FITS: Framework-based Instruction-set Tuning Synthesis for Embedded Application Specific Processors

Allen Cheng Gary Tyson† Trevor Mudge

Advanced Computer Architecture Lab
The University of Michigan
Ann Arbor, MI 48109-2122
{accheng,tnm}@eecs.umich.edu

†Department of Computer Science
Florida State University
Tallahassee, FL 32306-4530
tyson@cs.fsu.edu

ABSTRACT

We propose a new instruction synthesis paradigm that falls between a general-purpose embedded processor and a synthesized application specific processor (ASP). This is achieved by replacing the fixed instruction and register decoding of general purpose embedded processor with programmable decoders that can achieve ASP performance with the fabrication advantages of a mass produced single chip solution.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization General**]: Instruction set design (e.g. RISC, CISC, VLIW); C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Algorithms, Performance, Design

Keywords

16-bit ISA, Instruction synthesis, Embedded processor, ASP, Instruction encoding, Reconfigurable processors, Configurable architecture, Code density, Low-power, Energy efficient

1 INTRODUCTION

In recent years, embedded systems have received increased attention due to the rapid market growth for high-performance portable devices such as smart phones, PDAs and digital cameras. These applications require more instruction throughput while retaining strict limits on cost, area, and power. This requirement necessitates a new system architecture that can exploit the special characteristics of these embedded applications to meet the ever-tighter constraints. An emerging popular strategy to meet the challenging cost, performance, and power demands is to move away from general-purpose designs to application-specific designs tailored to the requirements of a particular application. An application-specific processor (ASP) is a processor designed for a particular application. Thus, an ASP design contains only those capabilities necessary to execute its target workload. The result is that ASPs can achieve levels of performance and efficiency that are unattainable in general-purpose processors.

In this paper, we present a cost-effective Framework-based Instruction-set Tuning Synthesis (FITS) technique for designing embedded ASPs. FITS offers a tunable, general-purpose processor solution to meet the code size, performance, and time to market constraints with minimal impact on area. FITS delays instruction set synthesis until after processor fabrication. Post fabrication synthesis is performed by replacing the fixed instruction and register access decoder of conventional designs with programmable decoders. Through the programmable decoders, we can optimize the instruction encoding, address modes, and operand and immediate bit widths to match the requirements of the target application. FITS is cost-effective in that: (1) it reduces the code size by synthesizing 16-bit ISAs with minimal performance degradation for many embedded applications that would normally require 32-bit ISAs; (2) it reduces power consumption by requiring a smaller on-chip cache and by deactivating those parts of the datapath that are not mapped to any instructions of the synthesized architecture; (3) it reduces cost and time to market for new products by utilizing a single processor platform across a wide range of applications, while retaining the ability to optimize the instruction set and register organization for the specific needs of each application. The datapath of a FITS processor would be similar to a general-purpose embedded processor such as ARM, containing a full range of functions, but would map only a subset of those to the synthesized instruction set. By only mapping those operations that a particular application needs to the synthesized instruction set, it is possible to encode all instructions in a short, 16-bit format while retaining all of the special purpose operations that can be found in a large instruction embedded processor.

The remainder of this paper is organized as follows: Section 2 describes related work in instruction set synthesis for embedded systems. Section 3 identifies the characteristics of embedded applications. Section 4 presents the FITS design methodology and architectural innovations. Section 5 examines the effectiveness of using a short instruction encoding in the FITS paradigm. We conclude and discuss future work in Section 6.

2 RELATED WORK

In today's embedded computing industry, one commonly adopted ISA scheme is dual instruction sets. Dual instruction set processors, such as the ARM [2], MIPS16 [5], ST100 [9], and ARCtangent-A5 [1], address the limited memory and energy constraint by supporting a 16-bit instruction set along with the 32-bit instruction set. Performance critical code is compiled to the 32-bit ISA and the

DAC 2004, June7-11, 2004, San Diego, California, USA.

