ELSEVIER

Applied Mathematical Modelling 25 (2000) 83–98

APPLIED MATHEMATICAL MODELLING

www.elsevier.nl/locate/apm

Dynamic load-balancing of finite element applications with the DRAMA library

A. Basermann [a], J. Clinckemaillie [b], T. Coupez [c], J. Fingberg [a], H. Digonnet [c], R. Ducloux [d], J.-M. Gratien [b], U. Hartmann [a], G. Lonsdale [a,*], B. Maerten [e], D. Roose [e], C. Walshaw [f]

[a] *C &C Research Laboratories, NEC Europe Ltd., Rathausallee 10, D-53757 St. Augustin, Germany*
[b] *ESI S.A., Rue Saarinen 20, Silic 270, 94578 Rungis Cedex, France*
[c] *CEMEF (Ecole des Mines/Armines), Rue Claude Dauness, BP 207, 06904 Sophia Antipolis, France*
[d] *Transvalor S.A., Les Espaces Delta, BP037, 06901 Sophia Antipolis, France*
[e] *K.U. Leuven, Department of Computer Science, Celestijnenlaan 200A, B-3001 Heverlee-Leuven, Belgium*
[f] *Centre for Numerical Modelling and Process Analysis, University of Greenwich, Park Row, Greenwich, London SE10 9LS, UK*

Received 1 June 2000; accepted 1 August 2000

Abstract

The DRAMA library, developed within the European Commission funded (ESPRIT) project DRAMA, supports dynamic load-balancing for parallel (message-passing) mesh-based applications. The target applications are those with dynamic and solution-adaptive features. The focus within the DRAMA project was on finite element simulation codes for structural mechanics. An introduction to the DRAMA library will illustrate that the very general cost model and the interface designed specifically for application requirements provide simplified and effective access to a range of parallel partitioners. The main body of the paper will demonstrate the ability to provide dynamic load-balancing for parallel FEM problems that include: adaptive meshing, re-meshing, the need for multi-phase partitioning. © 2000 Elsevier Science Inc. All rights reserved.

Keywords: Dynamic load-balancing; Software library; Mesh partitioning; Finite element

1. Introduction

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 high performance computing (HPC) technology will thus only be able to be exploited when efficient parallel adaptive solvers can be realised. As discussed in [1], the resulting requirement for HPC software is dynamic load-balancing, which for many mesh-based applications means dynamic mesh re-partitioning.

* Corresponding author.
E-mail address: lonsdale@ccrl-nece.technopark.gmd.de (G. Lonsdale).

PII: S0307-904X(00)00043-3

Major advances have been made in recent years in the two areas which formed the starting point for the activities in the DRAMA project [2]: the development of parallel mesh-partitioning algorithms suitable for dynamic re-partitioning (re-allocation of sub-meshes to processes at run-time); the migration and optimisation of industrial-strength simulation codes to HPC platforms using the message-passing paradigm. The DRAMA project brought together the developments in parallel partitioning and parallel FE applications to ensure that the potential of scalable computing can be achieved for fully-functional industrial simulation, which includes efficient adaptive meshing (and re-meshing) options. The parallel dynamic re-partitioning routines can also handle the full complexity and range of finite elements as used in industrial structural mechanics codes, as exemplified by the applications within the project.

The central product of the project was the DRAMA Library [3,4] comprising various tools for dynamic re-partitioning of unstructured finite element applications. The target applications are those using parallelisation based on a partitioning of the mesh into sub-domains (irrespective of the methodology chosen to implement the distribution of computations). The core library functions 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. The DRAMA cost model is able to take account of dynamically changing computational *and* communication requirements. Furthermore, it is formulated in such a way that all information can be provided by the application based on its actual local data and measured costs (via code instrumentation). The library provides the application program sufficient information to enable an efficient migration of the data between processes. Section 2 of this paper will give an introduction to the format of the library and indicate how it can be exploited within application codes.

Via the DRAMA Library, dynamic load-balancing can be achieved which enables scalable, efficient parallel FE applications, even with adaptive mesh refinement (coarsening) and re-meshing. As a by-product of this approach, a fully parallel mesh *generator* has been developed that exploits the parallel re-partitioning of adaptively generated meshes. The mesh re-allocation approach to dynamic load balancing has been demonstrated and validated by the leading industrial codes PAM-CRASH (for crashworthiness simulation, [5,6]), PAM-STAMP (for metal stamping/deep-drawing and related simulations [7,8]), FORGE3 (for forging with viscoplastic incompressible materials, [9,10]). Section 3 will present parallel performance behaviour for the above applications using the DRAMA library.

Despite this emphasis on the validation codes within the project, the library has been designed to be general purpose. With the final DRAMA library accessible as public domain software [2], it is hoped that a wide range of applications will be able to make use of the project results.

2. The DRAMA library

The DRAMA Library is designed to be called by parallel message-passing (MPI) mesh-based (and in particular finite element) applications. The 'expectation' of such applications is for the rapid provision of information about:

(a) a re-partitioning of the mesh which balances the costs occurring in the application;
(b) 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 library design and re-partitioning modules included have taken these requirements into account by the careful definition of the cost model and library interface.

2.1. The DRAMA cost model and library interface

Full details of the cost model and library interface can be found in [11,12]. The DRAMA Library is written in C with message-passing via MPI. The library may be called by applications written in both Fortran and C.

