need further investigation.

2.1 Purpose

The purpose of this deliverable is a revision of the submitted prototype pre-processing tool PPStee (cf. D5.1.3, [3]) considering the definition and analysis of system, data format and algorithms done earlier (cf. deliverables D5.1.1 and D5.1.2, [1] and [2], respectively).

2.2 Acknowledgement

This work made use of the facilities of HECToR, the UK's national high-performance computing service, which is provided by UoE HPCx Ltd at the University of Edinburgh, Cray Inc and NAG Ltd, and funded by the Office of Science and Technology through EPSRC's High End Computing Programme.

3 Previous work

3.1 Analysis and system definition

CRESTA Deliverable 5.1.1, Pre-processing: analysis and system definition for exascale systems [1] demands the following property that is crucial for an exascale-ready simulation: main target of pre-processing is to guarantee a good overall simulation load balance. In particular, this claim implies that the term load balance is not only applied to the solver. Load balance has to treat all parts of a simulation and therefore includes all simulation costs of pre-processing, solution calculation and post-processing.

All simulation parts must be brought closer together. Separate tools for pre-processing of simulation data or solver data, for post-processing of the results and for visualisation will corrupt the overall simulation performance. An awareness of a complete simulation cycle rather than just a solver must evolve.

A tighter integration of pre-processing and post-processing becomes vital in the exascale regime. This can be achieved by extension of information flow between the specific simulation parts. Interfaces among pre-processing and simulation core, simulation core and subsequent result analysis and to visualisation routines can provide needed information. Additionally, the interfaces can implement methods for steering that enforce specific behaviour.

A closely linked simulation cycle performs better in general and offers additional chances of performance improvement, both, automated or user-steered. Result analysis as part of the post-processing can, for example, automatically initiate a repartitioning of the mesh following a specific event or time step thereby optimising the load balance mid-simulation. Using remote-rendering techniques, it is possible to inspect the simulation and intermediate results and thus adjust solver parameters, mesh properties or result analysis properties.

3.2 Data format and algorithms

CRESTA Deliverable 5.1.2, Pre-processing: data format and algorithms [2] analyses pre-processing tools and, in particular, partitioning libraries and data formats and algorithms the partitioners use. The primary recommendation focuses on the specific layout of data. A minimal set of graph data helps to keep down the memory footprint of huge geometries or other input data; this becomes more important for exascale simulations where the ratio of memory to computational power is predicted to decrease further. Moreover, it lowers necessary intercommunication of threads that will also suffer from computational power leaving communication network speed behind.

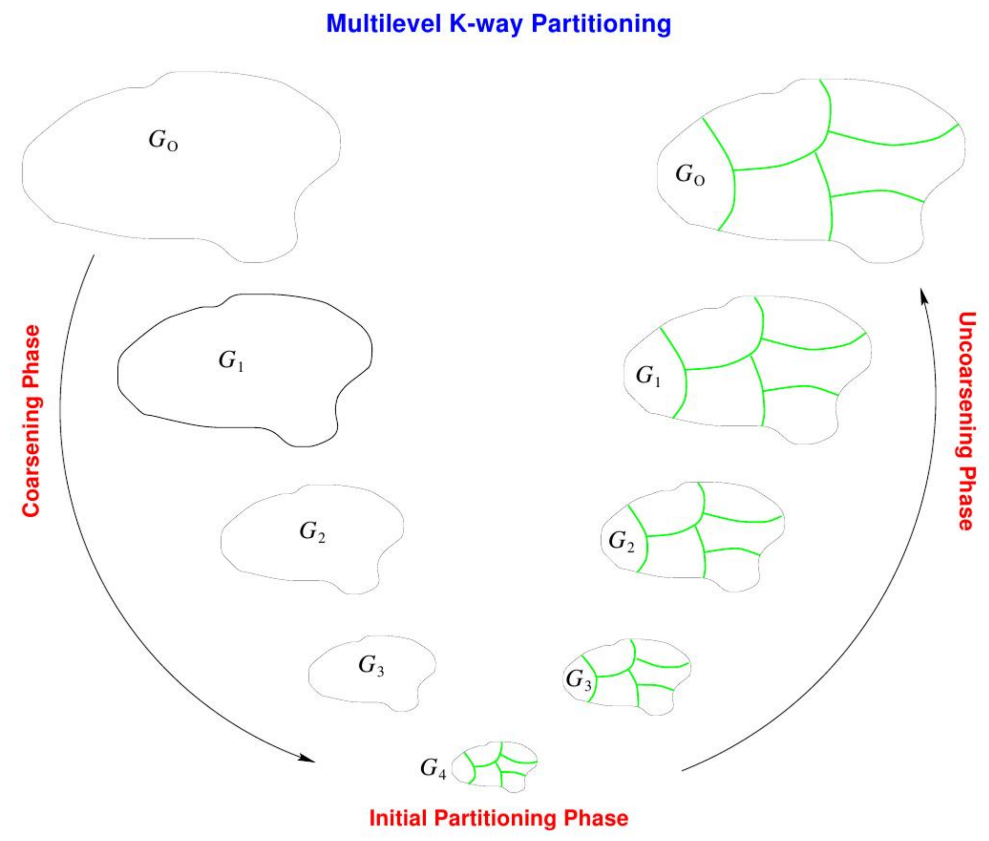

Figure 1: Multilevel k-way graph partitioning; [4]

