Microprocessors and Microsystems 38 (2014) 788–802

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Parallel distributed scalable runtime address generation scheme for a coarse grain reconfigurable computation and storage fabric

Nasim Farahini [a,*], Ahmed Hemani [a], Hassan Sohofi [a], Syed M.A.H. Jafri [a], Muhammad Adeel Tajammul [a], Kolin Paul [b]

[a] Royal Institute of Technology, KTH, Sweden
[b] Indian Institute of Technology Delhi, India

ARTICLE INFO

Article history:
Received 24 December 2013
Revised 12 May 2014
Accepted 23 May 2014
Available online 11 June 2014

Keywords:
Streaming address generation
CGRA
Parallel distributed DSP
Code compaction

ABSTRACT

This paper presents a hardware based solution for a scalable runtime address generation scheme for DSP applications mapped to a parallel distributed coarse grain reconfigurable computation and storage fabric. The scheme can also deal with non-affine functions of multiple variables that typically correspond to multiple nested loops. The key innovation is the judicious use of two categories of address generation resources. The first category of resource is the low cost AGU that generates addresses for given address bounds for affine functions of up to two variables. Such low cost AGUs are distributed and associated with every read/write port in the distributed memory architecture. The second category of resource is relatively more complex but is also distributed but shared among a few storage units and is capable of handling more complex address generation requirements like dynamic computation of address bounds that are then used to configure the AGUs, transformation of non-affine functions to affine function by computing the affine factor outside the loop, etc. The runtime computation of the address constraints results in negligibly small overhead in latency, area and energy while it provides substantial reduction in program storage, reconfiguration agility and energy compared to the prevalent pre-computation of address constraints. The efficacy of the proposed method has been validated against the prevalent address generation schemes for a set of six realistic DSP functions. Compared to the pre-computation method, the proposed solution achieved 75% average code compaction and compared to the centralized runtime address generation scheme, the proposed solution achieved 32.7% average performance improvement.

1. Introduction

Address generation is a dominant factor in the latency, area and energy cost of the DSP applications and other scientific applications dominated by vector processing; it can be as high as 70% in applications like image processing after loop transformations and memory hierarchy optimizations [1]. Parallel distributed coarse grain reconfigurable fabrics are emerging as promising architectural solutions that provides hardware like efficiency while retaining the benefits of programmability. However, the main focus of research in this field has been to speed up the functional computation with the efficiency of address generation receiving some but less than deserved attention [2–4]. We argue that the overall computational and silicon efficiency benefits of the hardware like parallel distributed solutions would be diluted unless these solutions also provide a matching parallel distributed address generation scheme. Providing such a scheme for a coarse grain reconfigurable computation and storage fabric is the focus of this paper.

Addresses in DSP applications are almost always functions of loop indices that in turn are constrained by the loop bounds. Loop bounds can themselves be compile time static or be functions of other loop indices and bounds. Further, these functions for addresses and loop bounds can be affine or non-affine. To meet these address generation requirements DSP architects have adopted two broad categories of solutions, as shown in Fig. 1.

One is to pre-compute the addresses and the loop bounds at compile time; this solution is also known as loop unrolling and has been employed by many coarse grain reconfigurable architectures [5,6]. Note that in this context, loop unrolling is mentioned as a technique to pre-compute the addresses and address bounds in each instance of the unravelled loop and not the technique used

* Corresponding author.
E-mail addresses: farahini@kth.se (N. Farahini), hemani@kth.se (A. Hemani), sohofi@kth.se (H. Sohofi), jafri@kth.se (S.M.A.H. Jafri), tajammul@kth.se (M.A. Tajammul), kolin.paul@gmail.com (K. Paul).

http://dx.doi.org/10.1016/j.micpro.2014.05.009
0141-9331/

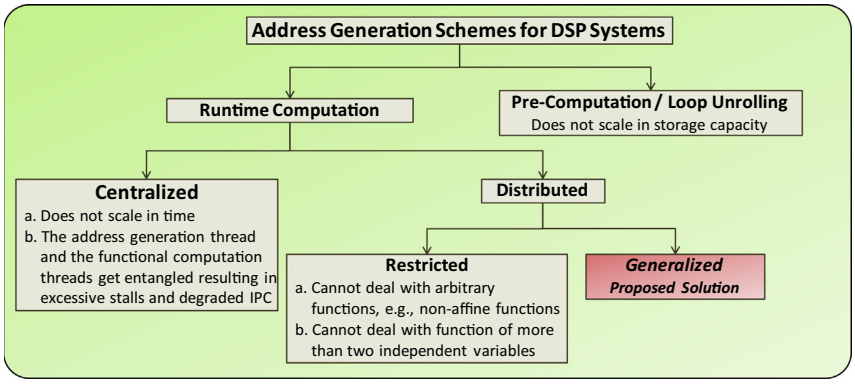

Fig. 1. Categories of Address Generation Schemes.

in the compiler optimization context where it is used to increase parallelism and utilization. This solution does not scale in space, as the storage required would increase with the outer loop bounds and the level of loop nesting. This is especially problematic for true parallel distributed solutions where not just the computation but also the control is distributed. Distributed control implies distributed program storage and if each such program storage unit needs to be dimensioned to cope with loop unrolled program, the overall cost of program storage would be unreasonably high [7].