The interface between the application code and the library is designed around the DRAMA cost model (which results in an objective 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 allows the prediction of the effect on the performance of moving some parts of the mesh to other processes. Calculation and communication speeds of the processors are taken into account by a combination of hardware specific parameters and costs that are based on time measurements and enumeration (e.g. byte counts for data exchange) provided by application code instrumentation. Heterogeneous machine architectures can also be taken into account within the cost model by the provision of processor-specific parameters for computation and communication characteristics (though support within the partitioners is still under development, see for example [13]). 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. Hence, the DRAMA cost model includes both per element and per node computational costs and element–element, node–node, and element–node data dependencies. Indeed, within a finite element code, part of the computations may be performed element-wise, for example, a matrix-assembly phase, while other operations are node-based, such as the update of physical variables and nodal co-ordinates or the solution of systems of linear equations. Furthermore, the inter-sub-domain communication is frequently carried out using node lists.

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.

Different algorithmic parts in parallel application codes that are separated by explicit synchronisation points are defined as *phases* within the DRAMA cost model. The total cost is then given by the sum, over the phases, of the maximum cost, over all processes, per phase. The load-imbalance for each phase is the ratio of the maximum to average process costs for that phase.

The library contains various partitioning modules plus 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 data migration information (old $\leftrightarrow$ new mesh relationships). These new-to-old and old-to-new relations are necessary because the DRAMA library migrates only the mesh description. Subsequently, the application program has to migrate the data (temperature, stress, velocity, etc.) associated with the mesh.

In addition to the modules described above, the DRAMA library also contains some auxiliary functions. One such function is represented by routines to improve the numbering of elements and nodes, in order to reduce the computational time or the memory requirements within the application (e.g. by minimising the number of cache misses or the bandwidth of the

system matrix). In addition, several routines are provided which can be used to force sub-domain boundary re-location as a support tool for parallel re-meshing of the form described in Section 3.3.

2.2. Re-partitioning modules

Several types of mesh re-partitioners are available in the DRAMA library, based on mesh-migration, graph partitioning or co-ordinate partitioning. In addition to the references given below, detailed information can be found in the project deliverables [14,15].

The mesh-migration module migrates parts of the mesh to other sub-domains, using an iterative procedure in which *pairs of processes* perform load-balancing, using the DRAMA cost model as objective function. While theoretical convergence proofs show that in the worst case the number of iterations grows with the square of the number of processes, practical experience with finite element meshes shows that only a few iterations are needed to achieve nearly perfect load balance. For further information see [16,17].

The geometric partitioning module is based on recursive co-ordinate bisection (RCB). Although it is generally accepted that such a partitioner produces less than optimal partitions in comparison with more sophisticated techniques, the approach offers some interesting features: speed of execution, preservation of geometric proximity, moderate memory requirements. The RCB implementation in the DRAMA library is based on the version of Nakhimovski [18], which uses buckets (intervals on the co-ordinate axes) to reduce edge cut and also exploits the unbalanced recursive bisection of Jones and Plasman [19] to reduce data transfer.

The third mesh re-partitioning module is based on existing software packages for parallel graph re-partitioning: PARMETIS, developed by Karypis et al., University of Minneapolis [20,21] and PJOSTLE, developed by Walshaw, University of Greenwich [22]. PARMETIS is included in the DRAMA library, and the co-operation within the project with the University of Greenwich has led to a DRAMA interface to a modified version of PJOSTLE. Both packages contain recent developments enabling multi-constraint and multi-phase partitioning (see [23–25] and Section 3.2). Since the DRAMA library interface is completely mesh-based, a layer has been created around the graph re-partitioning modules. This layer within the library constructs a graph from the mesh-based input information and re-constructs from the output of the graph re-partitioners mesh-based information, to be used by the application program to perform the actual data migration.

Since the majority of the results presented in this paper are obtained with a graph-based re-partitioner, we now give further details of this module.

2.2.1. Graph-based re-partitioning methods

Both PARMETIS and PJOSTLE use multi-level techniques, in which a hierarchy of coarser graphs is constructed. At the coarsest level the graph is (re-)partitioned and during the uncoarsening process the partitioning is refined. The re-partitioning of the coarsest graph is based on a diffusion scheme. PJOSTLE uses the diffusion algorithm of Hu and Blake [26], which has a 'global view' of the optimisation problem, is synchronous and is best suited for applications whose load does not change excessively at each iteration. PARMETIS uses a wavefront diffusion variant [21,27], which is better suited to handle more unbalanced situations. PARMETIS also allows the use of a diffusion scheme that only has a 'local view', which works asynchronously and is highly parallel in nature. However, the local view might lead to a decrease in quality of the partitioning and thus to a higher run time of the application.

During the coarsening process, graph vertices are gathered together locally in each sub-domain to form 'coarse graph vertices'. PJOSTLE also allows global gathering, i.e. vertices can be gathered over the sub-domain interfaces.

Besides the diffusion-based techniques, PARMETIS provides two routines that compute completely new partitions (c.f. static partitioning) and re-map these intelligently onto the processes in order to minimise migration [27]. This approach is well suited for highly unbalanced cases.

2.2.2. Mesh to graph conversion

Within the DRAMA library, the mesh is converted in parallel into a distributed weighted graph. The vertex weights correspond to calculation costs and edge weights correspond to potential communication costs. These weights are taken from the data provided through the library interface and reflect the types of contributing nodes and elements occurring in the mesh.

Different graph representations can be selected, see Fig. 1. The selection should be based on the application requirements (speed vs. partition quality, available memory), the cost function model and the accuracy to which the cost model should be approximated. While the graph should represent a sufficiently good approximation of the cost model, experience shows that using a more accurate representation of the cost function does not always lead to a better partition, since the optimisation problem becomes more difficult to solve.

