J. Parallel Distrib. Comput. 96 (2016) 12–26

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Co-optimizing application partitioning and network topology for a reconfigurable interconnect

Deepak Ajwani[a,*], Adam Hackett[b], Shoukat Ali[c], John P. Morrison[d], Stephen Kirkland[b,1]

[a] Bell Labs, Alcatel-Lucent, Dublin 15, Ireland
[b] Hamilton Institute, National University of Ireland Maynooth, Ireland
[c] Elastica, San Jose, CA, United States
[d] The Centre for Unified Computing, University College Cork, Cork, Ireland

HIGHLIGHTS

- We give an algorithm to compute topologies of reconfigurable interconnect systems.
- We optimize, for reconfigurable networks, topology, routing, and task partitioning.
- We compute a high-throughput seed topology from structural properties of the task.

ARTICLE INFO

Article history:
Received 25 June 2013
Received in revised form
15 February 2016
Accepted 21 April 2016
Available online 28 April 2016

Keywords:
Network configuration algorithm
Reconfigurable interconnect topology
Optical circuit switch
Topology-aware graph partitioning
Stream-computing

ABSTRACT

To realize the full potential of a high-performance computing system with a reconfigurable interconnect, there is a need to design algorithms for computing a topology that will allow for a high-throughput load distribution, while simultaneously partitioning the computational task graph of the application for the computed topology. In this paper, we propose a new framework that exploits such reconfigurable interconnects to achieve these interdependent goals, i.e., to iteratively co-optimize the network topology configuration, application partitioning and network flow routing to maximize throughput for a given application. We also present a novel way of computing a high-throughput initial topology based on the structural properties of the application to seed our co-optimizing framework. We show the value of our approach on synthetic graphs that emulate the key characteristics of a class of stream computing applications that require high throughput. Our experiments show that the proposed technique is fast and computes high-quality partitions of such graphs for a broad range of hardware parameters that varies the bottleneck from computation to communication. Finally, we show how using a particular topology as a seed to our framework significantly reduces the time to compute the final topology.

1. Introduction

Optical circuit switches have recently been proposed as a low-cost, low-power and high-bandwidth alternative in the design of high-performance compute clusters (e.g., [41,27,4,39]). At the same time, these switches allow users to configure the network topology to suit the requirements of the application.

The option of configuring the interconnect opens up new possibilities for improvement in topology-aware graph partitioning approaches. Instead of asking the question "given an application graph G, how would you partition it on a set of compute nodes connected in topology H?" we are wondering "given an application graph G, how would you best interconnect the compute nodes to elicit the best possible partitioning and routing of G on the interconnect topology?" This research addresses this question by formulating an iterative strategy for co-optimizing the partitioning of the application graph and the configuration of the network topology.

There are two constraints that further complicate this issue. In a real system, a compute node has only a fixed number of ports to connect to the reconfigurable switch. Secondly, the reconfigurable switch has a limit on the maximum number of simultaneous links that it can maintain. Therefore, as an unavoidable part of our strategy, we also try to minimize the maximum traffic on the

* Corresponding author.
E-mail address: deepak.ajwani@alcatel-lucent.com (D. Ajwani).
1 Current address: Department of Mathematics, University of Manitoba, Winnipeg, Manitoba, Canada.

http://dx.doi.org/10.1016/j.jpdc.2016.04.010
0743-7315/© 2016 Elsevier Inc. All rights reserved.

interconnect while satisfying the above two constraints. Because our framework attempts to co-optimize topology configuration, application partitioning and interconnect routing, we refer to it as TPR co-optimizing framework.

Our approach is not tied to a particular communication pattern within the application. In our experimental results, we show performance gains for thousands of application graphs randomly selected, (with random communication patterns) from within the class of stream computing applications. Please note that our algorithm works for general graphs, even though our experiments are done on class of graphs that emulate stream computing applications.

The rest of this paper is structured as follows. We present the notations, definitions, the problem addressed and our key contributions in Section 2. Section 3 describes our main framework together with all the details of the individual steps. Our experiments with this framework are reported in Section 4. In Section 3.2, we show that seeding the framework with a good initial topology and then conducting a restricted search around it significantly reduces the time to compute the final topology. We describe some related work in Section 5 and conclude with future research directions in Section 6.