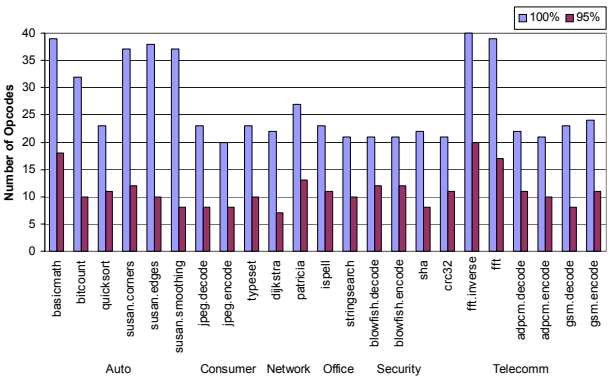

Figure 1: Dynamic Utilization of Opcodes

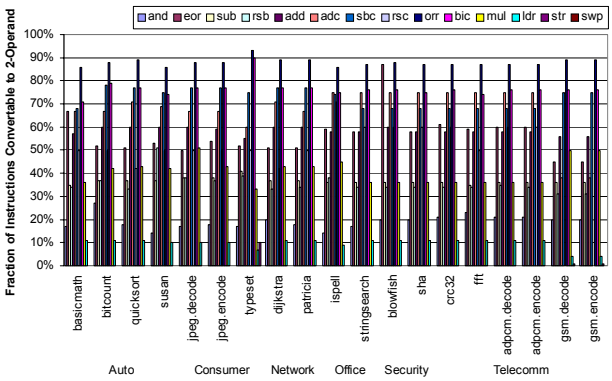

Figure 2: 2-Address Convertible Instructions

rest to the 16-bit ISA. However, this approach often requires manual code blending, which is not only time consuming but can be prone to inserting errors into an existing application. Recognizing this dilemma, ARM has introduced the Thumb-2 [3] architecture, which is a blended ISA that combines both 16-bit and 32-bit instructions in a single instruction set.

Our approach is different from others in that we believe the a 16-bit ISA can accommodate the requirements of almost all embedded applications without the support of some larger instructions. Since applications may not require the same set of instructions, we propose an architecture with the full range of functional capabilities found in a 32-bit embedded processor but only map a subset of instructions that a particular program needs to the 16-bit instruction format. Thus, rather than starting with a 32-bit ISA and looking for places to partially substitute it with its 16-bit counterpart, we move straight into the single 16-bit ISA scheme and utilize an instruction encoding synthesized to the requirements of each application.

3 EMBEDDED WORKLOAD ANALYSES

3.1 Opcode Space Requirement

The opcode space in an ISA specifies the number of different instructions a processor may perform. Higher utilization of instructions by an application means more instruction bits need be allocated to opcodes for that application. Figure 1 shows the dynamic ARM integer opcode usage from MiBench [6]. The 100% bar represents the number of opcodes utilized by each application. Among all 23 programs, 16 of them utilize 27 or less opcodes; 7 of them utilize between 32 to 40 opcodes. The 95% bar indicates the number of opcodes used the most: together, they account for 95% or more of total dynamic instructions. Ignoring less than 5% of total dynamic instructions reduces the opcode requirement significantly: 20 out of 23 programs use at most 13 opcodes, while the highest demand does not exceed 20. The reason for the significant reduction in opcode requirement is because not all opcodes are executed equally frequently. According to our simulation results, 55.6% opcodes, on average, were executed less than 1% of the time. These infrequently executed opcodes can be instead translated into software to save the instruction space without affecting performance significantly.

3.2 Operand Space Requirement

Instructions with three addresses prevail in many popular 32-bit RISC machines. Despite many advantages of having three register operands, two operands are often enough. Therefore, we statically and dynamically examined how often two addresses may suffice. Static statistics are important from a code size viewpoint, and dynamic statistics help us gauge power dissipation. Figure 2 illustrates the fraction of all three-address instructions that only need two addresses statically. We determine this value by calculating the fraction of time that a destination operand is the same as one of the source operands. Besides *load*, *store*, and *swap* instructions which tend to use three addresses by nature, the remaining 3-address integer instructions need only two operands 19% to 88% of the time. The dynamic profiling shows a even higher convertibility that ranges from 59% to 87% of the time. This encouraging result suggests intermixing two-address instructions and three-address instructions within an application. This approach trades off the expressive power of an instruction for a compact space.