2.2.2.1. Dual graph or element graph. The dual of the mesh is a widely used graph representation, where the weighted graph vertices correspond to mesh elements and the associated calculation costs. The edges represent the potential communication between neighbouring elements. Vertices are connected by an edge if the corresponding elements share an edge (in 2D) or a face (in 3D).

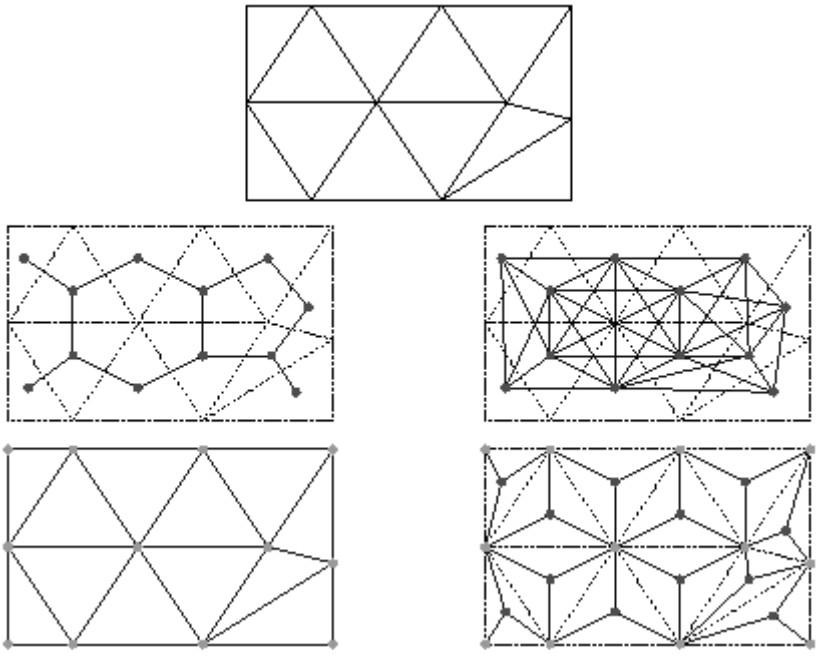

Fig. 1. A simple mesh and various graph representations. Middle row: dual graph; extended dual graph. Bottom row: nodal graph; combined graph.

The default option is that only the element-based calculation costs are taken into account, neglecting the node-based calculation costs. However, DRAMA also allows the distribution of the node-based calculation costs to the graph vertices (corresponding to elements).

2.2.2.2. Extended dual graph. For meshes with elements of different dimensions, the potential communication cannot be well represented by the dual graph. In the extended dual graph, graph vertices are connected by an edge when the corresponding elements share one or more nodes. Hence, certain connections between sub-domains that are lost in the classical dual graph, including connections between elements of different dimensions, are maintained. However, the extended dual graph can become very complex, requiring a lot of memory, especially for 3D tetrahedral meshes.

2.2.2.3. Generalised dual graph. This graph lies between the classical dual graph and the extended dual graph. As with the extended dual graph, it is well suited for meshes with different element types. However, not all elements sharing a node are joined by an edge of the graph. An element is only connected to those neighbouring elements which share a (local) maximum number of nodes.

2.2.2.4. Nodal graph. Here graph vertices correspond to mesh nodes and vertices are connected if they share an element. The default option is that only the node-based calculation costs are taken into account, but DRAMA also allows the distribution of the element-based calculation costs over the graph vertices (corresponding to nodes).

2.2.2.5. Combined graph. In this graph, both elements and nodes are represented by vertices, allowing a good representation of all calculation costs. Since finite element applications often use node lists for inter-process communication, graph edges represent communication requirements between elements and nodes. Hence, this graph is a simplification of the general combined graph that would have all kinds of element–element, node–node, and element–node connections.
Note that in order to construct the graph from the mesh, the 'inverse connectivity information' must be computed. While the 'connectivity information' gives for each element the list of nodes belonging to that element, the 'inverse connectivity information' gives for each node the list of elements to which it belongs. For each sub-domain, the inverse connectivity is determined for both local and non-local nodes, which requires communication between processes storing neighbouring sub-domains.

2.2.3. Graph to mesh conversion

The output of graph partitioners is just an array indicating for each graph vertex to which process (sub-domain) it should be migrated. In the case of the dual graph, this array only gives a new distribution for the mesh elements, while a new distribution for the nodes still has to be determined. A similar situation holds in the case that the nodal graph has been used. Within the DRAMA library, the new distribution of all mesh data is determined, along with a new numbering, and the old-to-new and new-to-old relations between the old and new partitioning of the mesh. Note that the computation of both the new numbering and the old-to-new and new-to-old relations requires communication.

3. Performance of finite element applications with DRAMA

The applications used to validate the DRAMA approach and library within the project are representative of a range of finite element software having a natural requirement for a

re-partitioning library as parallelisation aid. The simulation codes use time-marching as basic solution procedure and both explicit (PAM-CRASH/-STAMP) and implicit (FORGE3) time-integration methods are included. Causes of load-imbalance, and resulting degradation of scalability, are: (a) a dynamic behaviour of (element-based) computational cost and of the communication patterns; (b) meshes which are changing during the calculation – adaptive meshing or re-meshing, including re-shaping, refinement and de-refinement. 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.

