Dynamic load balancing in a nutshell*

Fourestier Sébastien

January 2008

Abstract

Until now, parallel computing has been the only conceivable method to solve computational mechanics issues which handle colossal amounts of computer memory and work. Its effectiveness is closely linked to load repartition among processors. Quite naturally, several tools devoted to load balancing had appeared, static ones are easier to puzzle out but are not suitable for applications where work significantly evolve during execution. Notably: adaptive mesh refinement (AMR), contact detection in transient dynamics, adaptive physics models, particle simulations and simulations. The aim of this paper is

1. to present the load balancing problem;
2. to analyse existing dynamic load balancing algorithms;
3. to outline future research outlooks.

1 Dynamic load balancing problem

To execute efficiently a task on a parallel computer, one has to assure the following, often antagonistic, properties upon computing nodes:

1. work is balanced;
2. the communication amount is minimised;

These properties can be guaranteed by a well-considered data distribution over computing units which can be generally formalized as a problem which takes in input a weighted undirected graph modeling the task to execute, where nodes represent data and associated computing work and edge represent communication amount [2]:

Definition 1.1. k-partitioning problem

Let $G = (V, E)$ be an undirected graph and $w_V^t : V \to \mathbb{R}$, $w_E^t : E \to \mathbb{R}$ be two t-dependent applications which respectively associate a positive weight to edges and nodes. Let k be destination architecture computing nodes cardinal, then k-partitioning problem can be expressed as follows:
For a given t, find a subset family of V, $(V_i)_{1 \leq i \leq k}$ such as:

1. $\forall (i,j) \in [\![1,k]\!], \left| \sum_{v \in V_i} w_V^t(v) - \sum_{v \in V_j} w_V^t(v) \right|$ is minimal;
2. $\sum_{1 \leq i < j \leq k} \sum_{\substack{\{u,v\} \in E \\ (u,v) \in V_i \times V_j}} w_E^t(u,v)$ is minimal.

Remark 1.1. We suppose in this paper that the underlying architecture is homogeneous (*i.e.* parallel computer nodes have identical computing power and communication cost is uniform). With unhomogeneous architectures such as NUMA ones, a second graph is used to represent destination architecture and in this case, load balancing problem is an extension to *k-partitioning problem* generally called *graph placement problem.*

*This note is a summary of Bruce Hendrickson and Karen Devine publication entitled "*Dynamic load balancing in computational mechanics*" [1]

k-partitioning problem resolution can be carried out statically or dynamically (before or during task execution) by tools called "load balancers". Although dynamic load balancers are harder to figure out, they are suitable to handle tasks whose computing work associated to data significantly evolves during execution, *i.e.* w_V^t and/or w_E^t applications are closely dependent to t.

By definition, a dynamic load balancer has to be run during task execution, this can lead to slow down. As a consequence, it should satisfy the following extra property:

3. load balancer resource usage is not prejudicial to task execution;

Dynamic load balancers are called up several times during task execution, as data sending to computing nodes is expensive, they should strive to compute new data distribution starting from existing one. Thus, another property dynamic load balancer should try to achieve is:

4. data distribution should be computed incrementally.

According to task knowledge (fine-grained task description as a graph or an inaccurate one which does not take communications in consideration), several dynamic load balancer families can be defined. These families handle a subproblem of *general load balancing problem*. We will first focus on independent task dynamic distribution problem.

2 Provider/consumer model

When work can be divided into independent tasks weighted with corresponding load, definition 1.1 can be simplified:

Definition 2.1. Independent task distribution problem

Let V be a task set, $w : V \to \mathbb{R}$ be an application which associates a weight to tasks and k be destination architecture computing nodes cardinal, then independent task distribution problem can be expressed as following:
Find a subset family of V, $(V_i)_{1 \leq i \leq k}$ such as:

$$\forall (i,j) \in [\![1,k]\!], \left| \sum_{v \in V_i} w(v) - \sum_{v \in V_j} w(v) \right| \text{ is minimal.}$$

Task weights can be computed with regard to:

- computational work associated to the task;
- dynamic load balancing overhead: task communication cost (proportional to task size).

Task communication cost usually not included in weight since it is often negligible compared to computing work and optimisations based on communication-computation overlap could be easily achieved.

This problem could be dynamically solved by a provider/consumer model where nodes are provider, consumer or both:

- providers maintain untreated task sets (at beginning task sets correspond to the V subset family of definition 2.1);
- consumers ask for work so as to prevent starvation.

Remark 2.1. For heterogeneous architectures, if provider has knowledge about consumer computing power then it can use task weights to achieve better load balancing.

