Submitted to **Parallel Computing**, Special Issue on "Graph Partitioning and Parallel Computing"

Dynamic cost modelling and load balancing for mesh-based applications

A. Basermann[1], J. Fingberg[1], G. Lonsdale[1], B. Maerten[2], R. Ducloux[3], C. Walshaw[4]

[1] C & C Research Laboratories
NEC Europe Ltd.
Rathausallee 10,
D-53757 St. Augustin
Germany

[2] K.U. Leuven
Dept. Computer Science
Celestijnenlaan 200A
B-3001 Heverlee-Leuven
Belgium

[3] Transvalor S.A.
Les Espaces Delta
BP037
06901 Sophia Antipolis
France

[4] U. Greenwich
Centre for Numerical Modelling
and Process Analysis
Wellington St, Woolwich
London SE18 6PF
UK

Abstract: The DRAMA project is a European Commission (ESPRIT)-funded project which has been initiated to support the take-up of large scale parallel simulation in industry by dealing with one of the main problems which restricts the use of message-passing simulation codes - the inability to perform dynamic load balancing. A concentration on message-passing parallelisation corresponds to the target of addressing large scale and thus highly scalable parallel applications. The central product of the project will be a library comprising various tools for dynamic re-partitioning of unstructured finite element applications. The particular focus of the project is on the requirements of industrial Finite Element codes, with library evaluation and validation being performed using industrial software packages, but codes using Finite Volume formulations will also be able to make use of the project results. The core library functions will perform a **parallel** computation of a mesh re-allocation that will re-balance the costs of the application code based on the DRAMA cost model. This paper will discuss the design features of the library, which allow a general approach to load identification, modelling and minimisation. Results will be presented which both justify the inclusion of single-phase/uni-constraint graph partition components and point ahead to the requirements for multi-phase/multi-constraint versions.

Key Words: dynamic load balancing, mesh re-partitioning, parallel Finite-Element applications

1 Introduction

The effectiveness of parallel computing has been clearly demonstrated for scientific applications ranging from purely academic problems to simulation codes used within industrial design. However, particularly in the latter area, the advancement beyond "exploratory installations" requires fully efficient, fully scalable solutions to all problems of interest to the end-user and involving all code functions. In many areas of simulation, a crucial component for efficient numerical computations is the use of solution-driven adaptive features: locally-adapted meshing or re-meshing; dynamically changing computational tasks. The full advantages of HPC technology will thus only be able to be exploited when efficient parallel adaptive solvers can be realised. As discussed in [11], the resulting requirement for HPC software is for dynamic load balancing, which for many mesh-based applications means dynamic mesh re-partitioning. The

ESPRIT project DRAMA (project No. 24953, “Dynamic re-allocation of meshes for parallel Finite Element applications") has been initiated to address this issue, with a particular focus being the requirements of industrial Finite Element codes, but codes using Finite Volume formulations will also be able to make use of the project results.

Given the importance of the theme and the generality of the requirement for dynamic load balancing of HPC applications, it is clear that the DRAMA objectives will be shared with other international developments. The following brief discussion highlights the differences between DRAMA and other activities and positions the project’s approach w.r.t. those activities.

Two major aspects distinguish DRAMA from alternative approaches: the development of a general purpose *library*; the evaluation and validation with leading commercial software for industrial simulation. The latter feature, in addition to ensuring the industrial relevance of the approach, includes state-of-the-art 3-D parallel adaptive meshing technology in development at the DRAMA project partner, CEMEF. The library approach is a most important distinction and a strength of the project, since it should ensure a widespread take-up of the project results, especially since the final DRAMA library will be put into the public domain. A general-purpose approach is also indicated as a future direction, though as an object-oriented tool rather than a standard library, in [11]. The alternative to the library approach is of course to develop load balancing & mesh re-allocation options within a particular application and many groups are, and have been, investigating such an approach.

As will be discussed in more detail below, the DRAMA library performs re-partitioning of the ***mesh*** (based on all costs arising on that mesh), and supports the data re-allocation with old-new mesh numbering information. The partitioning of the mesh is maintained even when using (appropriately interfaced and/or modified) graph partitioners inside the library, due to the mesh-to-graph (and inverse mapping) components. Thus the project is addressing weaknesses in graph partitioning methods which are under discussion within state-of-the-art forums (see, for example, [10]). It will be seen in Section 3 that the graph partitioning component is indeed able to minimise mesh-based costs. Within the DRAMA library, two state-of-the-art tools will be included: ParMETIS as an integral part and JOSTLE via a compatible interface (a form of "*plug-and-play"* option). Those two tools are widely accepted and have a significant number of users (though mainly for static partitioning).

A development that is to a certain extent complementary to the approach taken by DRAMA is the PLUM environment ([4]). The PLUM environment for dynamically balancing a hierarchically refined mesh focuses on minimising the costs of data repartitioning via a heuristic remapping algorithm. It represents the mesh by a fixed dual graph (with variable weights to represent refinement levels) of the coarsest mesh. A more general research environment, UG ([2]), allows applications to build on parallel, multilevel components for unstructured meshes and includes dynamic load balancing via the DDD tool ([3]). UG's primary aim is to be a tool for the exploration of new discretisation schemes, solvers and error estimators. Consequences of the focus on research are that Fortran interfacing has not yet been considered and absolute performance (in terms of data structures allowing high optimisation across all HPC platforms) has not been a driving issue.

By performing dynamic mesh re-allocation within the application, the DRAMA approach is to increase the efficiency of that application and thus of **its** exploitation of the HPC system being used. The focus on important simulation codes for industrial design is at the same time a focus on applications for which the turn around time of individual jobs plays a critical role. Many projects and developments have taken an alternative route to dynamic load balancing, which is not directly comparable with the DRAMA developments: Via interaction with the operating system

they target the optimal use of the system resources rather than the scalability of individual important applications.

Following an overview of the DRAMA project, and in particular of the applications whose requirements are helping to drive the library design, we will discuss how well a general library and cost model is able to be used to capture the occurring costs and introduce new developments which will be investigated to address the major shortcoming for some applications: separated computational sections, for example due to multiple synchronisation points, which require a "multi-phase" or "multi-objective" minimisation approach. Results will be presented which illustrate that the graph partitioning components are able, through the mesh $\leftrightarrow$ graph transformations, to minimise the DRAMA cost function. In addition to performance results for parallel library execution on representative application meshes, it will be seen that initial results with the full Forge3 code show total application gains when using the DRAMA library.