Another crucial fact concerns algorithms used. Partitioners implement different strategies in their algorithm to solve the NP-complete problem of distributing input data to all threads (Figure 1 shows an example of one of those algorithm strategies, the multilevel K-way partitioning approach of ParMETIS; for details see [2]). A particular partitioner and its method may suit a specific simulation on a specific system quite well. However, simulations designed today must be adaptable to system and system architectures that are not yet built. Thus a chance to compare and fit algorithms to the simulation and its input on a given system in terms of scalability and performance must be preserved.

3.3 PPStee

PPStee is an interface for pre-processing steering (a detailed description can be found in CRESTA Deliverable 5.1.3, Pre-processing: first prototype tools for exascale mesh partitioning and mesh analysis, [3]). Its main purpose is to optimise the overall simulation load balance and can be extended by further steering capabilities covering other important exascale-relevant duties like automated mesh manipulation or fault tolerance. Figure 2 shows a chart for information flow for a simulation using PPStee.

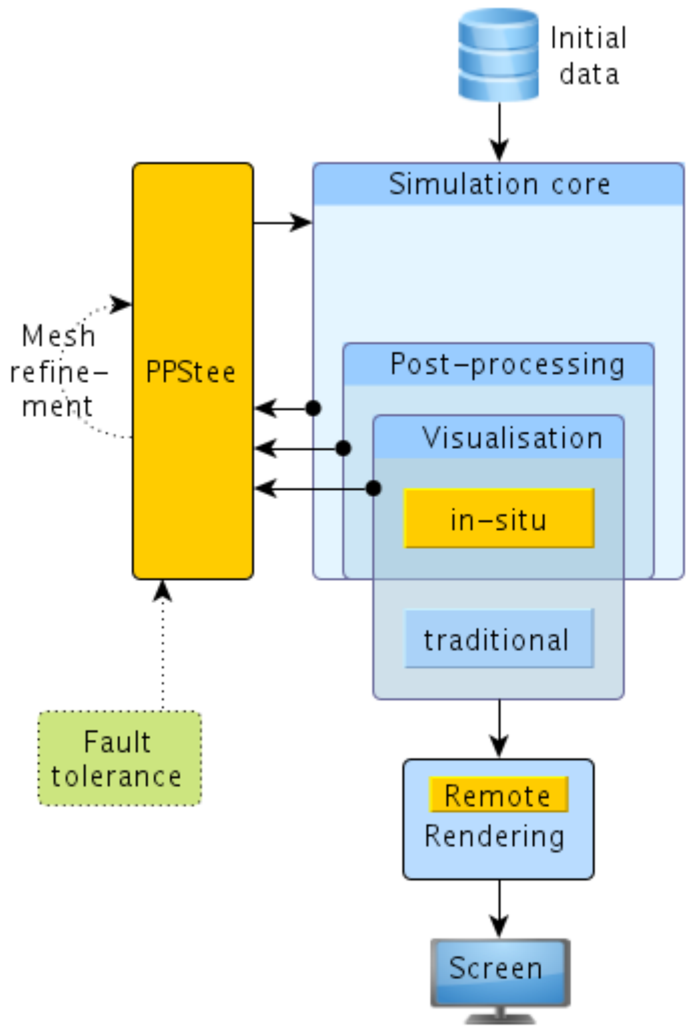

Figure 2: PPStee flow chart; [3]

PPStee offers a standardised access to three different partitioning tools, i.e. ParMETIS, PTScotch and Zoltan. Exchange of the partitioner used is designed to be as easy as possible to facilitate convenient access to all partitioners. Data format is kept simple and compatible among all partitioners, yet it is (almost) minimal satisfying recommendations listed in [2].

Different simulation phases, called stages, are supported. Computation, visualisation and other simulation parts can submit their communication and computation costs separately. This way PPStee is able to ensure a load balance for the entire simulation cycle.

Additionally, PPStee remains adjustable to future requirements. New partitioning tools can be integrated easily; stages and stage setup can be altered to match simulation needs. Mesh refinement techniques, user-driven or automated, can be built into the pre-processing step. An interface to a fault tolerance framework providing system health and status data could steer partitioning of simulation data in future.

