ELSEVIER

Parallel Computing 27 (2001) 869–881

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Dynamic multi-partitioning for parallel finite element applications

A. Basermann [a,*], J. Fingberg [a], G. Lonsdale [a], B. Maerten [b], C. Walshaw [c]

[a] *C & C Research Laboratories, NEC Europe Ltd., Rathausallee 10, D-53757 St. Augustin, Germany*
[b] *K.U. Leuven, Department of Computer Science, Celestijnenlaan 200A, B-3001 Heverlee-Leuven, Belgium*
[c] *Centre for Numerical Modelling and Process Analysis, University of Greenwich, Park Row, Greenwich, London SE10 9LS, UK*

Received 30 June 2000; received in revised form 29 September 2000; accepted 12 October 2000

Abstract

The central product of the DRAMA (Dynamic Re-Allocation of Meshes for parallel Finite Element Applications) project is a library comprising a variety of tools for dynamic re-partitioning of unstructured Finite Element (FE) applications. The input to the DRAMA library is the computational mesh, and corresponding costs, partitioned into sub-domains. The core library functions then perform a parallel computation of a mesh re-allocation that will re-balance the costs based on the DRAMA cost model. We discuss the basic features of this cost model, which allows a general approach to load identification, modelling and imbalance minimisation. Results from crash simulations are presented which show the necessity for multi-phase/multi-constraint partitioning components.

Keywords: Dynamic load balancing; Cost model; Multi-phase/multi-constraint partitioning; Crash simulation

1. Introduction

The DRAMA (Dynamic Re-Allocation of Meshes for parallel Finite Element Applications) project [1] was initiated to support the take-up of large-scale parallel

* Corresponding author.
E-mail address: basermann@ccrl-nece.de (A. Basermann).

0167-8191/01/$ - see front matter
PII: S0167-8191(01)00072-2

simulation in industry by dealing with the main problem which restricted the use of message passing simulation codes – the inability to perform dynamic load balancing. The central product of the project is a library comprising a variety of tools for dynamic re-partitioning of unstructured Finite Element (FE) applications. The starting point for the DRAMA library is a (discretisation) mesh distribution into sub-domains that results in imbalanced costs of the application code. The core library functions then perform a parallel computation of a mesh re-allocation that will re-balance the costs based on the DRAMA cost model.

The main goal of the paper is to show the necessity for multi-phase/multi-constraint re-partitioning components [9,10,16] for cost re-balancing. These new partitioning methods allow load balancing for several computational phases – e.g., stress–strain analysis and contact treatment in the industrial crash analysis code PAM-CRASH^{TM} [6] – that are separated by synchronisation points.

In Section 2, we present the basic features of this cost model which allows a general approach to load identification, modelling and imbalance minimisation. Section 3 briefly describes the library interface, Section 4 treats the principles of current mesh partitioning methods. Approaches to dynamic load balancing with the DRAMA library, in particular for several computational phases, are discussed in Section 5. In Section 6, different re-partitioning methods are evaluated by means of test cases from PAM-CRASH^{TM}, whereas Section 7 is devoted to concluding remarks.

2. DRAMA cost model

The DRAMA cost model [3,12,13] explicitly considers calculation costs w_i per sub-domain i and communication costs $c_{i,j}$ between sub-domains i and j of the parallel application code. For the load-balancing re-partitioning algorithms, it results in an objective cost function F. The 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.

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. The DRAMA mesh consists of nodal coordinates and of a list of nodes per element which is a native data structure (element connectivity) in most finite element applications.

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^{TM} code [6], 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 allows the construction of *virtual elements* [4,12,13] which represent the occurring costs of such dependencies (see also

Section 5). A virtual element is included in the DRAMA mesh in the same way as a real element: as an additional connectivity list of its constituent nodes.

Specific types u identify calculation cost parameters per element or per node that refer to different kinds of elements, different material properties, or generally different algorithmic parts of the application code requiring different kinds of operations. Communication cost parameters per element–element, node–node, and element–node connection depend on the amount of data that potentially have to be transferred for a link between two objects of types u_1 and u_2.

