Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 186 (2007) 1311–1321
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

On the Hamiltonian cycle mapping onto 3-D torus interconnection network based on base-b reflected gray codes

Saleh H. Al-Sharaeh

Department of Computer Science, King Abdallah II for Information Technology, Jordan University, Amman, Jordan

Abstract

In this paper we present a 3D large volume simulation decomposition and mapping technique onto a 3D torus interconnection network, based on base-b reflected gray codes. Such simulation leads to fast execution plus an improvement of total execution time. The minimization of the execution time of the simulation is due to minimization of packet routing in the 3D interconnection torus network. Applying the algorithm on real 3D space plasma simulation of the Aurora region of the Earth's Ionosphere was observed to be almost ideal speedup. The efficiency of the implementations was shown to be constant close to one, as the size of the 3D simulation increases.

Keywords: Parallel simulation; Speedup; Plasma; Hamiltonian; Gray codes

1. Introduction

High Performance Applications (HPC) are the most demanding of computing power. Due to perceived difficulties with programming in a multicomputing environment, interconnection topologies play a vital role [1–4]. The Massively Parallel Processing (MPP) Cray T3D/T3E architecture employs a 3D torus interconnection topology. Each Processing Element (PE) has local memory. Such memory is physically distributed, but it is logically shared because each PE can access any other PE memory without involving the microprocessor in that PE [5,6]. Cray Research Inc. introduced the first phase which has 256 processors followed by the T3E with 1024 processors, and recently Cray XD1 [6] a shared memory system. In this new generation of MPP based on 3D torus interconnection topologies, each node has 16M bytes that are physically distributed, but are logically shared. When a processor needs data at a certain address, which exists in another processor, that processor number combined with the address requested will formulate the effective address, and then a very fast search engine handles the communication without interrupting the computation processor. Latency can be effectively hidden in this manner by transparently overlapping communication with computation, which makes this new generation very effective and scalable.

E-mail address: salsharaeh@yahoo.com

0096-3003/$ - see front matter
doi:10.1016/j.amc.2006.07.163

The aforementioned characteristics of the T3D make this kind of architecture suitable to our application, and with a proper mapping methodology, the algorithm developed was ported to a 3-D torus (the Cray T3D interconnection topology). The method based on base-b reflected Gray code minimized the inter-processor communication delay.

2. Task assignment, mapping, and scheduling

The targeted hardware, Cray T3D, is made up of a set of processing nodes that are interconnected by some form of interconnection topology, in this case 3D Torus. In a homogenous environment, the processing elements all have the same computing capability, and more often than not, the network is regular in structure. In a heterogeneous environment, many of the computing engines differ from one another, the network may be very irregular in shape and the nodes may be dispersed over long geographical distances.

In general, the act of allocating the task graph to the target topology can be described in terms of three basic operations: assignment, scheduling, and mapping [5]. The assignment operation involves the clustering or partitioning of tasks into processing groups in a manner that ensures a one-to-one correspondence between the number of groups and the number of processing elements. Scheduling is the process of determining the order in which the tasks (and possibly communications) are to execute within each processing group. The mapping problem involves assigning processing groups to individual processing elements for execution, in a way that match the communication structure present between groups with the topology of the inter-processor communication network. Each of these three operations is NP complete and can occur in a static (preprocessing) or dynamic (run time) environment. For many algorithms, optimal allocations can be made statistically by carefully examining the structure of the algorithm. One technique that has been used successfully for such problems employs base-b reflected Gray codes [7,8].

3. Parallel computing terminology

It is important that one understands how to evaluate the time complexity of a given algorithm, which is often described using the big "O" notation. The definition of the Order Notation "O" is as follows [9]. Assume f and g are functions over the domain of the natural numbers. Then O($f(n)$) [read "order at most $f(n)$] is the set of all $g(n)$ such that there exist positive constants c and n_0 so that $|g(n)| \leqslant cf(n)$ for all $n > n_0$.

The complexity of a parallel algorithm such as the total sum of n numbers is a function of the problem size. In a parallel representation, the execution time is the maximum over all inputs of size n, of the time elapsed from when the first processor begins execution of its portion of the problem until the last processor terminates execution. Determining the sum of n values on a sequential computer has time complexity O(n). Porting the same algorithm on a parallel computer that has $n/2$ processors, the sum can be determined in order O($\log n$), if the partial sums are computed in a treelike fashion [10].

Let S be a single parameter, which represents the effective size of a given algorithm. Then $W(S)$ would be the total workload (i.e. total number of computational operations) that is required by the best sequential algorithm to solve the problem. The execution time of $W(S)$ on a single processor is then $T_1(S) = W(S)/\Delta$, where Δ is the computational rate or capacity of the processor. Restructuring the problem for parallelism requires that a portion of the original workload W_i be assigned to each of the i processors in the system, with $W(S) = \sum_P W_i(S)$. The mapping of the application problem onto P processors introduces extra overhead and sometimes-extra workload that does not exist in the best sequential case. Let O_i be the overhead function associated with processor i. This overhead function reflects the time that each processor in the system is not involved in useful computation. It includes the time spent performing redundant computations; the time spent in communication and the time the processor is idle. Thus, the total overhead for P processors and problem size S is equal to $h(S,P) = \sum_P O_i$. The parallel time $T(P)$ is then the time it takes the algorithm to run on P processors, and it is given by

$$T_P(S) = \frac{1}{P}\left[\sum_P \left(\frac{W_i(S)}{\Delta} + O_i\right)\right]. \tag{1}$$