2 The DRAMA Project

In the following, a brief introduction to the activities of the DRAMA project will be given. The reader is referred to the project web-site [12] for further information about the project and its partners, and in particular to the (public) project deliverables made available there. As stated above, the central product of the project will be the DRAMA Library. The library interface, and underlying cost model, have been designed such that all information can be provided by the application based on its actual local data and measured costs (via code instrumentation). This aspect will be addressed for two of the "DRAMA applications" in Section 3. An overview of these applications, whose role in the project is to evaluate and validate the DRAMA library, will be given in Section 2.1. Section 2.2. summarises the design decisions taken in the construction of the library. Detailed information concerning the DRAMA Cost model and library interface are given in [13,14,20,21].

2.1 The DRAMA Project Applications

The mesh re-allocation approach to dynamic load balancing will be demonstrated and validated by the leading industrial codes PAM-CRASH (for crashworthiness simulation), PAM-STAMP (for metal stamping / deep-drawing and related simulations), FORGE-3 (for forging with viscoplastic incompressible materials). Despite this emphasis on the validation codes within the project, the library has been designed to be general purpose. All the DRAMA applications use time-marching as basic solution procedure and both explicit (PAM-CRASH/-STAMP) and implicit (FORGE3) methods are included. Causes of load-imbalance, and resulting degradation of scalability, are: (a) a dynamic behaviour of computational cost per element and of the communication patterns; (b) meshes which are changing during the calculation - adaptive meshing or re-meshing, including reshaping, refinement and coarsening. The self-impacting contact-impact algorithms used in PAM-CRASH are extreme cases of the former. Adaptive meshing is essential for codes like FORGE3 or PAM-STAMP where the large deformations would otherwise result in extremely severe distortions of the mesh elements.

2.1.1 FORGE-3 & Parallel Adaptive Remeshing

FORGE3 from Transvalor is an implicit finite element code designed for the simulation of three-dimensional metal forming. It is able to simulate the large deformations of viscoplastic incompressible materials with unilateral contact conditions. The code is based on a stable mixed velocity/pressure formulation using tetrahedral unstructured meshes and employs an implicit time

stepping technique. Central to the Newton iteration dealing with the non-linearity arising from the behaviour of material and the unilateral contact condition is an iterative procedure based on a conjugate residual method for the solution of the large linear system.
The parallelisation of the full code, including adaptive re-meshing, was done within the EUROPORT project ([24]) employing a mesh partitioning approach. For forging simulations, the capability for re-meshing is a unique, competitive advantage of the FORGE3 code. A functioning 3-D parallel re-meshing procedure has been established which requires a repartitioning stage ('element migration'), not only to avoid load imbalance but also to deal with the interface re-meshing. This has to date been achieved via a centralised re-allocation process, which becomes a bottleneck, especially for large problems or when a large number of processors is used.
The reader is referred to [5,8,9] and the references therein for further information.

2.1.2 PAM-CRASH & PAM-STAMP

The PAM-CRASH and PAM-STAMP codes are two of the ESI/PSI Group products that are built around the PAM-SOLID core solver libraries. The most crucial components within the crashworthiness code PAM-CRASH are the contact-impact algorithms whose main feature, from the DRAMA viewpoint, is the dynamically changing computation and communications costs. Contact-impact algorithms are also crucial to the simulations performed by PAM-STAMP, but the much more significant parallelisation requirement is the efficient handling of adaptive meshing since around 90% of stamping applications rely on the adaptive meshing features. In contrast to the re-meshing approach adopted by FORGE3, PAM-STAMP uses a mesh-refinement (and coarsening) strategy based on the original user-defined mesh. The current message-passing versions have been further developed from the prototypes produced within the EUROPORT ([24]) and EUROPORT-D ([15]) projects.
Leaving details of the algorithms and their parallelisation to [6,19] and the references therein, a summary of the solution methods of the PAM-SOLID-based codes is as follows: an explicit time-integration; a non-linear finite element method using a Lagrangian unstructured mesh; element-wise stress-strain calculations supplemented by penalty method contact-impact algorithms to detect and correct penetration of structural components.

2.2 The DRAMA Library

The DRAMA Library is designed to be called by parallel message-passing (MPI) finite element (in general, mesh-based) applications; the library itself is written in C and C++ and exploits MPI, it may be called by applications written in both Fortran and C. The "expectation" of such applications is for the rapid provision of information about: a re-partitioning of the mesh which balances the costs occurring in the application; the interaction between processes required to achieve the re-partitioning. Given the normal complexity and application dependence of such algorithms, the actual data migration would not be expected of the library. Thus, the DRAMA library and its re-partitioning algorithms must be efficient, parallel (operating on distributed data) and must also take the current partition into account, in order to avoid high communication costs during the resulting data migration. Furthermore, it should be based on actual occurring costs, rather than some abstract heuristic. The current library design and re-partitioning modules included has taken these requirements into account by the careful definition of the cost model and library interface. A summary of this strategy would be: "The DRAMA Library is designed to balance in parallel the actual costs occurring on the application's finite element mesh".

2.2.1 DRAMA Cost model and Library Interface

The interface between the application code and the library is designed around the DRAMA cost model (which results in an objective cost function for the load balancing re-partitioning algorithms) and the instrumentation of the application code to specify current and future computational and communication costs. The DRAMA cost model provides a measure of the quality of the current distribution and is used for the prediction of the effect on the computation of moving some parts of the mesh to other sub-domains. Calculation and communication speeds of the processors are taken into account by a combination of hardware specific parameters and costs which are based on time measurements and enumeration provided by application code instrumentation. Heterogeneous machine architectures can also be taken into account in this way. The essential feature is that the cost model is **mesh-based**, so that it is able to take account of the various workload contributions and communication dependencies that can occur in finite element applications. Being mesh-based, the DRAMA cost model includes both per element and per node computational costs and element-element, node-node, and element-node data dependencies (for communication).
In addition to data dependencies between neighbouring elements and nodes in the mesh, dependencies between arbitrary parts of the mesh can occur. For the PAM-CRASH code, such data dependencies originate within the contact-impact algorithms when the penetration of mesh segments by non-connected nodes is detected and corrected. The DRAMA cost model (and of course the library interface) allows the construction of "virtual elements" which represent the occurring costs of such dependencies.

