Position Paper: Reaching intrinsic compute efficiency requires adaptable micro-architectures

Mark Wijtvliet, Luc Waeijen, Michaël Adriaansen, and Henk Corporaal

Eindhoven University of Technology, 5612 AZ, The Netherlands,
{m.wijtvliet, l.j.w.waeijen, h.corporaal}@tue.nl &
m.adriaansen@student.tue.nl

Abstract. Today's embedded applications demand high compute performance at a tight energy budget, which requires a high compute efficiency. Compute efficiency is upper-bound by the technology node, however in practice programmable devices are orders of magnitude away from achieving this intrinsic compute efficiency. This work investigates the sources of inefficiency that cause this, and identifies four key design guidelines that can steer compute efficiency towards sub-picojoule per operation. Based on these guidelines a novel architecture with adaptive micro-architecture, and accompanying tool flow is proposed.

Keywords: adaptive micro-architecture, intrinsic compute efficiency, spatial layout

1 Introduction

Modern embedded applications require a high computational performance under severe energy constraints. Mobile phones, for example, have to implement the 4G protocol, which has a workload of about 1000 GOPS [11]. Due to battery capacity limitations, the computation on a mobile phone has a budget of about 1 Watt. Thus, under these requirement, each computational operation can only use $1pJ$ of energy. Another example is ambulatory healthcare monitoring, where a patients' vital signs are monitored over an extended period of time. Because these devices have to be mobile and small, energy is very limited. An added constraint is that the compute platform has to be programmable, as the field of ambulatory healthcare is still developing, and improved algorithms and new applications are developed at a fast rate.

To support such embedded applications, a computational operation has an energy budget in the sub-pico joule domain. However, current programmable devices do not have a high enough compute efficiency to meet this requirement. One of the most compute efficient microprocessors, the ARM Cortex-M0, has an compute efficiency of $5.1{}^{pJ}/_{op}$ at 40nm low-power technology [8]. The intrinsic compute efficiency (ICE) of 45nm technology is $1{}^{pJ}/_{op}$ [6]. There is thus a gap between the ICE and the achieved efficiency of at least a factor 5. However, to support compute intensive embedded applications, processors more powerful than the Cortex-M0 are needed, which increases the gap up to several orders of

magnitude [6]. In order to meet the demands of modern embedded applications, this gap has to be closed.

This work investigates why programmable processing devices do not meet the ICE and how this can be improved in section 2. It is concluded that an adaptive micro-architecture should be leveraged to improve parallelism exploitation and save energy on dynamic control and data transport, two of the largest sources of inefficiency. Based on this observation a novel architecture is proposed in section 3, designed to narrow the efficiency gap. The tool flow and compiler approach are discussed in section 4. Section 5 concludes this work.

2 Discussion and Related Work

The achieved compute efficiency (ACE) of current programmable architectures is several orders of magnitude lower than the ICE [6]. Hameed et al. find a similar gap (500×) between general purpose processors and Application Specific Integrated Circuits (ASICs), which come very close to the ICE. There are many sources of inefficiency for general purpose processors that contribute to this gap. Hameed et al. identified various of these sources [4]. They extend a Tensilica processor [2] with complex instructions and configurable hardware accelerator to support a target application. This brings the compute efficiency for the application within 3× of the ICE, but at the expense of generality. The resulting architecture is highly specialized. Based on their optimizations it can be concluded that largest sources of overhead are:

1. Dynamic control, e.g, fetching and decoding instructions.
2. Data transport, e.g., moving data between memory, caches and register files.
3. Mismatch between application and architecture parallelism, e.g., 8-bit add on 32-bit adder.

The first two sources of overhead can be attributed to sequential execution. A large amount of energy is used because the processor fetches and decodes a new instruction every cycle. This can be mitigated by using spatial layout (execution in parallel). By increasing the number of issue slots, it is possible to achieve a single instruction steady-state, such that no new instructions need to be fetched for an extended period of time. Refer to the for-loop in Fig. 1 for an example. The loop-body contains operations A, B, C and control flow computation 'CF'. The loop in the figure can be transformed from the sequential version (left) to the spatial version (right) by software-pipelining. The single-cycle loop body in Fig. 1 does not require any other instructions to be fetched and decoded. It can be observed that a general purpose processor with only one issue slot can never support single-cycle loops due to the control flow. This technique is already used in very long instruction word (VLIW) processors [9], but is only applicable if the number of issue slots and their compute capabilities match the loop. ASICs and Field Programmable Gate Arrays (FPGA) implement the extreme form of spatial-layout. By completely spatially mapping the application the need for instruction fetching and decoding is eliminated altogether.