2. Preliminaries

2.1. Notations

We refer to application graph as $G(V_G, E_G)$ (or simply G) and to avoid tedious notation, also use the same notation for contracted application graphs. The notation $H(V_H, E_H)$ (or simply H) is used to refer to the topology graph. The elements of V_G are referred to as vertices while elements in V_H are referred to as nodes or compute nodes. The notation N_P denotes the total number of processors in the supercomputer. Since the nodes in the topology graph correspond to the actual compute nodes in the architecture, we have $N_P = |V_H|$ (although they need not all be connected or have some computation load). We are interested in the mapping of vertices in application graph to nodes in the topology graph. The weight of a vertex or node u is denoted by $w_v(u)$, while the weights on an edge e of either the application graph or topology graph is referred as $w_e(e)$.

2.2. Problem definition

We are given a computational task graph $G(V_G, E_G)$ where the vertices denote computational kernels and the edges capture the dependencies between the different computational kernels. The weights on vertices denote the average amount of computation that needs to be performed at the corresponding kernel to produce one element of output. Similarly, the weight on an edge represents the average amount of data transfer between the kernels (corresponding to the two incident vertices) to produce one element of output.

We assume that the compute nodes in the high-performance system are identical with the same processing speed (hereafter denoted by S_{comp}). These compute nodes are connected through a reconfigurable switch, which can alter the topology to suit the application. We also assume that bandwidth on all links connected through the reconfigurable switch is identical (denoted by S_{comm}).

In order to run the application on the system, we need to map each vertex $v \in V_G$ to a compute node and route each edge $e \in E_G$ along some path in the network topology. Let $\mu(v)$ be a mapping that specifies the compute node to which a particular vertex v is mapped. Let $\rho(e)$ be the sequence of communication links that are used to route an edge e in E_G. Given such a mapping and a routing scheme, the computation load on a compute node $P_i \in V_H$ is $w_v(P_i) = \sum_{(u \in V_G) \wedge (\mu(u) = P_i)} w_v(u)$ and the communication load over a link $e \in E_H$ is $w_e(e) = \sum_{(e' \in E_G) \wedge (e \in \rho(e'))} w_e(e')$. Since all computation over the nodes and communication over the links happen concurrently, the throughput is constrained by the slowest element. We define the throughput of a node P_i to be $S_{\text{comp}}/w_v(P_i)$ and the throughput of a link e to be $S_{\text{comm}}/w_e(e)$. The compute throughput of the system is the minimum throughput of a node and the communication throughput of the system is the minimum throughput over a link. The throughput generated by the overall system is the smaller of the compute throughput and the communication throughput. Note that our definition of throughput arises out of stream computing applications, where we view the compute nodes and communication links as processing units running concurrently so that the overall throughput is equal to the throughput of the slowest processing unit (similar to the throughput of a fetch–decode–execute pipeline where fetch, decode and execute stages run concurrently). Nonetheless, other problem-specific definitions can be used (with an accompanying change to the performance vector in Section 3.3.3).

In a real system, a compute node has only a fixed number of ports to connect to the reconfigurable switch. Let this constraint be called the max-degree constraint, denoted as $\Delta_{\max}$. Also, the reconfigurable switch has a limit on the maximum number of simultaneous links that it can maintain. We refer to this limit as max-edges constraint, denoted as $E_{\max}$. Thus, the switch can configure any topology that satisfies the constraints that maximum degree in the topology is no more than $\Delta_{\max}$ and the total number of links is not more than $E_{\max}$. Note that these constraints on the space of configurable topologies are very natural and can easily arise in many other applications. Our goal is three-fold. (a) Compute a network topology graph H that is likely to elicit a high throughput mapping for the application graph G. (b) Compute a mapping of vertices in V_G to nodes in V_H to achieve a high computation throughput. (c) Compute a routing scheme for edges in E_G to communication links in H so as to minimize congestion and thereby provide high communication throughput.

A good topology is one that allows a mapping and a routing scheme to yield a high throughput (ideally close to the optimum). Since the definition of a good topology depends on the difficult problems of computing good mapping and routing schemes, it is not easy to compute. We therefore propose a framework where we derive a good initial topology based on the structural properties of the application graph and then iteratively improve this topology by performing local modifications.

Note that although the connections created by optical switch are directed in nature, we treat them as undirected. This is because engineers invariably pair these optical cables to keep the routing protocols simple. Often, the two optical fibers in an optical cable are used for making the data-transfer bidirectional.

2.3. Key contributions

Our key contributions are as follows.
(1) We propose a new framework that iteratively co-optimizes the discovery of a good network topology that is configurable within the constraints of the reconfigurable interconnect, and the computation of a good partitioning, mapping and routing solution for deploying the application on the computed topology to maximize throughput. In contrast to the existing literature on graph partitioning, mapping and routing, our problem is challenging because we are also optimizing for the topology while computing the mapping of an application graph to the topology. Thus in our framework, the topology is iteratively altered to alleviate the computation and communication bottlenecks that are identified with the mapping solution.