Referring to [20,21] for further details, we will repeat some of the definitions used in the DRAMA Cost Model here for ease of description in subsequent sections. All occurring costs may be assigned a '*type*', which allows the application to distinguish costs arising for different algorithmic components. An example is given, for nodal computation costs in Section 3.1.1.
Computational '*phases*' are computational sections separated by synchronisation points, for example, global communications steps arising due to the need to calculate global quantities (some reduction operation). Denoting the sum of all occurring costs on a processor, i, for phase, j, by F_i^j, then the total costs over the total number of phases (denoted by 'nphases'), F, is given by

$$\mathbf{F} = \sum_{\mathbf{j}=1,\ldots\mathbf{nphases}} \max_{\mathbf{i}=0,\ldots\mathbf{p}-1}(F_i^j). \tag{1}$$

The load imbalance λ^j, given by

$$\lambda^{\mathbf{j}} = \frac{\max_{\mathbf{i}=0,\ldots\mathbf{p}-1}(F_i^j)}{\mathbf{average}_{\mathbf{i}=0,\ldots\mathbf{p}-1}(\mathbf{F_i^j})} \tag{2}$$

The current library design includes several types of mesh repartitioners that may be selected by the application: mesh-migration ([7,22]), graph partitioning & geometric (co-ordinate-based, [1,11]) partitioning. A discussion of the various approaches and of the motivation for their choice for the DRAMA library are given in [14]; Section 2.2.2 below will give further information on the graph partitioning options, since these will be the subject of the cost function minimisation studies reported on in Section 3. The library builds upon the partitioning options with modules to provide the interface to the full DRAMA input mesh and the cost monitoring parameters and to deliver the full DRAMA output mesh and old ↔ new mesh relationships. The latter information is provided in a format that will allow the application code to directly build the appropriate

communication constructs (mailing lists) in order to perform the associated data migration corresponding to the re-partitioning.

2.2.2 Graph Partitioning within the DRAMA Library

'Classical' graph partitioning methods employing weighted graphs derived from either element or nodal mesh connections would be unable to fully account for the costs arising in a finite element application in general. The **mesh-to-graph** module of the DRAMA library constructs an appropriate weighted graph from the distributed mesh. Depending on the properties and the needs of the application, the resulting graph can be an 'element graph', a 'node graph', or a combined 'element-node graph'. The latter contains all possible relevant cost contributions for finite element codes. For a given partition, edges between nodes, elements or elements and nodes represent different communication requirements between processors. For instance, edges between elements and nodes lead to communication when a sub-domain possesses an element but not all its nodes. The combination of the mesh-to-graph module with a suitable graph partitioner results in a **mesh partitioner** based on the DRAMA cost model**.**

Within the current version of the DRAMA library, the subsequent graph partitioning is carried out by calling routines from PARMETIS, the software package developed by Karypis et al., University of Minneapolis ([18,23]). PARMETIS contains several strategies for graph re-partitioning; in particular a multilevel method based on 'diffusing' load to adjacent partitions. The idea behind this multilevel technique is that from the originally graph a hierarchy of coarser graphs is generated (by merging graph vertices to 'supervertices'). A careful re-partitioning of the coarsest graph is computed, and then this new partitioning is successively 'projected' onto the next finer graph and improved (again via load diffusion). In addition to this graph partitioning module the co-operation with the University of Greenwich has developed a DRAMA interface to a modified version of the JOSTLE mesh partitioning software ([25]) which extends the range of particular options which will be investigated and validated.

3 Cost Capture and Minimisation, Performance Results

In the discussion of the DRAMA library above, emphasis was placed on the development of a cost model, and corresponding library interface, which represented the actual application costs occurring on the (finite element) mesh. Questions raised by this are: Can the application represent its costs using the cost model ? Can the cost model be optimised by the combination of mesh-to-graph module and graph partitioner ? Can that optimisation be performed in an acceptable time for realistic meshes ? What is the benefit for the application ?

At the time of writing, not all questions can be answered completely:

The very first question is dependent on the code's communication structures and is in part the subject of current developments to introduce multi-phase/-objective partitioning. Section 3.1 will discuss the instrumentation of the PAM-CRASH and Forge3 codes and will show that even the dynamically changing computation costs of contact-impact algorithms in PAM-CRASH can be successfully "captured". The need for the multi-phase partitioning approach will be demonstrated using the latter application.

The final question on performance can only be fully answered when final benchmarking has been completed, but very promising first results **are** available, as will be seen in Section 3.3. The main focus of that section, however, will be to demonstrate that the DRAMA cost function may indeed be effectively handled using the mesh-to-graph plus graph partitioner approach and that the parallel re-partitioning may be handled in acceptable time scales.

3.1 Cost Capture

Avoiding details of the full set of parameters which may be provided to define the cost model in general ([13,21]), the crucial parameters to be provided are numbers of operations (for nodes, elements) for various types of occurring computations and numbers of bytes to be communicated across particular data dependencies (element-node, etc.). The important point is that actual timings can be used – from code instrumentation – to provide the computational times and code monitoring (counting) may be included to gather the appropriate communication volumes. Since communication latency is, from an instrumentation point of view, equivalent to load imbalance, time measurements are not used for communication costs.

Before turning to an example from PAM-CRASH, where code instrumentation is essential to correctly model the dynamically changing computation and communication structures related to contact-impact calculations, it should be noted that in some cases detailed instrumentation is not necessary:

For the Forge3 code, the main computational cost of the iterative solver is in the matrix vector product, which is performed at each iteration. It is done independently on each subdomain and the global vector is built by adding the contributions obtained on the different processors. The floating operations reside in the computation of matrix-vector products while the communications take place during the addition. The performance of the solver depends only on the number of iterations (and thus the number of matrix vector products) which varies as $O(N^{3/2})$ (however, this can vary with the preconditioner), N being the number of mesh nodes. The matrix vector product remains a short step in itself, the cost being in the great number of matrix vector products to be done. From the above, and the excellent agreement between predicted and actual costs, it can be seen that an 'instrumentation' of the code based on numbers of mesh nodes is perfectly sufficient to provide the information to the DRAMA library interface and means that the introduction of wall-clock timers around the computations is unnecessary.

To illustrate the possibilities with code instrumentation, we take an example from the instrumentation of the PAM-CRASH code. The numbers of operations per node of type 'type', $nop^n(type)$, are set adaptively after every 1000 time-steps (a "monitoring interval") from timings over the monitoring interval, $\Delta t^{calc,n}(type)$, based on the individual routine and code section timings within each time-step (using MPI_WTIME):

$$nop^n(type) = \Delta t^{calc,n}(type) \; / \; N^n(type),$$

where $N^n(type)$ is the number of occuring nodes of that type. For instance, the node type may be a particular sub-set of nodes involved in a particular contact-calculation, where the sub-set may be defined over the monitoring interval. The monitoring interval would be chosen to fit to the expected frequency of DRAMA library calls (which, it should be pointed out, may be used simply to check on load balance and may not necessarily result in a re-partitioning calculation).

