A Repartitioning Hypergraph Model for Dynamic Load Balancing

Abstract

- In this paper, we present a novel repartitioning hypergraph model for dynamic load balancing that accounts for both communication volume in the application and migration cost to move data, in order to minimize the overall cost.

- Use of a hypergraph-based model allows us to accurately model communication costs rather than approximating them with graph-based models.

- the new model can be realized using hypergraph partitioning with fixed vertices and describe our parallel multilevel implementation within the Zoltan load-balancing toolkit.

I. Introduction

Objective and trade-offs of Repartitioning problem :

1. balanced load in the new data distribution;
2. low communication cost within the application (as determined by the new distribution);
3. low data migration cost to move data from the old to the new distribution;
4. short repartitioning time.

- Total application execution time :

$$t_{tot}=(t_{comp}+t_{comm})+t_{mig}+t_{repart} \qquad (1)$$

- t_{comp} : application's computation times,
- t_{comm} : application's communication times,
- t_{mig} : data migration time,
- t_{repart} : repartitioning time,
- α : indicates how many iterations of the application are executed between each load-balance operation.
- t_{comp} and t_{repart} can be ignored; the cost function to be minimized by the repartitioning algorithm reduces to :

$$cost_{time}=t_{comm}+t_{mig} \qquad (2)$$

- the time spent in communication is proportional to the amount of data being sent. Thus, the
- cost function to be minimized by the repartitioning algorithm becomes :

$$cost_{vol}=b_{com}+b_{mig} \qquad (3)$$

- b_{comm} : amount of data sent in each iteration
- b_{mig} : amount of data sent during migration.
- In this work, we present **a repartitioning-hypergraph model** that minimizes the sum of total communication volume in the application and migration cost to move data, as stated in(3).

II. Previous work on dynamic load balancing

1. Dynamic load-balancing approaches

Three main methods: **scratch-remap method**, **incremental method** and **repartitioning method**.

a) Scratch-remap method

The computational model representing the modified structure of the application is partitioned from scratch without accounting for existing part assignments. Then, old and new partitions are remapped to minimize the migration cost.

b) Incremental method

Existing part assignments are used as initial assignments and incrementally improved by using a sub-optimal cost function that minimizes either data migration cost (diffusive methods) or application communication cost (refinement methods).

c) Repartitioning method

Existing part assignments are taken into account to minimize both data migration cost and application communication cost.

2. Computational Models of Dynamic Load Balancing methods

There are three computational models : **coordinate-based models**, **graph-based models**, **hypergraph-based models**.
(refer to the article [8] for more details)

a) **coordinate-based models**
such as Recursive Coordinate Bisection and Space-Filling Curves
b) **graph-based models**
c) **hypergraph-based models**

Category	Property	Coordinate based	Graph based	Hypergraph based
Scratch-remap	Migration cost	high	high	high
	Communication cost	high	low	low
	Communication model	none	approximate	accurate
Incremental	Migration cost	moderate	low	low
	Communication cost	high	moderate	moderate
	Communication model	none	approximate	accurate
Repartitioning	Migration cost	n/a	low	low
	Communication cost	n/a	low	low
	Communication model	none	approximate	accurate

III. Preliminaries

- A hypergraph H = (V,N) is defined by a set of vertices V and a set of nets (hyperedges) N, each net $n_j \in N$ is a non-empty subset of vertices. A weight ω_i can be assigned to each vertex $v_i \in V$, and a cost c_j can be assigned to each net $n_j \in N$
- P = {V1, V2, . . . , Vk} is called a k-way partition of H if each part Vp, p = 1, 2, . . . , k, is a non-empty, pairwise-disjoint subset of V and $¿_{p=1}^{k} V_p = V$
- A partition is said to be balanced if

$$W_p < W_{avg}(1+\epsilon)\, for\, p = 1,2,\dots,k \qquad (4)$$

- where $W_p = \sum_{v_i \in V_p} \omega_i$ and $W_{avg} = \frac{\sum_{v_i \in V} \omega_i}{k}$ and $\epsilon > 0$ is a predetermined maximum tolerable imbalance.
- In a given partition P, a net that has at least one vertex in a part is considered to be connected to that part.
- The **connectivity** λj of a net nj denotes the number of parts connected by nj under the partition P of H.
- A net nj is said to be **cut** if it connects more than one part (i.e., $_j$> 1).
- CutCost(H, P) denote the cost associated with a partition P of hypergraph H.

$$CutCost(H,P) = \sum_{n_j \in N} c_j(\lambda_j - 1) \qquad (5)$$

- This cost metric exactly corresponds to communication volume in parallel computing
- The standard hypergraph partitioning problem is the task of dividing a hypergraph into k parts such that :
  - ✔ the cost (5) is minimized
  - ✔ the balance criterion (4) is maintained.

1. Hypergraph Partitioning with Fixed Vertices

- The standard hypergraph partitioning problem is the task of dividing a hypergraph into k parts such that :
  - ✔ the cost (5) is minimized
  - ✔ the balance criterion (4) is maintained.