3.1. Memory requirements of the library

Before describing execution times and partitioning quality results for benchmark test cases, we comment briefly on the memory requirements of the DRAMA library. A detailed analysis of the memory requirements of both the graph-based re-partitioning module and the mesh-migration module is given in [15]. The results can be summarised as follows:

(a) *Graph-based re-partitioning module*. An important drawback of the graph-based re-partitioning component of the DRAMA library is the high memory requirement. The inverse connectivity, needed to construct the graph representation of the mesh, requires a large amount of extra memory, but the largest amount of extra memory is required when calling PARMETIS or PJOSTLE. Both tools require an additional memory that is proportional to the number of graph vertices and edges. Precise estimates of the memory requirements are difficult to compute because of the multi-level approach employed. A great deal depends on the mesh and its distribution of elements and nodes. The graph structure will influence the coarsening process and thus also the amount of memory used. For single phase partitioning, using reasonable assumptions for the reduction of the number of vertices during the graph coarsening, the following estimate can be given for the maximum memory usage per process (sub-domain), mem_max, **in bytes**:

$$\text{mem\_max} = 8\ \text{nelem} + 8\ \text{nconn} + 12\ \text{nnode} + 80\ \text{nvtx} + 56\ \text{nedge},$$

where nelem is the number of local elements, nconn the length of the mesh connectivity list, nnode the number of local nodes, nvtx and nedge are the numbers of local graph vertices and graph edges (it is assumed that co-ordinates are stored as 32-bit floating point numbers).

For the dual graph representation of a tetrahedral mesh, we have $\text{nedge} < \text{nconn} = 4\ \text{nelem}$. Then the estimate simplifies to

$$\text{mem\_max} = 344\ \text{nelem} + 12\ \text{nnode}.$$

Using the nodal graph for a typical FORGE3 mesh, the number of edges is about twelve times the number of nodes. The number of elements is usually four to five times the number of nodes. This leads to the estimate

$$\text{mem\_max} = 40\ \text{nelem} + 764\ \text{nnode}.$$

The combined graph would lead to the estimate

$$\text{mem\_max} = 344\ \text{nelem} + 316\ \text{nnode}$$

but experiments show that there is no need to use the combined graph in case of a 'homogeneous' tetrahedral mesh.

Experiments with FORGE3 tetrahedral meshes show that the derived dual and nodal graphs coarsen very well (coarsening fraction ≈ 0.55). Taking this into account, we obtain the following sharp estimates:

$$\text{mem\_max} = 318\ \text{nelem} + 12\ \text{nnode}$$

for the dual graph, and

$$\text{mem\_max} = 40\ \text{nelem} + 668\ \text{nnode}$$

for the nodal graph. A comparison between the latter estimates and the actual memory requirements indicates that the formulae above lead to a slight underestimation of the memory requirements (less than 10%).

(b) *Mesh-migration module*. The mesh-migration module (see Section 2.2) also needs the inverse connectivity information. A detailed analysis shows that the memory requirements for tetrahedral meshes, as used by FORGE3, are

$$\text{mem\_max} = 92\ \text{nelem} + 44\ \text{nnode}$$

which is considerably less than those for the graph-based re-partitioning module, but still high compared with the mesh itself.

In order to reduce the memory required by the DRAMA library, the library allows memory blocks (work space) to be passed to it from the application or the dynamic allocation of any additional memory (see [12]). Although the memory requirements for the DRAMA library are very high, one has to take into account that (subsequently, when the additional memory used by the DRAMA library is de-allocated or returned) the application program may also need a large amount of additional memory, in order to be able to migrate parts of the mesh data. Clearly, if the application data use (nearly) all memory available, dynamic load balancing via mesh re-partitioning is not possible.

3.2. PAM-CRASH & PAM-STAMP

The PAM-CRASH and PAM-STAMP codes are two of the ESI Group products that are built around the PAM-SOLID core solver libraries. This means that they share the same basic algorithms and computational kernels, but include different algorithms and routines for application-specific functions. 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 communication costs. Contact–impact algorithms are also crucial to the simulations performed by PAM-STAMP, but a 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.

Leaving details of the algorithms and their parallelisation to [5,6] 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.

The PAM-SOLID parallelisation includes several synchronisation points, which require multi-partitioning. A global synchronisation separates the stress–strain and contact–impact calculations and further communications within each time-marching cycle result in a multi-phase application.

For simplicity, and taking the magnitudes of computational contributions into account, the codes are currently handled as a two-phase application: contact–impact and stress–strain plus time-integration (and all remaining calculations).

The DRAMA library handles multi-phase applications by producing a partition which correctly accounts for costs arising in the separate phases. An alternative approach would be to independently partition the mesh (or sub-meshes) per phase and perform data migration between phases. This approach has been successfully exploited for solid dynamics simulations on large-scale parallel machines for the Pronto code [28].

3.2.1. PAM-CRASH with multi-partitioning

In this section we present results for multi-partitioning [23–25] of meshes produced by PAM-CRASH simulations with industrial benchmark models (courtesy of Audi AG and BMW AG). The objective is to demonstrate that the DRAMA library can be effectively employed for multi-phase applications. Dynamic load-balancing with DRAMA for PAM-CRASH is functional, and the overheads of re-partitioning have been shown not to be significant: on 8 compute nodes of the NEC Cenju-4 the total elapsed time of 17040 s included only 77 s for the DRAMA library calls and subsequent data migration. However, the possible total gains that can be achieved with the existing contact–impact algorithmic implementation in PAM-CRASH are limited:

(a) for small numbers of processes, the contact–impact phase of the calculations has been optimised in recent years to the point where the total run-time is very much dominated by the already well-balanced stress–strain calculations;

(b) the existence of a non-scaling computational section and (pseudo-) all-to-all communication, which cannot be handled by the DRAMA library's partitioners, produces a dominating cost for larger numbers of processes, particularly when the re-partitioning attempts to balance the contact phase across many processes.

For this reason, full code performance results are not presented.

The various partitioning options used are given in Table 1, where all acronyms for the particular partitioning options are explained in [20–24]. Partitioner 6 is a single-phase partitioner added for comparison.

