Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 379 – 388

Procedia Computer Science

2013 International Conference on Computational Science

Topology Aware Task stealing for On-Chip NUMA Multi-Core Processors

B.Vikranth, Rajeev Wankar[1], C.Raghavendra Rao

b.vikranth@cvr.ac.in, CVR College Of Engineering, Hyderabad, India
wankarcs@uohyd.ernet.in, School Of Computers and Information Sciences, University Of Hyderabad, Hyderabad, India
crrcs@uohyd.ernet.in, School Of Computers and Information Sciences, Hyderabad, India

Abstract

"The On Chip NUMA Architectures (OCNA) introduce a new challenge namely memory-latency to the scheduling methods. The language run-times and libraries try to explore the processing power of these multiple cores by mapping the user-created tasks on to these cores by using suitable scheduling algorithms with load balancing support to improve throughput. The popular load balancing techniques used are work-sharing and work-stealing and many run-time systems such as Cilk, TBB and wool implement task stealing algorithm to schedule the tasks on to the cores by multiplexing the program generated tasks on to the native worker threads supported by the operating system. But the task stealing strategy applied in present run-time systems assumes the sharing the last level cache (LLC) and common shared bus among all cores on Chip Multi Processor. It tries to optimize the utilization without considering the presence of multiple On Die DRAM controllers and their topological arrangements. Current task stealing technique also suffers from problem of randomly choosing the victim worker queue. In this paper we address these issues and propose a solution for these problems by suggesting few optimizations. Our proposed task stealing strategy dynamically analyzes the topology of the underlying hardware connections and models the group of cores and connections as a logical topology tree. This logical tree is translated into multiple worker pools called stealing domains. By restricting the task stealing within these domains, this strategy is implemented and shows an average of 1.24 times better performance on NAS Parallel Benchmark programs compared to popular runtimes Cilk and OpenMP.

Keywords: multi-core, task stealing, work sharing, worker queue, load balancing, On Chip NUMA Multi Core.

1.Introduction

The modern trend in multi-core processors is to pack multiple groups of cores onto chip with point-to-point links using the latest communication technology such as QPI[11] links from Intel or Hyper Transport links from AMD. These groups of processors also integrate multiple on chip memory controllers to reduce the memory bandwidth problem and support scalability. So, modern many-core processors can be treated as On-Chip-NUMA Architectures (OCNA) where the memory address space is divided among multiple dies and

[1] * Corresponding author. Tel.: +91-9440-679-869; fax: +9140-23010780.
E-mail address: wankarcs@uohyd.ernet.in.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.201

sockets. In this context, it is a common observation that the work load distribution and data distribution between the various processors is often not uniform. A particular processor may get heavily loaded while the others are idle. To avoid this kind of non uniform work load distribution, a load balancing algorithm is commonly employed. The sender initiated algorithms generally employ a centralized dispatcher which is responsible for dispatching various processes to the available processors. This phenomenon is called as work-sharing. All sender initiated strategies work with this phenomenon.

On the other hand each of the processors would be maintaining their individual local queues of tasks assigned to them by the central dispatcher. If this queue gets exhausted, i.e. if a processor completes execution of all of its assigned tasks then it is in an idle state becomes a thief. These architectures which are following receiver initiated algorithms would implement a technique where the thief-worker would steal tasks from the victim worker and executes those tasks while making a decision on which processor to choose and which task to choose. This is called as task stealing. The method of choosing the victim is randomly done in the present run time systems. Randomly choosing the victim worker may cause two types of delays namely:

Steal miss: choosing the random victim worker queue may result a success or a failure. A failure may be resulted if the randomly chosen queue is also under loaded or empty.

Remote steal: The second possibility is the randomly chosen run-queue that may belong to the other socket or other group of cores DRAM in which case, the stealing of task results task migration and data migration. In this paper, we analyze and address these issues and propose a new method to improve it.

Nomenclature

A	Worker: is a native thread which is part of worker pool. The number of workers generally is equal to the number of cores in the system.
B	Thief: A worker becomes a thief when it has no work to do. Therefore, its run-queue is empty. It is eligible to steal task from other run queues.
C	Victim: A worker becomes a victim when its task queue reaches a threshold value level.

1.1.Structure

The rest of the paper is organized in the following manner. In section 2 the related work is covered and in section 3 the problems identified in existing task stealing strategy are discussed. Section 4 contains the detailed description of the proposed algorithm and the result analysis is presented in section 5.

2. Motivation

Task stealing algorithm has been enhanced by various optimization strategies. [9, 12, 13] address the issues based on profiling data which require the tools to capture it. Theses profile based strategies do not consider the topology of core and memory interconnections. In contrast to this, we propose a topological aware, non-profile and dynamic task stealing strategy. This proposed strategy is based on capturing the topology of underlying hardware which does not cause any overhead since building topology tree is one time process and that too during the initialization of the library. This optimized task stealing strategy can be directly opted into user-level run-time libraries such as Cilk[4], OpenMP and TBB[14]. To the best of our knowledge, the work proposed in the paper provides better solution for On Chip NUMA Multi Core runtime environments.