4 PPStee and HemeLB

4.1 Integration

The specific integration of the pre-processing steering interface PPStee into a mature code such as HemeLB is straight forward (cf. [3], section 2.2). The integration is divided into two steps, bare code changes and some additional changes in the build system. In the first step, PPStee substitutes the partitioner call. HemeLB already uses the library ParMETIS to evenly distribute computational domains among all processes. Thus, we have all mesh data set up correctly and we save the trouble to assemble the arrays needed. We simply replace the ParMETIS call as shown in [3], section 3.1 and make sure to include PPStee's header file instead of ParMETIS' header.

Additionally, we adjust HemeLB's build system according to the needs of PPStee. Since ParMETIS is already included, we add PPStee and all other partitioners we want to use to the dependencies of the build target HemeLB. For an update of include and library directory settings, the very entries for ParMETIS are well-used as guides. Put altogether, the code changes are small and it is easy to find the right places, in spite of the size of a project like HemeLB.

4.2 Proof of concept

For a proof of concept, we have integrated and built HemeLB with PPStee and two partitioners, ParMETIS and PTScotch. Our first test system is a small one and provides an Intel Xeon E5520 with 8 real and 16 virtual cores. We used five different process counts between 4 and 16 and the minimal time of three runs for each data set. Figure 3, Figure 4 and Figure 5 depict the runtime results of the three different HemeLB data sets R15-L45, R15-450 and R30-L900, respectively.

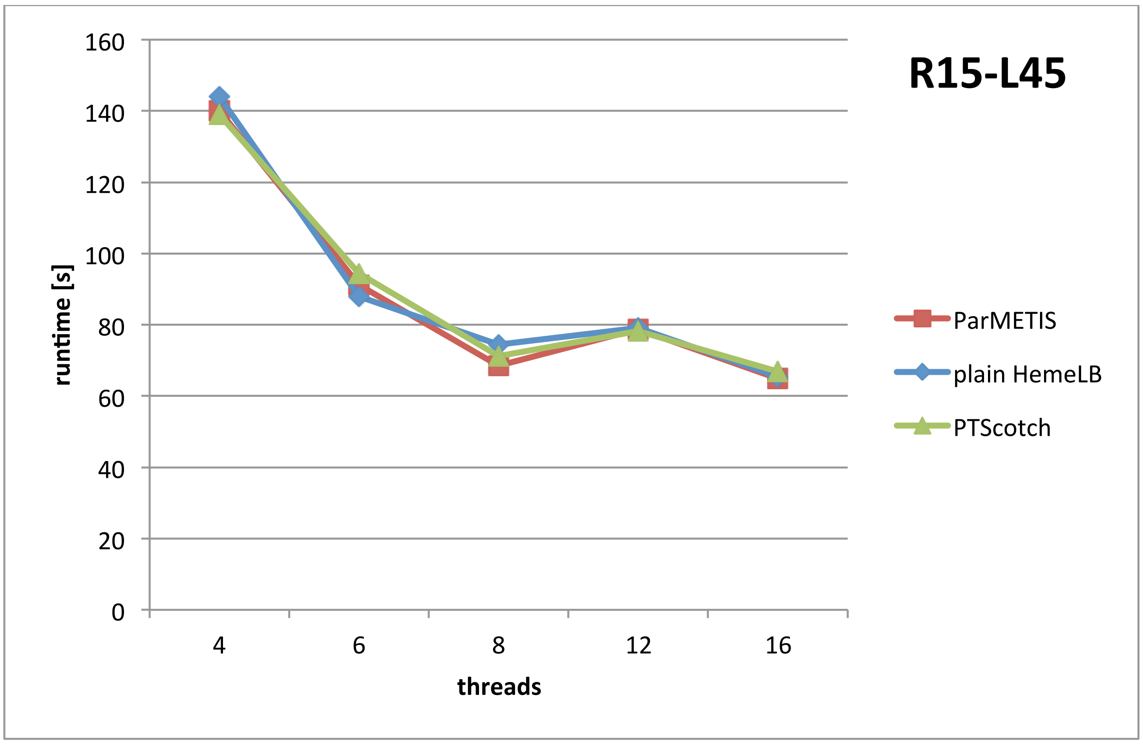

Figure 3: HemeLB runtimes for data set R15-L45 with plain HemeLB, HemeLB with PPStee using ParMETIS and HemeLB with PPStee using PTScotch on Intel Xeon E5520