Tables 2 and 3, for the Audi and BMW models, respectively, give the minimum, maximum and mean total computational weights (as provided to the DRAMA Library from the application) for a 16-domain partition. The cost categories 'FE' and 'CO' are the costs for the stress–strain (+ time-integration) phase and the contact–impact phase. In addition to the total edge cut of the partition, load imbalance factors (as given by the DRAMA Cost Model based on the application parameters) per phase and in total are given. The load imbalance is defined as the ratio of the maximum to average process costs, a perfect load balance giving a value of 1.0. Near perfect load balance is achieved with the multi-partitioning approaches load imbalance of around 1% or less is

Table 1
Repartitioning methods

Method	Partitioner
1	METIS_mCPartGraphkway, sequential
2	MJostle, sequential
3	MOC_PARMETIS_Partkway
4	MOC_PARMETIS_SR
5	Mjostle, parallel
6	PARMETIS_RepartGDiffusion,single phase

Table 2
Audi benchmark, computational weights per phase, cut edges and load imbalance factors

Method	FE [min max]	CO [min max]	Edge cut	Imbalance [FE CO] total
1	[10428 10437]	[126 134]	2193	[1.000 1.011] 1.001
2	[9354 10536]	[132 134]	3116	[1.001 1.011] 1.010
3	[10095 10548]	[128 136]	2308	[1.011 1.026] 1.011
4	[10158 10632]	[126 136]	2682	[1.019 1.026] 1.019
5	[10380 10518]	[130 136]	2641	[1.008 1.026] 1.008
6	[9075 11070]	[0 1128]	4154	[1.061 8.513] 1.155
	Mean: 10432.1	Mean: 132.5		

Table 3
BMW benchmark, computational weights per phase, cut edges and load imbalance factors

Method	FE [min max]	CO [min max]	Edge cut	Imbalance [FE CO] total
1	[19647 19653]	[232 246]	4018	[1.000 1.006] 1.000
2	[17238 19992]	[238 246]	8472	[1.017 1.006] 1.017
3	[19281 19857]	[220 248]	4096	[1.010 1.014] 1.011
4	[19245 19848]	[234 250]	4017	[1.010 1.022] 1.010
5	[19422 19953]	[238 252]	5029	[1.015 1.030] 1.016
6	[18525 20706]	[0 982]	6863	[1.054 4.014] 1.090
	Mean: 19650	Mean: 244.6		

achieved for both modules, in comparison with single phase results of 15% and 9% (for the Audi and BMW models, respectively).

3.2.2. PAM-STAMP with parallel adaptive mesh refinement

Highly effective deployment of the DRAMA library with PAM-SOLID-based codes should be expected with the PAM-STAMP code, since this code relies heavily on the use of adaptive meshing, which will generate load imbalance within the stress–strain calculations. Furthermore, the typical contact–impact algorithms used within PAM-STAMP applications display better scaling properties than those used within PAM-CRASH and the calculations are fairly well distributed across the elements. The results included below will show that the dynamic costs of the adaptive finite elements are effectively re-partitioned using DRAMA.

Unfortunately, the full impact of dynamic load-balancing for PAM-STAMP cannot be demonstrated with the parallel prototype integrated with the DRAMA library during the lifetime of the project. The prototype code includes (physically unnecessary) nodal calculations for the time integration of nodes corresponding to the elements modelling the tools: the 'null shells' which are treated as rigid bodies and used to model the motion of the punch, die and blankholder. These calculations are removed in more recent versions of the standard PAM-STAMP code. The impact on the prototype PAM-STAMP with DRAMA was twofold: the normally dominant finite element stress–strain calculations (upon which the design for the use of the DRAMA library in the PAM-codes is based) become subsidiary to the null shell nodal costs; nodal costs remain imbalanced since restrictions in the data migration and re-partitioning approach within the PAM-codes mean that the DRAMA nodal partitions cannot be used. In addition, the parallel (message-passing) version with adaptive meshing is a recently developed feature and currently subject to robustness problems when more than one level of refinement is introduced (thus additionally limiting the impact of re-partitioning).

Although other options for handling the prototype code are available within the DRAMA library (an example being node-by-node cost modelling), the fact that the future versions of parallel PAM-STAMP will resolve the above issues meant that further investigation with the prototype code was not deemed appropriate, nor was it feasible within the project time-frame.

The results presented in this section are taken from full PAM-STAMP simulations with an industrial benchmark model: a general motors fender with an initial mesh having 5180 refinable elements on the metal blank, which can result in around 30,000 elements at the end of the simulation, depending on the type of refinement parameters set. The cost analysis is only for the finite element phase, which includes the use of adaptive mesh refinement. The cost modelling strategy adopted for the locally refined meshes was to sum all costs for refined elements to produce a modified cost for the original mesh element – this guarantees that parent–child element sets are distributed to a process (which simplifies handling within the application code).

