Parallel Computing 48 (2015) 108–124

Contents lists available at ScienceDirect

Parallel Computing

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

A data-driven paradigm for mapping problems

Peng Zhang[a,*], Ling Liu[b], Yuefan Deng[c,d]

[a] Biomedical Engineering Department, Stony Brook University, NY 11794-8151, United States
[b] Marine and Atmospheric Science Department, Stony Brook University, NY 11794-5000, United States
[c] Applied Mathematics Department, Stony Brook University, NY 11794-3600, United States
[d] National Supercomputer Center in Jinan, Shandong 250101, China

ARTICLE INFO

Article history:
Received 30 September 2013
Revised 13 January 2015
Accepted 5 May 2015
Available online 13 May 2015

Keywords:
Data mapping
Task mapping
Parallel computing
Data movement matrix

ABSTRACT

We present a new data-driven paradigm for solving mapping problems on parallel computers. This paradigm targets at mapping data modules, instead of task modules, onto multiple processing cores. By dependency analysis of data modules, we devise a data movement matrix to reduce the need of manipulating task program modules at the expenses of handling data modules. To visualize and quantify the complex maneuver, we adopt the parallel activities trace graphs introduced earlier. To demonstrate the procedure and algorithmic values of our paradigm, we test it on the Strassen matrix multiplication and Cholesky matrix inversion algorithms. Mapping tasks has been more widely studied while mapping data is a new approach that appears to be more efficient for data-intensive applications that are becoming prevalent for today's parallel computers with millions of cores.

Published by Elsevier B.V.

1. Introduction

Cellular networks such as torus and mesh interconnects [1–5] are deeply exploited for the communication sub-systems to accommodate the ever-increasing needs of coupling millions of processing cores in most of today's advanced parallel computers [1,6]. The growing complexities of such communication sub-systems coupled with intensified performance requirements of such sub-system make it a serious challenge to design the state-of-the-art communication sub-system and the programming model for handling large data sets [7–15]. The programming model [16,17] currently adopted in many applications are facing tight bottlenecks even at the parallelism for tens of thousands of processes. For emerging supercomputers with millions of cores, we need to develop new programming paradigms and, unfortunately, untangling and streamlining the processes of non-uniform remote memory accesses, heterogeneous I/O, and numeral computation for such large-scale systems [17] is a daunting task. For a small aspect, researchers have studied the task mapping paradigm (TMP) to map task modules onto processing cores for minimizing unnecessary communications and processes idling [7,18] and then tested on various application for many parallel computers demonstrated significant performance improvement [8–15,19].

TMP is generally formulated in the graph theoretic terms [7,14], in which both the application requirement and the network information are represented in graphs. The parallel application is represented as a directed acyclic graph (DAG), in which a vertex is a compute task module that encapsulates a set of operations with a specific data set. The network is represented as a DAG whose vertex represents a processor and the edge represents the direct link between two adjacent processors. The task mapping seeks the map for mapping the task graph onto the network graph, i.e., assignment of task modules onto processors. The objective of the task mapping is to essentially reduce communication cost by strategically placing task modules on the appropriate

* Corresponding author. Tel.: 6318890809.
E-mail address: Peng.Zhang@StonyBrook.edu, pzhang99@gmail.com (P. Zhang).

http://dx.doi.org/10.1016/j.parco.2015.05.002
0167-8191/Published by Elsevier B.V.

processors, while balancing the compute load on processors. For this, the hop-bytes metric is usually used for measuring the solution quality of the mapping algorithm [7,9,10,12–14,18].

The TMP formulation, naturally, concentrates on optimizing communications by manipulating the placement of task modules and rather than data modules. From the data perspective, we study a new paradigm for the mapping problem that describes an algorithm in this paper. This paradigm substitutes the data modules for the task modules and the data movement for the inter-task communication and dependency. Our new data-driven mapping paradigm (DMP) seeks the best assignment of data modules to processor cores to minimize the total execution time including local computation and data availability as well as idling.

This paper is organized as follows: the basic concepts and formulism of data-driven mapping paradigm are discussed in Section 2 in which the parallel activities trace graphs are devised for help decouple the computation and communication activities. Section 3 demonstrates the applications of the DMP to the Strassen's matrix multiplication and Cholesky matrix inversion algorithms. This section illustrates and compares the basic concepts of data and task dependency graphs and matrices. Discussions and conclusions are drawn in the last section.

