AFRL-VS-TR-2003-1024

**AFRL-VS-TR-
2003-1024**

POWER-ADAPTIVE MICROARCHITECTURE AND COMPILER DESIGN FOR MOBILE COMPUTING

Rajiv Gupta et al.

**The University of Arizona
Department of Computer Sciences
Tucson, AZ 85721**

January 2003

Final Report

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

**AIR FORCE RESEARCH LABORATORY
Space Vehicles Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776**

REPORT DOCUMENTATION PAGE

Form Approved OMB No.
0704-0188

1. REPORT DATE (DD-MM-YYYY)
28-01-2003

2. REPORT TYPE

3. DATES COVERED (FROM - TO)
xx-07-2000 to xx-11-2002

4. TITLE AND SUBTITLE
POWER-ADAPTIVE MICROARCHITECTURE AND COMPILER DESIGN FOR MOBILE COMPUTING
Unclassified

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME AND ADDRESS
The University of Arizona
Department of Computer Sciences
Tucson, AZ85721

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME AND ADDRESS
,

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APUBLIC RELEASE
,

13. SUPPLEMENTARY NOTES

14. ABSTRACT
refer to atch

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:

17. LIMITATION OF ABSTRACT
Public Release

18. NUMBER OF PAGES
20

19. NAME OF RESPONSIBLE PERSON
Mosher, Jan
Janet.Mosher@kirtland.af.mil

a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified

19b. TELEPHONE NUMBER
International Area Code
Area Code Telephone Number
DSN

AFRL-VS-TR-2003-1024

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data, does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report has been reviewed by the Public Affairs Office and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

This material is based on research sponsored by the Air Force Research Laboratory under agreement number F29601-00-1-0183. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

If you change your address, wish to be removed from this mailing list, or your organization no longer employs the addressee, please notify AFRL/VSSE, 3550 Aberdeen Ave SE, Kirtland AFB, NM 87117-5776.

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

This report has been approved for publication.

JAMES LYKE
Project Manager

KIRT S. MOSER, DR-IV
Chief, Spacecraft Technology Division

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 28-01-2003	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From - To)* 1 Jul 00 – 1 Nov 02

4. TITLE AND SUBTITLE Power-Adaptive Microarchitecture and Compiler Design for Mobile Computing	**5a. CONTRACT NUMBER** F29601-00-1-0183
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Rajiv Gupta, Santosh Pande, Soner Onder	**5d. PROJECT NUMBER** DARP
	5e. TASK NUMBER SE
	5f. WORK UNIT NUMBER BN
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	**8. PERFORMING ORGANIZATION REPORT NUMBER**

The University of Arizona
Dept. of Computer Science
Gould-Simpson Bldg., Rm. 746
Tucson, AZ 85721

Georgia Tech. University
Atlanta, GA 30332

Michigan Tech. University
Houghton, MI 49931

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Space Vehicles Directorate 3550 Aberdeen Ave., SE Kirtland AFB, NM 87117-5776	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-VS-TR-2003-1024

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

13. SUPPLEMENTARY NOTES

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government

14. ABSTRACT

This project considered various sources of power consumption in general purpose high-performance and embedded processors and developed techniques for lowering the power consumption without significant sacrifice in performance. We have designed low power data caches and low power external data buses that can be used for both superscalar and embedded processors. For superscalar processors we have designed low complexity memory disambiguation mechanism, power efficient instruction issue mechanism, and load/store reuse techniques. For embedded processors we have developed techniques that allow us to achieve performance while operating on compacted code and data. The various techniques that were developed have been implemented as part of gcc compiler and the FAST simulation system. Experimentation was

15. SUBJECT TERMS

Power efficient architectures, compiler techniques

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	SAR	20	**19b. TELEPHONE NUMBER** *(include area code)* (505) 846-5812

Table of Contents

Abstract

This project considered various sources of power consumption in general purpose high-performance and embedded processors and developed techniques for lowering the power consumption without significant sacrifice in performance. The developed techniques can be divided into the following three categories:

Low power data caches and data buses.
These techniques were developed techniques for lowering the power consumed by on-chip memory and external data buses associated with the processors. They are useful for both high-performance and embedded processors since in both types of processors on-chip memory and external buses consume significant part of the total power. These techniques are based upon compression/encoding of frequent values. We have also developed compiler support for carrying out data compression for reducing power consumed by the memory subsystem.