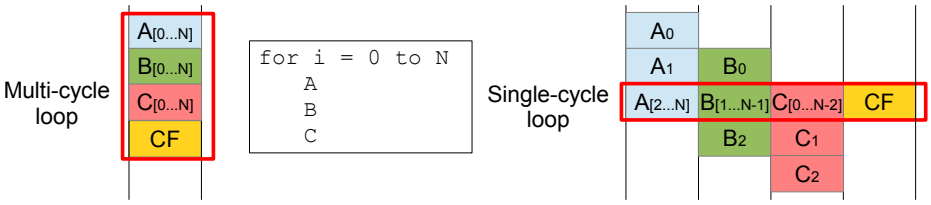

Fig. 1. Program execution example for multi- and single-cycle loops

The second source of inefficiency, data transport, is reduced substantially by adapting the data-path to the application in such a way that the register file (RF) and memory system are bypassed as much as possible like in explicit data-paths [12]. The memory system and the RF are two of the main energy users in a processor [4]. Thus, by keeping data in the pipeline, the overall energy usage can be reduced significantly.

The third source of inefficiency can be addressed by adapting the micro-architecture to the application. Applications have varying types of parallelism: bit-level (BLP), instruction-level (ILP) and data-level (DLP). BLP is exploited by multi-bit functional units, such as a 32-bit adder. ILP is exploited with multiple issue slots, such as very long instruction word (VLIW) processors. Finally, DLP is exploited by single instruction multiple data (SIMD) architectures. Different applications expose different types and amounts of parallelism. When the micro-architecture is tuned to the application, such as in an ASIC or FPGA, the mix of different types of parallelism can be exploited in the optimal manner.

Micro-architecture adaptation is the key to achieve a higher compute efficiency. FPGAs and ASICs do this, but at an unacceptable price. For ASICs the data-path is adapted for one set of applications, so it loses generality. FPGAs are configured at gate-level, which requires many memory cells to store the hardware configuration (bitfile). These cells leak current resulting in high static power consumption [7, 3]. Furthermore the dynamic power is also high [1], due to the large configurable interconnect. Additionally efficiently compiling for FPGAs is hard due to the very fine granularity. Although there are High-Level Synthesis tools that reduce the programming effort, they cannot always provide high quality results [13] because of this.

Summarizing, to achieve high compute efficiency, overhead of adaptability should be reduced, while still supporting:

1. Single instruction steady state, e.g., single-cycle loops
2. Data transport reduction, e.g., explicit bypassing
3. Application tailored exploitation of parallelism, e.g., VLIW with matching SIMD vector lanes
4. Programmability

3 Architecture Proposal

In this section an energy-efficient architecture is proposed that ticks all requirement boxes from section 2. An adaptive micro-architecture is realized by separating control units, such as instruction fetch (IF) and instruction decoder (ID), from the functional units (FU). Each ID can be connected to one or more FUs through a control network, and FUs are interconnected via a data network. These networks use switch-boxes that configured before the application is executed, and remain static during execution, much like FPGAs. The number of switch-boxes is much smaller than in an FPGA, and multiple bits are routed at once. Therefore the proposed architecture requires significantly less configuration bits, thereby avoiding the high static energy usage that FPGAs suffer from. There are various FU types that are considered: Arithmetic Logic Units, Load Store Units, RFs and Branch Units. The adaptive micro-architecture enables high energy efficiency while attaining high compute performance.

3.1 Single instruction steady state

It is possible to construct VLIW-like micro-architectures by grouping IDs in a common control group, and connecting them to FUs, as shown in Fig. 2. In this figure a three issue-slot VLIW is shown. By adapting the number of issues slots to the application, single instruction steady state is supported. Thus reducing instruction fetch and decode, resulting in lower dynamic energy usage. Multiple ID control groups enable the construction of multiple independent VLIWs.

3.2 Data transport reduction

Reduction of data transport is achieved by directly connecting FUs through a switch-box network. This allows results from one FU to bypass the RF and memory, and directly flow to the next FU. Complex data-flow patterns, such as butterfly patterns in the fast Fourier transform, can be wired between the FUs. This reduces RF accesses that otherwise would have been required to accomodate these patterns. The special case of data-flow patterns where each compute node performs the same operation, such as reduction trees, can be supported with only one ID to control the entire structure.