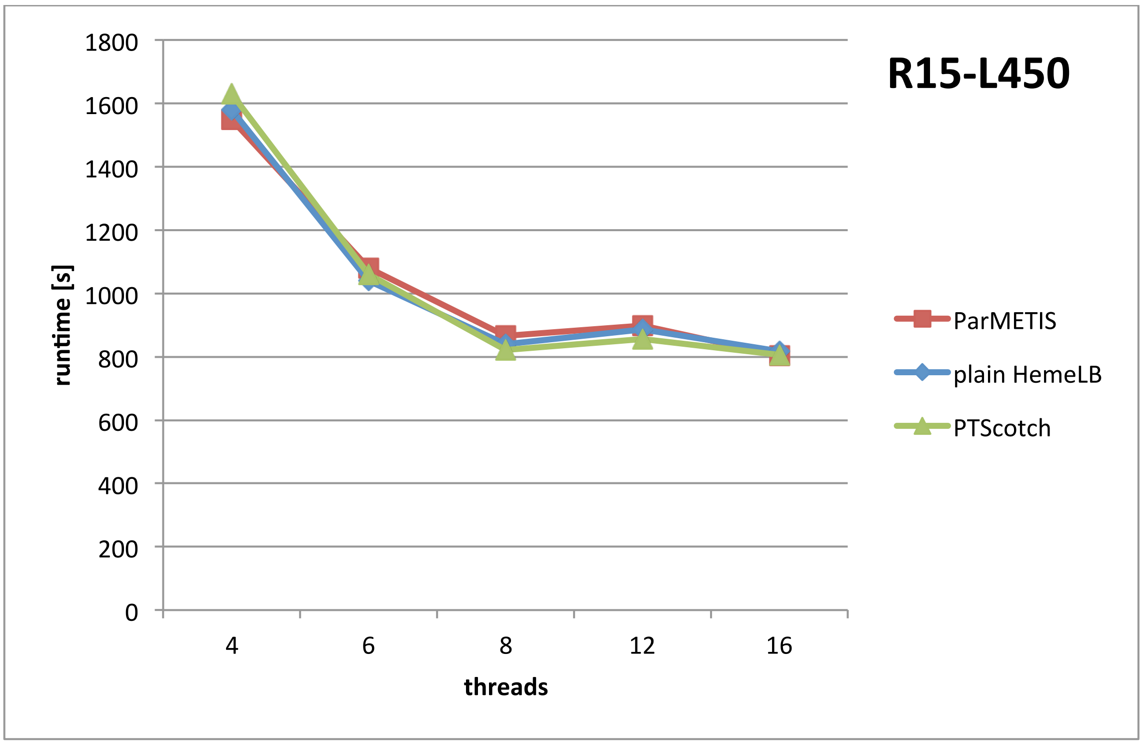

Figure 4: HemeLB runtimes for data set R15-L450 with plain HemeLB, HemeLB with PPStee using ParMETIS and HemeLB with PPStee using PTScotch on Intel Xeon E5520