Different algorithmic parts in parallel application codes that are separated by explicit synchronisation points are defined as *phases* within the DRAMA cost model. DRAMA evaluates the costs per phase *iphase*. The PAM-CRASH^{TM} code, e.g., can be considered to consist of essentially two sections; stress–strain computations including time integration (FE phase) and contact treatment (contact phase) with a global synchronisation in between and also at the end of each computing cycle.

The determination of cost parameters requires application code instrumentation. Number of operations per element/node of type u, $nop_i(u)$ can be specified by counting operations *or* by time measurements. The sum over all phases of total calculation times per phase and counting total number of computational operations allow the determination of calculation speeds s_i^{calc}. For communication, the number of bytes $noc(u_1, u_2)$ that has potentially to be transferred for a link between two objects of types u_1 and u_2 and communication speeds $s_{i,j}^{\text{comm}}$ have to be specified (latency is not considered). $s_{i,j}^{\text{comm}}$ essentially depends on the specific communication protocol. A suited communication model considering the message length has to be chosen. Moreover, a correspondence between types and phases must be given.

With these parameters, the DRAMA cost model can be written in the following form:

$$
\begin{aligned}
F &= \sum_{iphase} \max_i F_i^{iphase}, \\
F_i^{iphase} &= w_i^{iphase} + \sum_j c_{i,j}^{iphase}, \\
w_i^{iphase} &= \sum_u N_i(u) \frac{nop_i(u)}{s_i^{\text{calc}}} \\
c_{i,j}^{iphase} &= \sum_{u_1 u_2} N_{i,j}(u_1, u_2) \frac{noc(u_1, u_2)}{s_{i,j}^{\text{comm}}},
\end{aligned}
$$

$N_i(u)$ is the number of elements/nodes of type u and $nop_i(u)/s_i^{\text{calc}}$ is the computational cost of an object of this type. Since only the ratio is relevant both $nop_i(u)$ and s_i^{calc} may be specified as relative values if this makes instrumentation easier. $N_{i,j}(u_1, u_2)$ is the number of elements/nodes in a sub-domain boundary region, and $noc(u_1, u_2)/s_{i,j}^{\text{comm}}$ is the potential communication cost for a link between two objects of types u_1 and u_2.

3. DRAMA library interface

The interface between the application code and the library is designed around the DRAMA cost model and the instrumentation of the application code to specify current and predict future computational and communication costs [4]. Thus the application code has to provide DRAMA, per sub-domain, with the current mesh description, i.e., the element–node connectivity including the type information. The elements can be either real or virtual elements. The nodal coordinates are given in addition.

Moreover, the application code places the calculation and communication cost parameters per type at DRAMA's disposal as well as the correspondence between types and phases.

DRAMA returns the new partition in terms of a new numbering of local elements and nodes together with the relationships between old and new numbering systems and the coordinates of the new set of nodes local to a process. The relationships between old and new numbering systems support the application code in building send and receive lists.

4. Mesh partitioning

Distributing the mesh across a parallel computer so that the computational load is evenly balanced and the data locality maximised is known as mesh partitioning. It is well known that this problem is NP-complete, so in recent years much attention has been focused on developing heuristic methods, many of which are based on a graph corresponding to the communication requirements of the mesh, e.g. [7]. A particularly popular and successful class of algorithms which address this partitioning problem are known as multi-level algorithms. They usually combine a graph contraction algorithm which creates a series of progressively smaller and coarser graphs together with a local optimisation method which, starting with the coarsest graph, refines the partition at each graph level. Although the refinement usually only explores a very localised portion of the solution space, it appears that the multi-level enhancement adds a global quality to the final partition and that very high partition quality can be achieved, *in parallel*, independent of the initial partition [14].

