Microelectronics Journal 62 (2017) 57–64

Contents lists available at ScienceDirect

Microelectronics Journal

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

Energy-efficient data prefetch buffering for low-end embedded processors

Muhammad Yasir Qadri[a], Nadia N. Qadri[b], Martin Fleury[a,*], Klaus D. McDonald-Maier[a]

[a] School of Computer Science and Electronic Systems, University of Essex, Colchester Co4 3SQ, UK
[b] Department of Electrical Engineering, COMSATS Institute of Information Technology, Wah Campus, Pakistan

ARTICLE INFO

Keywords:
Control words
Data prefetch
Embedded processor
Micro-architecture

ABSTRACT

An energy-efficient architecture should jointly optimize energy consumption and throughput, as captured by the Energy-Delay-Square Product (ED^2P) metric. This paper introduces a prefetch data buffer micro-architecture, which achieves that goal with the aid of software-inserted control words to govern the prefetch process. The proposed architecture is aimed at low-end embedded processors, which, so as to reduce energy consumption, lack a cache-based memory hierarchy. By identifying after compilation which data should be prefetched and modifying the object code, the rate of prefetch misses is reduced. And by pre-computing memory addresses using auxiliary software after compilation and modifying the object code, address computation by hardware at run time is avoided, reducing pipeline stalls and, thus, improving throughput. Additionally in the case of branches, by prefetching two data items at any one time, alternative instruction outcomes are anticipated. The paper contains results from running a range of well-known and representative benchmarks on the proposed architecture. There was an improvement of 6 – 20% compared to an unbuffered architecture in execution times when tested over those seven benchmarks. Furthermore, the average ED^2P for the buffered architecture when normalized against the same architecture without buffering was found to vary between 54% and 90% according to benchmarking, though there is a cost in code size increase. That is to say, for the benchmarks tested there was a net energy efficiency improvement of between 10% and 46% in comparison with the equivalent unbuffered architecture with a lower area overhead.

1. Introduction

The growing proliferation of embedded battery-powered devices, often performing complicated tasks, leads to the prioritization of energy-efficient design optimizations. Such design optimizations strive to strike a balance between energy usage and throughput in a system and do not simply attempt to reduce energy consumption. In some energy-efficient designs, for some applications, there might even be a negative impact on energy consumption. However, the throughput can to rise to compensate, as quantified in the Energy Delay Product (EDP) metric [1].

The proposed architecture gives preference to a data prefetch buffer rather than a data cache. Though cache-based memory hierarchies are the norm for general-purpose PC architectures, in the embedded world system, architectures may employ alternatives to caches. The energy usage and chip area take-up of caches these can both be considerable. In addition, if the cache is not carefully tuned the cache miss ratio will also increase, leading to processor idling. Thus it is [2] that caches are a problematic feature in battery-powered embedded systems. Prior research [3–5] confirms that caches may be responsible for as much as 50% of a low-end processor's energy budget. For the most part, not herein, a cache-like structure underpins software-prefetching schemes, i.e. software prefetching assists already present hardware cache memories. However, the proposed scheme does *not* require a cache-like structure to be present, which is why it is likely to be more effective. Instead, the proposed architecture with a data prefetch buffer replaces the typical cache memory, and, hence, the inherent disadvantages of such caches (i.e. compulsory, conflict, and capacity misses). Comparing to the typical cache, the proposed prefetch buffer requires: much smaller storage, is more area efficient, and less power consumption.

Compiler-controlled prefetching of data [6] is one way that a data prefetch buffer can take the place of a data cache in an energy-efficient manner. However, prefetching typically suffers from an increased memory bandwidth. This increase is caused by unnecessary prefetches, owing to false predictions by the particular algorithm employed. (For previous research on prefetch buffering refer to Section 2). This paper proposes a software-controlled prefetch buffering architecture that, through the mechanism of control-word insertion, removes such false predictions. The proposal also has a number of additional advantages, including a reduction in pipeline stalls arising from memory address

* Corresponding author.
E-mail address: fleum@essex.ac.uk (M. Fleury).

http://dx.doi.org/10.1016/j.mejo.2017.01.014
Received 20 July 2016; Received in revised form 17 November 2016; Accepted 2 January 2017

calculations. We believe that the introduction of a varying instruction size, when control words are introduced is justified by the gains made. The Acorn RISC Machine (ARM) in its Thumb variant also includes 16-bit and 32-bit instructions.

Low-end embedded microprocessors and microcontrollers typically have on-chip memory. This is a way to reduce the number of additional components needed in an embedded application. Such an arrangement also results in single-cycle access to the memory, which this paper's proposal takes advantage of. In a further simplification, the instruction pipeline changes from the basic five-stages of a Reduced Instruction Set Processor (RISC) to just two stages. Both the 8-bit Atmel AVR [7] and the Peripheral Interface Controller (PIC) microcontroller families [8] support on-chip memory and two-stage pipelines, which the proposed software-based prefetching technique exploits. Both families also have a Harvard architecture, which allows instructions and data to be accessed simultaneously. As these microcontroller families are extensively deployed, the proposal in this paper is of wide generality and applicability to embedded applications. For example, by 2013 an AVR was present on every one of 700,000 official Arduino boards[1] and Microchip, PIC's manufacturer, output a 2013 press release[2] stating that it supplies one billion processors per year. According to an SAE article of 2014 entitled “Market for 8-bit chips remains strong”, Costlow points out that such processors account for 24% of the automotive microcontroller market, which figure is expected only to decline to 22% by 2018. As the vast majority of applications for low-end processors are in embedded computing, not in general-purpose computing, our proposal is geared towards embedded computing applications. Though, as Section 2 describes, pioneering work has gone on in the past within the general field of pre-fetching, we believe there is still scope for improvements, even though these improvements will now be focussed on specific domains.