Of course, a certain level of averaging is thus introduced, but the application is able to choose a monitoring interval that is appropriate for the expected computational variation. By the use of virtual elements, dynamic dependencies and computational costs may be defined (and instrumented in the same way as static costs) and passed to the DRAMA library. We will see below, that the combination of actual costs can be modelled by the DRAMA cost model, with the restriction that the total costs should be the sum of costs for particular element and node computations plus the pure communications costs – a condition which is not satisfied when multiple synchronisation points occur within the application.

The PAM-CRASH code includes synchronising communications which mean that stress-strain calculations and contact-impact calculations are separated, i.e. the solution algorithm has

two distinct phases. The current cost model and previously available partitioning algorithms within the DRAMA library cannot take account of the two separated computational phases. Thus, trying to balance all occurring costs via a single DRAMA call, which implicitly assumes one phase without synchronisation points, may result in subsequent execution times which are either minimally improved or even worse (dependent on the relative imbalances and computational weights of the two phases). This is demonstrated in Figure 1 where time-histories of the DRAMA cost function per processor are displayed for each of the 16 processors before and after the calculation of a new partition with the DRAMA library. The cost function for this example included the stress-strain calculations and all dynamic costs from the contact-impact algorithms, but some global costs were omitted in order to highlight the dynamic behaviour. The DRAMA cost function was evaluated after every 1000 time-steps. Figure 1 also includes the corresponding total elapsed time for the code, Ttot, for the same monitoring interval, measured by calls to MPI_WTIME around the full time-step. The same global costs, which were omitted from the cost function, were subtracted from Ttot in order to enable a direct comparison with the cost function. Otherwise all remaining computation, idle and communications times are included. The Ttot times are of course independent of the process/processor number.

The restriction of the cost modelling to a single computational phase is the reason for the larger discrepancy between the total code elapsed time and the highest total cost model time after the DRAMA repartitioning, since increased idle times are introduced into the first phase, by moving elements from partitions with higher loads in the second (contact-impact phase). As a result, the code elapsed times after the DRAMA repartitioning partition are actually in excess of before it. *However, it can be seen that the DRAMA library has succeeded in minimising the cost function*: the spread of cost function histories after repartitioning is reduced by about a factor of two. The "migration" of elements away from the contact-impact-dominated processors is clearly visible when comparing the partitions in Figure 2. These results show that the basis for accurate cost capture is available. The approach to be followed within the DRAMA project is to use separate the costs into phases and to employ partitioners which can take account of this separation. First results will be presented in Section 3.2.3.

Figure 1: Comparison of the **total costs** included in the DRAMA cost function for the 2 different executions (initial, top, and DRAMA-produced, bottom) together with the total code elapsed time (Ttot) for the same intervals. Ttot is the upper line in both cases. All costs are normalised to seconds per time-step.