3.3 Immediate Space Requirement

We classify all immediates into three categories: branch immediates, ALU immediates, and memory immediates. The branch immediates determine the PC-relative branch target. The ALU immediates are constant values used by ALU instructions to process data. The memory immediates are constant offsets used by load and store instructions to calculate the effective memory addresses. Figure 3 shows the static distribution of instructions with each type of immediates embedded. On average, 71% of instructions contains immediates: 30.7% being ALU immediates; 23.5% being memory immediates; 16.8% being branch immediates. Given their dominant usage across the entire benchmark suite, it is important for any good processor design to manage immediates efficiently. Thus, the next thing we look at is the number of different immediates existing in the program. If the number of unique value is small, we may offload these immediates from instructions to a table and then compress the instruction space by substituting original larger immediate fields with smaller table index fields. On average, there are 6020 different branch immediates, 648 different ALU immediates, and 464 different memory immediates, as shown in Figure 4.

Dynamic analysis allows us to identify the most frequently used immediates, and thus allows us to pinpoint the greatest need of immediates and allocate the instructions bits accordingly. Our results show that, on average, 97.7% of executed instructions contain immediates: 53.9% being ALU immediates; 32.2% being memory immediates; 11.7% being branch immediates. For the number of different immediates utilized in a program, there are, on

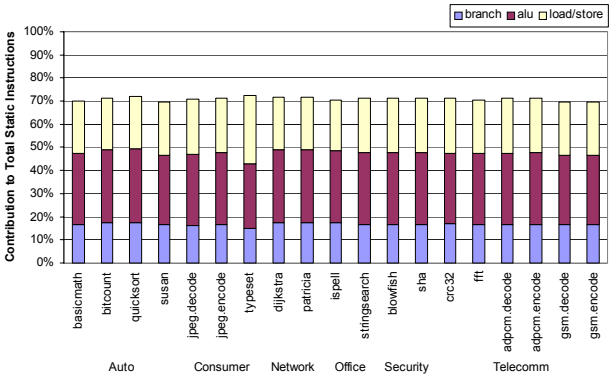

Figure 3: Distribution of Immediate Instructions

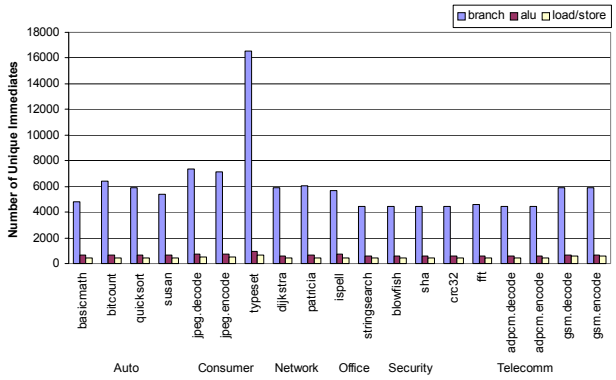

Figure 4: Utilization of Unique Immediate Constants

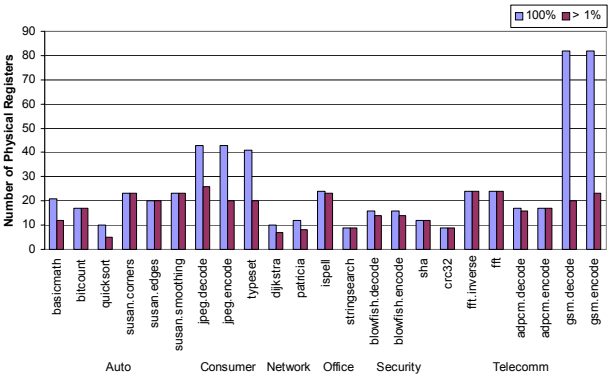

Figure 5: Utilization of Physical Registers

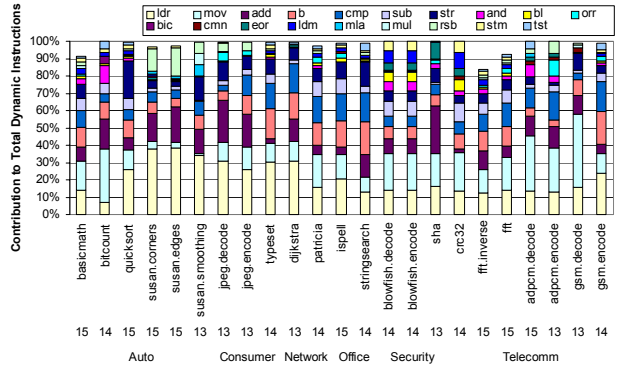

Figure 6: Synthesized FITS Instruction Set

average, 335 different branch immediates, 113 different ALU immediates, and 197 different memory immediates.

