A Partitioning Programming Environment for a Novel Parallel Architecture

R. Hartenstein, J. Becker, M. Herz, R. Kress, U. Nageldinger

Universitaet Kaiserslautern
Erwin-Schroedinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, e-mail: abakus@informatik.uni-kl.de

Abstract

The paper presents a partitioning and parallelizing programming environment for a novel parallel architecture. This universal embedded accelerator is based on a reconfigurable datapath hardware. The partitioning and parallelizing programming environment accepts C-programs and carries out both, a profiling-driven host/accelerator partitioning for performance optimization in a first step, and in a second step a resource-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable resources.

1. Introduction

From empirical studies [23] it can be concluded that the major amount in computation time is due to rather simple loop constructs. Studying DSP and image preprocessing algorithms on a von Neumann platform we found examples where up to 94% of the execution time elapsed for address computation overhead [1]. Parallelizing compilers have been developed, where compilation is based on loop transformations [3], [26]. But the hardware structures are not reflecting the structure of the algorithms very well, which substantially restricts the exploitation of inherent parallelism.

Custom Computing Machines (CCMs: [11], [18]) use application-specific hardware for speed-up. FPGA-based CCMs (FCCMs [4] [5]) use field-programmable hardware as a general purpose accelerator co-processor. But application development here usually requires hardware experts, due to the lack of a suitable general paradigm. The von Neumann paradigm does not efficiently support "soft" hardware because of its extremely tight coupling between instruction sequencer and ALU: processor architectures fall apart, as soon as the data path is changed. So another paradigm is desirable like the one of Xputers, which supports "soft" ALUs like the rALU concept (reconfigurable ALU) [7] or the rDPA approach (reconfigurable data path array) by Rainer Kress [14] [15].

For this new class of hardware platforms a new class of compilers is needed, which generate both, sequential and structural code: i. e. partitioning compilers. This paper introduces a parallelizing compilation method with two levels of partitioning: host/accelerator partitioning and a structural/sequential partitioning (second level).

First the target hardware is briefly summarized. Section 3 explains the partitioning and parallelizing programming environment targeting conventional C programs onto a host using an Xputer as universal configurable accelerator.

2. The Target Hardware

The target hardware consists of a host workstation that uses the Xputer [6] [7] [8] as a universal hardware accelerator (see figure 1). With Xputers the highest speed-up is obtained for algorithms, which iterate the same set of operations over a large amount of data. Such operations are mapped onto a reconfigurable parallel arithmetic-logic unit (rALU). All input and output data to the complex rALU operations is buffered in a scan window (SW), which is a kind of smart register file, the location sequence of which in memory space is determined by a data sequencer. All data in the scan windows can be accessed in parallel by the rALU.

Each of up to seven Xputer modules run independently from each other and the host until a task is completed. Reconfigurations can be performed independently from other running tasks. Each module generates an interrupt to the host when the task is finished. The host may run concurrently to such tasks. The basic structure of an Xputer module consists of three major parts (Figure 1):

- two-dimensionally organized data memory
- reconfigurable arithmetic & logic unit (rALU)
- reconfigurable data sequencer (DS)

Within iterations the access sequence to data, typically organized in arrays, often shows regularity which allows to describe address sequences generically from a few parameters (see section 3.2). The sequence of memory locations (examples in figure 6) reached by such generic address sequences (GAS) we call scan patterns (SP). By this basic sequencing mechanism Xputers are deterministically data-driven (non-von Neumann): i. e. one or multiple (generic address generators) GAGs are used instead of an instruction sequencer, generating addresses only by hardware without eating memory cycles. During a SP operation execution is