Superscalar processors.
In context of high performance processors we have developed low complexity memory disambiguation mechanism, power efficient dynamic instruction issue mechanism, and load/store reuse techniques.

Embedded processors.
In context of embedded processors we developed techniques which allow us to achieve performance while operating on compacted code and data. It is desirable to use compacted code and data beacuse it greatly reduces the power consumed by memory.

The various techniques that were developed have been implemented as part of gcc compiler and the FAST simulation system. Experimentation was carried to demonstrate the utility of the techniques.

1. Introduction

Recently, power consumption has become one of the biggest challenges in high-performance desktop systems. This is because the drive toward increasing levels of performance has pushed clock frequencies higher and has increased the processor complexity. Both increases come at a cost of high power consumption. The costs associated with packaging, cooling and power delivery have thus jumped to the forefront in the microprocessor industry.

To get an idea of the trends in power consumption of today's processor consider the following table taken from the power study on Alpha processors. The Alpha processor family is multi-issue, out-of-order execution high performance processor. We can see clearly the drastic growth of the power and its density as well. This increase would also negatively impact the processor reliability if the power dissipation keeps increasing at this rate. Even though reducing the supply voltage is well known as an efficient way of controlling power consumption, its benefits are more than offset by the increased complexity and frequency. This calls for creative architecture solutions that can focus on high level trade offs between power and performance.

Alpha Model	Power (W)	Frequency(MHz)	Die size(mm^2)	Voltage(V)
21064	30	200	234	3.3
21164	50	300	299	3.3
21264	90	575	314	2.2
21364	>100	>1000	340	1.5

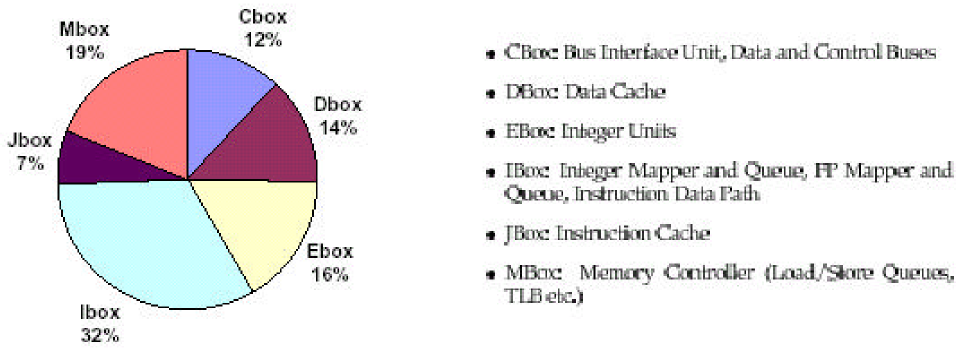
Table 1: Power Trends for Compaq Alpha.

The need to limit the power consumption is also crucial for portable computer platforms such as cellular phones, palm handhelds and pocket PCs because those devices are battery powered. Given the type of applications being written for mobile devices, there is an increasing demand for delivering high quality multimedia output. Since the advances in battery technology are limited, designing low power processors that can operate with a light weight battery for long duration is imperative. Table 2 shows the trends in power consumption for typical embedded processors: ARM7 to ARM10. They are simpler designed in-order execution pipelined processors. We can see from the table that the range of power consumption is only the order of MilliWatt. The reason for this big difference is due to much simpler architecture design. Future embedded processors will have more complex structure such as deeper pipeline length and branch prediction. The designs will resemble high performance processors but under different constraints. Therefore, limiting the power consumption is also becoming more and more important for embedded processors.

ARM Model	Power(mW/MHz)	Frequency(MHz)	Die size(mm^2)	Voltage(V)
ARM720T	0.2	<100	1.8	0.9
ARM920T	0.35	<230	6.0	0.9
ARM1020E	0.8	<375	6.9	0.9

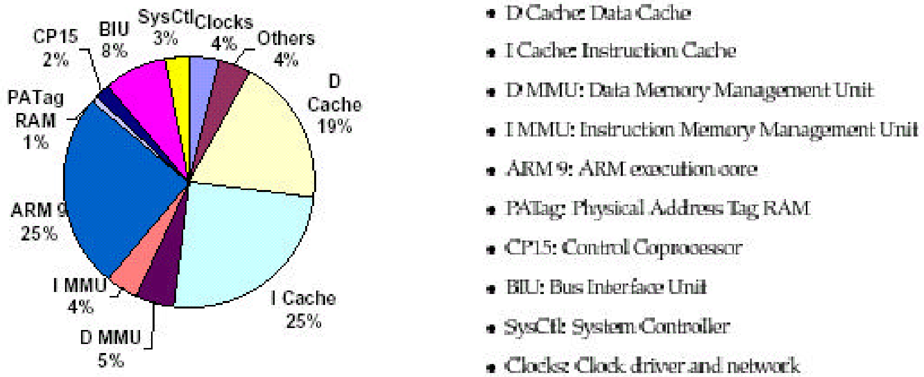
Table 2: Power Trends for ARM Family (0.13 micron technology).