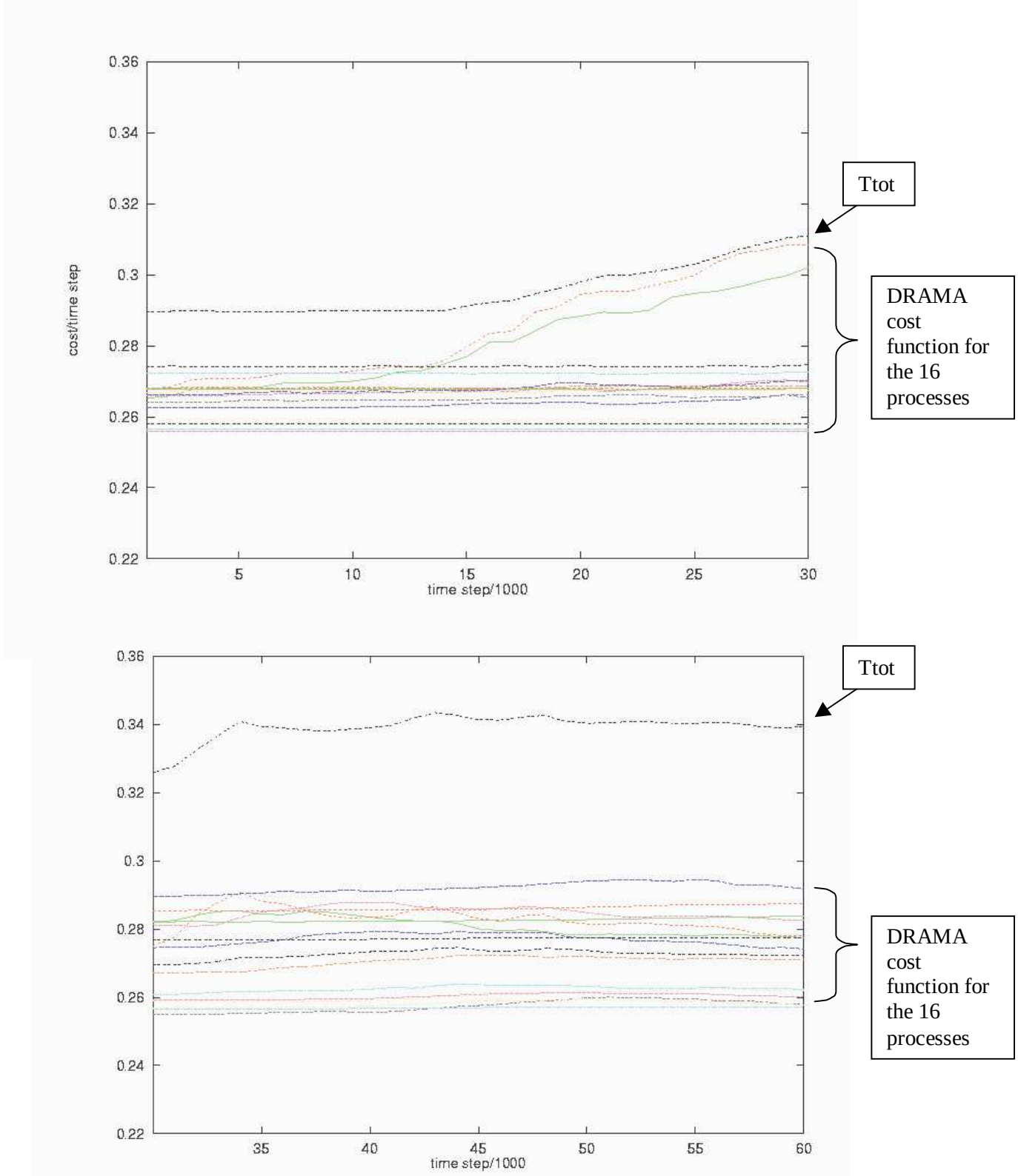
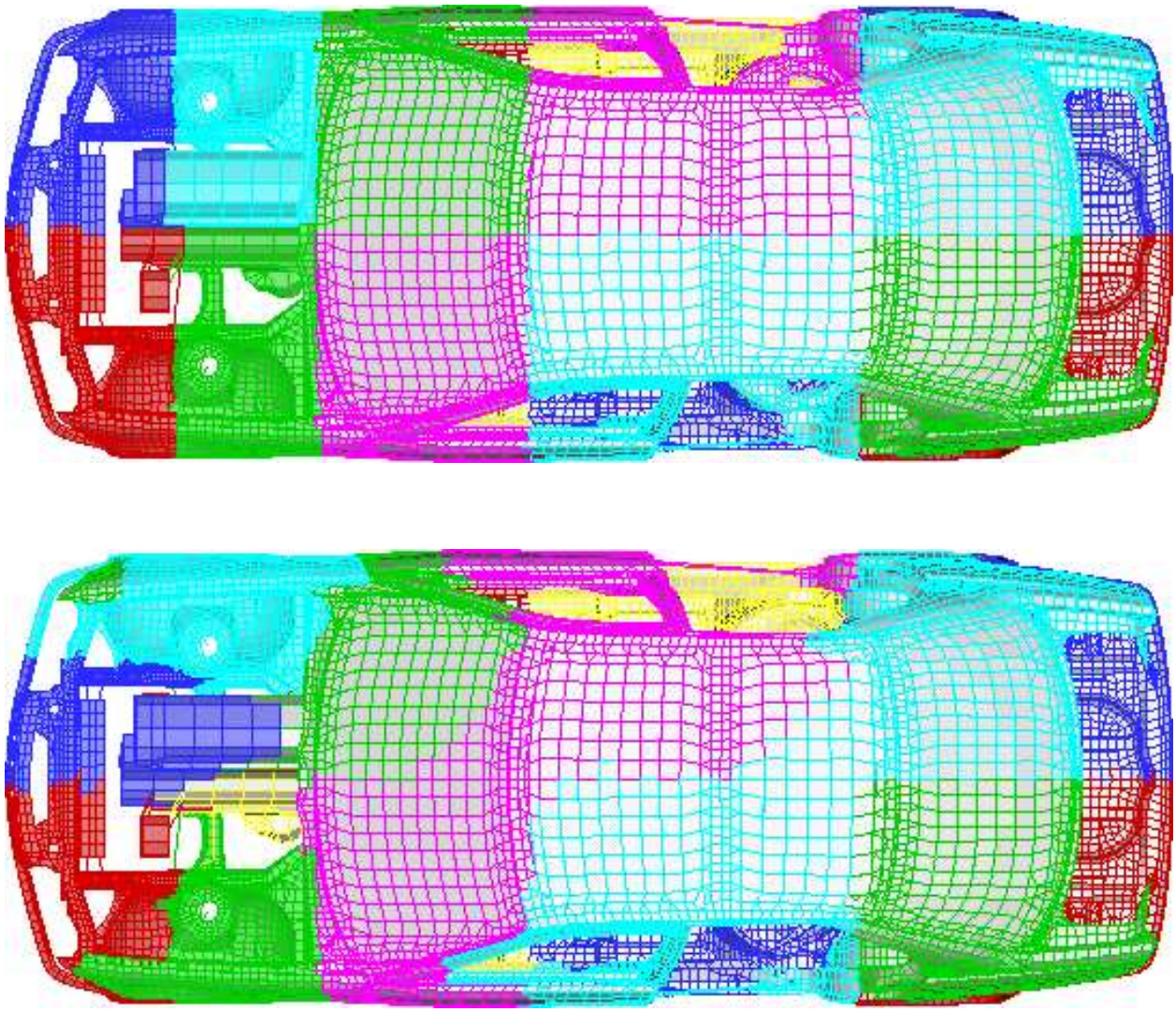


Figure 2.: The initial mesh showing the initial partition (top) and DRAMA repartition (bottom). (Courtesy of BMW AG)



3.2 Cost Minimisation via Graph Partitioning

3.2.1 Single-phase Partitioning for FORGE3

For a typical simulation with FORGE3, the finite element mesh consists only of tetrahedral elements; i.e. all elements are of the same type. For the following tests we have used a mesh consisting of 231846 tetrahedral elements and 49666 nodes, partitioned into 4 subdomains, that has been re-meshed such that the current partitioning is heavily unbalanced: about half of the mesh resides on one processor.

ParMETIS has been called with the global diffusion option and with various values for the allowed load imbalance (1%, 2%, and 5% imbalance), PJOSTLE was used with both the local and global options. Both were used with element and node graph representations. The dual graph is constructed by taking into account only the element based calculation costs (neglecting the node based calculation costs). In a second step the nodes are partitioned based on the element partitioning. The nodal graph is constructed by taking into account only the node based calculation costs (neglecting the element based calculation costs). The elements get a partitioning based on the partitioning of the nodes. The re-partitioning results are reported in Table 1.
Before and after re-partitioning, we have computed the predicted execution time for one iteration of the implicit solver by evaluating the cost function, F (Section 2.2.1, equation (1)). Note that both the element-based and the node-based calculation costs are taken into account when evaluating the cost function. Also presented are $\min_{i=0\ldots p-1} F_i$, the $\text{average}_{i=0\ldots p-1} F_i$ and the load imbalance λ (here, the superscript for phase number has been dropped). Also presented are the number of elements and nodes moved and the re-partitioning time on the NEC Cenju-4 (using 4 processors). The re-partitioning time is the time needed by the DRAMA library. The relative execution time of ParMETIS is for the nodal graph and the dual graph about 25% and 40%, respectively.

The results in Table 1 show that excellent re-partitioning is obtained for both the nodal and the dual graph, despite the fact that the node-based calculation costs and the communication latency are neglected in the dual graph representation. The nodal graph representation clearly leads to a good solution in much less time and with less memory usage. For the Forge3 code, there is no real advantage in using a combined graph and certainly not one that would justify the even higher memory requirements.

