D5.1.3 – Pre-processing: first prototype 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:	M18
Submission date:	31/03/2013
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)	George Mozdzynski (ECMWF), David Henty (UEDIN)

Dissemination level	
PU	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	20/09/2012	First notes.	Gregor Matura (DLR)
0.2	26/02/2013	First version of the deliverable	Gregor Matura (DLR)
0.3	28/02/2013	First revision	Gregor Matura (DLR)
0.4	15/03/2013	Merging first internal revisions	Gregor Matura (DLR)
1.0	20/03/2013	Final version of the deliverable for submission	Gregor Matura (DLR)

Table of Contents

Index of Figures

1 Executive Summary

This deliverable is a software deliverable, providing a first prototype interface for pre-processing steering named PPStee. In this document we provide a brief overview of the software.

The analysis (and system definition) of pre-processing ([1]) illustrated the main target of pre-processing: an overall simulation load-balance. All simulation costs, for example work load and communication time, must be included in the calculation of load-balance to guarantee the best-possible system performance. Work load and communication costs do not only arise in the simulation core, i.e. the solver and its associated tasks, but also in the other parts surrounding a simulation. Input data preparation, post-processing and (remote) rendering move closer to the simulation core and likewise the system, especially in the exascale regime.

This has to be addressed properly by a tighter coupling of pre-processing within the simulation cycle. Pre-processing cannot be seen as an external pre-simulation component any more. Simulation-intermediate interaction is necessary.

To facilitate this new role of pre-processing, an exchange of information with the simulation core, post-processing and other simulation parts is needed. These interfaces must be designed and installed. They pass useful data between these components and introduce the option of steering. This enables direct performance gains for each part because of better adaption to the input data. Furthermore the simulation becomes a lot more interactive and thus user friendly.

As the simulation parts merge, new capabilities emerge. For example, the transition from a run-and-stop fashion to a repeating simulation loop allows for the possibility of an optimised repartitioning. Timing measurements of the old cycle can be used directly to improve the partition quality of the following cycle and additionally determine whether it is reasonable to redistribute the data in this cycle in the first place. In contrast to a non-interactive simulation, calculation costs may vary in between cycles and so has the partitioning.

With a pre-processing interface in mind, a suitable data format has to be specified (described in D5.1.2, pre-processing: data format and algorithms, [2]). The main constraint is the desire for a minimal amount of data, but that is sufficiently large to retain all the details of the input data. This ensures both low communication times during data transfer and a minimal memory footprint.

Based on this simulation input data and its layout, pre-processing should provide an algorithm properly adjustable to the unique simulation data structure and its needs. It may be necessary to compare different load-balancing methods in terms of scalability and performance of the resulting partitioning.

This deliverable is a software deliverable, providing a first prototype interface for pre-processing steering named PPStee. This software feeds into the simulation cycle a graph or mesh data and various communication costs and work load from all simulation loop components. It uses state-of-the-art partitioning libraries to provide an overall simulation load-balance and can be extended with further functionality such as mesh manipulation methods or connection to a fault tolerance framework.

In this document we sketch features and properties of PPStee and show advantages and disadvantages of its architecture. We illustrate the integration into a simulation work flow in terms of both data flow in combination with PPStee and actual implementation using a basic usage example. We point out the current software status and future work.

The PPStee source code and its documentation can be found in the CRESTA SVN (/wp5/preprocessing/).

2 Introduction to PPStee – a pre-processing interface

PPStee is an interface for pre-processing steering. It ships as a library and is implemented in the pre-processing phase of a simulation. Supplied with information on simulation data, its main purpose is to optimise the overall simulation load-balance starting with initial data distribution and not necessarily ending after visualisation of the simulation results.

2.1 Properties

PPStee is built around various partitioning tools, namely ParMETIS[3], PTScotch[4] and Zoltan[5]. These are established and widely used libraries. They provide partitioning capabilities which are mostly congruent among each other. Each of them can be used to retrieve a decent load-balance for a simulation. Yet they have been developed independently and use different approaches to compute the partitioning. This leads to a partitioning of different quality depending on the input data and therefore on the simulation. PPStee offers an easy-to-implement mechanism to swap the choice of partitioning tool with only slight changes to the code. By doing so, the obtained timings can lead to a better choice of partitioner for the simulation and therefore directly improve load-balance.