Typically in these methods, the load-balance constraint – that the computational load is evenly balanced – is simply satisfied by ensuring that each processor has an approximately equal share of the mesh entities (e.g. the mesh elements, such as triangles or tetrahedra, or the mesh nodes). Even in the case where different mesh entities require different computational solution times (e.g. boundary nodes and internal nodes) the balancing problem can still be addressed by weighting the corresponding graph vertices and distributing the graph weight equally. However, for applications such as contact-impact, with *multiple loops* over *subsets* of the mesh entities interspersed by global communications, this simple cost model breaks down.

In these cases, two different partitioning approaches have been developed. The multi-constraint partitioning method of Karypis and Kumar [9] views the mesh as a

graph partitioning problem with multiple load-balancing constraints. The vertices of the graph are given a vector of weights (representing the contribution to each balancing constraint) and the refinement strategy takes each constraint into account when testing whether or not it is able to migrate vertices from one sub-domain to another.

In contrast, the multi-phase strategy of Walshaw et al. [16] consists of a graph manipulation wrapper around an almost unmodified 'black box' multi-level graph partitioner. The partitioner is then used to partition each phase individually although based on partitioning results from previous phases.

5. Dynamic load balancing with DRAMA

5.1. Basic features

The goal of any load balancing method is to improve the performance of applications which have computational requirements that vary with time. The DRAMA library is targetted primarily at mesh-based codes with one or more phases. It offers a multiplicity of algorithms allowing the different needs of a wide range of applications (Finite Element, Finite Volume, adaptive mesh refinement, contact detection) to be covered. The DRAMA library contains geometric, e.g., recursive coordinate bisection, topological (graph) and local improvement (direct mesh migration) methods [2]. It enables the use of leading graph partitioning algorithms through internal interfaces to ParMetis [8,10,11] and PJostle [15,16].

In comparison with the direct use of graph partitioners, DRAMA has the following advantages:

(1) DRAMA's interface is mesh-based. Since an element–node connectivity list is an essential component of mesh-based application codes DRAMA can be easily integrated. Mesh to abstract graph conversion is performed within DRAMA.
(2) Beside graph partitioners, DRAMA offers local improvement (migration) and geometric methods. Thus DRAMA is more general.
(3) DRAMA supports cost capturing and cost monitoring.
(4) DRAMA supports the application code in building new mailing lists after the re-partitioning.
(5) DRAMA allows different element/node type management.

Thus, DRAMA provides pre-defined solutions for most mesh-based application codes.

Many applications consist of several phases separated by explicit or implicit global synchronisation points. This is a challenging problem that requires each phase to be balanced independently. Fig. 1 (left) illustrates the situation for two processors and two phases. Both phases show distinct load imbalance. If both phases depend on each other as for the stress–strain and contact phases in PAM-CRASH^{TM} – the computations refer to the same mesh in both phases – balancing the aggregate costs of both phases is of no use, the two phases have to be balanced separately.

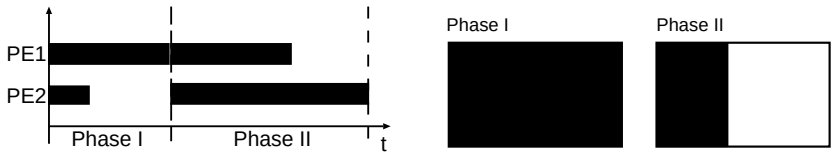

Fig. 1. Left: Load imbalance in two phases separated by two sychronisation points. Right: Operations in two phases on different parts of the same mesh.

There are two approaches to this problem, one is to work with a separate division of objects for each phase [5], the other is to balance each phase on a common partition. The first strategy is advantageous if all computational sections (phases) of the code work on the entire model. It requires fast communication between the different decompositions in each computing cycle. If the code works on different parts of the model (mesh) in different phases it can be favourable to maintain a single mesh decomposition and save communication time. The latter situation is displayed in Fig. 1 (right). The first phase refers to the whole mesh, the second only to the left part of the mesh. For example, for a frontal impact simulation with PAM-CRASH^{TM}, stress–strain is computed for the whole mesh whereas contact detection and correction is mainly performed in the front part of the car model.