Further, the re-partitioning time (approx. 4.3s) is reasonable compared to the time between two calls to the DRAMA library. Indeed, one integration time step (approx. 2000 iterations of the implicit solver) requires about 56 seconds and re-partitioning is typically done every 10 to 20 time steps. If one then looks at the predicted gains by using the DRAMA library (taking the nodal graph and re-partitioning every 20 time-steps) we see:

Original code: $\text{time} = 20 \times 2000 \times 4.886 \times 10^{-2} = 1954.40$ secs,
With ParMETIS: $\text{time} = 20 \times 2000 \times 2.821 \times 10^{-2} + 4.34 = 1132.74$ secs,
With PJOSTLE: $\text{time} = 20 \times 2000 \times 2.693 \times 10^{-2} + 6.18 = 1083.38$ secs.

Thus, as demonstrated by the actual Forge3 results below, the prediction of the modelling is for significant gains when using the DRAMA library. What has otherwise been clearly demonstrated is that the DRAMA Cost Model can be effectively minimised by the graph partitioning modules.

Table 1: Results for various graph representations with a FORGE3 mesh consisting of 231846 tetrahedral elements and 49666 nodes and divided in 4 subdomains. The initial mesh distribution is heavily unbalanced (about half of the mesh resides on one processor).
ParMETIS has been called with 1%, 2%, and 5% imbalance, PJOSTLE with local and global options.

Dual graph						
		ParMETIS			PJOSTLE	
	Initial	1%	2%	5%	Local	global
$F_c = \max F_i$	4.886×10^{-2}	2.861×10^{-2}	2.821×10^{-2}	2.822×10^{-2}	2.795×10^{-2}	2.790×10^{-2}
$\min F_i$	1.382×10^{-2}	2.740×10^{-2}	2.695×10^{-2}	2.671×10^{-2}	2.555×10^{-2}	2.526×10^{-2}
average F_i	2.686×10^{-2}	2.795×10^{-2}	2.748×10^{-2}	2.737×10^{-2}	2.689×10^{-2}	2.657×10^{-2}
Imbalance, λ	81.92%	2.33%	2.64%	3.12%	3.97%	4.99%
Elements moved		60037	56725	52331	118573	101891
Nodes moved		11961	11276	10397	24913	21515
Re-partitioning time (secs)		10.92	10.76	10.74	16.90	16.99
Memory used		40 MB required for largest domain				
Nodal graph						
		ParMETIS			PJOSTLE	
	Initial	1%	2%	5%	Local	global
$F_c = \max F_i$	4.886×10^{-2}	2.836×10^{-2}	2.821×10^{-2}	2.877×10^{-2}	2.696×10^{-2}	2.693×10^{-2}
$\min F_i$	1.382×10^{-2}	2.728×10^{-2}	2.706×10^{-2}	2.634×10^{-2}	2.682×10^{-2}	2.676×10^{-2}
average F_i	2.686×10^{-2}	2.792×10^{-2}	2.757×10^{-2}	2.755×10^{-2}	2.690×10^{-2}	2.684×10^{-2}
Imbalance, λ	81.92%	1.57%	2.32%	4.44%	0.25%	0.31%
Elements moved		57335	56869	54910	98553	99052
Nodes moved		11236	11157	10717	20874	20981
Re-partitioning time (secs)		4.32	4.34	4.34	6.35	6.18
Memory used		25 MB required for largest domain				

3.2.2 DRAMA Library Costs for PAM-CRASH

DRAMA library costs for Forge3 meshes were given in Table 1, for partitioning with both nodal and element graphs. Since the natural option for the PAM-CRASH code is the use the combined graph, and because the PAM-CRASH meshes have a very different structure, especially since virtual elements are employed for the contact-impact cost capture (with non-local communications links), additional performance tests with the DRAMA library are reported here. As in Section 3.2.1, single-phase partitioning is used. Tests were performed with a BMW benchmark model (c.f. Figure 2) on the NEC Cenju-4 such that the mesh after 1000 time-steps was first input to the DRAMA cost function (i.e. library routine DRAMA_COSTFUN - see [13]) and then to the DRAMA Library to calculate a new partition. The re-partitioning calculation was performed using the partitioning options: ParMETIS-2.0 local diffusion

(ParMETIS_RepartLDiffusion) with load-imbalance threshold value 1.05. In practice, the expectation is for a full DRAMA Library call (calculating a new partition) to be performed with a frequency closer to every 10000 cycles. In fact, the subsequent time-steps will have a higher computational cost, as the contact-impact calculations become more expensive.

Based on the results of Table 2, a full DRAMA library call is expected to give an overhead of approx. 2% elapsed time, when called after every 10000 cycles and when using 63 compute P.E.s. The overhead is naturally smaller for fewer processors. The degradation in scaling of the DRAMA cost function is due to the use of collective operations within the cost function. The time-histories displayed in Figure 1 show a behaviour which is smooth enough to suggest that no real benefit would be obtained from cost function sampling at less than 1000 cycle intervals. Thus, the cost function overheads are, for all but the largest process numbers, insignificant.

Table 2: Execution times for DRAMA Cost function and DRAMA Library calls on the NEC Cenju-4 together with the total solution times for the first 1000 cycles. PAM-CRASH, BMW benchmark model.

Number of node processes	**Execution times in seconds**		
	DRAMA Cost Function	**DRAMA Library**	**PAM-CRASH for 1000 time-steps (including DRAMA costs)**
2	**0.0256**	**6.28**	**753.5**
4	**0.0526**	**6.84**	**397.8**
8	**0.0421**	**9.66**	**213.5**
16	**0.0613**	**11.7**	**127.2**
32	**0.132**	**13.1**	**81.4**
48	**1.17**	**13.9**	**71.7**
63	**0.713**	**14.2**	**63.6**

3.2.3 Multi-phase Partitioning for a PAM-CRASH Model

In this section we report on preliminary results with partitioning for the multiple-phase costs arising in PAM-CRASH (as discussed in Section 3.1). Experiments have been made to compare single-phase partitioning with the new multi-constraint option ([16,17]) in (sequential) METIS version 4.0 and a new multi-phase partitioning introduced in JOSTLE. The simplified benchmark case taken is a box-beam, comprising only 4-node shell elements, with the contact area limited to its lower part. In order to more clearly demonstrate the effects under consideration, the contact-impact phase costs, which occur only on dynamically changing sub-meshes, were scaled to be around three times larger than the shell element (stress-strain calculation) costs that occur over the full model. The contact pairs are modelled by 5-node virtual elements corresponding to a penetrating node and a surface segment made of 4 nodes. Repartioning was triggered after 10000 cycles and the resulting partitions from different balancing schemes are shown in Figure 3 and detailed in Table 3. Load balance quantities given in Table 3 are defined as follows:

λ_1 for shell calculations (phase 1), λ_2 for contact calculation (phase 2), $\lambda_{tot,1}$ for the total load balance assuming no synchronisation between phases and $\lambda_{tot,2}$ for the total load balance considering synchronisation between phases.