The data format of PPStee orientates towards compatibility and a minimal footprint. The possibility of direct data access without additional copy operations and a minor overhead for internally used data improves memory consumption. The overhead is not entirely needed but can save on cost-intensive collective communications. These can occur if a conversion of the native partitioner data format is calculated. Nevertheless, the PPStee data format is designed to be capable of this conversion and to do it quickly and cheaply.

PPStee's main task is to balance the simulation load. This is not a new approach; all the mentioned partitioners do so. Yet, PPStee provides a disruptive feature: it incorporates different simulation stages by default. Hence not only the computation of the simulation core is balanced. Other parts, such as visualisation, can provide their calculation and communication load on the data too. These parts are naturally present in exascale simulations due to the unfeasibility of off-situ processing of the huge data amounts. A true simulation-covering load-balance is gained.

The modular architecture and the flexible data format make PPStee easily adjustable. New partitioning tools, whether they are developed directly in PPStee or stand-alone, can be integrated with minor effort. Further stages can be added if the need arises. PPStee offers several places to introduce fault tolerance techniques. For example, an extension to PPStee could take care of a redundant backup of the graph data which in turn is used to recover lost data if one processor dies. In addition, PPStee could prevent the usage of this processor and the simulation can continue with minimal delay. Furthermore, mesh refinement routines are conceivable which can alter the submitted graph data automatically or adjusting it to the system's structure.

2.2 Integration into the simulation work flow

In general, PPStee does not allocate any memory (with the exception of a tiny amount of private data) and is not responsible for any data movement[1]. The simulation keeps track of the graph or mesh data and the accessibility of this data throughout the simulation lifetime. This way the integration of PPStee into an existing code is kept simple and the least disruptive for the simulation's data flow. The responsibility for the data belongs completely to the simulation.

[1] Obviously, more advanced features such as mesh manipulation techniques break this general rule.