Immediately after the compilation of application code, our software inserts control words, which cause a processor to prefetch data required by the next instruction in the two-stage instruction pipeline. In this way, the method avoids static pointer-based data references and associated address computations. More generally, the prefetch technique is implementable either by an additional software tool operating at compile time, the choice herein, or by an enhanced compiler directly, not used by us. Specifically, during the software creation phase or after program compilation, control words are placed at a location at least one instruction ahead. As a result, during execution the data required can be fetched without pipeline stalls. Therefore, this architecture provides greater energy efficiency when compared to an unbuffered architecture with lower area overhead. As with other software prefetching schemes, our proposal leads to what could be for some applications a significant increase in code size owing to the need to store some 32-bit rather than 16-bit instructions to accommodate control words. The significance will depend on the size of the application code, which might anyway fit within the existing on-chip memory.

Although the previously-mentioned EDP [1] is a widely adapted metric to evaluate energy and delay effects, Martin et al. [9] recommends a weighted approach, using another metric i.e. energy-delay-square-product (ED^2P), which in [9] is alternatively called energy-time-square (ET^2), as T is delay. This metric is very useful in evaluating trade-offs between the circuit-level power consumption and the overall energy efficiency of the system [10]. The ET^2 (or alternatively ED^2P) metric was first introduced by Martin et al. in [9] in order to evaluate the asynchronous MIPS3000 processor. The validity of the metric was later analysed by Martin in [11,12]. In [13], ED^2P is defined as $ED^2P = EPI{\cdot}CPI^2 = EPC{\cdot}CPI^3$, where EPI is the energy per committed instruction, EPC is energy per cycle, and CPI is cycles per instruction. For a complete execution of benchmark i, ED^2P can be calculated as: $ED^2P_{\eta i} = \eta_i$. $EPC.CPI^3$, where η_i is the number of instructions executed. However, herein we use the notation ED^2P instead of $ED^2P_{\eta i}$ as a simplification.

In evaluating a design [13], ED^2P highlights performance more than EDP does. To first order, ED^2P is also independent of variations in voltage and frequency. A mathematical analysis of the advantages of using ET^2 over ET can be found in [11], where it is said that “The energy-delay product $E \times t$ is often used to compare designs but is unfortunately not an acceptable metric”. Indeed, some authors, for example [14], even suggest using the cube of delay to weight the delay more than the energy of a system. However, we adopt a more moderate approach. The focus of the architecture is to achieve greater throughput for an overall reduced active cycle for a processor. Thus, the authors of this paper consider ED^2P to be the most appropriate metric for the research presented in this paper. Motivated and guided in that way, in this paper we introduce a novel prefetch data buffering micro-architecture for low-end, embedded processors with on-chip memories, which provides increased energy efficiency.

Finally in this introduction, notice that a brief outline of some of our ideas has been filed as a U.S. patent application [15], though without relevant prior research papers, consideration of the context, or performance results and analysis, as now occurs in this paper. This paper also includes a longer description of the innovation and broadens the treatment. Our ideas are also applicable to instruction prefetching. In [16] we did exactly that, thus confirming the benefit of the ideas contained in this paper for data prefetching.

The rest of this paper is arranged as follows. Section 2 is a review of related work in this field before going on to describe the proposed architecture in Section 3. In Section 4, the area and power overheads of the software-controlled prefetching are compared against those of the original unbuffered architecture. A detailed analysis of energy consumption reduction and throughput improvement occurs using various benchmark applications. Finally, Section 5 rounds up the paper with some concluding remarks.

2. Related research

A significant trend in low-power cache design [17] is to include an additional small extra data buffer, which is accessed directly by the embedded system. Therefore, these designs require the buffer to be accessed first, preventing altogether direct access to the original caching structure. The intention of such designs is to save energy by achieving a high hit rate to the small intermediate buffer. Notice that a larger buffer does not result in the same energy savings. Investigation of these intermediate hardware buffers or caches is the inspiration behind substituting software control of data prefetch. Crucially, however, the current paper avoids a cache-based memory hierarchy. On the other hand, purely hardware-guided prefetching of data into caches, e.g. [18], may be energy intensive [19] and is certainly not suitable for multicore platforms (For some counter-examples of energy efficient hardware prefetching consult [20]).

Most recently, there has been interest in introducing machine learning into prefetching. Work in [21] considers the risk of aggressive prefetching saturating the memory bandwidth of a multicore processor, for example, with a 40% risk of hardware prefetching harming the performance of Intel's Sandyridge i7-2600K processor. Instead [21] considers dynamically combining hardware- and software-based prefetching in the Adaptive Resource Efficient Prefetching (AREP) framework. That framework examines a selection of prefetch configurations in order to choose the one with the least impact on performance. The work in [21] reports an 8% increase on average in performance from applying AREP. The automatic prefetching tuner (PATer) for the POWER8 processor [22] provides a way of tuning the prefetch configuration. The need arises because the POWER8 processor has a 25-bit hardware register in which the cache prefetch configuration can

[1] According to Cuartielles in 2013 on the Arduino FAQ at http://www.arduino.cc/en/Main/FAQ

[2] Available at http://www.microchip.com/pagehandler/en-us/press-release/microchips-12-billionth-pic-mi.html