3.3 Application tailored exploitation of parallelism

The varying amount of ILP in an application can be exploited by the configurable VLIW structures. DLP is captured by constructing SIMD-type vector-lanes within each issue slot, as shown in Fig. 2, where issue-slot 3 has a vector width of four. BLP is addressed by combining multiple narrower FUs into wider units, e.g., combine two 16-bit FUs into one 32-bit unit. This allows efficient support of multiple data-widths, e.g., processing 8-bit pixels for an image application in one case, and supporting 32-bit fixed point for health monitoring applications.

3.4 Programmability

The possible configurations in the proposed architecture all bear a strong resemblance to VLIW processors with an explicit data-path and issue slots with vector lanes. This requires a compiler which supports explicit bypassing, which is described in more detail in section 4.

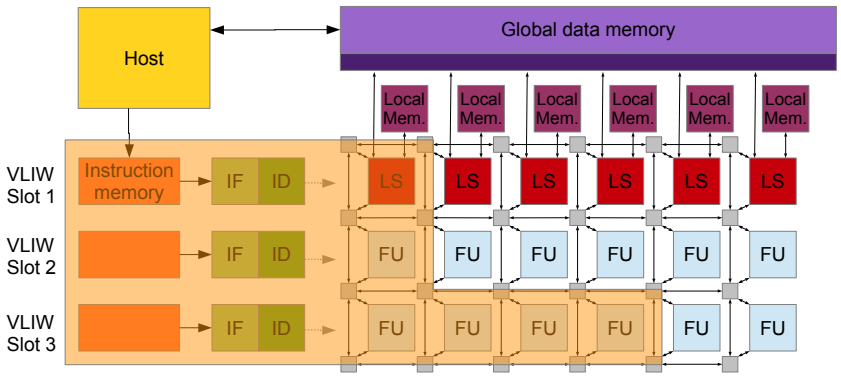

Fig. 2. Proposed architecture

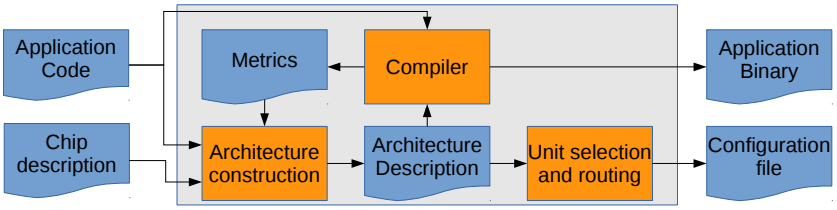

Fig. 3. Toolflow

4 Tool flow

Many architectures have been published in literature, but few of them are used in industry, often because of the lack of mature tools. The development of the tool flow for the highly flexible proposed architecture is challenging. A chip description file that lists the available resources and interconnect options is used as input to the tool flow, as is shown in Fig. 3. Generation of the hardware description (HD) and mapping of data-flow patterns to FUs is done based on this file. The tools to generate synthesizable HD are already implemented, but require integration with the full toolflow. The most challenging part, the construction of the compiler, is in progress. Section 4.1 discusses the challenges and various approaches to deal with them.

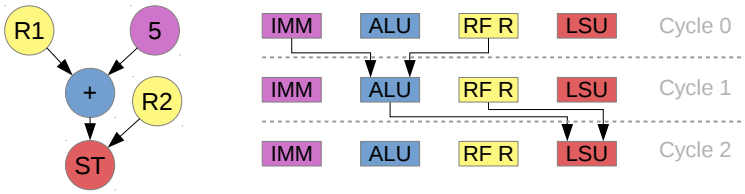

Fig. 4. Resource and dependence graphs

4.1 Compiler

Designing the compiler is particularly challenging because of the explicit data-path of the proposed architecture, and code has to be generated for all possible combinations of IDs and FUs. In addition to the tasks of a regular compiler, the compiler needs to route data between FUs. This is similar to compilers for transport triggered architectures [5].