The second solution is to compute addresses and loop bounds at runtime. Address Generation Units (AGUs) embedded with the storage units have long been used in DSP systems [8] but are often restricted to one or two dimensional affine addressing. AGUs are programmed with the loop bounds that are computed by the main computation unit. In another variant of address generation used in modern VLIWs, some ALUs (issues) are reserved for address and loop bound computation, working as a complement to the AGUs [9].

The problem with this approach is that the state machines for functional, address and loop bound computation may not be compatible and would result in large number of stalls and degrade the IPC (Instruction per Cycle) as addressed in [10]. Yet another variant to the runtime computation of addresses and loop bounds that is commonly adopted by the software centric solutions that rely on accelerators to speed up one or more inner loops is to centrally compute the loop bounds and/or addresses by the software processor and distribute them to the accelerators [11,12]. This approach is also not scalable as the centralized computation of multiple address and loop bounds can potentially become a bottleneck, especially when the ambition is to not just accelerate a few inner loops but to map the complete DSP sub-system to a parallel distributed coarse grain reconfigurable fabric; in essence, the entire DSP sub-system, a modem or a codec, becomes an accelerator controlled by a system controller.

The address and loop bound computation scheme proposed in this paper is based on runtime and distributed computation, thus avoiding the space scalability problem of the pre-computation approach and the latency scalability problem of the centralized computing. Secondly, the proposed scheme is general and not restricted to affine functions. Lastly, the address and loop bounds computation is an independent thread that does not get entangled with the functional computation thread, thus avoiding the problem of stalls and IPC degradation observed in VLIWs.

The runtime nature of the address generation scheme serves as a code compaction mechanism compared to the pre-computation or the loop unrolling method. This benefit is amplified by the distributed nature of the scheme that minimizes the energy required to move address between the address computation unit(s) where it is generated and the storage units where it is consumed. As the proposed scheme targets a coarse grain reconfigurable fabric, the code compaction also enables agile and low power parallel distributed reconfiguration. The coarse grain reconfigurable fabric has an adaptive power management scheme where the degree of parallelism of an algorithm is decided at runtime depending on the deadlines and the resources available. For this scheme to be effective it is essential for the reconfiguration to be agile and low power and the proposed scheme makes a difference as we demonstrate and quantify it in Section 6.

The rest of the paper is organized as follows. In Section 2, we review the state of the art in address generation scheme and relate it to the proposed scheme. This is followed by Section 3 where we introduce the terminology and the core idea at an abstract level to emphasize the generality of both the requirement and the proposed solution. Section 4 presents the specific Coarse Grain Reconfigurable computation and storage fabric that has been enhanced with the proposed runtime address generation scheme. Section 5 elaborates the proposed scheme by first presenting the details of the Runtime Address Constraints Computation Unit (RACCU) followed by the strategy to map address generation requirements between AGUs and RACCUs. This section ends with a concrete example of how an application is mapped to the enhanced address generation scheme and its working and timing explained in terms of pseudo assembly code. In Section 6, we validate the efficacy of the proposed scheme by comparing it against the pre-computation approach and the centralized runtime address generation scheme. We also do cost benefit analysis in this section. Lastly, we summarize and draw conclusions.

2. Related work

DSP architects have long recognized the benefits of dedicated hardware resources called AGU (Address Generation Unit) for generating addresses as elaborated in Dake Liu's textbook on the topic [32]. AGUs work in parallel with the main computational unit involved in functional computation. Talavera et al. [36] have comprehensively classified the address generation requirements and hardware. Commercial DSPs like TI TMS320C55x [13] and AT&T 16xx [34] have AGUs that supports commonly used DSP addressing modes like bit-reverse addressing, circular buffer etc. These commercial DSPs also support what is known as zero overhead loops with the auto-increment/decrement features connected to the address generation hardware. To go beyond the standard address generation schemes, e.g., with non-affine or non-linear functions or compute address bounds dynamically these DSPs that are typically VLIWs adopt one of the three following strategies. One is to pre-compute or unroll the loop. The pre-computation/loop unrolling strategy, as argued before does not scale in space. The second