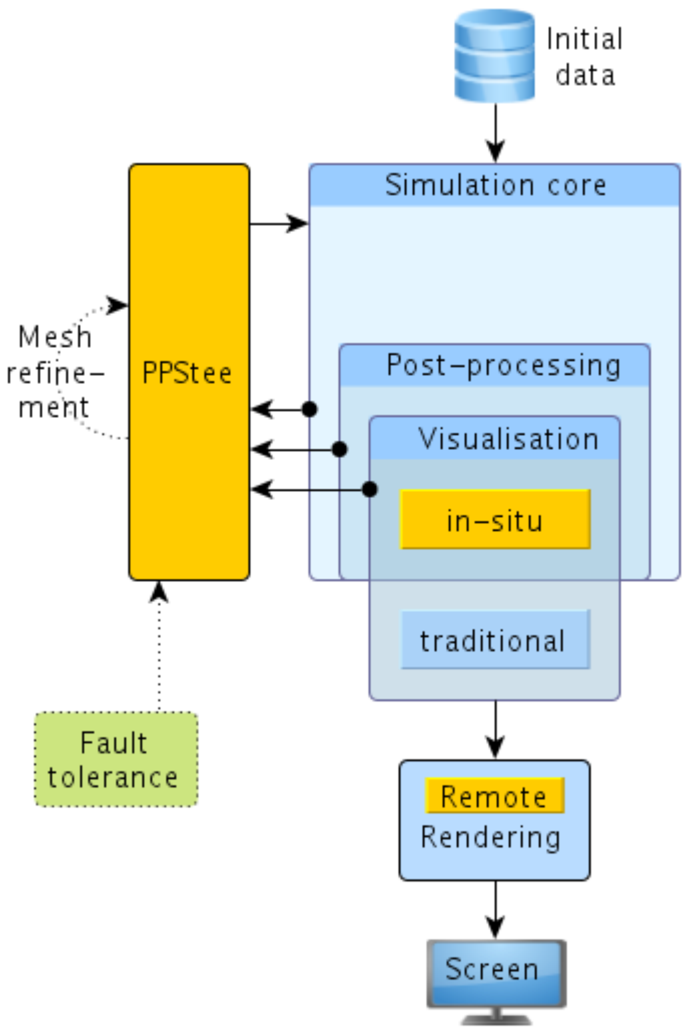

Figure 1: PPStee flow chart

The work and data flow is illustrated in Figure 1: PPStee flow chart. The simulation core reads the initial data, usually some kind of geometry, from the file system and submits this data in the form of a graph to PPStee. Additionally, it provides the work load and communication costs that it will use as (graph) weights, where work load matches weights for the vertices and communication is mapped onto the edges. These weights can be estimates based on former simulation runs or precise prediction based on a proper investigation of the code. Also, a posteriori measurement should be used to improve the reliability of these figures.

Furthermore, all other simulation components should submit their weights too, whenever possible. For example, this includes the work load of any result data analysis carried out in the post-processing phase. This is especially interesting for cost-intensive calculation and communication in in-situ visualisation, as these presumably are of the same order of magnitude as the solver costs. Basically, every task done within the simulation code and executed on the cluster should provide its cost to guarantee a gapless load-balance throughout the full simulation loop.

Finally, the simulation core retrieves a partitioning either directly after the initial submission of the graph and all corresponding weight estimates or, in subsequent cycles, triggers a calculation of an updated partitioning. This repartitioning should be based on timing and costs measurements of previous cycles and thus is better balanced than the initial partitioning. Due to the data responsibility, the simulation core compares the re-partition to the current partition and decides whether it is worth the effort to move the corresponding data.

In summary, the result is a load-balance covering the complete simulation loop. For a single-loop simulation this result requires estimates from former runs, for a simulation traversing multiple cycles the result becomes even better using adaption and repartitioning.

Apart from this main data flow, some insertions will be possible in the future. Mesh manipulation techniques could be applied after initial graph submission or between cycles. After initial graph submission, the mesh could be smoothed or used to generate a finer mesh. Between the cycles, the result of a previous data analysis in the post-processing component could trigger a refinement of spots in the mesh where this modification leads to a more accurate or faster solution.

Additionally, a fault tolerance framework could interact with PPStee and steer the distribution of data. If, for example, some nodes drop out graph and load data could be adjusted to the new cluster status. Obviously, a decent data backup and recovery mechanism would be required.

2.3 Advantages

PPStee's main advantage is the standardised partitioner access. Once PPStee's data structures are created and filled with the according graph data the partitioner is chosen arbitrarily. This introduces the option to independently change the partitioner used. Then, timing measurements and other tests can be used to reveal the best-suited partitioner for the simulation.

PPStee relies mainly on established external partitioning tools. Their mature and methodologically sound algorithms are used and provide partitioning at a state-of-the-art level. Additionally, PPStee's basic data containers can be used to manufacture a partitioning routine particularly tailored for the user's needs. Later, a direct integration in PPStee is possible.

PPStee comes with little programming overhead. If a partitioner is already implemented in a simulation the changes required to make use of PPStee is minimal. PPStee provides function signatures very similar to those native to the partitioners. All data structures can be kept and used, making data handover and reception of the resulting partition relatively easy.

PPStee requires only a small amount of additional memory. PPStee uses only a little auxiliary data for internal book keeping. The full graph data can be passed by reference thus keeping memory obligations at the simulation side. Data access is read-only; whether it can be freed afterwards depends on its usage: for example stage weights should be kept alive if the simulation will do more than one cycle and thus needs a later repartitioning.

2.4 Disadvantages

PPStee accesses only basic routines of the partitioning libraries although most of them provide extended features which may improve the partitioning quality. This certainly is a side effect of the standardised access. On the other hand, this very access helps to indicate whether a further investigation of these extended features is reasonable. Also, if a specific extended routine becomes crucial in the future it can be integrated into PPStee.

Another point to mention is the insertion of another software layer by PPStee. Although this should not negatively affect the simulation, it does increase the complexity and may lead to undesired or faulty behaviour which may become harder to track.