Let us look at the power distribution of the Alpha 21264 and ARM 920T. Figure 1 is taken from the power study in Alpha processors and Figure 2 is taken from a tutorial on low-power processor.



- CBox: Bus Interface Unit, Data and Control Buses
- DBox: Data Cache
- EBox: Integer Units
- IBox: Integer Mapper and Queue, FP Mapper and Queue, Instruction Data Path
- JBox: Instruction Cache
- MBox: Memory Controller (Load/Store Queues, TLB etc.)

Figure 1: Power Distribution for Alpha 21264



- D Cache: Data Cache
- I Cache: Instruction Cache
- D MMU: Data Memory Management Unit
- I MMU: Instruction Memory Management Unit
- ARM 9: ARM execution core
- PATag: Physical Address Tag RAM
- CP15: Control Coprocessor
- BIU: Bus Interface Unit
- SysCtl: System Controller
- Clocks: Clock driver and network

Figure 2: Power Distribution for ARM920T

In the Alpha 21264, the EBox and IBox represent the integer and floating point execution processor core. The summed power expenditure of those two is 47%, much higher than the ARM processor core (25%). This is because the 21264 has complex four-issue out-of-order speculative execution pipelines while the ARM 920T has only a single-issue in-order execution pipeline. The more complex the design the higher power it consumes. The IMMU, DMMU, PATag RAM and the CP15 in ARM 920T contribute in various memory requests and handlings, similar to the function of the MBox of the 21264. The total power of those components is 12%, less than the 19% for MBox in 21264. One reason for that is the ARM uses the virtual address caches to partially avoid the address translation. The BIU and SysCtl in ARM 920T is comparable to the CBox of 21264. The former has total 11% of power consumption and the latter has 12%.

This project addressed the various sources of power consumption in high-performance and embedded processors. The developed techniques can be divided into the following three categories:

- *Low power data caches and data buses.* These techniques were developed techniques for lowering the power consumed by on-chip memory and external data buses associated with the processors. They are useful for both high-performance and embedded processors since in both types of processors on-chip memory and external buses consume significant part of the total power. These techniques are based upon compression/encoding of frequent values. We have also developed compiler support for carrying out data compression for reducing power consumed by the memory subsystem.

- *Superscalar processors.* In context of high performance processors we have developed low complexity memory disambiguation mechanism, power efficient dynamic instruction issue mechanism, and load/store reuse techniques.
- *Embedded processors.* In context of embedded processors we developed techniques which allow us to achieve performance while operating on compacted code and data. It is desirable to use compacted code and data beacuse it greatly reduces the power consumed by memory.

2. Low Power Data Caches and Data Buses

2.1 Frequent Value Locality

By analyzing the behavior of a set of benchmarks, we have demonstrated that a small number of distinct values tend to occur very frequently in memory. On an average, only eight of these frequent values were found to occupy 48% of memory locations for the benchmarks studied. In addition, we demonstrated that the identity of frequent values remains stable over the entire execution of the program and these values are scattered fairly uniformly across the allocated memory.

We have developed three different algorithms for finding frequent values and experimentally demonstrated their effectiveness. Each of these algorithms is designed to suit a different application scenario. We have considered algorithms for the following scenarios and demonstrated their effectiveness.

1. *Find Once for a Given Program*. This method finds a fixed frequent value set through a profiling run which is then used by the application in all later execution runs. This is a purely software based approach. Thus once the values are known, they must be communicated to any hardware based application either through compiler generated code or operating system support. Moreover, if the frequent value set is sensitive to the program input, this approach will cause loss in performance.

2. *Find Once Per Run of the Program.* This method finds a fixed frequent value set during each execution run of the program. The set of values is found through limited online profiling during the initial execution of the program after which the values are fixed and profiling ceases. These values are then used by the application during for remainder of the execution. In other words the fixed frequent value set is found during each execution and therefore the frequent value set being sensitive to program input is not a problem for this method. This approach uses specialized hardware for finding the values. Therefore no compiler or operating support is required to communicate the values to the hardware.