3 Geometric load balancers

When the task to execute on a parallel computer is based on a geometric issue, communications and computing work can be closely linked to this geometry. In this case, geometric knowledge of the task can be sufficient to achieve load balancing. By assigning to every object a location in the space, a specific load balancing problem can thus be expressed:

Definition 3.1. Geometric k-partitioning problem

Let V^t be a t-dependent dot set and k be destination architecture computing nodes cardinal, then geometric k-partitioning problem can be expressed as following:
For a given t, find a subset family of V^t, $(V_i^t)_{1\leq i\leq k}$ such as:

$$\forall (i,j) \in [\![1,k]\!], \left|\mathrm{card}(V_i^t) - \mathrm{card}(V_j^t)\right| \text{ is minimal.}$$

There exist several geometric partitioning methods, we will focus an geometric recursive bisection and tree based methods [3].

3.1 Recursive bisection

One of the simplest ways to partition a dot set is to use Recursive Coordinate Bisection algorithm (RCB) [4]. At each step, the dot set is bisected by means of an orthogonal plane to the most elongated coordinate axis in a way to obtain two sets of the same cardinal. This algorithm cannot achieve elegantly a k-partitioning since k is not a power of 2. Hence, an improvement of this algorithm called Unbalanced Recursive Bisection [5] has been figured out.

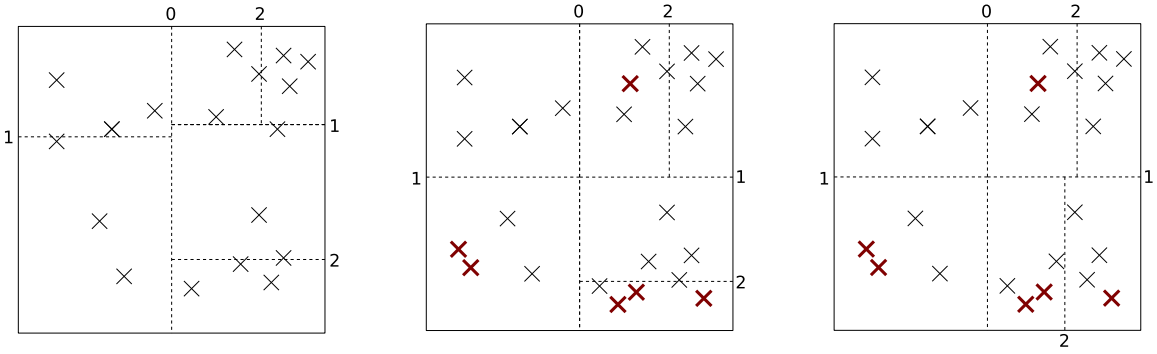
Let S_1, S_2 be the two sections of S according to an orthogonal plane to a coordinate axis and $dim(S) = n$, URB algorithm recursively bisects S assuring that:

1. $\exists p \in [\![1, k-1]\!], \mathrm{card}(S_1) = p\dfrac{\mathrm{card}(S)}{k}$;
2. subsets aspect ratio is the best:
   Let $l_{k,i}$ be the length of the interval obtained by the projection of S_k on the ith coordinate axis, then

$$\forall k \in \{1,2\}, \max_{i,j\in[\![1,n]\!]} \left(\frac{l_{k,i}}{l_{k,j}}\right) \text{ is minimal.}$$

In addition to be cost effective in terms of resource needs, these load balancers can be executed incrementally. Figure 1 compares incremental and no-incremental URB execution. URB incremental repartitioning recursively changes cutting plane position while conserving $t-1$ coordinate axis choice. Furthermore, it ensure:

$$\mathrm{card}\, S_1^t = p^{t-1}\frac{\mathrm{card}(S^t)}{k}.$$



(a) Initial URB partitioning of the set.

(b) New incrementally computed URB partitioning.

(c) New URB partitioning computed from scratch.

Figure 1: An URB parallel partitioning example with $k = 6$ and $dim(S) = 2$. In (b) and (c) new dots are added to the set, they are bold. Cutting plans are numbered according to recursive creation order.

3.2 Space filling curves based partitioners

This geometric partitioner class constructs a tree so as to order dots. Compared to recursive bisection, the leafs of space filling curve tree correspond to dots (fine-grain) instead of dot sets. The root represents the global space. Then, recursively, the space is geometrically divided along each coordinate axis (without taking into account section cardinals) in $2^{dim(V^t)}$ sub-spaces which are considered as children of previous level space. Tree thus obtained, a traversal is made to obtain a dot ordering. Eventually, this sequence is equally sliced to obtain a solution to the *geometric k-partitioning problem*.

These partitioners are incremental. Compared to recursive bisection class, they are a little faster but they produce a slightly worse partitioning.

4 Graph partitioners

Graph partitioners [2, 6], by using a fine-grain representation of the task thanks to a graph, produce a high-quality solution to *k-partitioning problem* defined in section 1. We will first focus on spectral approach. Then, the emphasis will be put on local graph load balancers. Eventually, multi-level improvement of graph methods will be described.