The performance presented in Fig. 2 is for the above model using 200 mesh refinement steps (based on a 1° angle criterium) and a maximum of one refinement level (the final number of blank

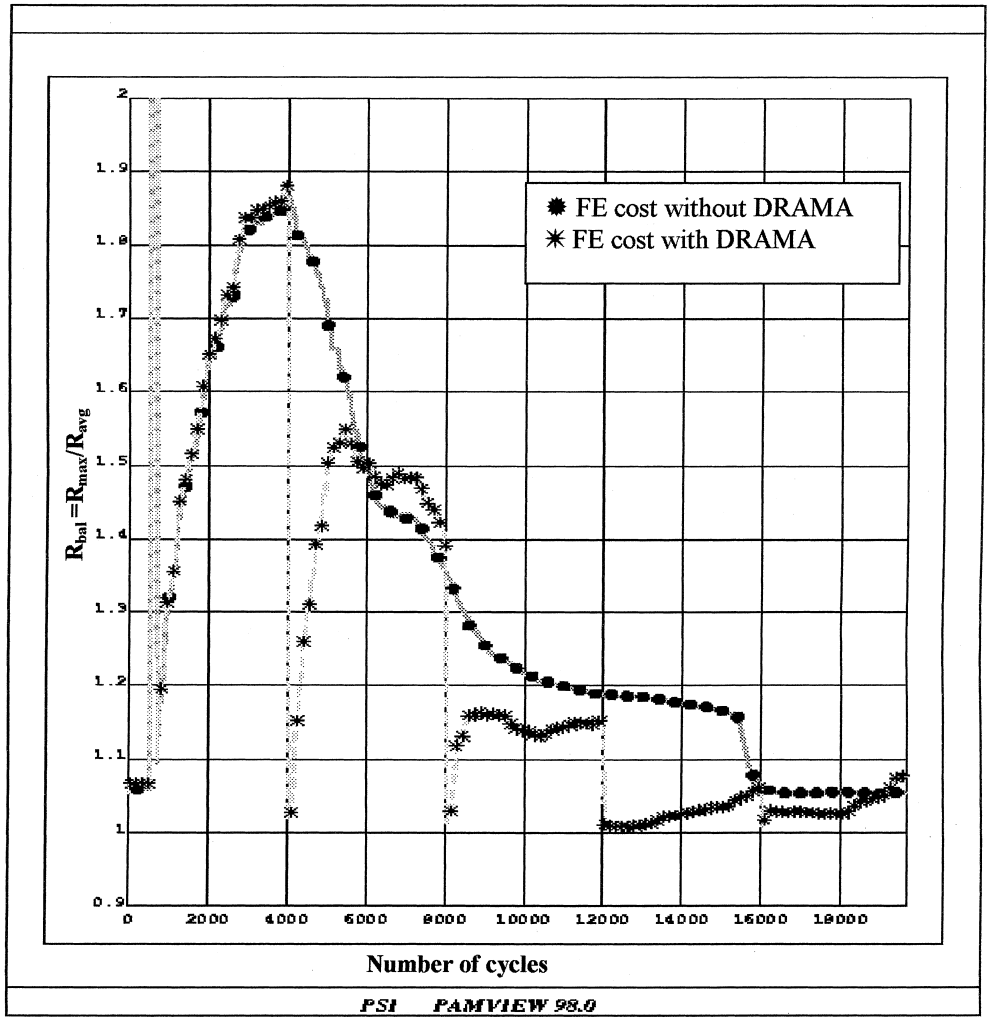

Fig. 2. Comparison of F.E. imbalance development (R_{bal} per cylce), for the Fender Benchmark with/without DRAMA repartitioning on 8 compute nodes, NEC Cenju-4.

shell elements is 10,146). DRAMA re-partitioning was performed at intervals of 4000 computational cycles.

Fig. 2 demonstrates the evolution of imbalance using the ratio of slowest (R_{max}) to average (R_{avg}) times per process spent in the finite element routines, R_{bal}:

$$R_{bal} = R_{max}/R_{avg}.$$

The comparison is made between a standard run without DRAMA re-partitioning and the run with DRAMA both using eight compute nodes on the NEC Cenju-4. Two issues are apparent: the mesh refinement generates high imbalance in the early phases of the simulation that decreases as the simulation proceeds; with values of R_{bal} close to 1.03 at the re-partitioning points (the multiples of 4000 cycles), DRAMA is very effectively balancing the FE costs.

The reason for decreasing imbalance over the length of the simulation is that the final mesh includes refinement which is fairly equally distributed over the model. What is also clear is that one would in practice use more frequent DRAMA re-partitioning. The DRAMA library actually allows for monitoring of load imbalance, by using a computationally inexpensive cost/imbalance prediction function, so that in practice one could perform re-partitioning at time points determined by the application at run time.

3.3. FORGE-3 and parallel meshing/re-meshing

FORGE3 from Transvalor is an implicit finite element code designed for the simulation of 3D 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. The parallelisation of the full code, including adaptive re-meshing, uses a mesh partitioning approach [9,10]. For forging simulations, the capability for re-meshing is a unique, competitive advantage of the FORGE3 code. The 3D parallel re-meshing procedure requires a re-partitioning stage, not only to avoid load imbalance but also to deal with the interface re-meshing.

A scalable parallel mesher/re-mesher has been developed by combining the DRAMA mesh-migration module and the serial mesher, MTC, from CEMEF using the following strategy:

- Begin with a coarse mesh distributed (partitioned into sub-domains) over the processes.
- Re-mesh each sub-domain independently at fixed interfaces.
- Mark the nodes at the sub-domain boundaries which are to become internal nodes after re-partitioning (to improve the mesh quality in terms of both element shape and size).
- Do a re-partitioning step, whereby forcing the marked nodes to become internal nodes.
- Mark the nodes to be re-meshed (the marked ones which were previously at the interfaces).
- Re-mesh in the neighbourhood of the marked nodes.
- Re-partition taking only computational costs into account and iterate if needed.

With this approach the number of operations is almost the same as in the sequential case. The above strategy allows for the use of the existing sequential mesher on each process, while assuring that partition-induced mesh features are removed.