Figure 3:Different partitions into four subdomains: initial partition (top, left), ParMETIS global diffusion (top, middle), ParMETIS static *k*-way (top, right), METIS multi-constraint (bottom, left), PJOSTLE diffusion (bottom, middle) and JOSTLE multi-phase (bottom, right)

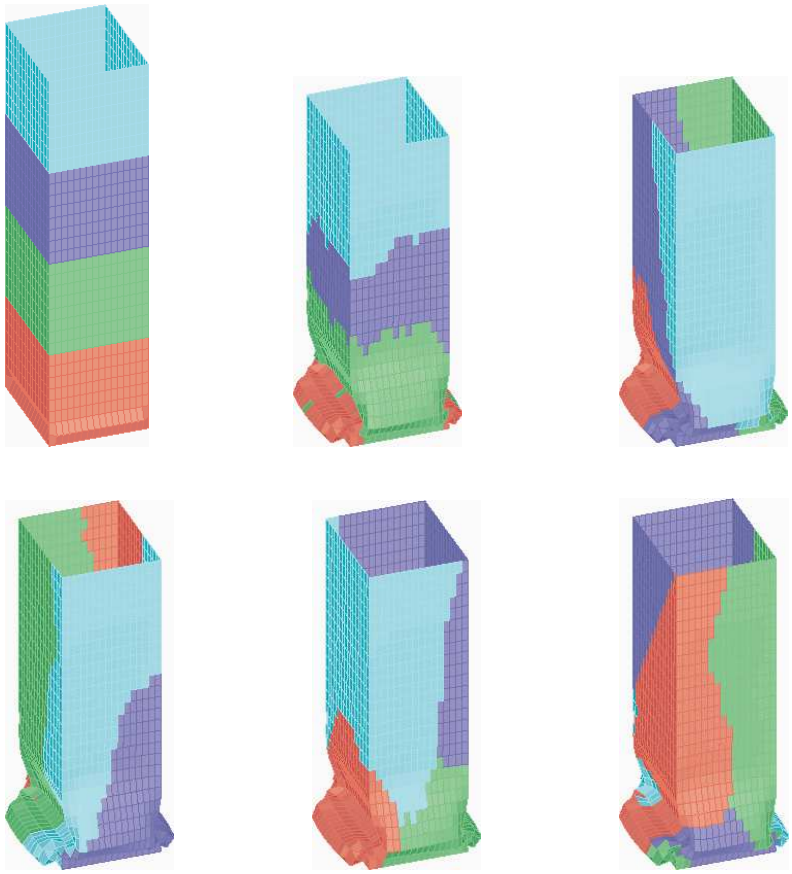


Table 3: Distribution of shell elements and contact pairs (CPs) per subdomain for different partitioning methods together with the the load imbalance factors.

	Initial		ParMETIS				METIS		PJOSTLE		JOSTLE	
			Diffusion		Static (k-way)		Multi-constraint		Diffusion		Multi-phase	
	shell	CPs	shell	CPs	shell	CPs	shell	CPs	shell	CPs	shell	CPs
PE 0	512	118	306	100	415	61	512	30	330	91	515	30
PE 1	512	0	553	118	599	0	512	30	516	27	515	29
PE 2	512	0	594	0	445	57	512	28	601	0	507	30
PE 3	512	0	595	0	589	0	512	30	601	0	511	29
λ_1, λ_2	1.00	4.00	1.162	3.390	1.170	3.085	1.000	1.017	1.174	3.085	1.006	1.017
$\lambda_{tot,1}$	1.442		1.011		1.026		1.002		1.004		1.007	
$\lambda_{tot,2}$	1.442		1.307		1.302		1.002		1.455		1.007	

Considering the results of Table 3, it is clear that only multi-constraint and multi-phase partitioning (with values of $\lambda_{tot,2}$ close to one) could succeed in improving the performance of the application although the other partitioning schemes minimised the aggregate cost function in each case (as can be seen from the values of $\lambda_{tot,1}$). The aggregate cost function, however, does not consider the presence of two synchronisation points and thus neglects idle times that are high in the single-phase approach.

3.3 FORGE3 Performance

The following full code measurements have been performed on a four processor DEC SMP machine. Two benchmark cases have been used. The DRAMA library plus data migration costs for the two models for the completed simulations are also given below. Note that the DRAMA library times are much smaller than those presented in Section 3.2: in addition to the use of a very large mesh there, a much larger load-imbalance was present in the input mesh. For the results with remeshing in sections 3.3.1 and 3.3.2, the initial imbalance is around 10%.

3.3.1 Forging of a Connecting Rod

The forging of a connecting rod has been completely simulated. Depending on the number of remeshings performed, the total number of elements is approximately 16000. The use of DRAMA improved the overall job times distinctly: the example took 77 minutes with DRAMA, whereas 97 minutes were needed with the original algorithm. The reduced CPU time is due to faster remeshing and to the fact that we use the node partition provided by DRAMA to increase the solver accuracy.
Times for both the DRAMA library re-partitioning and subsequent data migration were both below one second for each of the 39 remeshing steps. Thus, the total dynamic load balancing overhead is below 2% for this case.

3.3.2 Forging of a Brass Tap

The second benchmark is the simulation of the forging of a brass tap. The mesh size is approximately 120000 elements. The following results refer to the first 50 time steps of the simulation. For this partial simulation, the DRAMA version required 151 minutes as opposed to the 205 minutes needed with the original code. Five remeshing steps were performed.
In this case, each DRAMA library call required 4 seconds and the data migration is again below one second per remeshing step. Thus the total dynamic load balancing overhead is below 0.5%.

4 Concluding Remarks

With the ultimate release of the DRAMA library into the public domain, the aim of the DRAMA project is to enable a widespread exploitation of the library as a tool to allow efficient use of HPC platforms. By defining an interface for general mesh applications, with a choice of (state-of-the-art) re-partitioning components, a large section of the scientific computing community will be able to achieve scalable performance with their complex applications. The scope of the library exploitation is mesh-based applications that include dynamic and adaptive meshing or re-meshing and/or dynamically changing computational loads on otherwise static meshes. Though the project includes evaluation using industrial finite element codes, the mesh interface definition is also directly applicable to finite volume codes.

It has been shown that the conversion routines from the general mesh interface to an appropriate graph representation allow graph partitioners to minimise the (mesh-oriented) DRAMA Cost function. First results have shown that the overhead of the parallel execution of the library partitioners is low enough to allow major gains to be achieved within the application code. Future project activities will include rigorous evaluation and benchmarking for industrial problems.