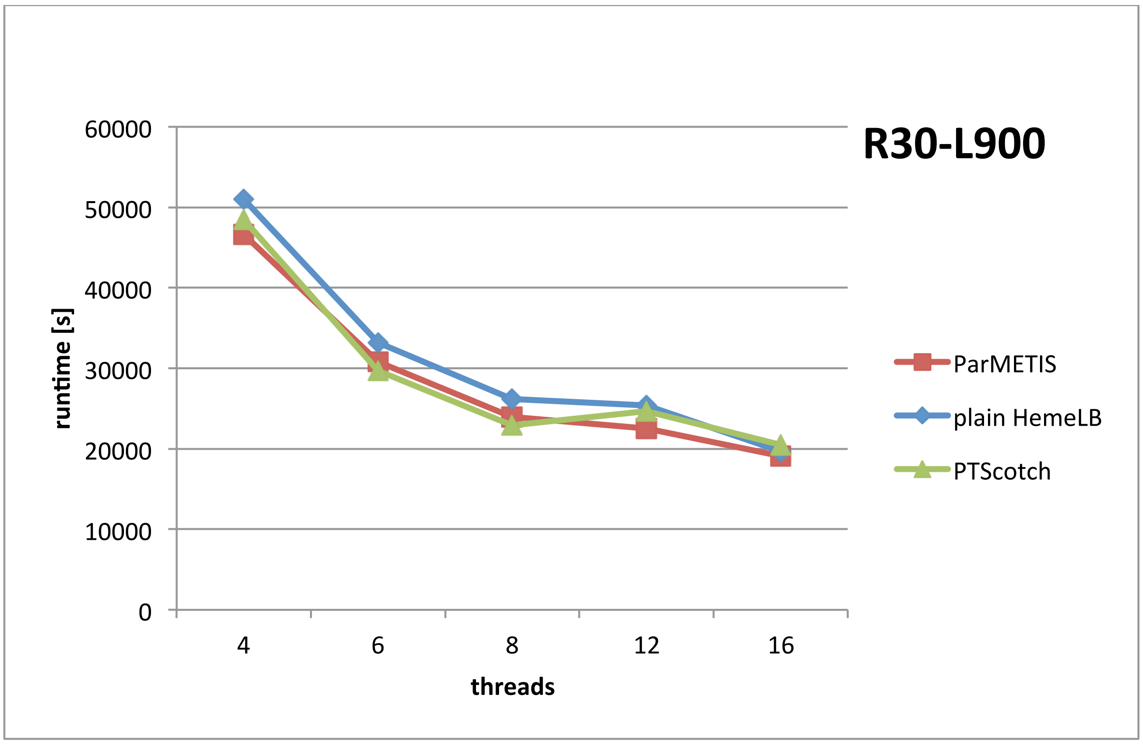

Figure 5: HemeLB runtimes for data set R30-L900 with plain HemeLB, HemeLB with PPStee using ParMETIS and HemeLB with PPStee using PTScotch on Intel Xeon E5520

In all three charts we find a close match of runtimes for all three methods, i.e. plain HemeLB (using ParMETIS), HemeLB with PPStee using ParMETIS and HemeLB with PPStee using PTScotch. There are slight performance gains and losses here and there; however, a general tendency with respect to a method or process count cannot be identified.

Concluding for low process counts, we see no penalty in runtime when using PPStee. On the one hand, this proves that PPStee does not introduce any significant overhead in computation time. On the other hand, PPStee offers the possibility for an improvement in runtime when using another partitioner or other data sets, in particular in case of large data sets and high process counts.

4.3 Measurements

The following measurement data were collected on HECToR. In its current stage (Phase 3), HECToR is a CRAY XE6 system and contained in 30 cabinets and comprise of a total of 704 compute blades. Each blade contains four compute nodes giving a total of 2816 compute nodes, each with two 16-core AMD Opteron 2.3GHz Interlagos processors. Each 16-core processor shares 16 GB of memory.

HemeLB was modified as described in section 4.1 to incorporate PPStee in three different versions. Each version uses the initial HemeLB mesh data setup in ParMETIS' data layout and passes these data to one of the three partitioners ParMETIS, PTScotch and Zoltan. We used a plain, i.e. unmodified, HemeLB version as a reference and always used fully populated nodes adding up to the specified thread count.

The data sets "data_01M" and "data_02M" are cylindrical test geometries with about 1 million and 2 million lattice sites, respectively. They run for three cycles and 1000 steps per cycle. The third data set is a bifurcation geometry with approximately 650 thousand lattice sites and runs for 1000 time steps.

Figure 6, Figure 7 and Figure 8**Error! Reference source not found.** depict runtime values in seconds on 32 up to 2048 (512 for bifurcation) cores for all four configurations and data sets "data_01M", "data_02M" and bifurcation, respectively.

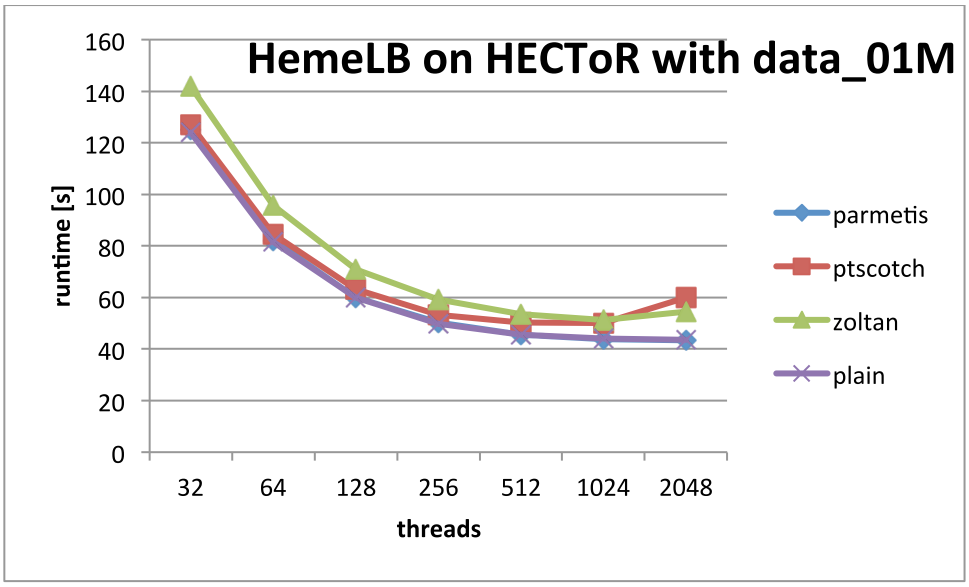

Figure 6: HemeLB runtimes for data set data_01M with plain HemeLB and HemeLB with PPStee using ParMETIS PTScotch and Zoltan on HECToR

