Journal of King Saud University – Computer and Information Sciences (2017) 29, 85–92

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

Register allocation for fine grain threads on multicore processor

D.C. Kiran [a,*], S. Gurunarayanan [b], Janardan P. Misra [a], Munish Bhatia [a]

[a] Department of Computer Science and Information Systems, Birla Institute of Technology and Science Pilani, 333031 Rajasthan, India
[b] Department of Electrical Electronics and Instrumentation, Birla Institute of Technology and Science Pilani, 333031 Rajasthan, India

Received 13 October 2014; revised 3 April 2015; accepted 14 April 2015
Available online 23 November 2015

KEYWORDS

Multicore;
Compiler;
Fine grain parallelism;
Scheduling;
Register allocation

Abstract A multicore processor has multiple processing cores on the same chip. Unicore and multicore processors are architecturally different. Since individual instructions are needed to be scheduled onto one of the available cores, it effectively decreases the number of instructions executed on the individual core of a multicore processor. As each core of a multicore processor has a private register file, it results in reduced register pressure. To effectively utilize the potential benefits of the multicore processor, the sequential program must be split into small parallel regions to be run on different cores, and the register allocation must be done for each of these cores. This article discusses register allocating heuristics for fine grained threads which can be scheduled on multiple cores. Spills are computed and its effect on speed-up, power consumption and performance per power is compared for a RAW benchmark suite.

1. Introduction

Coupled with technological advancement in the field of computer architecture and relentless demand for faster processing has led to the development of multicore processors. A multicore processor has multiple processor cores on same processor chip. Each individual core has a separate register file and is capable of executing complete Instruction Set Architecture (ISA). In order to exploit the capabilities of multicore processors, a significant amount of research in the area of code parallelization and multiprocessing has been carried out. An application running on a multicore system does not guarantee the performance improvement until the application has been explicitly designed to take the advantage of multiple cores present on the processor chip. To develop an application that exploits multicore, predominantly two approaches are followed. The first approach is to develop an explicitly parallel code that can be scheduled on multiple cores of a given processor and the other approach is using a compiler to extract fine grained parallelism by identifying the sets of instructions that can be executed in parallel. Currently, several new programing

* Corresponding author.
E-mail addresses: dck@pilani.bits-pilani.ac.in (D.C. Kiran), sguru@pilan.bits-pilani.ac.in (S. Gurunarayanan), jpm@pilan.bits-pilani.ac.in (J.P. Misra), munish.bhatia.cse@gmail.com (M. Bhatia).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2015.04.001

models and different ways to exploit threads and data-level parallelism are being explored which help in coarse grained parallelism. There is very little effort from the research community toward the exploitation of compiler driven fine grained parallelism in a sequential program.

The multicore processors can be made to exploit fine grained parallelism of a given code by exposing the low level architectural details to the compiler and operating systems (Zhong, 2008). The architecture can be designed to support the minimal set of operations required for executing an instruction and the task of extracting the fine grained parallelism can be left for compilers and run time environment to achieve. The runtime environment can manage resource allocation, extracting parallel constructs for different cores, and scheduling based on information generated by the compiler. Some of the architectures supporting these features are Power4 (Tendler et al., 2002), Cyclops (Cascaval et al., 2002) and RAW (Waingold et al., 1997) architecture. The multicore environment has multiple interconnected tiles and on each tile there can be one RISC like processor or core. Each core has instruction memory, data memory, PC, functional units, register files, and source clock. FIFO (queue) is used for communication. Here the register files are distributed, eliminating the small register name space problem. The challenge in achieving a performance gain from fine-grain parallelism is identification of the fine grained thread from a given single threaded application and scheduling these threads on different cores of the multicore processor. There has been considerable focus on improving performance through automated fine-grain parallelization, where a sequential program is split into parallel fine grained threads and are scheduled onto multiple cores (Kiran et al., 2011a,b; Kiran et al., 2012). In general, the multicore processors have a private register file, L1 data and Instruction cache and shared L2 cache. The limited size of L1 data cache, warrants the optimal amount of data to be brought into the cache. The poor choices in the placement of data can lead to the increased memory stalls and low resource utilization. The fine grained threads that are scheduled onto different cores need to be allocated registers from a respective register file of the core on which they are scheduled.