- Hypergraph partitioning with fixed vertices is a more constrained problem. In this problem, in addition to the input hypergraph H and the requested number of parts k, a fixed-part function f(v) is also provided as an input to the problem.

- denoted by :
  - o $f(v) = -1$ i.e the vertex v is free i.e it is allowed to be in any part in the solution P.
  - o $f(v) = q \; for \; 1 \leq q \leq k$ i.e the vertex is fixed in part q i.e it is required to be in V_q in the final solution P.

2. Multilevel Partitioning Paradigm

Multilevel partitioning consists of three phases: *coarsening*, *coarse partitioning* and *refinement*. (see more details in ref [1] and [2])

a) Coarsening phase.

- hierarchy of smaller hypergraphs that approximate the original one is generated

b) Coarse partitioning phase.

- The smallest hypergraph obtained at the end of the coarsening phase is partitioned.

c) Refinement phase

- the coarse partition is projected back to the larger hypergraphs in the hierarchy and improved using a local optimization method.

IV. Repartitioning Hypergraph Model

- We call the period between two subsequent load-balancing operations an ***epoch*** of the application.

- An epoch consists of one or more computation iterations and the computational structure and dependencies of an epoch can be accurately modeled with a computational hypergraph.

- The hypergraph that models the j^{th} epoch of the application is donated by $H^j = (V^j, N^j)$ and the number of computation iterations in that epoch by α_j.

- Load balancing for the first epoch is achieved by partitioning H_1 using a static partitioner.

- Here the repartitioning hypergraph model appropriately captures both application communication and data migration costs associated with an epoch.

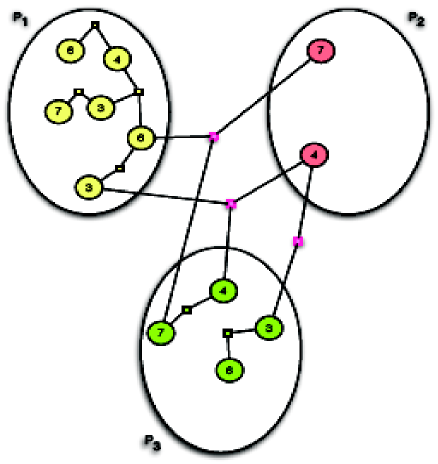
- To model migration costs in epoch j, we construct a repartitioning hypergraph $\bar{H}^j = (\bar{V}^j, \bar{N}^j)$ by augmenting H^j with k new vertices corresponding to each of the k parts, and $|V^j|$ new hyperedges using the following procedure:

  o Scale each net's cost (representing application communication) in N_j by $_j$ while keeping the vertex weights intact.

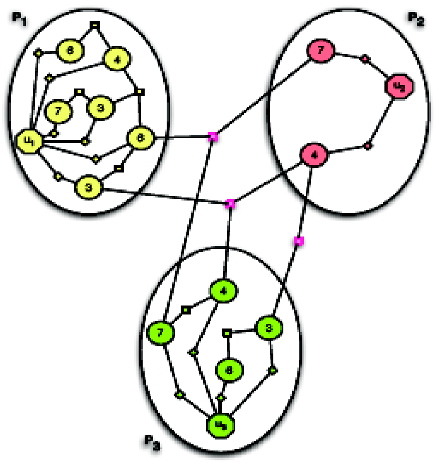
  o Add a new part vertex u_i with zero weight for each part *i*, and fix those vertices in respective parts; i.e., $f(u_i)=i \; for \; 1 \leq i \leq k$ Hence $\bar{V}^j = V^j \cup (u_i | 1 \leq i \leq k)$

- For each vertex $v \in V^i$,add a migration net n_v between *v* and u_i if *v* is assigned to part *i* at the beginning of epoch *j*. Set the migration net's cost c_v to the size of the data associated with v, since this migration net represents the cost of moving vertex *v* to a different part.

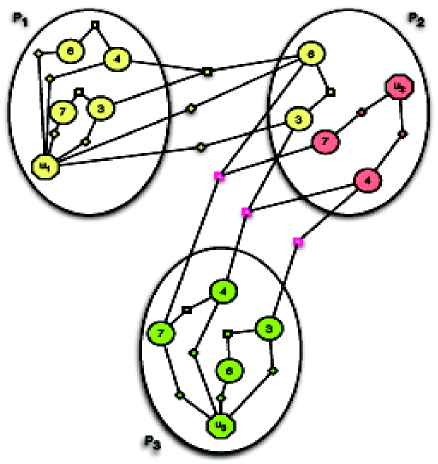
$$\begin{aligned} cost_{vol} &= b_{com} + b_{mig} \\ cost_{vol} &= \alpha_j CutCost(H^j, P^j) + \sum_{n_v \in (\bar{N}^j - N^j)} c_v(\lambda_v - 1) \\ cost_{vol} &= \alpha_j \sum_{n_j \in N^j} c_j(\lambda_j - 1) + \sum_{n_v \in (\bar{N}^j - N^j)} c_v(\lambda_v - 1) \end{aligned} \qquad (7)$$