5.2. Application to crash simulation

For crash simulation, we follow the single mesh decomposition strategy because it is much easier to implement in the existing application code. In the following, we show the results for the stress–strain phase and contact phase of PAM-CRASH^{TM} exploiting the new multi-phase/multi-constraint options of Jostle and Metis [9,10,16].

The graph-partitioning for crash simulation is built upon a combined graph of elements and nodes [2] because a part of the computation is node-based and a part element-based. Fig. 2 displays a combined graph for four-node shell elements. An element is linked with all its four nodes. The connection to other elements is via common nodes.

The basic objects during contact detection in PAM-CRASH^{TM} are pairs of nodes and segments of a surface, the segment being defined by four nodes. These objects

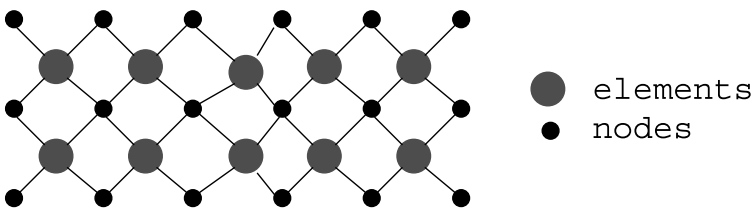

Fig. 2. Combined graph for four-node shell elements.

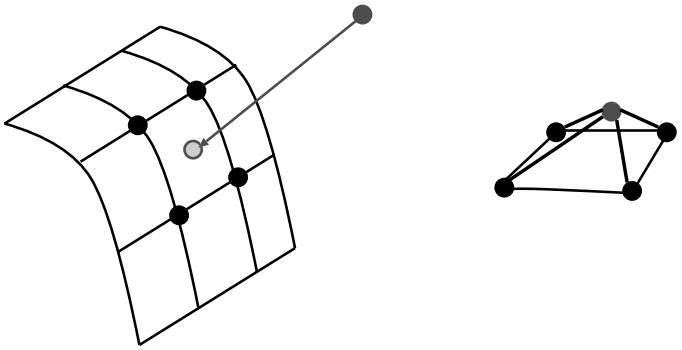

Fig. 3. Contact pair: virtual five-node element.

are passed to the DRAMA library as virtual five-node elements in the DRAMA mesh format [4]. Fig. 3 shows a shell element that is potentially penetrated by a node. The penetrating node and the four nodes of the shell element form a contact pair, a virtual five-node element.

6. Evaluation of different partitioning techniques

To demonstrate the viability of the DRAMA approach we present the results obtained with multi-constraint (mc) Metis and multi-phase Jostle (MJostle) for a box-beam model with PAM-CRASH^{TM}. We start from an initial partition with 44% imbalance ($\lambda = 1.44$). This imbalance is caused by contact calculations in the lowest domain. The cost weights for contact calculations are artificially increased for this small test case to illustrate the effect of multi-partitioning. For the industrial models AUDI and BMW below, realistic weights are applied. After 10,000 simulation cycles, we compute a re-partitioning with single-phase/uni-constraint ParMetis static (stat) as well as PJostle diffusion and compare the resulting distribution with the multi-phase/multi-constraint approaches.

The load imbalance factors are defined as

$$\lambda^1 = \frac{\max_{i=0..p-1}(w_i^1)}{\overline{w_i^1}} \quad \text{(for the stress–strain phase only)},$$

$$\lambda^2 = \frac{\max_{i=0..p-1}(w_i^2)}{\overline{w_i^2}} \quad \text{(for the contact phase only)},$$

$$\lambda^1_{\text{tot}} = \frac{\max_{i=0..p-1}\left(\sum_j^{nphases} w_i^j\right)}{\overline{\sum_j^{nphases} w_i^j}} \quad \text{(neglecting synchronisation)},$$

$$\lambda^2_{\text{tot}} = \frac{\sum_j^{nphases} \max_{i=0..p-1}(w_i^j)}{\overline{\sum_j^{nphases} w_i^j}} \quad \text{(considering synchronisation)},$$