Various register allocation approaches are proposed in the past (Chaitin, 1982; Norris and Pollock, 1994; Gupta et al., 1994; Callahan and Koblenz, 1991; Lueh et al., 2000; Chow and Hennessy, 1984; Briggs et al., 1989; Poletto and Sarkar, 1999; Mossenbock and Pfeiffer, 2002; Fu and Wilk, 2002; Burkard et al., 1984; Todd et al., 1996). Most of the register allocation algorithms assume that the CPU has regular register file and these algorithms fail to adapt themselves for irregular architectures. Several solutions have been proposed for irregular architectures, but without considering the specific implementation details, it is difficult to achieve optimal register allocation (Koes and Goldstein, 2005; Kong and Wilken, 1998; Scholz and Eckstein, 2002). In the case of a multicore processor, each core of the processor has an individual register file and optimal register allocation is of utmost importance. Multicore architecture is one area which expects new thinking for register allocation. The proposed work explores the various likely steps needed for register allocation for multicore architecture.

This paper proposes two register allocation heuristics referred as heuristic 3 and heuristic 4 for fine grained threads which can be scheduled on multicore processor. Results are compared with heuristic 1 and heuristic 2 which are existing register allocation approaches. The proposed register allocation heuristics along with considering multiple private register files on each core, constructs the interference graph incrementally by checking the register pressure as opposed to existing register allocation approaches which construct the global interference graph and then perform simplification to reduce register pressure.

The rest of the paper is organized as follows. Section 2 describes background of the proposed work. It also discusses some of the recent works pertaining to sub-block creation or fine grained thread extraction and scheduling techniques. Section 3 gives a detailed description of the proposed register allocation technique for multicore environment. Through an illustrative example the steps involved in the proposed algorithm is presented in Section 3.4. Analysis and discussion of the results are presented in Section 4, Section 5 presents the conclusion and direction for future work.

2. Background

This section introduces the background of the proposed work. The proposed work performed in conjunction with following work.

- Parallel region formation or extracting fine grain threads (Kiran et al., 2011a).
- Scheduling parallel regions or fine grain threads on to multiple cores (Kiran et al., 2011b, 2012).

The work flow of the compiler is shown in Fig. 1. Two additional passes are introduced into the normal flow of the compiler. The Fine grained extractor module and the scheduler module. The Fine grained extractor module analyzes the basic blocks of CFG and divides each of them into multiple sub-blocks. The scheduler module generates the multiple schedules which can be concurrently executed on different cores of the processor. The register allocator module carries out the register assignment operation using Chaitin's register allocation approach (Chaitin, 1982).

2.1. Fine grain thread

A fine grain thread is a sub-block formed by analyzing the instruction dependency in the basic block of the control flow graph (CFG) of a program Kiran et al., 2011a. The fine grain thread extractor module in Fig. 1 creates sub-blocks. The sub-blocks created are disjoint and can run in parallel. In Fig. 3, the CFG has 4 basic blocks (B_p). The disjoint set operations

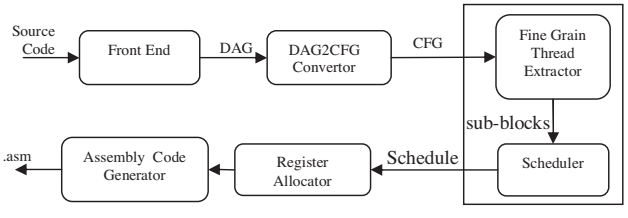

Figure 1 Flow of compiler.