DYNAMIC MULTI-PARTITIONING FOR PARALLEL FINITE ELEMENT APPLICATIONS

A. BASERMANN, J. FINGBERG AND G. LONSDALE
C&C Res. Lab., NEC Europe Ltd., Rathausallee 10, 53757 St. Augustin, Germany

B. MAERTEN
K. U. Leuven, Dept. Comp. Science, Celestijmemlaan 200A, B-3001 Heverlee-Leuven, Belgium

C. WALSHAW
U. Greenwich, Centre for Num. Modelling and Proc. Analysis, Wellington St., Woolwich, London SE18 6PF, UK

1 Introduction

The DRAMA project [1] has been initiated to support the take-up of large scale parallel simulation in industry by dealing with the main problem which restricts 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 repartitioning 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. We discuss the basic features of this cost model which allows a general approach to load identification, modelling and imbalance minimisation. First results are presented which show the necessity for multi-phase/multi-constraint partitioning components.

2 DRAMA Cost Model

The DRAMA cost model [7] 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 code,[4] 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* [2,7] which represent the occurring costs of such dependencies (see also 4). 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.

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 type 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 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.

Cost parameter determination requires application code instrumentation. Numbers 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 numbers of computational operations allow the determination of calculation speeds s_i^{calc}. For communication, the number of bytes $noc(u_1, u_2)$ that have potentially to be transferred for a link between two objects of type u_1 and u_2 and communication speeds $s_{i,j}^{comm}$ have to be specified (latency is not considered). $s_{i,j}^{comm}$ 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 has the following form.

$$F = \sum_{iphase} \max_i F_i^{iphase} \quad , \quad F_i^{iphase} = w_i^{iphase} + \sum_j c_{i,j}^{iphase}$$

$$w_i^{iphase} = \sum_u N_i(u) \frac{nop_i(u)}{s_i^{calc}} \quad , \quad c_{i,j}^{iphase} = \sum_{u_1 u_2} N_{i,j}(u_1, u_2) \frac{noc(u_1, u_2)}{s_{i,j}^{comm}}$$

$N_i(u)$ is the number of elements/nodes of type u and $nop_i(u)/s_i^{calc}$ 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 $noc(u_1, u_2)/s_{i,j}^{comm}$ is the potential communication cost for a link between two objects of type 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 future computational and communication costs.[2] 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 Dynamic Load Balancing with DRAMA

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 targeted 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

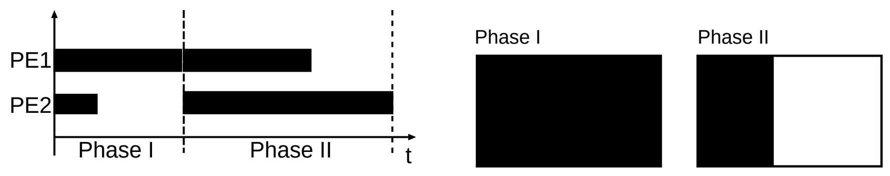

Figure 1. Left: Load imbalance in two phases separated by two sychronisation points. Right: Operations in two phases on different parts of the same mesh.

geometric (RCB), topological (graph) and local improvement (direct mesh migration) methods.[3] It enables the use of leading graph partitioning algorithms through internal interfaces to ParMetis and PJostle.[5,8]

Over using the graph partitioners directly, 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 managment.

Thus, DRAMA provides pre-defined solutions for most mesh-based 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. Figure 1 (left) illustrates the situtation 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 phase in PAM-CRASH — the computations refer to the same mesh in both phases — balancing the aggregate costs of both phases is of no use, both phases have to be balanced separately. There are two approaches to this problem, one is to work with a separate division of objects for each phase, 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 Figure 1 (right). The first phase refers to the whole mesh, the second only to the left part of the mesh. For a frontal car crash simulation against a rigid wall with PAM-CRASH, stress-strain is computed for the whole mesh whereas contact detection and correction is mainly performed in the front part of the car model.