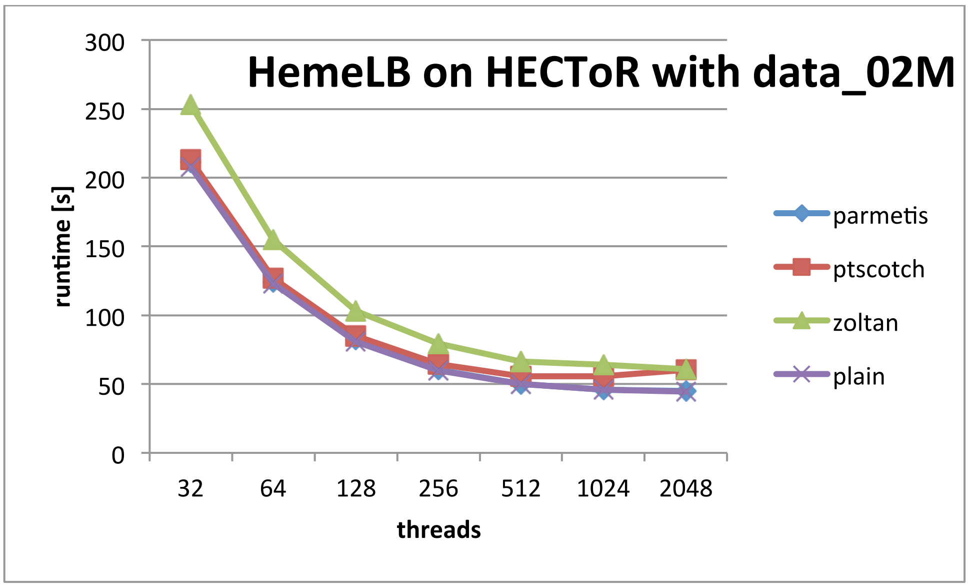

Figure 7: HemeLB runtimes for data set data_02M with plain HemeLB and HemeLB with PPStee using ParMETIS PTScotch and Zoltan on HECToR

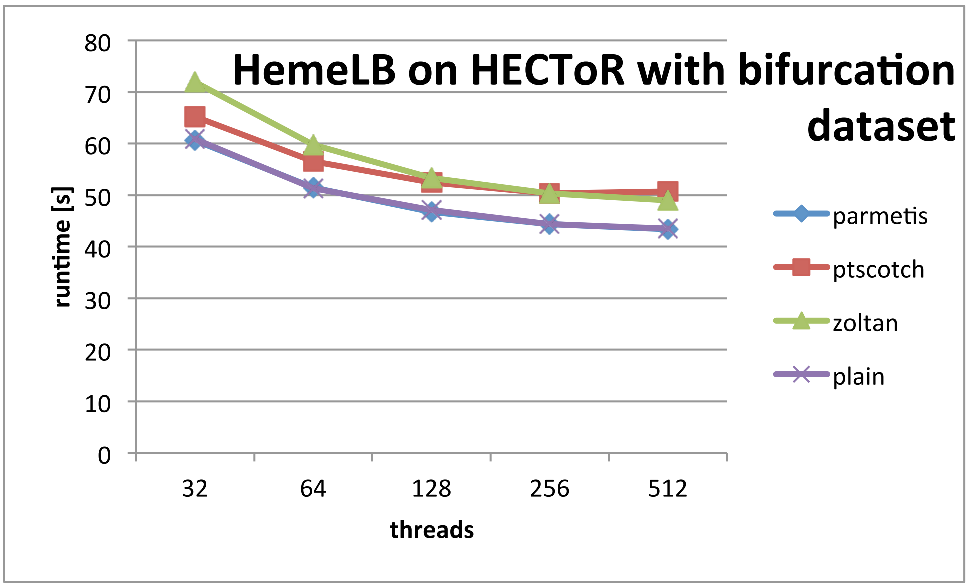

Figure 8: HemeLB runtimes for bifurcation (50um) data set with plain HemeLB and HemeLB with PPStee using ParMETIS PTScotch and Zoltan on HECToR

We see a very close match for all datasets between HemeLB with PPStee using ParMETIS and plain HemeLB that uses ParMETIS by default. PPStee with PTScotch stays close to both ParMETIS versions for the cylindrical geometries but shows a growing increase in runtime for large thread counts.

For the bifurcation geometry, this increase is also present and the gap between PPStee with PTScotch and the ParMETIS versions is slightly bigger.

The runtime graph of PPStee with Zoltan has the same shape as the graph of the ParMETIS versions for all geometries. However, it lacks an almost constant amount of runtime of about 20% for all geometries and core numbers.

In general, we detect a far better scaling behaviour for both cylindrical datasets than for the bifurcation geometry.