The new parallel re-meshing strategy used in FORGE3 is similar to the parallel meshing algorithm given above and is realised by calling DRAMA twice per computational cycle: first before the re-meshing, specifying the nodes we want to be on the same sub-domain, then a second time after the re-meshing to achieve a good load balance. This strategy was not foreseen at the beginning of the DRAMA project, but has been made possible by the speed of the re-partitioning and data migration, which takes only a few seconds (as can be seen in Section 3.3.1) while one time step, or a re-meshing takes several minutes.

3.3.1. Parallel mesh generation/re-meshing

Results are presented here for homogeneous refinement over all sub-domains. Commencing with a mesh distributed over processes and given a new target mesh size, the re-mesher and the re-partitioner are run successively until the size is satisfied and the partition is well balanced. The parallel homogeneous refinement is defined by iteration between the two following steps:

- re-meshing with blocked interfaces;
- re-partitioning with migration of interfaces.

Performance figures are given in Table 4 for the parallel re-mesher with a 3D test case (executed on the NEC Cenju-4). The test consists of dividing the mesh size by a factor 2. The initial mesh has 4092 nodes and 20,418 elements and the final one around 22,000 nodes and 116,000 elements.

3.3.2. Parallel re-meshing with DRAMA for FORGE3

In addition to performance testing, the DRAMA-interfaced version of FORGE3 has been validated for robustness and quality of the results produced using the standard FORGE3 validation test suite. With the FORGE3 strategy of local re-meshing combined with interface movement, the most important aspect for validation of DRAMA use is that the final complete re-meshing is achieved. Comparisons of shape, temperature field, strain and forging loads for the pipe connector test case, which is the most difficult case from the re-meshing viewpoint, showed that the simulation accuracy was maintained with the new version exploiting DRAMA.

Fig. 3 shows speed-ups on the pipe connector problem with FORGE3 using DRAMA, employing both the mesh migration module and graph partitioning (with ParMeTis), on a PC-cluster at NEC, comprising dual processor Pentium-pro PCs with a Myrinet switch. Table 5 shows the relative performance, for several examples from the validation suite, with respect to the parallel FORGE3 code without DRAMA; as can be seen, significant computational savings are made. The results in Table 5 were all produced on a 4-CPU, DEC SMP system, the meshes are all relatively small (the Conrod case involves approximately 5000 tetrahedral elements, the other two cases have initial tetrahedral meshes with around 5000 elements, which result in approximately 15,000 elements at the end of the simulation). For these meshes, the need for efficient handling of re-partitioning is more pronounced than with larger meshes.

4. Concluding remarks

The results presented in Section 3 demonstrate that the DRAMA library can be used to support dynamic load-balancing for complex mesh-based applications. The interface has been designed to be appropriate for such applications and takes into account the possible need for multi-phase partitioning.

With the 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-

Table 4
Performance for the 3D test case on the NEC Cenju-4

# proc	1	2	4	8	16
Re-meshing time (s)	1253	701	446	259	167
Re-partitioning time (s)	0	7	21	23	25
Total time (s)	1253	708	467	262	192
Speed-up	1	1.79	2.68	4.78	6.52
Efficiency	1.00	0.88	0.67	0.60	0.40

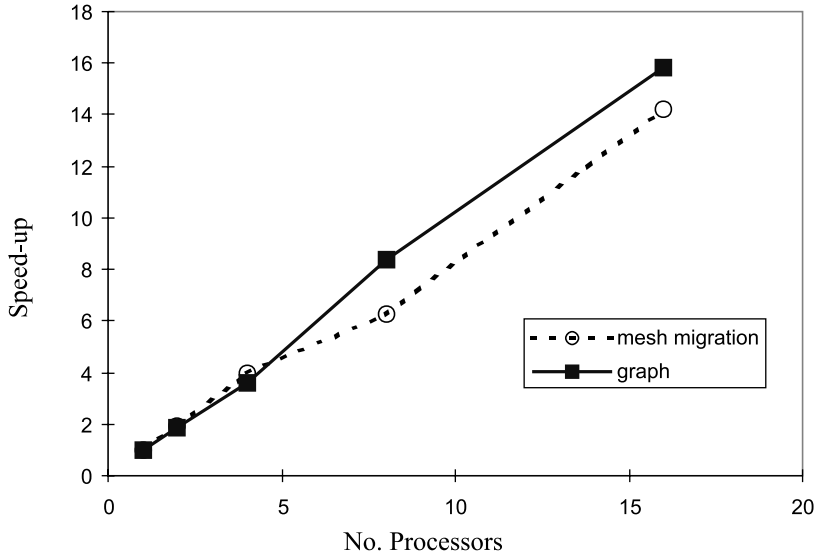

Fig. 3. Speed-up for the pipe connector on the NEC PC-cluster. Results obtained with both the mesh migration and the graph-based re-partitioning modules are presented.

Table 5
Comparisons of total elapsed times (in seconds) for three test cases with the parallel FORGE3 code: original and new DRAMA versions

Case	Conrod	Pipe connector	Tap fitting
Original version	5820	43500	54500
Drama version	4620	34040	36066

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 DRAMA project focused on finite element codes, the mesh interface definition is also directly applicable to finite volume codes.

Finally, it should be pointed out that the DRAMA library provides only a *starting point* for the creation of efficient parallel (MPI-based) codes. There is still the need for the applications codes to implement the data migration and to create data structures within their code that allow parallel dynamic/adaptive mesh partitioning. Future extensions of the DRAMA library may include additional support for the data migration within the application code. It is hoped that feedback from DRAMA users will help to steer these and other future developments.

Acknowledgements

The DRAMA Consortium would like to thank the Department of Computer Science, University of Minnesota for their ongoing co-operation in providing various versions of the MeTis and ParMeTis software and all the participants in the three DRAMA Steering Workshops

who assisted the progress of the project through many fruitful discussions. The support of the European Commission through the ESPRIT IV (Long Term Research) Programme is gratefully acknowledged.