3. *Continuously Changing During Program Run.* This method maintains a changing frequent value set by carrying out continuous profiling of the program during each execution run. Moreover profiling is carried out by specialized hardware. Under this method an application can benefit from adaptation of the frequent value set during a given run.

Additional details of load-store reuse techniques can be found in [7,8].

2.2 Frequent Value Data Cache

Next we demonstrates how this *frequent value* phenomenon can be exploited in designing a cache that trades off performance with energy efficiency. We propose the design of the *Frequent Value Cache* (FVC) in which storing a frequent value requires few bits as they are stored in encoded form while all other values are stored in unencoded form using 32 bits. The data array is partitioned into two arrays such that if a *frequent value* is accessed only the first data array is accessed. This approach greatly reduces the energy consumed by the data cache. The reduction in energy is achieved at the cost of an additional cycle needed to access nonfrequent values. Our results show: (a) the time spent on encoding and decoding of values typically has little or no impact on the cache access time; (b) while the reduction in dynamic energy consumed by a *frequent value* access ranges from 34% to 84%, the increase in the dynamic energy consumed by a *nonfrequent value* access ranges from 0.0005% to 0.0678% for a read and from 0.776% to 4.679% for a write; and (c) experiments with some of the SPEC95 benchmarks show that on an average a 64Kb/64-value FVC provides 28.8% reduction in L1 cache energy 3.38% increase in execution time delay, 26.1% reduction in energy-delay product and 31.3% reduction in power dissipation over a conventional 64Kb cache. Several alternate low power data cache designs and their evaluations can be found in [8,9,10].

2.3 Frequent Value Encoding for Low Power Data Buses

Since the I/O pins of a CPU are a significant source of energy consumption, work has been done on developing encoding schemes for reducing switching activity on external buses. Modest reductions in switching can be achieved for data and address busses using a number of general purpose encoding schemes. However, by exploiting the characteristic of memory reference locality, switching activity at address bus can be reduced by as much as 66%. Till now no characteristic has been identified that can be used to achieve similar reductions in switching activity on the data bus. We have discovered a characteristic of values transmitted over the data bus according to which a small number of distinct values, called frequent values, account for 32% of transmissions over the external data bus. Exploiting this characteristic we have developed an encoding scheme that we call the FV encoding scheme. To implement this scheme we have also developed a technique for dynamically identifying the frequent values which compares quite favorably with an optimal offline algorithm. Our experiments show that FV encoding of 32 frequent values yields an average reduction of 30% (with on-chip data cache) and 49% (without on-chip data cache) in data bus switching activity for SPEC95 and mediabench programs. Moreover the reduction in switching achieved by FV encoding is 2 to 4 times the reduction achieved by the bus-invert coding scheme and 1.5 to 3 times the reduction achieved by the adaptive method. Additional details can be found in [12].

2.4 Data Compression Transformations for Dynamically Allocated Data Structures

We introduce a class of transformations which modify the representation of dynamic data structures used in programs with the objective of compressing their sizes. We have developed the common-prefix and narrow-data transformations that respectively compress a 32 bit address pointer and a 32 bit integer field into 15 bit entities. A pair of fields which have been compressed by the above compression transformations are packed together into a single 32 bit word. The above transformations are designed to apply to data structures that are partially compressible, that is, they compress portions of data structures to which transformations apply and provide a mechanism to handle the data that is not compressible. The accesses to compressed data are efficiently implemented by designing data compression extensions (DCX) to the processor's instruction set. We have observed average reductions in heap allocated storage of 25\% and average reductions in execution time and power consumption of 30\%. If DCX support is not provided the reductions in execution times fall from 30% to 12.5%. Additional details of this technique can be found in [11].

3. Superscalar Processors

3.1 Dynamic Instruction Issue Mechanism

While the central window implementation in a superscalar processor is an effective approach to waking up ready instructions, this implementation does not scale to large instruction window sizes as those that are required by wide issue superscalars of the future. We have developed a new wake-up algorithm that dynamically associates explicit wake-up lists with executing instructions according to the dependences between instructions. Instead of repeatedly examining a waiting instruction for wake-up till it can be issued, this algorithm identifies and considers for wake-up a fresh subset of waiting instructions from the instruction window in each cycle thus reducing the energy consumed by the issue logic. This subset of instructions are the ones present in the wake-up lists of completing instructions. The direct wake-up microarchitecture (DWMA) that we present is able to achieve approximately 80%, 75% and 63% of the performance of a central window processor at high issue widths of 8, 16 and 32 respectively when an effective memory disambiguation mechanism is employed for load speculation. Additional details of this technique can be found in [2].