4.4 Analysis

The measurements show a close match of plain HemeLB and PPStee with ParMETIS for all datasets. Since both versions use ParMETIS' data format and partitioning algorithms we can deduce that the only difference between both versions, i.e. the PPStee modifications, do not impose any noticeable runtime overhead on the simulation. Thus PPStee is safely applicable in the matter of runtime even if later change of the partitioner used does not prove to be beneficial for the simulation at hand.

PPStee with PTScotch is level-pegging with both ParMETIS versions for both cylindrical geometries and loses ground for the bifurcation dataset. The reason for this behaviour is unclear but could be related to the reduced regularity of the bifurcation geometry as compared to the cylindrical ones.

On the other hand, HemeLB with PPStee using PTScotch shows scaling problems: starting at 256 cores, runtime begins to increase slightly for the bifurcation dataset and more drastically in case of the cylindrical geometries. For the bifurcation dataset, this effect diminishes the aforementioned general deficiency of this geometry, yet it is impossible to predict which of both will dominate for larger core numbers. This scaling behaviour could also be caused by regularity issues of the underlying geometries.

The Zoltan configuration draws attention by constant gap in runtime compared to both ParMETIS versions. The shapes of the curves for the Zoltan versions match the corresponding ParMETIS curves quite well and thus indicate a correct, i.e. equivalent, scaling for both partitioners. Therefore, the reason for the gap, in particular, because it is constant over core counts, could be ascribed to the data layout issues: HemeLB passes its mesh data in ParMETIS' data format and a conversion is needed to pass these data on to Zoltan.

In total, we experienced an easy integration of PPStee into HemeLB. It is comfortable to switch the partitioner used and consequently easy to obtain partitioning results with other partitioners beside the one that was used originally. We see no drawbacks in runtime for the initially used partitioner ParMETIS and got the chance to compare it to PTScotch and Zoltan. For core numbers up to at least 512, the results encourage the use ParMETIS within HemeLB.

5 PPStee and OpenFOAM

5.1 OpenFOAM characteristics

An analysis of simulations with OpenFOAM (compare [7], CRESTA Deliverable 6.1.1, Roadmap to exascale) exhibits crucial points regarding any modifications to the OpenFOAM simulation chain. A simulation based on OpenFOAM is not to be seen as a monolithic simulation, i.e. each part of the simulation process that leads to the end result is a distinct phase in the simulation (Figure 9 depicts a possible sequence of simulation phases) and moreover a distinct executable program. Additionally, each of these simulation part programs is not necessarily parallel.

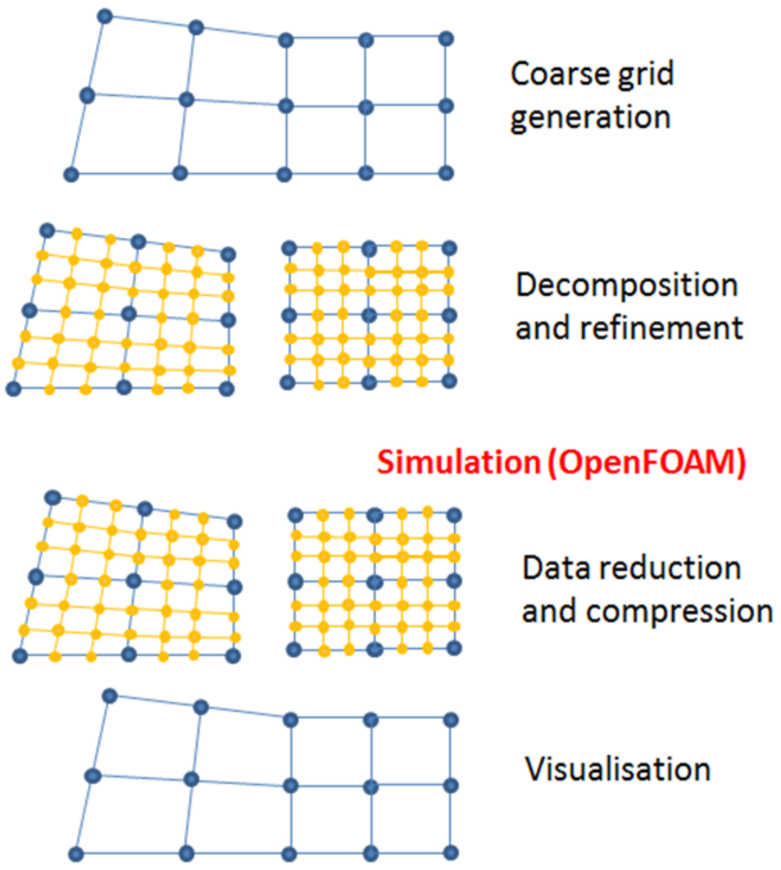

Figure 9: Schematic view of the overall work process of OpenFOAM

These phases are separated by design but, of course, have to exchange their input and output data respectively. This data transfer between two phases is done via hard disk and there is currently no way to bypass this behaviour. It's hardly surprising that this exposes a really bad scalability as seen for the simulation in its entirety.