That not all re-partitioning problems can be solved with a single-phase re-partitioning has been demonstrated with the PAM-CRASH code, where multiple computational phases occur. The current research developments into multi-constraint or multi-phase partitioners are also being investigated within the DRAMA framework and the first results presented here are most encouraging. The future focus will be to ensure that the improvements offered by these partitioners can be realised in parallel implementations.

Acknowledgements

The authors would like to thanks all our colleagues from the DRAMA project for their support and for the many lengthy and fruitful discussions without which this work would not have been possible. Particular thanks go to ESI for providing access to the PAM-CRASH code and to BMW AG who kindly permitted the use of the model whose mesh is illustrated in Figure 2.
The support of the European Commission through the ESPRIT IV (Long Term Research) Programme is gratefully acknowledged.

5 References

[1] S. A. Attaway, E. J. Barragy, K. H. Brown, D.R. Gardner, B. A. Hendrickson and S. J. Plimpton, Transient solid dynamics simulations on the Sandia/Intel Teraflop computer, *Proceedings of the ACM/IEEE SC97 Conference* (available on CD-ROM from ACM Member Services, email - orders@acm.org), Technical Paper, 1997

[2] P. Bastian, K. Birken, K. Johanssen, S. Lang, N. Neuss, H. Rentz-Reichert and C. Wieners, UG - A flexible toolbox for solving partial differential equations, *Computing and Visualisation in Science*, to appear.

[3] K. Birken, Dynamic distributed data (DDD) in a parallel programming environment - DDD Reference Manual, *Forschungs- und Entwicklungsberichte RUS-23*, Rechenzentrum der Universität Stuttgart, 1994

[4] R. Biswas and L. Oliker, Experiments with repartitioning and load balancing adaptive meshes, *NASA Ames Technical Report NAS-97-021*, 1997

[5] J.L. Chenot, L. Fourment, T. Coupez, R. Ducloux, E. Wey, FORGE3 - a general tool for practical optimization of forging sequence of complex three-dimensional parts in industry, *ImechE C546/033* , 1998

[6] J Clinckemaillie, B Elsner, G Lonsdale, S Meliciani, S Vlachoutsis, F de Bruyne and M Holzner, Performance issues of the parallel PAM-CRASH code, *Int. J. Supercomputer Applications and High Performance Computing*, **11(1),** 3-11,1997

[7] T. Coupez, Parallel adaptive remeshing in 3D moving mesh finite element, *Numerical Grid Generation in Computational Field Simulation, Vol. 1 (B.K. Soni et.al Editors)*, 783-792, Mississippi University, 1996

[8] T. Coupez, S. Marie and R. Ducloux, Parallel 3D simulation of forming processes including parallel remeshing and reloading, *Numerical Methods in Engineering '96 (Proceedings of 2nd ECCOMAS Conference, J.-A. Désidéri et. al. Editors)*, 738-743, Wiley, 1996

[9] T. Coupez and S. Marie, From a direct solver to a parallel iterative solver in 3D forming simulation, *Int. J. Supercomputer Applications and High Performance Computing*, **11(4),** 205-211, 1997

[10] B. Hendrickson, Graph partitioning and parallel solvers: Has the Emperor no clothes ?, *Proceedings Irregular '98, Lecture Notes in Computer Science* ***1457***, Springer-Verlag, 1998

[11] B. Hendrickson and K. Devine, Dynamic load balancing in computational mechanics, *Comp. Meth. Applied Mechanics & Engineering*, to appear.

[12] The DRAMA Consortium, *Project Homepage:* http://www.cs.kuleuven.ac.be/cwis/research/natw/DRAMA.html

[13] The DRAMA Consortium, Library Interface Definition, *DRAMA Project Deliverable* ***D1.2a***, in [12], 1998

[14] The DRAMA Consortium, Report on Re-Partitioning Algorithms and the DRAMA Library, *DRAMA Project Deliverable* ***D1.3a***, in [12], 1998

[15] EUROPORT-D ESPRIT HPCN Project No. 21102, *Project Homepage*: http://www.gmd.de/SCAI/europort-d/

[16] G. Karypis, and V. Kumar, Multilevel Algorithms for Multi-Constraint Graph Partitioning, *Proceedings of the ACM/IEEE SC98 Conference* (available on CD-ROM from IEEE Computer Society Membership Services, email cs.info@computer.org), Technical Paper, 1998

[17] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. *Technical Report TR 98-019, Department of Computer Science, University of Minnesota*, 1998.

[18] G. Karypis, K. Schloegel and V. Kumar, PARMETIS Parallel graph partitioning and sparse matrix ordering library, *Version 1.0, Dept. of Computer Science, University of Minnesota*, 1997

[19] G. Lonsdale, A. Petitet, F. Zimmermann, J. Clinckemaillie, S. Meliciani and S. Vlachoutsis, Programming crashworthiness simulation for parallel platforms, *Mathematical and Computer Modelling*, to appear.

[20] B. Maerten, A. Basermann, J. Fingberg, G. Lonsdale, D. Roose, Parallel dynamic mesh re-partitioning in FEM codes, *Advances in Computational Mechanics with High Performance Computing* (Proceedings of the 2nd Euro-Conference on parallel and distributed computing for computational mechanics, B.H.V. Topping Ed.), Saxe-Coburg, 163-167, 1998

[21] B. Maerten, D. Roose, A. Basermann, J. Fingberg, G. Lonsdale, DRAMA: A library for parallel dynamic load balancing of finite element applications, *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, March 22-24, 1999, CD-ROM, to appear.

[22] C. Ozturan, H. L. de Cougny, M. S. Shephard and J. E.Flaherty, Parallel adaptive mesh refinement and redistribution on distributed memory computers, *Comp. Meth. Mech. Engnrg.*, **119,** 123-137, 1994

[23] K. Schloegel, G. Karypis and V. Kumar, Multilevel diffusion schemes for repartitioning of adaptive meshes, *J. Parallel and Distributed Computing*, **47,** 109-124, 1997

[24] K. Stüben, H. Mierrendorff, C.-A. Thole and O. Thomas, Parallel industrial Fluid Dynamics and Structural Mechanics codes, H. Liddell, A. Colbrook, B. Hertzberger and P. Sloot (Eds.), *Lecture Notes in Computer Science* ***1067,*** 90-98, Springer-Verlag, 1996

[25] C. Walshaw, M. Cross and M. Everett, Dynamic load-balancing for parallel adaptive unstructured meshes, *Parallel processing for scientific computing (M. Heath et. al. Eds.)*, SIAM, 1997