A Heuristic for Partitioning Parallel Computation

Weizhen Mao *and* David M. Nicol
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187-8795

Abstract

Parallel computation can usually be viewed as a weighted undirected graph, where graph nodes typically represent computation, and edges represent communication. One seeks to distribute the total workload (computation and communication) by partitioning the graph into subgraphs and assigning each subgraph to a processor such that every processor has approximately the same amount of workload. In this paper, we prove the intractability of this graph partition problem, present a greedy heuristic, and analyze the performance of the algorithm.

Keywords: algorithms, load partitioning and balancing, optimization.

1 Introduction

The problem of assigning workload in a parallel system has long been considered as important, and in the general case, as intractable. A significant amount of research has addressed the issue of finding good, if not optimal, workload assignments.

Parallel computation can usually be viewed as an undirected graph with node weights and edge weights, reflecting the relative computation and communication volumes, respectively. For any subgraph, define an *internal edge* to be one with both endpoints in the subgraph and an *external edge* to be one with just one endpoint in the subgraph. Viewing the subgraph as the set of nodes assigned to a processor, the sum of the node weights of the subgraph is a measure of computation cost and the sum of the external edge weights of the subgraph is a measure of communication cost. Define the *load* of a subgraph to be the sum of the weights of its nodes and its external edges. If the original graph is partitioned into p subgraphs, then the *bottleneck* cost of the partition is taken to be the maximum load among all subgraphs. Our goal is to determine the optimal partition that minimizes the bottleneck cost.

The partitioning problem with bottleneck cost as objective function was studied in various literatures, such as [3] and [5]. In a previous paper [4] we showed that certain equi-partitions are optimal, but that, surprisingly, there exist cases where the optimal partition is *not* an equi-partition. Further, it was proved in [2] and [4] that when the graph is a k-ary n-cube the partition is optimal if each subgraph is a set of nodes clustered as tightly as possible. In this paper, we will use a rather different approach, focusing on design and analysis of algorithms that yield near-optimal partitions.

We organize this paper as follows. In Section 2, we define the problem and its variations. In Section 3, we present our algorithm (heuristic), PAIRING. In Sections 4 and 5, we study the performance of the algorithm. Finally, we conclude in Section 6.

2 Problem formulations

We first define the following GRAPH PARTITION problem and show its NP-completeness.

GRAPH PARTITION

INSTANCE: An undirected graph $G = (V, E, w_1, w_2)$, where V is a set of nodes, E is a set of edges, $w_1 : V \to Z^+$ is a node weight function, and $w_2 : E \to Z^+$ is an edge weight function, a positive integer p, where $1 \le p \le |V|$.

QUESTION: Is there a partition of V into $V_1, V_2, \ldots, V_p$ such that for a given $B > 0$,

$$\max_{1 \le i \le p} \{ \sum_{v \in V_i} w_1(v) + \sum_{e=(u,v), u \in V_i, v \notin V_i} w_2(e) \} \le B \ ?$$

Theorem 1. GRAPH PARTITION *is NP-complete.*

Proof Reduction from the NP-complete PARTITION problem [1]. □

We can also define the following variations and show their NP-completeness (omitted).

UNIT-WEIGHT GRAPH PARTITION

INSTANCE: An undirected graph $G = (V, E)$, where V is a set of nodes, and E is a set of edges, a positive integer p, where $1 \leq p \leq |V|$.

QUESTION: Is there a partition of V into $V_1, V_2, \ldots, V_p$ such that for a given $B > 0$,

$$\max_{1 \leq i \leq p} \{|V_i| + |E(V_i)|\} \leq B,$$

where $E(V_i)$ is the set of external edges of V_i?

EQUAL-WEIGHT GRAPH PARTITION INTO EQUAL-SIZED SUBSETS

INSTANCE: An undirected graph $G = (V, E, w_1, w_2)$, where V is a set of $s \cdot p$ nodes, E is a set of edges, $w_1 : V \to \{c_1\}$ for some constant c_1 is the node weight function, and $w_2 : E \to \{c_2\}$ for some constant c_2 is the edge weight function, a positive integer p, where $1 \leq p \leq |V|$.