3.2 Dynamic Memory Disambiguation

Memory dependence prediction allows out-of-order issue processors to achieve high degrees of instruction level parallelism by issuing load instructions at the earliest time without causing a significant number of memory order violations. We developed a simple mechanism which incorporates multiple speculation levels within the processor and classifies the load and the store instructions at run time to the appropriate speculation level. Each speculation level is termed as a color and the sets of load and store instructions are called color sets. We show how this mechanism can be incorporated into the issue logic of a conventional superscalar processor and show that this simple mechanism can provide similar performance to that of more costly schemes resulting in reduced hardware complexity and cost. The performance of the technique was evaluated with respect to the store set algorithm. At very small table sizes, the color set approach provides up to 21% better performance than the store set algorithm for floating point

Spec-95 benchmarks and up to 18% better performance for integer benchmarks using harmonic means. Additional details of this technique can be found in [1].

3.3 Load-Store Reuse

We carried out experimental studies that show that even programs compiled using optimizing compilers contain very high levels of load and store reuse opportunities. The presence of these load and store reuse opportunities can be viewed as an opportunity for reducing on-chip cache activity and hence the energy consumed by the cache. However, taking advantage of this opportunity is a non-trivial task because a load and store reuse mechanism will itself consume energy. To get an overall reduction in energy consumed we must save more energy in the cache than is consumed by the reuse mechanism. Our experiments with an aggressive reuse mechanism resulted in a net increase in energy used.

We carried out the design and evaluation of a reuse mechanism which was carefully tuned to achieve two objectives: (i) capture and eliminate a large fraction of reusable load and store instructions; and (ii) perform reuse using less energy than what is saved in the cache. A number of strategies are used to minimize the activity in the reuse unit. For programs with high levels of reuse we obtain IPC improvements of up to 55% and net cache energy savings of up to 47%. Even more importantly, in programs where little reuse is found, the net increase in energy consumed is less than 3%. In contrast to traditional filter cache designs which trade-off energy reductions with higher execution times, our approach reduces both energy and execution time. Additional details of load-store reuse techniques can be found in [4,5,6].

3.4 Functional Unit Management

We have developed a novel approach which combines compiler, instruction set, and microarchitecture support to turn off functional units that are idle for long periods of time for reducing static power dissipation by idle functional units using power gating. The compiler identifies program regions in which functional units are expected to be idle and communicates this information to the hardware by issuing directives for turning units off at entry points of idle regions and directives for turning them back on at exits from such regions. The microarchitecture is designed to treat the compiler directives as hints ignoring a pair of off and on directives if they are too close together. The results of experiments show that some of the functional units can be kept off for over 90% of the time at the cost of minimal performance degradation of under 1%. Additional details of this technique can be found in [3].

4. Embedded Processors

4.1 Profile Guided Selection of ARM and Thumb Instructions

The ARM processor core is a leading processor design for the embedded domain. In the embedded domain, both memory and energy are important concerns. For this reason the 32 bit ARM processor also supports the 16 bit Thumb instruction set. For a given program, typically the Thumb code is smaller than the ARM code. Therefore by using Thumb code the I-cache activity, and hence the energy consumed by the I-cache, can be reduced. However, the limitations of the Thumb instruction set, in comparison to the ARM instruction set, can often lead to generation of poorer quality code. Thus, while Thumb code may be smaller than ARM code, it may perform poorly and thus may not lead to overall energy savings. We performed a comparative evaluation of ARM and Thumb code to establish the above claims and present analysis of Thumb instruction set restrictions that lead to the loss of performance. We developed profile guided algorithms for generating mixed ARM and Thumb code for application programs so that the resulting code gives significant code size reductions without loss in performance. Our experiments show that this approach is successful and in fact in some cases the mixed code outperforms both ARM and Thumb code. Additional details of this technique can be found in [14].