3.4 Register Space Requirement

To minimize the memory traffic and reduce code size, we analyzed the maximum number of physical registers necessary to eliminate any memory spills when compiling a program. Figure 5 shows the register usage statistics using the MIRV [7] compiler augmented with our profiling tools. The analyses were performed at the procedure level. The 100% bar shows the maximum number of physical registers a program ever needs. However in real programs, some procedures are called more often than the others. We argue that in order to achieve a better resource utilization, procedures that execute less than 1% of the time yet demand many registers (e.g. Main and Init) should be given less consideration. As a result of this policy, 3 programs need 8 or less physical registers; 7 programs need 9-16 physical registers; 13 programs need 17-26 physical registers.

4 SYNTHESIS FRAMEWORK

4.1 FITS Methodology

FITS is an application-specific hardware software co-design approach that matches microarchitectural resources to application performance needs while improving code-density. FITS does application-specific customization at the instruction set level utilizing programmable decoders for instruction decode and register access. A FITS processor consists of a fairly large set of functional units, including standard ALU operations as well as a set of occasionally useful instructions (e.g. Multiply/accumulate, looping instructions, etc.). Limitations on the functions provided are due to chip area goals, not instruction set size limits. This can greatly increase the number of similar operations, such as saturating add, because the additional circuitry to add saturation to an add operation is minimal. Since instruction space encoding is decoupled, it is possible to add many instructions that may only be useful to a small subset of applications. With a programmable decoder, FITS can tune an ISA to include only those operations necessary for a single application. Moreover, FITS is extremely flexible in terms of the range of underlying microarchitecture that it can work with: from general-purpose DSPs or embedded processors to application-specific customized data-paths. FITS provides the same level of customization as many ASPs, trading somewhat greater chip area requirements for eliminating the need to synthesize a new chip for each application.

To tune a FITS processor, a FITS aware compiler analyzes the instruction and register requirements of an application before instruction selection and register allocation. We currently use profile information (as shown in the previous section), but we are exploring new optimization heuristics using static dataflow information to perform the code transformation. Once code generation is complete, the compiler can specify the register organization and instruction decoding to perform for the application. This configuration information is then downloaded to a non-volatile state in the FITS processor. At this point, the processor instruction set and register file organization is complete. If this application is later upgraded with increased functionality, FITS can re-configure the decoders to match the new requirements of the application. In general, FITS can transform any general-purpose machine into an application-specific processor platform with over-provisioned resources that can be dynamically configured to adapt to the needs of different applications.