One approach for scheduling for an explicit data-path is list scheduling using a resource graph (RG). The RG has a node for every FU at every clock cycle in the schedule, shown in Fig. 4. Scheduling is done by mapping nodes from the data dependence graph onto the nodes in the RG. However, scheduling deadlocks can occur when the result of a scheduled operation can not be routed to its destination, because the required pass-through resource became occupied during scheduling. Guaranteeing that values can always be read from the RF prevents these scheduling deadlocks. There are two methods to guarantee this. One always allocates a temporary route to the RF. Another method generates max flow graphs to check if all data can reach the RF [10]. Instead of preventing deadlocks, they can also be resolved by a backtracking scheduler that unschedules operations if their result can not be routed.

5 Conclusions

Various sources of inefficiency in programmable devices have been investigated, and methods to reduce these inefficiencies have been discussed. Four design guidelines have been established that will steer programmable devices in the direction of sub-picojoule compute efficiency, while not sacrificing generality and performance of these devices. A novel architecture with adaptable micro-architecture which adheres to these guidelines has been proposed, and its tool flow has been described. The proposed architecture is an adaptable mix between multi-core VLIW, SIMD, and FPGA architectures, which allows efficient mapping of an application, by using the best from each of these architectures. A synthesizable hardware description the architecture is available, and will be used in future work for validation of the guidelines presented here, and further development of the architecture.

References

[1] Amara Amara, Frédéric Amiel, and Thomas Ea. "FPGA vs. ASIC for low power applications". In: *Microelectronics Journal* 37.8 (2006), pp. 669 – 677. ISSN: 0026-2692. DOI: http://dx.doi.org/10.1016/j.mejo.2005.11.003. URL: http://www.sciencedirect.com/science/article/pii/S0026269205003927.

[2] Inc. Cadence Design systems. *Tensilica Customizable Processor IP*. URL: http://ip.cadence.com/ipportfolio/tensilica-ip (visited on 11/20/2015).

[3] Lanping Deng, K. Sobti, and C. Chakrabarti. "Accurate models for estimating area and power of FPGA implementations". In: *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*. Mar. 2008, pp. 1417–1420. DOI: 10.1109/ICASSP.2008.4517885.

[4] Rehan Hameed et al. "Understanding Sources of Inefficiency in General-purpose Chips". In: *SIGARCH Comput. Archit. News* 38.3 (June 2010), pp. 37–47. ISSN: 0163-5964. DOI: 10.1145/1816038.1815968.

[5] Jan Hoogerbrugge. *Code generation for transport triggered architectures*. TU Delft, Delft University of Technology, 1996.

[6] Akash Kumar et al. *Multimedia Multiprocessor Systems: Analysis, Design and Management*. Embedded Systems. Springer Netherlands, 2010. ISBN: 978-94-007-0083-3. DOI: 10.1007/978-94-007-0083-3.

[7] Fei Li et al. "Architecture Evaluation for Power-efficient FPGAs". In: *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*. FPGA '03. New York, NY, USA: ACM, 2003, pp. 175–184. ISBN: 1-58113-651-X. DOI: 10.1145/611817.611844. URL: http://doi.acm.org/10.1145/611817.611844.

[8] ARM Ltd. *ARM Cortex-M0 Specifications*. URL: http://www.arm.com/products/processors/cortex-m/cortex-m0.php (visited on 11/19/2015).

[9] Yi Qian, Steve Carr, and Philip Sweany. "Loop fusion for clustered VLIW architectures". In: *ACM SIGPLAN Notices* 37.7 (2002), pp. 112–119.

[10] Dongrui She et al. "Scheduling for register file energy minimization in explicit datapath architectures". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*. Mar. 2012, pp. 388–393. DOI: 10.1109/DATE.2012.6176502.

[11] C.H. Van Berkel. "Multi-core for mobile phones". In: *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.* 2009, pp. 1260–1265. DOI: 10.1109/DATE.2009.5090858.

[12] Luc Waeijen et al. "A Low-Energy Wide SIMD Architecture with Explicit Datapath". English. In: *Journal of Signal Processing Systems* 80.1 (2015), pp. 65–86. ISSN: 1939-8018. DOI: 10.1007/s11265-014-0950-8. URL: http://dx.doi.org/10.1007/s11265-014-0950-8.

[13] M. Wijtvliet, S. Fernando, and H. Corporaal. "SPINE: From C loop-nests to highly efficient accelerators using Algorithmic Species". In: *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*. Sept. 2015, pp. 1–6. DOI: 10.1109/FPL.2015.7294015.