Table 1
Distribution of shell elements and contact pairs (CPs) per sub-domain, different partitioning methods, box-beam model

PE	Initial		ParMetis stat		PJostle diff		Metis mc		MJostle	
	Shell	CPs	Shell	CPs	Shell	CPs	Shell	CPs	Shell	CPs
0	512	118	415	61	330	91	512	30	515	30
1	512	0	599	0	516	27	512	30	515	29
2	512	0	445	57	601	0	512	28	507	30
3	512	0	589	0	601	0	512	30	511	29
$\lambda_{1,2}$	1.00	4.00	1.17	2.07	1.17	3.09	1.00	1.02	1.01	1.02
λ^1_{tot}	1.442		1.026		1.004		1.002		1.007	
λ^2_{tot}	1.442		1.302		1.455		1.002		1.007	

where $\overline{x_i}$ denotes the mean value of all x_i, $i = 0..p-1$. Note that the real load imbalance is given by λ^2_{tot}.

From Table 1 we see that only multi-partitioning methods can improve the performance of the application; they are the only schemes giving a total imbalance λ^2_{tot} close to 1. Of course, the other schemes minimised the aggregate cost-function as can be seen from the values of λ^1_{tot} but neglecting the two synchronisation points only results in an increased idle time.

To evaluate the performance of different partitoning methods for more realistic cases we compare the results obtained with test meshes of an AUDI and a BMW car model which originate from PAM-CRASH™ simulations of a frontal impact with a rigid wall. The mesh data are stored after 10,000 cycles from a total of around 90,000 cycles. The two models consist of 4-node shell and 2-node beam elements. The initial total load imbalance of the AUDI model is 12.1% (load imbalance factors: 1.0002 for stress–strain, 10.642 for contact), the initial total load imbalance of the BMW model is 3.1% (load imbalance factors: 1.0002 for stress–strain, 3.524 for contact). For the graph representation of the mesh we use a combined graph [2] consisting of elements and nodes where the connections are only between elements and nodes, i.e., node–node and element–element connections are omitted. For the BMW model we also consider an element graph representation, the classical extended dual graph [2,4], where elements are connected if they share one or more nodes.

In the following, the methods listed in Table 2 [8–11,15,16] are tested for repartioning.

Method 6 is a single phase partitioner and is added for comparison reasons, all other methods are multi-phase/multi-constraint algorithms. Methods 1 and 2 are sequential multi-partitioners, all other methods are parallel. *MOC_PARMETIS_SR* (method 8) is a re-partitioner that should minimise load imbalance and the difference between the current and the new partitions. The latter was not investigated here but will be checked in detail in future tests.

Tables 3 and 5 show the distribution of shell elements, beam elements, nodes and contact pairs (CPs) per processor (PE) for the AUDI and the BMW models with 16

Table 2
Re-partitioning methods

Method	Graph representation	Partitioner
1	Combined graph	METIS_mCPartGraphkway, sequential
2	Combined graph	MJostle, sequential
3	Combined graph	MOC_PARMETIS_Partkway
4	Combined graph	MOC_PARMETIS_SR
5	Combined graph	MJostle, parallel
6	Combined graph	PARMETIS_RepartGDiffusion, single phase
7	Extended dual graph	MOC_PARMETIS_Partkway
8	Extended dual graph	MOC_PARMETIS_SR
9	Extended dual graph	MJostle, parallel

Table 3
Distribution of shell elements, beam elements, nodes and contact pairs (CPs) per sub-domain, AUDI, method 1