Here we follow the single mesh decomposition strategy because it is much easier to implement in existing applications. We show first results for the FE phase and contact phase of PAM-CRASH exploiting the new multi-phase/multi-constraint options of Jostle and Metis.[6]

The graph-partitioning is built upon a combined graph of elements and nodes [3] because a part of the computation is node-based and a part element-based. The basic objects during contact detection are pairs of nodes and segments of a surface, the segment being defined by four nodes. These objects are passed to the DRAMA library as virtual five-node elements in the DRAMA mesh format.[2]

5 Evaluation of different partitioning techniques

To demonstrate the viability of the DRAMA approach we show first results obtained with multi-constraint (mc) Metis and multi-phase Jostle (MJostle) for a box-beam model with PAM-CRASH. We start from an initial partition with 44% imbalance ($\lambda = 1.44$). The reason for this imbalance are 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 repartitioning 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

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

Table 1. Distribution of shell elements and contact pairs (CPs) per sub-domain for different partitioning methods together with the load imbalance factors for the 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_{tot}^1	1.442		1.026		1.004		1.002		1.007	
λ_{tot}^2	1.442		1.302		1.455		1.002		1.007	

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

$\overline{x_i}$ denotes the mean value of all x_i, $i = 0..p-1$. λ_{tot}^1 neglects synchronisation points, whereas λ_{tot}^2, the real load imbalance, considers them.

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_{tot}^2 close to one. Of course, the other schemes minimised the aggregate cost-function as can be seen from the values of λ_{tot}^1 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 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 80,000 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%, the initial total load imbalance of the BMW model is 3.1%. For the graph representation of the mesh we use a combined graph [3] consisting of elements and nodes where the connections are only between elements and nodes. We consider the methods listed in Table 2.[5,6,8]

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* is a re-partitioner that should minimise load-imbalance and the difference between the current and the new partition. The latter was not investigated here but will be checked in detail in future tests. Tables 3 and 4 show minimum, maximum, and mean total computational weights per

Table 2. Re-partitioning 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

phase of 16 sub-domains for the AUDI and the BMW model with all partitioning methods considered. *FE* and *CO* are the costs for the stress-strain phase and the contact phase. Cut edges as well as load imbalance factors per phase and total load imbalance factors are given in addition.

The results in Tables 3 and 4 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.

6 Conclusions

As demonstrated by tests with meshes 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.

Acknowledgements

First, the authors would like to thank all our colleagues from the DRAMA project. 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, Project Homepage:
`http://www.cs.kuleuven.ac.be/cwis/research/natw/DRAMA.html`

Table 3. Total computational weight per phase, cut edges, and imbalance, AUDI.

meth.	FE [min max]	CO [min max]	edge cut	imbalance [FE CO] tot.
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 4. Total computational weight per phase, cut edges, and imbalance, BMW.

meth.	FE [min max]	CO [min max]	edge cut	imbalance [FE CO] tot.
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		

2. The DRAMA Consortium, *Updated Library Interface Definition*, DRAMA Project Deliverable D1.2b,[1], 1999.
3. The DRAMA Consortium, *Report on Re-Partitioning Algorithms and the DRAMA Library*, DRAMA Project Deliverable D1.3a,[1], 1998.
4. J. Clinckemaillie, B. Elsner, G. Lonsdale, S. Meliciani, S. Vlachoutsis, F. de Bruyne and M. Holzner, *Performance issues of the parallel PAM-CRASG code*, Int. J. Supercomputer Applications and High Performance Computing, **11 (1)**, page 3-11, 1997.
5. G. Karypis and V. Kumar, *ParMetis: Parallel graph partitioning and sparse matrix ordering library*, University of Minneapolis, tech. rep. #97-060.
6. G. Karypis and V. Kumar, *Multilevel Algorithms for Multi-Constraint Graph Partitioning*, University of Minneapolis, tech. rep. #98-019.
7. B. Maerten, D. Roose, A. Basermann, J. Fingberg, and 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*, SIAM, Philadelphia, CD-ROM, 1999.
8. C. Walshaw, M. Cross, and M. Everett, *Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes*, J. Par. Dist. Comput., Vol. 47, No. 2, pp. 102-108, 1997.