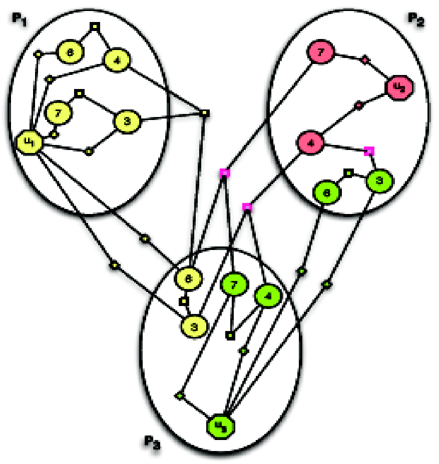
(a)



(b)



(c)



(d)

		c	d
b_{comm}		4	3
b_{mig}		2	4
$cost_{vol}$	$\alpha=۱$	$۱*۴+۲=۶$	$۱*۳+۴=۷$
	$\alpha=۱۰$	$۱۰*۴+۲=۴۲$	$۱۰*۳+۴=۳۴$

V. Parallel Repartitioning Tool

1. Coarsening Phase

- we approximate the original hypergraph with a succession of smaller hypergraphs with similar connectivity and equal total vertex and edge weight.

- Parallel matching is performed in rounds. In each round, each processor broadcasts a subset of candidate vertices that will be matched in that round. Then, all processors concurrently compute their best match for those candidates and the global best match for each candidate is selected.
- For fixed-vertex partitioning, vertices that are fixed to different parts, are not allowed to match.
- There are three scenarios in which two vertices match:
  - Both vertices are fixed to the same part,
  - Only one of the vertices is fixed to a part,
  - Both are not fixed to any parts (i.e.,both are free vertices).

2. Coarse Partitioning Phase

- In the coarse partitioning phase, we construct an initial partition of the coarsest hypergraph available.

3. Refinement Phase

- The code is based on a localized version of the successful Fiduccia-Mattheyses method,
- The algorithm performs multiple pass-pairs and in each pass, each free vertex is considered to move to another part to reduce the cut metric.

4. Handling Fixed Vertices in Recursive Bisection

- Zoltan uses recursive bisection, to obtain a k-way partition. This recursive bisection approach can be extended easily to accommodate fixed vertices.
- Then, the multilevel partitioning algorithm with fixed vertices described above can be executed
- This scheme is applied recursively in each bisection.

VI. Experimental Results

1. Repartitioning Approaches

- **Repartitioning technique:**
  - Three categories: *scratch-remap*, *incremental* and *repartitioning*.
  - Only refinement approaches within the incremental techniques category are considered.

- **Cost model:**
  - Coordinate-based models are not considered.

- **Optimization method:**
  - Distinction between single-level versus multi-level partitioners

Partitioner	Repartitioning technique	Cost model	Optimization method	Software
Z-repart	repartitioning	hypergraph	multilevel	Zoltan
Z-SL-repart	repartitioning	hypergraph	single level	Zoltan
Z-scratch	scratch-remap	hypergraph	multilevel	Zoltan
Z-SL-refine	iterative	hypergraph	single level	Zoltan
M-repart	repartitioning	graph	multilevel	ParMETIS
M-scratch	scratch-remap	graph	multilevel	ParMETIS

Properties of the partitioners used in the experimental evaluation.

We compare six different partitioners given in Table 2 that collectively cover all options with respect to each of the three aspects considered.

2. Dynamically Perturbed Data Experiments (refer to section 6.2 of [3])

- Two different methods are used to dynamically perturb the data in the experiments :
  - **Dynamic structure perturbation**
    The first method introduces biased random perturbations that change the structure of the data.
  - **Dynamic weight perturbation**
    The second method simulates adaptive mesh refinement.

Name	**|V |**	**|E|**	**vertex degree**			**Application Area**
			min	max	avg	
xyce680s	682,712823,2321	209	2.4	VLSI	design	
slac6M	5,955,366	11,766,788	2	4	4.0	Finite element mesh
cage15	5,154,859	47,022,346	2	46	18.2	DNA electrophoresis

Properties of the test datasets; \|V \| and \|E\| are the numbers of vertices and graph edges, respectively.

- The results indicate that our new hypergraph repartitioning method *Z-repart* performs better than *M-repart* in terms of minimizing the total cost in the majority of the test cases.

- Therefore, *Z-repart* provides a more accurate trade-off between communication and migration costs than *M-repart* to minimize the total cost.

3. Adaptive Mesh Refinement Experiments (ref to [3])
4. Term-by-Document Experiments (ref to [3])

References :

[1] : Parallel Hypergraph Partitioning for Scientific Computing.
[2] : Hypergraph based dynamic load balancing for adaptive scientific computations.
[3] : A Repartitioning Hypergraph Model for Dynamic Load Balancing.
[4] : Graph partitioning model for parallel computing.
[5] : A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs.
[6] : Partitioning and Load Balancing for Emerging Parallel Applications and Architectures.
[7] : Dynamic Load Balancing in Computational Mechanics.
[8] : Partitioning and dynamic load balancing for the numerical solution of partial differential equations.