PE	Shell	Beam	CPs	Nodes	Phase FE	Phase CO
0	1735	6	67	1652	10428	134
1	1738	0	63	1715	10428	126
2	1737	3	67	1744	10431	134
3	1739	0	67	1661	10434	134
4	1739	0	64	1612	10434	128
5	1730	17	63	1622	10431	126
6	1735	6	67	1696	10428	134
7	1735	6	67	1732	10428	134
8	1725	29	67	1567	10437	134
9	1738	3	67	1719	10437	134
10	1739	0	67	1612	10434	134
11	1739	0	67	1702	10434	134
12	1730	17	66	1670	10431	132
13	1739	0	67	1586	10434	134
14	1739	0	67	1644	10434	134
15	1674	129	67	1710	10431	134
Total	27711	216	1060	26644	166914	2120
Mean					10432.125	132.5

sub-domains. Multi-partitioner 1 is applied. *phase FE* and *phase CO* are the costs for the total stress–strain phase and the contact phase. The cost for a beam element is about half the cost for a shell element whereas the cost for a contact pair is about one third of the cost for a shell element. These ratios are realistic for PAM-CRASH^{TM} and were confirmed by timings within the application code. Therefore, the number of shell elements is multiplied by 6, the number of beams by 3, and the number of contact pairs by 2 to obtain the total costs per processor for the stress–strain phase and the contact phase.

In Tables 4 and 6, minimum, maximum, and mean total computational weights of all 16 sub-domains are shown per phase for all partitioning methods considered. Cut edges as well as load imbalance factors per phase (λ^1 and λ^2) and total load imbalance factors (λ^2_{tot}) are given in addition.

The results in Tables 3–6 demonstrate that all multi-partitioning methods are able to achieve nearly perfect load balance. The multi-partitioners reach a load imbalance of around 1% and less for both models whereas the single phase partitioner ends up with a load imbalance of about 15% for the AUDI and of about 9% for the BMW. Table 6 does not show significant load imbalance differences for using the combined

Table 4
Total computational weight per phase, cut edges, and imbalance, AUDI

Method	FE [min max]	CO [min max]	Edge cut	Imbalance [FE CO] total
1	[10 428 10 437]	[126 134]	2193	[1.000 1.011] 1.001
2	[9354 10 536]	[132 134]	3116	[1.001 1.011] 1.010
3	[10 095 10 548]	[128 136]	2308	[1.011 1.026] 1.011
4	[10 158 10 632]	[126 136]	2682	[1.019 1.026] 1.019
5	[10 380 10 518]	[130 136]	2641	[1.008 1.026] 1.008
6	[9075 11 070]	[0 1128]	4154	[1.061 8.513] 1.155
	Mean: 10432.1	Mean: 132.5		

Table 5
Distribution of shell elements, beam elements, nodes and contact pairs (CPs) per sub-domain, BMW model, method 1

PE	Shell	Beam	CPs	Nodes	Phase FE	Phase CO
0	3267	16	122	2933	19 650	244
1	3266	18	116	2937	19 650	232
2	3275	0	123	3005	19 650	246
3	3269	12	122	2783	19 650	244
4	3275	0	123	2950	19 650	246
5	3259	32	122	2910	19 650	244
6	3259	31	123	2812	19 647	246
7	3257	36	123	2850	19 650	246
8	3271	8	123	2784	19 650	246
9	3266	19	123	2738	19 653	246
10	3246	58	123	3108	19 650	246
11	3271	8	123	2841	19 650	246
12	3275	0	122	2807	19 650	244
13	3272	6	123	2875	19 650	246
14	3230	90	123	3123	19 650	246
15	3258	34	123	2920	19 650	246
Total	52 216	368	1957	46 376	314 400	3914
Mean					19 650	244.625

Table 6
Total computational weight per phase, cut edges, and imbalance, BMW

Method	FE [min max]	CO [min max]	Edge cut	Imbalance [FE CO] total
1	[19 647 19 653]	[232 246]	4018	[1.000 1.006] 1.000
2	[17 238 19 992]	[238 246]	8472	[1.017 1.006] 1.017
3	[19 281 19 857]	[220 248]	4096	[1.010 1.014] 1.011
4	[19 245 19 848]	[234 250]	4017	[1.010 1.022] 1.010
5	[19 422 19 953]	[238 252]	5029	[1.015 1.030] 1.016
6	[18 525 20 706]	[0 982]	6863	[1.054 4.014] 1.090
7	[19 470 19 758]	[238 248]	15 447	[1.006 1.014] 1.006
8	[19 422 19 809]	[234 246]	17 301	[1.008 1.006] 1.008
9	[19 566 19 818]	[242 248]	18 629	[1.009 1.014] 1.009
	Mean: 19 650	Mean: 244.6		