QUESTION: Is there a partition of V into $V_1, V_2, \ldots, V_p$ such that $|V_i| = s$ for $1 \leq i \leq p$ and that for a given $B > 0$,

$$\max_{1 \leq i \leq p} \{|V_i|c_1 + |E(V_i)|c_2\} \leq B?$$

SIMPLE GRAPH PARTITION

INSTANCE: An undirected graph $G = (V, E)$, where V is a set of $p \cdot s$ nodes and E is a set of edges.

QUESTION: Is there a partition of V into $V_1, V_2, \ldots, V_p$ such that $|V_i| = s$ for $1 \leq i \leq p$ and that for a given $B > 0$,

$$\max_{1 \leq i \leq p} \{|E(V_i)|\} \leq B?$$

3 A Heuristic

The following two approaches are usually used to study NP-complete problems: (1) Design optimal algorithms for subsets of instances, and (2) Design heuristics (approximation algorithms) for the problem, and analyze their performances over all instances. In this section, we will design a greedy algorithm for the GRAPH PARTITION problem.

We say that a partition is an *equi-partition* if all subgraphs in the partition have about the same node count. Although there exist examples in which optimal partitions are not equi-partitions, the following theorem shows that when the node weights are large enough (massive computation) and the edge weights are relatively small (fast communication links), the optimal partition with respect to the bottleneck cost is an equi-partition.

Theorem 2. *In any graph of $p \cdot s$ nodes with $w_1(v) = w \geq |E|, \forall v \in V$ and $w_2(e) = 1, \forall e \in E$, the optimal partition $\{V_1, V_2, \ldots, V_p\}$ has $|V_i| = s, \forall i$.*

Proof Assume that in the optimal partition, there exist V_i and V_j such that $|V_i| \geq |V_j|+2$. We can show that the cost contributed by V_i and V_j to the cost function, i.e., $cost(V_i, V_j) = \max\{|V_i| \cdot w + |E(V_i)|, |V_j| \cdot w + |E(V_j)|\}$, is no less than that when a node in V_i is moved to V_j. □

In the environment of high-performance computing, massive computation is done in a high-speed network. So from now on, we adopt the following assumptions: (1) $|V| = n = 2^k$, and $p = 2^l$, for $0 \leq l \leq k$; (2) $w_1(v) = w$, for all $v \in V$, where $w \geq |E|$; and (3) $w_2(e) = 1$, for all $e \in E$.

The heuristic we will present is in fact a greedy algorithm, which merges node subsets connected by the heaviest-weighted edges until the size of the partition is achieved, while keeping the node counts of the subgraphs as closely as possible. We call this algorithm PAIRING. For simplicity, assume that nodes in G are labeled. Each step/round/pass in PAIRING is defined as follows.

```
while there is v in V that is not paired do
   choose an unpaired node with the lowest
      label, say v;
   pair v with an unpair node via an edge
      with the maximum weight
   (if more than one candidate, choose the one
      with the lowest label);
   mark both nodes as paired;
combine every two paired nodes as one;
relabel the new nodes according to the order
   the pair was created;
merge the corresponding edges by summing up
   the edge weights.
```

4 When is PAIRING Optimal?

In this section, we will examine a few special types of graphs which play important roles in parallel systems and determine if PAIRING is optimal for these graphs.

Lemma 1. *Let V_i be any subset of m nodes in a hypercube and $I(V_i)$ be the number of internal edges in V_i. Then*

$$I(V_i) \leq \frac{m}{2} \cdot \log_2 m.$$

Proof By induction on m. □

Theorem 3. *PAIRING is optimal for hypercubes.*

Proof In a hypercube of $n = 2^k$ nodes, $degree(v) = k, \forall v \in V$. After $k - l$ steps of PAIRING, the graph becomes a hypercube of $p = 2^l$ nodes with edge weights 2^{k-l}. The bottleneck cost of the partition obtained by PAIRING is therefore $2^{k-l}w + l2^{k-l}$.