References

[1] B. Hendrickson, K. Devine, Dynamic load balancing in computational mechanics, Comput. Meth. Appl. Mech. Engrg., to appear.
[2] The DRAMA Web-site:http://www.ccrl-nece.technopark.gmd.de/DRAMA/.
[3] G. Lonsdale, et al., DRAMA: Dynamic re-allocation of meshes for parallel finite element applications, in: Developments in Computational Mechanics with High Performance Computing, Proceedings of the Euro-CM-Par 99 Conference March 1999, Weimar, Civil-Comp Press, 1999.
[4] B. Maerten, et al., DRAMA: a library for parallel dynamic load balancing of finite element applications, in: P. Amestoy, et al. (Eds.), Euro-Par' 99 Proceedings, Lecture Notes in Computer Science, vol. 1685, 1999, pp. 313–316.
[5] J. Clinckemaillie, et al., Performance issues of the parallel PAM-CRASH code, Int. J. Supercomput. Appl. High Performance Comput. 11 (1) (1997) 3–11.
[6] G. Lonsdale, et al., Programming crashworthiness simulation for parallel platforms, Math. Comput. Model. 31 (2000) 61–76.
[7] E. Haug, et al., Transport vehicle crash safety and manufacturing simulation in the perspective of high performance computing and networking, Future Generation Comput. Systems 10 (1994) 173–181.
[8] E. Haug, et. al., Industrial sheet metal forming simulation using explicit finite element methods, FE-Simulation of 3D sheet metal forming process in automotive industry, VDI Berichte 894, Zürich, Switzerland, 14–16 May 1991.
[9] T. Coupez, S. Marie, From a direct solver to a parallel iterative solver in 3D forming simulation, Int. J. Supercomput. Appl. High Performance Comput. 11 (4) (1997) 205–211.
[10] T. Coupez, S. Marie, R. Ducloux, Parallel 3D simulation of forming processes including parallel remeshing and reloading, in: J.-A. Dé sidéri, et al. (Eds.), Proceedings of Second ECCOMAS Conference on Numerical Methods in Engineering '96, Wiley, 1996, pp. 738–743.
[11] DRAMA Project Deliverable, D1.1b, Final DRAMA cost model, in [2], 1999.
[12] DRAMA Project Deliverable, D1.2c, Updated library interface definition, in [2], 1999.
[13] C. Walshaw, M. Cross, Multilevel mesh partitioning for heterogeneous communication networks, Future Generation Comput. Systems, to appear.
[14] DRAMA Project Deliverable, D1.3a, Report on re-partitioning algorithms and the DRAMA library, in [2], 1998.
[15] DRAMA Project Deliverable, D1.3d, Final report on re-partitioning algorithms, in [2], 1999.
[16] T. Coupez, Parallel adaptive remeshing in 3D moving mesh finite element, in: B.K. Soni, et al. (Eds.), Numerical Grid Generation in Computational Field Simulation, vol. 1, Mississippi University, 1996, pp. 783–792.
[17] T. Coupez, H. Digonnet, R. Ducloux, Parallel meshing and remeshing, Appl. Math. Modelling, this volume.
[18] I. Nakhimovski, Bucked-based modification of the parallel recursive co-ordinate bisection algorithm, Linkoping Electronic Articles in Computer and Information Science, ISSN 1401-9841, vol. 2(015), available via http://www.ep.liu.se/ea/cis/1997/015/, 1997.
[19] M.T. Jones, P.E. Plassman, Computational results for parallel unstructured mesh computations, Technical Report UT-CS-94-248, Computer Science Department, University of Tennessee, 1994.
[20] G. Karypis, K. Schloegel, V. Kumar, PARMETIS parallel graph partitioning and sparse matrix ordering library, Version 2.0, Technical Report, Department of Computer Science, University of Minnesota, 1998.
[21] K. Schloegel, G. Karypis, V. Kumar, Multilevel diffusion schemes for repartitioning of adaptive meshes, J. Parallel Distrib. Comput. 47 (1997) 109–124.
[22] C. Walshaw, M. Cross, M. Everett, Parallel dynamic graph partitioning for adaptive unstructured meshes, J. Parallel Distrib. Comput. 47 (1997) 102–108.
[23] G. Karypis, V. Kumar, Multilevel algorithms for multi-constraint graph partitioning, Technical Report TR-98-019, Department of Computer Science, University of Minnesota, 1998.
[24] C. Walshaw, M. Cross, K. McManus, Multiphase mesh partitioning, Appl. Math. Modelling, this volume.
[25] A. Basermann, et al., Dynamic multi-partitioning for parallel finite element applications, in: E.H. Hollander et al. (Eds.), Parallel Computing: Fundamentals and Applications, Proceedings of the ParCo99 Conference, Delft, September 1999, Imperial College Press, 2000, pp. 259–266.

[26] Y.F. Hu, R.J. Blake, An optimal dynamic load balancing algorithm, Technical Report DL-P-95-011, Daresbury Laboratory, Warrington, UK, 1995.
[27] K. Schloegel, G. Karypis, V. Kumar, Dynamic repartitioning of adaptively refined meshes, in: Proceedings of the Supercomputing '98. http://www.supercomp.org/sc98/, 1998.
[28] S.A. Attaway, et al., Transient solid dynamics simulations on the Sandia/Intel Teraflop computer, in: Proceedings of the ACM/IEEE SC97 Conference (available on CD-ROM from ACM Member Services, email: orders@acm.org), Technical Paper, 1997.