5.2 Applicability of PPStee

The specific design of a simulation using OpenFOAM (cf. section 5.1) reveals that computational load balance of, for example, the solver is not the primary problem when trying to improve scalability of this simulation. The inevitable disk IO between the phases has a much higher impact on the overall performance of the simulation. If this problem is not resolved properly, an integration of PPStee in one of the parallel simulation parts will not increase overall simulation performance.

However, there is way to use PPStee to improve the situation a little bit. If there was a monolithic program that aggregates each phase of a simulation based on OpenFOAM, PPStee could be at least used to balance the disk IO of the parallel phases. PPStee's stages and their intrinsic weights (cf. section 2.1, Properties, in [3]) introduce the possibility to consider each phase's amount of disk IO separately. At the end of the day, this might lead to an overall performance gain while, of course, never reach the exascale regime.

6 Revision of system, data format and algorithms definition

6.1 Confirmed properties

In section 4.1 we demonstrated the integration of PPStee into a mature code, i.e. CRESTA's co-design application HemeLB. The integration is straight forward when the application, as in the case of HemeLB, already uses a partitioner. The changes in the actual code are minimal. The position of modifications needed do not require deep code insights as they are easy to find by the partitioner call. And slight additions in the build system complete the simple integration.

Runtime results of HemeLB with PPStee, presented in sections 4.2 and 4.3, confirm the claim of almost no overhead introduced by usage of PPStee. Additional memory for graph data, beside the memory used anyway, is not required, thus PPStee's minimal data layout works out well as intended. Much more conveniently, there are no runtime drawbacks when using PPStee; at least this is true (as demonstrated in section 4.3) for the partitioner used originally. For the other partitioners, it may be necessary to tune the partitioners' parameters, see section 6.3 below.

The detailed analysis of the case study of HemeLB (see section 4) also shows one of PPStee's primary features: a switch of the partitioner used among the partitioners supported is easy. This way, the simulation and its graph and input data can be tested for its compatibility with ParMETIS, PTScotch or Zoltan quite similarly and without further effort.

6.2 Comparison with ITAPS

The Center for Interoperable Technologies for Advanced Petascale Simulations (ITAPS, [8]) delivers interoperable and interchangeable mesh, geometry, and field manipulation services. The identically named set of tools and libraries, ITAPS, focuses on various mesh manipulation techniques including front tracking, mesh quality improvement via smoothing and swapping, adaptive mesh refinement and dynamic load balancing.

However, ITAPS mostly pursues the classical character of pre-processing, i.e. a simulation part located prior to the simulation core essentially providing various improvement methods to the initially-raw input mesh. Although visualisation tools are included, there is currently no automated way to analyse and re-use visualisation output in a subsequent simulation cycle. Here, the difference to PPStee becomes apparent: PPStee, along with the other WP5 tools, is designed to close the simulation cycle thereby gaining overall simulation performance over dedicatedly-separate simulation architectures.

6.3 Further investigation needed

Beside the confirmed properties of PPStee described so far, we stumbled upon points where further investigation is needed. For a start, we have seen that swap of the partitioning library used is easy. However, the measurements (see section 4.3) also show a constant deficiency when using Zoltan. This may correlate to the specific graph data structure of HemeLB, but it may point to inherent conversion problems concerning Zoltan's graph data, too. Additional examination could improve the way PPStee organises its data.

When graph data conversion is excluded as the underlying problem, the constant gain in runtime of HemeLB with PPStee and Zoltan could indicate an issue with the default partitioner values. Currently, PPStee calls Zoltan, and the other two partitioning libraries, without a specification of special parameters, hence default parameters are applied. If these default parameter set does not interact well with the simulation graph data it has to be customised accordingly. Additionally, partitioners offer individual routines that can be used to adapt to the graph data. It is an open question which specific routines and parameters can be applied to adjust a partitioner to a simulation. A significant performance gain may be possible and thus it should be investigated.

The measurements of runtime of HemeLB with PPStee and PTScotch (compare, particularly, Figure 6 and Figure 7) reveal another aspect that needs further attention. The figures show a serious problem with scalability for PTScotch starting around 512 threads. Analogous to Zoltan, reasons may be an incompatibility of unmodified HemeLB graph data with PTScotch or the lack of usage of individual routines and parameters PTScotch provides and needs to be set. Further runtime measurements with higher thread counts are necessary to investigate the scaling behaviour beyond 2048 cores, primarily for PTScotch but for ParMETIS and Zoltan as well.

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

[5] 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/

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

[7] CRESTA Deliverable 6.1.1, Roadmap to exascale

[8] ITAPS, Interoperable Technologies for Advanced Petascale Simulations, http://www.itaps.org