We next prove that for any partition of size $p = 2^l$ with equal-size subsets, its bottleneck cost is no less than $2^{k-l}w + l2^{k-l}$. This is left to the reader. □

Theorem 4. *PAIRING is optimal for toroidal meshes when the nodes in each subset of the partition obtained by PAIRING are placed in squares, and not optimal otherwise.*

Proof Omitted. □

Theorem 5. *PAIRING is not optimal for meshes.*

Proof By counterexamples. □

Theorem 6. *PAIRING is not optimal for meshes with diagonals.*

Proof By counterexamples. □

5 Hierarchically Defined Graphs (HDG)

In this section, we show that PAIRING also works well for a class of *hierarchically defined graphs* (HDG). We first define $HDG = HDG(0) \cup HDG(1) \cup HDG(2) \cup \cdots$, where

- $HDG(0)$ contains graph $(\{v_0\}, \emptyset)$.
- $HDG(k)$ contains graphs of 2^k nodes, each of which includes two graphs in $HDG(k-1)$ as subgraphs with one additional edge connecting them.

For instance, graph $(\{v_0, v_1\}, \{(v_0, v_1)\})$ is the only member in $HDG(1)$, and $HDG(2)$ has 4 graphs, one of which is $(\{v_0, v_1, v_2, v_3\}, \{(v_0, v_1), (v_1, v_2), (v_2, v_3)\})$. We observe that HDG represents the group of parallel computation which consists of clusters connected by a single communication link, corresponding to the most economical communication pattern.

Now let us consider $HDG(k)$. Any graph in $HDG(k)$ has 2^k nodes and $2^k - 1$ edges, and is connected. Therefore, $HDG(k)$ is in fact a set of trees with 2^k nodes. Note that not all trees with 2^k nodes are in HDG, such as a tree of a root with three children. It is easy to verify that PAIRING is optimal for graphs in $HDG(k)$ when $p = 2^0, 2^1$ and 2^k. We now assume $2 \le l \le k-1$.

Theorem 7. *PAIRING is optimal for $G \in HDG(k)$ if G is constructed by the definition of HDG in such a way that for any subset V_i of 2^{k-l} nodes in the partition obtained, the total degree of the nodes is no larger than 2^{k-l+1}.*

Proof Omitted. □

Next, we give a worst-case performance bound of PAIRING on arbitrary $HDG(k)$.

Theorem 8. *For any instance $G \in HDG(k)$,*

$$cost(PAIRING) - cost(OPT) \le \log p - 2$$

and

$$\frac{cost(PAIRING)}{cost(OPT)} \le \frac{33}{32}.$$

Proof For any subset V_i in the partition obtained by PAIRING, assume $degree(V_i) \le 2(2^{k-l} - 1) + \delta$, where $\delta \ge 0$ is an integer. Consider the passes after the graphs have at least 2^{k-l} nodes, where in each pass δ may be increased by 1. Since there are l such passes, $\delta \le l$.

In the partition obtained by PAIRING, we have $cost(PAIRING) \le 2^{k-l}w + 2(2^{k-l} - 1) + \delta - 2(2^{k-l} - 1) \le 2^{k-l}w + l$. In the optimal partition, we have $cost(OPT) \ge 2^{k-l}w + \lceil \frac{2(2^k-1)}{p} \rceil - 2(2^{k-l}-1) = 2^{k-l}w + 2$. Therefore, the first inequality holds. According to our assumptions, $w \ge |E| = 2^k - 1$, and $2 \le l \le k-1$. Therefore, the second inequality also holds. □

6 Conclusion

In this paper we presented a greedy partitioning algorithm and proved its optimality in the sense of minimizing the bottleneck cost for important parallel architectures such as hypercubes and some toroidal meshes. We also showed that the algorithm works very well for a class of hierarchically defined graphs. For future research, we will like to study the performance of the heuristic for arbitrary parallel computation.

Acknowledgement

Weizhen Mao was supported by NSF grant CCR-9210372. David Nicol was supported by NASA grant NAS1-19480 and NSF grant CCR-9201195.

References

[1] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[2] W. Mao and D. M. Nicol, "Combinatorics of k-ary n-cubes with Applications to Partitioning", Proceedings of the Fourth International Conference for Young Computer Scientists, 1995, pp. 662–669.

[3] D. M. Nicol and W. Mao, "Automated Parallelization of Timed Petri Net Simulation", Journal of Parallel and Distributed Computing, to appear.

[4] D. M. Nicol and W. Mao, "On Bottleneck Partitioning of k-ary n-cubes", Parallel Processing Letters, to appear.

[5] D. M. Nicol and D. R. O'Hallaron, "Improved Algorithms for Mapping Parallel and Pipelined Computations", IEEE Trans. on Computers, Vol. 40, 1991, pp. 295–306.