3 Implementation details

3.1 Basic usage

In this section we describe how to use PPStee based on an example implementation. We assume an existing code that initialises its data and then does a standard ParMETIS call:

```
ParMETIS_V3_PartKway(
    vtxdist, xadj, adjncy,
    vwgt, adjwgt,
    wgtflag, numflag, ncon, nparts,
    tpwgts, ubvec, options, edgecut,
    part,
    comm);
```

to retrieve a partitioning named `part`. Other partitioners can be used analogously.

We start by initialising a `PPSteeGraph` object with the graph data we have, i.e. `vtxdist` for the global vertex distribution and `xadj` and `adjncy` for the thread-local adjacency structure:

```
// get graph (as ParMETIS type)
PPSteeGraph graph =
    PPSteeGraphParmetis(MPI_COMM_WORLD, vtxdist, xadj, adjncy);
```

Next, we construct weights objects derived from the graph as these have to be compatible. We fill in weights for the computation and visualisation part. These weights denote the work load (vertex weights, `xadj`) and communication time (edge weights, `adjncy`) each simulation part needs.

```
// construct and set weights for computation
PPSteeWeights wgtCmp(&graph);
wgtCmp.setWeightsData(vwgt_c, adjwgt_c);
// construct and set weights for visualisation
PPSteeWeights wgtVis(&graph);
wgtVis.setWeightsData(vwgt_v, adjwgt_v);
```

Now, we establish an instance of the interface's main object and submit our graph and weights data.

```
// get interface
PPStee ppstee;
// submit graph
ppstee.submitGraph(graph);
// submit weights
```

```
ppstee.submitNewStage(wgtCmp, PPSTEE_STAGE_COMPUTATION);
ppstee.submitNewStage(wgtVis, PPSTEE_STAGE_VISUALISATION);
```

Finally, we trigger the calculation of the partitioning and get the desired partitioning.

```
// calculate partitioning
PPSteePart* part;
ppstee.getPartitioning(&part);
```

3.2 Software status

PPStee is shipped as source code with a CMake build system configuration for cross-platform support. The source code contains Doxygen mark-ups for automated generation of the documentation and example files to demonstrate usage. The current version 0.1.0 supports ParMETIS and PTScotch; Zoltan support will be available soon. Basic functionality is proven in a standalone system test. Mesh manipulation methods and fault tolerance techniques are not implemented.

The first CRESTA co-design application that will use PPStee is HemeLB with integration of PPStee still on-going. Currently we are testing the interaction of HemeLB and PPStee to guarantee an eventual operational availability. A switch of the partitioner will be tested; load-balance and performance measurements are planned to compare the quality of partitioning results. In conclusion, we obtain the partitioner that suits HemeLB data the best.

3.3 Future work

Future work focuses on further tests and a revision of PPStee. We will test PPStee components and functions in detail, both separately and in combination with the CRESTA co-design applications. HemeLB will be of prime interest; later, the integration of a more evolved version of PPStee will be tested with other applications such as OpenFOAM or Elmfire.

We wish to analyse PPStee by comparing the stated aims versus achieved goals. Does PPStee really provide a better overall simulation performance, especially regarding load-balance? Is the integration of PPStee into an existing project simple enough? Is it too simple because it misses relevant advanced features of the partitioners? We will also compare PPStee with other frameworks for pre-processing that cover similar features to PPStee. For example, Interoperable Technologies for Advanced Petascale Simulations (ITAPS, [6]) focuses on various mesh manipulation techniques and load-balance but the provided visualisation support is not included in the load-balance calculation.

This revision process of PPStee and its functionality will be examined in-depth in CRESTA Deliverable 5.1.4 “pre-processing: revision of system, data format and algorithms definition for exascale systems” [7].

4 References

[1] CRESTA Deliverable 5.1.1, pre-processing: analysis and system definition

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

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

[4] 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/

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

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

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

Annex A.

A.1 Documentation

PPStee documentation is provided in a separate PDF file or can be downloaded in the CRESTA SVN (/wp5/preprocessing/).