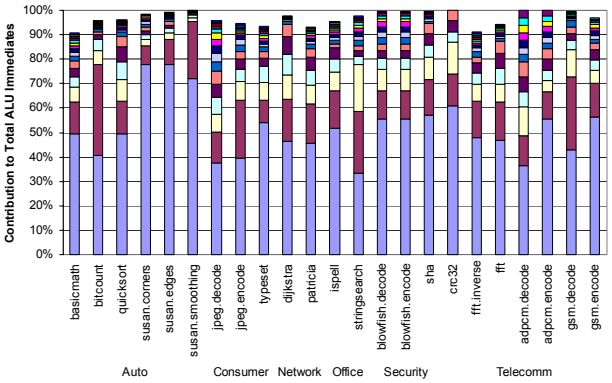

Figure 7: Synthesized Top 16 ALU Immediates

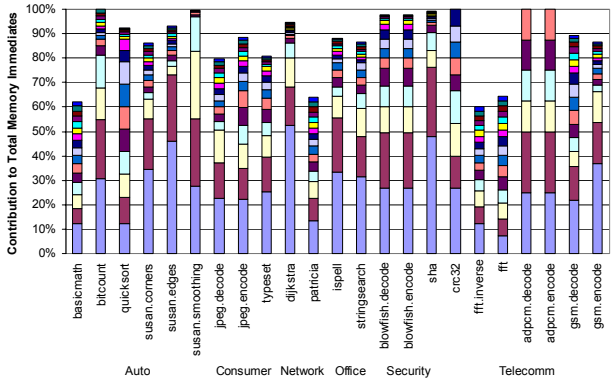

Figure 8: Synthesized Top 16 Memory Immediates

4.2 Synthesis Heuristic

The compiler must make tradeoffs in the instruction selection phase of optimization. This may include software emulation of rarely used instructions. In almost all cases the instruction set mapping includes a Base Instruction Set (BIS) and a Supplemental Instruction Set (SIS). A BIS includes instructions found across all applications (e.g. add); a SIS includes instructions required to make the instruction set Turing-complete. BIS and SIS together contain enough functionality to simulate any instructions not mapped for an application. In addition to BIS and SIS instructions, FITS will include a set of application-specific instructions (taken from the set of functional units in the microarchitecture) necessary for the application to meet any performance goals. An application-specific instruction set (AIS) is determined by evaluating the performance of various 16-bit encoding methods. Register allocation is also designed to trade off the register file size and encoding with register spill frequency.

Moreover, FITS uses a utilization-based dictionary compression technique to encode immediate operands. FITS identifies the most frequently accessed immediates and places them in programmable, non-volatile memory storage. Then, we could replace the bigger instruction immediate with a smaller index into this storage. This approach is similar to [4] except FITS can dynamically reconfigure the total immediate field width and adjust widths of other instruction fields accordingly to best reflect the application's requirements.

5 EXPERIMENTS AND RESULTS

We evaluated the effectiveness of FITS across a wide range of embedded applications in MiBench. We used the SimpleScalar toolset [8] to examine the quality of the synthesized instruction set. As shown in Figure 6, BIS includes *load*, *move*, *add*, *branch*, and *compare.* SIS includes *subtract*, *store*, *and*, *branch with link*, and *or*. The size of AIS is different from one application to another. The union of entire suite's AIS consists of: *bit clear*, *compare negative*, *exclusive or*, *load multiple*, *multiply accumulate*, *multiply*, *reverse subtract*, *store multiple*, and *test bits*. Depending on the individual execution characteristics, most applications include 3 to 5 AIS instructions. On average, BIS, SIS, and AIS account for 69.1%, 17.9%, and 10.8% of total dynamic execution needs respectively, totaling 97.8%; the other 2.2% are simulated using multiple BIS and SIS instructions.

Figure 7 shows the quality of synthesized ALU immediates. With as few as 16 immediates, they capture an average of 96.9% of total references made to the entire ALU immediate space. It is also interesting to observe that 51.8% of total accesses were captured by the most frequently referenced immediate. This technique applies to memory immediates in a lesser extent. On average, 87.4% of total references made to the entire memory immediate space were captured as shown in Figure 8.

6 CONCLUSIONS AND FUTURE WORK

The goal of this research is to argue for a new approach to the design of a class of embedded processors. We feel that by delaying the mapping of instruction set to microarchitecture to a point after chip fabrication, it will be possible to match the dense coding capabilities of ASP while retaining the fabrication advantages of a single chip design. Using the FITS design methodology enables a cost-effective 16-bit ISA synthesis solution while reducing design time and complexity, by decoupling the microarchitectural enhancements available on chip from the encoding issues of mapping to the subset of instructions required by a single application. Our analysis shows that for a wide range of embedded applications it is feasible to utilize a 16-bit instruction format, but that each application may require a different selection of operations and storage components. By delaying instruction assignment and register file organization until a program is loaded, it is possible to aggressively design the microarchitecture, including operations that are only occasionally useful, without the code bloat that would occur on a conventional machine.

7 REFERENCES

[1] ARCtangent-A5 microprocessor Technical Manual, ARC Cores, http://www.arccores.com.

[2] ARM7TDMI technical Manual. ARM Ltd., http://www.arm.com

[3] ARM Thumb®-2 Core Technology, ARM Ltd., http://www.arm.com/armtech/Thumb-2.

[4] C. Lefurgy, E. Piccininni, and T. Mudge, "Reducing Code Size with Run-time Decompression," in Proceedings of 6th International Symposium on High-Performance Computer Architecture (HPCA), Jan. 2000, pp. 218-227.

[5] K. D. Kissell, "MIPS16: High-density MIPS for the Embedded Market," in Proceedings of Real Time Systems, 1997.

[6] MiBench Version.1.0, http://www.eecs.umich.edu/mibench.

[7] MIRV Compiler Group, http://www.eecs.umich.edu/mirv.

[8] SimpleScalar LLC, http://www.simplescalar.com

[9] ST100 Technical Manual, STMicroelectronics, http://www.st.com.