The Performance Impact of Granularity Control and Functional Parallelism*

José E. Moreira† Dale Schouten‡ Constantine Polychronopoulos‡
moreira@watson.ibm.com {schouten, cdp}@csrd.uiuc.edu

† IBM T. J. Watson Research Center
Yorktown Heights, NY 10598-0218
‡ Center for Supercomputing Research and Development,
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St. Urbana, IL 61801-2307

Abstract. Task granularity and functional parallelism are fundamental issues in the optimization of parallel programs. Appropriate granularity for exploitation of parallelism is affected by characteristics of both the program and the execution environment. In this paper we demonstrate the efficacy of dynamic granularity control. The scheme we propose uses dynamic runtime information to select the task size of exploited parallelism at various stages of the execution of a program. We also demonstrate that functional parallelism can be an important factor in improving the performance of parallel programs, both in the presence and absence of loop-level parallelism. Functional parallelism can increase the amount of large-grain parallelism as well as provide finer-grain parallelism that leads to better load balance. Analytical models and benchmark results quantify the impact of granularity control and functional parallelism. The underlying implementation for this research is a low-overhead threads model based on user-level scheduling.

Keywords: dynamic scheduling, functional parallelism, task granularity, parallel processing, threads.

1 Introduction

The magnitude to which runtime overhead affects performance has been widely demonstrated [2, 3, 12]. In order to alleviate this problem [12] and other subsequent studies provided an environment that allows the user to control the number of parallel tasks a given parallel application generates. Given a fixed number of resources, a user or compiler can restrict the maximum number of parallel tasks of a parallel application to less than or equal to a predetermined amount.

* This work was supported by the Office of Naval Research under grant N00014-94-1-0234. Computational facilities were provided by the National Center for Supercomputing Applications. José Moreira was at the University of Illinois during the development of this research.
This paper reports on an implementation which employs the notion of dynamic granularity control. At any given time, the number of parallel activities a process generates is proportional to the number of physical resources allocated to that process. This allows the operating system to dynamically allocate a varying number of processors to different processes. In fact, the number of processors allocated to a particular process may vary over its lifetime.

The immediate impact of granularity control is the elimination of unnecessary overhead due to frequent context switching, creation and scheduling of tasks, additional interprocessor communication, and increased memory latency. Our method relies on a program representation which encapsulates the hierarchy of computations inherent in a parallel application. This allows for parallelism to be exploited first at the highest level of this hierarchy which corresponds to the outermost loops and the first-level function calls. Subject to resource availability, inner levels of parallelism are exploited by decomposing nested parallelism. A related focus of this work is the performance implications of the exploitation of functional (nonloop) parallelism. Our experiments indicate that functional parallelism can improve performance by a significant margin, even in situations where data (loop) parallelism is in abundance.

This paper is organized as follows: Section 2 describes the programming model and target machine architecture. Section 3 describes an autoscheduling threads model, nanoThreads. Queue management and granularity control issues are addressed in Sections 4 and 5 respectively. The environment used for our measurements is described in Section 6. An analytical model showing the benefits of the exploitation of functional parallelism and experimental results from synthetic benchmarks are presented in Section 7. The set of benchmarks used for more general measurements is listed in Section 8, and the results from these measurements are shown in Section 9. Finally, related work is discussed in Section 10 and concluding remarks are given in Section 11.

2 Machine and Programming Model

The target machine model is a shared address space multiprocessor with a multiprogramming environment. Therefore only a subset of the machine's processors will be allocated to a particular program. We call this subset of processors a partition and let this partition be time-variant, meaning that processors may be added or removed by the operating system during the execution of the job.

The program model is the hierarchical task graph [10], or HTG, combined with autoscheduling [19]. The HTG is an intermediate program representation that encapsulates data and control dependences at various levels of task granularity. This structure is used to generate autoscheduling code, which includes the scheduling operations directly within the program code. The HTG represents a program in a hierarchical structure, thus facilitating task-granularity control. Information on control and data dependences allows the exploitation of functional (task-level) parallelism, in addition to data (loop-level) parallelism. A brief summary of the properties of the HTG is given here and details can be found in [3, 8, 9, 10, 19].

The hierarchical task graph is a directed acyclic graph $HTG = (HV, HE)$ with unique nodes START and STOP $\in HV$, the set of vertices. Its edges, $HE$, are a union of