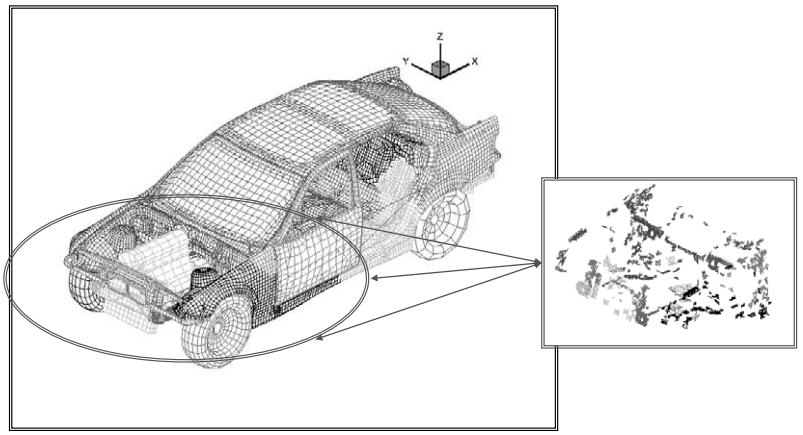

Fig. 4. Multi-constraint Metis (method 3) for a BMW PAM-CRASH^{TM} model: whole partition and partition of the contact area.

or the extended dual graph, but the number of cut edges is markedly higher for the extended dual graph.

A graphical representation of the handling of the contact phase is given in Fig. 4 for the BMW PAM-CRASH^{TM} model. Multi-constraint Metis (method 3) was applied to partition the BMW into 8 sub-domains. On the left, the whole partition is displayed, whereas the part of the partition where contact occurs is shown on the right. For a frontal crash, contact-impact mainly takes place in the front part of the car. Note that, due to multi-partitioning, all 8 subdomains share the contact area.

Fig. 5 shows the dynamic contact cost comparisons for a PAM-CRASH^{TM} simulation with the BMW model using re-partitioning after every 10,000 steps, on 8 processors of an NEC Cenju-4 system (R10000 processors, 400 Mflops, 200 MB/s maximum network transfer rate). The elapsed times for dynamic contact calculations

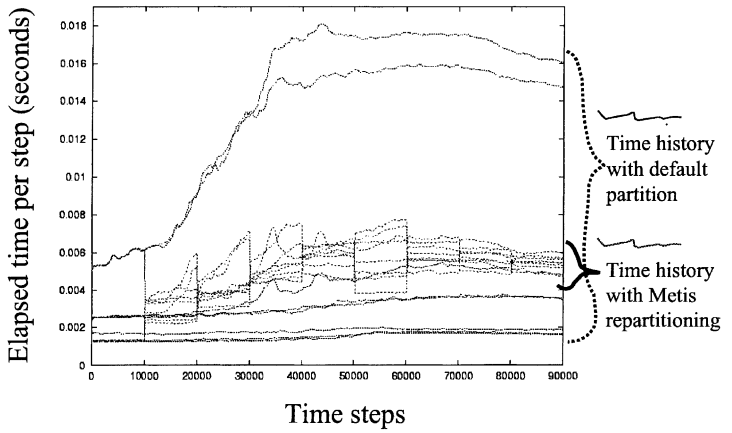

Fig. 5. Multi-constraint Metis (method 3) for a BMW PAM-CRASH™ model: dynamic contact cost comparisons for re-partitioning after every 10,000 steps (8 processors, Cenju-4).

per step on each of the 8 processors are displayed. Costs in the stress–strain phase are balanced with and without re-partitioning. With DRAMA multi-partitioning, load imbalance in the contact phase is markedly decreased (lower bracket).