2. Formulation

Like any numerical algorithms, parallel algorithms need to balance accuracy, efficiency, programming flexibilities among others. The data-driven mapping paradigm (DMP) we formulate for representing, managing and optimizing algorithms will demonstrate such balance and programmability. Section 2.1 introduces such a formalism while Section 2.2 shows the mechanism of the parallel activities trace (PAT) graphs for the parallel and highly tangled data movements.

2.1. Data-driven mapping paradigm

Definition 1. (data module): An algorithm is assumed to consist of a set of finite data modules $D = \{d_s\}$ $(1 \le s \le n)$ and task modules. A data module is the smallest unit of data that can be communicated as a whole packet over network and processed by a single processing core. If a data module exists before processing of the algorithm, it is referred to as an *initial* data module; otherwise, it is referred to as an *intermediate* data module. Clearly, expected results of an algorithm are a collection of intermediate data modules.

Definition 2. (data dependence): The generation of an intermediate data module d_s is assumed to depend on one or several data modules $d_{s1},\ldots, d_{sm}$ and the generation method f_s for d_s is specified in the algorithm: $d_s = f_s(d_{s1},\ldots, d_{sm})$. Each intermediate data module associates with one specific method. Then, d_s *depends* on $d_{s1},\ldots, d_{sm}$; d_s is a *dependent* on d_{si} and d_{si} is an *antecedent* of d_s where $i \in [1, m]$. The data dependence is always unidirectional. The initial data modules have no dependents while an intermediate data module may have multiple dependents and has at least one antecedent.

Definition 3. (data dependence graph and matrix): The data dependence is represented as a directed acyclic graph G_d (V_d, E_d). The vertices in V_d represent data modules and the edges in E_d represent direct dependence between data modules. If d_j is dependent on d_i, there is directed edge e_{ij} from d_i to d_j. The adjacent matrix of G_d is referred to as the *data dependence matrix* $A = [a_{ij}]_{n \times n}$ whose entry $a_{ij} = 1$ if and only if d_i is an antecedent of d_j and $a_{ij} = 0$ otherwise. The number of vertices representing intermediate data modules in DMP equals the number of atomic tasks in TMP.

Definition 4. (data relevance): If two data modules d_i and d_j are an antecedent of a same data module d_k, these two data modules d_i and d_j are *related to* each other through d_k. All of the antecedents of an intermediate data module d_k form a *relevant set* of d_k denoted as $\Omega_r(d_k)$.

The number of computing tasks in TMP can be calculated in the data dependency matrix A as shown by the following two theorems:

Theorem 1. Given a data dependency matrix $A = [a_{ij}]_{n \times n}$ of an algorithm, a relevant set of d_k is $\Omega_r(d_k) = \{d_k | a_{ik} = 1\}$.

Theorem 2. Given a data dependency matrix $A = [a_{ij}]_{n \times n}$ of an algorithm and let $K = \{k \in [1, n] \mid \Omega_r(d_k) \neq \Phi\}$, then the total number of computing tasks is $n - |K|$ where $|K|$ is the number of elements in the set K and Φ is the empty set.

Definition 5. (accomplishment criterion of an algorithm): It is always assumed in TMP that a processor executes a task that is assigned to it and thus an algorithm is accomplished after all tasks are executed. Similarly, in DMP, we assume that a processor executes a method $f_k(\Omega_r(d_k))$ for d_k as soon as it receives all of the data modules $d_{k1},\ldots, d_{km}$ in $\Omega_r(d_k) = \{d_{k1},\ldots, d_{km}\}$. Therefore, an algorithm is accomplished as long as every relevant set has been completely assembled on the processor.

We may assume, without losing generality, that any intermediate data module d_k depends on only two data modules, i.e., $|\Omega_r(d_k)| = 2$. Thus, an algorithm is accomplished as long as a pair of related data modules meets at a same processor.

Definition 6. (data movement matrix): As an algorithm is executed in parallel, a task needs to communicate with other tasks in TMP and, similarly, a data module must traverse the interconnection networks to join other relevant modules in DMP. Given k processors, p_1 through p_k, allocated for an algorithm with data modules $D = \{d_s\}$ $(1 \le s \le n)$, a matrix for directing data module movement between processors is denoted as $M = [m_{ij}]_{k \times n}$ whose entry $m_{ij} = 2$ iff d_j is initially mapped to p_i; $m_{ij} = 1$ iff p_i