4.1 Spectral approach

This approach focuses on Fiedler vector properties.

Definition 4.1. Adjacency matrix

Let $G = (V, E)$ be an undirected graph, G adjacency matrix noted $A = (a_{ij})_{i,j \in [\![1;n]\!]^2}$ is defined by:

$$\forall (i,j) \in [\![1;n]\!]^2, a_{ij} = \begin{cases} 0 & \text{if } i,j \notin E(G) \\ 1 & \text{if } i,j \in E(G) \end{cases}$$

Definition 4.2. Degree matrix

Let $G = (V, E)$ be an undirected graph, G degree matrix noted $D = (d_{ij})_{i,j \in [\![1;n]\!]^2}$ is defined by:

$$\forall (i,j) \in [\![1;n]\!]^2, d_{ij} = \begin{cases} \text{degree}(v_i) & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

Definition 4.3. Laplace matrix

Let $G = (V, E)$ be an undirected graph, G Laplace matrix noted Q is defined by: $Q = D - A$ where A and D are adjacency matrix and degree matrix respectively.

As Q is semidefinite, its eigenvalue can be shortered as following: $\lambda_1 = 0 \leq \lambda_2 \leq \cdots \leq \lambda_n$. It has been proved that λ_1 multiplicity is equal to G connected components number. If G is connected, then λ_2 is strictly positive. The Fiedler vector $X = (x_i)_{i \in [\![1;n]\!]}$, the eigenvector associated to λ_2, has the following property: more two vertex v_i and v_j are connected, more $|x_i - x_j|$ is small. According to this property, a solution to the k-partitioning problem is computed thanks to a recursive bisection of $(x_i)_{i \in [\![1;n]\!]}$ set taking median value as separator. Although spectral approach gives the highest quality partitioning, computation cost is high and it cannot be achieved incrementally.

4.2 Local iterative graph optimisation

This class of algorithm strives to improve local graph partitioning. However they cannot be easily used to compute a global load balancing, these algorithms are incremental, cost-effective and can be run asynchronously. These algorithms are usually made up of two steps:

1. choose work amount to migrate;
2. select objects to move.

Several methods can be used to carry out the first step:

- modeling workflow by an heat equation;
- demand-driven method;

- dimensional exchange (for hypercube architectures): in a loop over dimensions i, processes perform load balancing with there neighbours according to the ith dimension.

Most algorithms employed for second step are based on Kernighan and Lin (KL) algorithm. KL strives to improve local graph partitioning by achieving graph node exchanges at partition borders between two partitions owners. Exchanges which ensure best gain are chosen.

The Fiduccia-Mattheyes algorithm (FM) is an improvement of KL where objects are migrated to neighbours (without giving an object in exchange). It is more efficient in terms of complexity, however, as migration can create graph partition imbalance, migrations causing intolerable unbalance are not considered.

Several variants of both KL and FM algorithms exist according to gain definitions used. Gain definition often takes into consideration the following elements:

- relative or not gain;
- migration cost reduction;
- maintain/improve graph balance;
- edge cut reduction;
- aspect ratio optimisation.

4.3 Multi-level approach

The multi-level approach permits to efficiently partition very big graphs. It is composed of three steps:

1. contraction: The initial graph is iteratively contracted into smaller graphs with close topology;
2. partitioning: The smallest graph is partitioned using a standard graph partitioning method;
3. expansion: Partition is iteratively expanded to bigger graphs till the initial graph is reached.

5 Conclusion

Several dynamic load balancer classes have been introduced in this note. Geometric and global graph classes produce better balance than local load balancers since global load balancing is achieved. Although they are generic and produce the best quality partitions, global graph methods are very expensive both in terms of time and memory, moreover they are not incremental. Producer/consumer model, geometric partitioners and iterative load balancers are fast and need only a little memory.

Eventually, as several load balancer classes are complementary, load balancer hybridization is an interesting outlook so as to obtain better performances.

References

[1] Bruce Hendrichson and Karen Devine. *Dynamic load balancing in computational mechanics.*

[2] C. Chevalier. *Conception et mise en oeuvre d'outils efficaces pour le partitionnement et la distribution parallèles de problèmes numériques de très grande taille.*

[3] Ave Magnus Bruaset and Aslak Tveito. *Numerical Solution of Partial Differential Equations on Parallel Computers.*

[4] Horst D. Simon. *Partitioning of Unstructured Problems for Parallel Processing.*

[5] Mark T. Jones and Paul E. Plassmann. *Computational Results for Parallel Unstructured Mesh Computations.*

[6] F. Pellegrini. *PT-SCOTCH 5.0 User's guide.*