7. Conclusions

As demonstrated by test cases from a real industrial simulation code, all multi-partitioning methods achieve nearly perfect load balance whereas single phase partitioners fail to improve the initial imbalance. The new mesh distribution balances both computational phases simultaneously with small remaining imbalance.

For this simple instrumentation with all the computational costs carried by elements the two graph types considered give similar partitions. The main difference is a higher edge cut for the extended dual graph. The reason is the large number of neighbours each element has in this graph. We expect that the actual communication volume that occurs in the application is modelled more accurately by the combined graph because communication is node-based in most of the finite element applications.

Acknowledgements

The authors would like to thank 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. The support of the European Commission through ESPRIT IV (Long-Term Research) Programme is gratefully acknowledged.

References

[1] The DRAMA Consortium, Homepage: http://www.ccrl-nece.technopark.gmd.de/DRAMA.
[2] The DRAMA Consortium, Report on re-partitioning algorithms and the DRAMA library, DRAMA Project Deliverable D1.3a, in [1], 1998.
[3] The DRAMA Consortium, Final DRAMA cost model, DRAMA Project Deliverable D1.1b, in [1], 1999.
[4] The DRAMA Consortium, Updated library interface definition, DRAMA Project Deliverable D1.2c, in [1], 1999.
[5] S.A. Attaway, E.J. Barragy, K.H. Brown, D.R. Gardner, B.A. Hendrickson, S.J. Plimpton, Transient solid dynamics simulations on the Sandia/Intel Teraflop Computer, in: Proceedings Supercomputing'97, Technical Paper, 1997.
[6] J. Clinckemaillie, B. Elsner, G. Lonsdale, S. Meliciani, S. Vlachoutsis, F. de Bruyne, M. Holzner, Performance issues of the parallel PAM-CRASH code, Int. J. Supercomputer Applications and High Performance Computing 11 (1997) 3–11.
[7] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs, in: S. Karin (Ed.), Proceedings Supercomputing'95, ACM Press, New York, 1995.
[8] G. Karypis, V. Kumar, ParMetis: Parallel graph partitioning and sparse matrix ordering library, Technical Report # 97-060, 1997, University of Minnesota, Minneapolis.
[9] G. Karypis, V. Kumar, Multilevel algorithms for multi-constraint graph partitioning, in: Proceedings of the 10th Supercomputing Conference, 1998.
[10] K. Schloegel, G. Karypis, V. Kumar, Parallel multilevel algorithms for multi-constraint graph partitioning, Technical Report # 99-031, 1999, University of Minnesota, Minneapolis.
[11] G. Karypis, V. Kumar, Parallel multilevel k-way partition scheme for irregular graphs, SIAM Rev. 41 (1999) 278–300.
[12] B. Maerten, A. Basermann, J. Fingberg, G. Lonsdale, D. Roose, Parallel dynamic mesh re-partitioning in FEM codes, in: B.H.V. Topping (Ed.), Advances in Computational Mechanics with High Performance Computing, Proceedings of the Second Euro-Conference on Parallel and Distributed Computing for Computational Mechanics, Saxe-Coburg, 1998, pp. 163–167.
[13] B. Maerten, D. Roose, A. Basermann, J. Fingberg, G. Lonsdale, DRAMA: a library for parallel dynamic load balancing of finite element applications, in: Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Philadelphia, 1999.
[14] C. Walshaw, M. Cross, Parallel optimisation algorithms for multilevel mesh partitioning, Parallel Computing 26 (2000) 1635–1660.
[15] C. Walshaw, M. Cross, M. Everett, Parallel dynamic graph partitioning for adaptive unstructured meshes, J. Par. Distrib. Comput. 47 (1997) 102–108.
[16] C. Walshaw, M. Cross, K. McManus, Multiphase mesh partitioning, Appl. Math. Modelling 25 (2000) 123–140 (originally published as Univ. Greenwich Tech. Rep. 99/IM/51).