4.2 Enhancing the Performance of 16 Bit Thumb Code Through Instruction Coalescing

The ARM processor core is a leading processor design for the embedded domain. In the embedded domain, both memory and energy are important concerns. The 32 bit ARM processor addresses these concerns by supporting the 16 bit Thumb instruction set. For a given program, the Thumb code is typically 30% smaller than the ARM code and yields savings in instruction cache energy. However, the limitations of the Thumb instruction set, in comparison to the ARM instruction set, can often lead to generation of additional instructions. Thumb instruction counts are observed to exceed ARM instruction counts by 9% to 41%.

This paper presents a novel approach that enables Thumb code to perform like ARM code. We have observed that throughout Thumb code we can spot Thumb instruction pairs that are equivalent to a single ARM instruction. Therefore we have developed enhancements to the Thumb instruction set and the processor microarchitecture that allows a Thumb instruction pair to be translated into a single ARM instruction at runtime. We enhance the Thumb instruction set by incorporating Augmenting Instruction set eXtensions (AIX). Augmenting instructions are a new class of instructions which are entirely handled in the decode stage of the processor, that is, they do not travel through the remaining stages of the pipeline. A Thumb instruction pair that can be combined into a single ARM instruction is replaced by an AIXThumb instruction pair by the *compiler.* The AIX instruction is coalesced with the immediately following Thumb instruction to generate a single ARM instruction during the decode stage of the processor where the translation of Thumb instructions into ARM instructions takes place. The enhanced microarchitecture ensures that coalescing does not introduce pipeline delays and therefore coalescing results in reduction of both instruction counts and cycle counts. In our approach the compiler is responsible for identifying Thumb instruction pairs that can be safely coalesced because the safety of coalescing requires global analysis of the program and thus cannot be carried out entirely by the hardware. The ARM instruction set supports predicated execution while Thumb does not. Through the use of AIX instructions and coalescing hardware we are also able to incorporate a highly effective implementation of predicated execution in 16 bit mode.

Our experiments show that instruction counts for Thumb code exceed that of ARM code by 9% to 41%. However, instruction counts for AIXThumb code are lower than those for Thumb code by upto 25%. Thus, the combination of AIX instructions and instruction coalescing enhances the performance of 16 bit code considerably. Additional details of this technique can be found in [15].

4.3 Bit Section Instruction Set Extension of ARM

Programs that manipulate data at subword level, i.e. bit sections within a word, are common place in the embedded domain. Examples of such applications include media processing as well as network processing codes. These applications spend significant amounts of time packing and unpacking narrow width data into memory words. The execution time and memory overhead of packing and unpacking operations can be greatly reduced by providing direct instruction set support for manipulating bit sections. In this work we developed the Bit Section eXtension (BSX) to the ARM instruction set. We selected the ARM processor for this research because it is one of the most popular embedded processor which is also being used as the basis of building many commercial network processing architectures. We present the design of BSX instructions and their encoding into the ARM instruction set. We have incorporated the implementation of BSX into the Simplescalar ARM simulator from Michigan. Results of experiments with programs from various benchmark suites show that by using BSX instructions the total number of instructions executed at runtime by many transformed functions are reduced by 4.26% to 27.27% and their code sizes are reduced by 1.27% to 21.05%. Additional details of this technique can be found in [16].

4.4 Bitwidth Aware Global Register Allocation

Multimedia and network processing applications make extensive use of subword data. Since registers are capable of holding a full data word, when a subword variable is assigned a register, only part of the register is used. New embedded processors have started supporting instruction sets that allow direct referencing of bit sections within registers and therefore multiple subword variables can be made to simultaneously reside in the same register without hindering accesses to these variables. However, a new register allocation algorithm is needed that is aware of the bitwidths of program variables and is capable of packing multiple subword variables into a single register. This work develops one such algorithm.

The algorithm we propose has two key steps. First, a combination of forward and backward data flow analyses are developed to determine the bitwidths of program variables throughout the program. This analysis is required because the declared bitwidths of variables are often larger than their true bitwidths and moreover the minimal bitwidths of a program variable can vary from one program point to another. Second, a novel interference graph representation is designed to enable support for a fast and highly accurate algorithm for packing of subword variables into a single register. Packing is carried out by a node coalescing phase that precedes the conventional graph coloring phase of register allocation. In contrast to traditional node coalescing, packing coalesces a set of interfering nodes. Our experiments show that our bitwidth aware register allocation algorithm reduces the register requirements by 10% to 50% over a traditional register allocation algorithm that assigns separate registers to simultaneously live subword variables. Additional details of this technique can be found in [17].

4.5 Dual Memory Banks in Embedded Processors

Many modern embedded processors support partitioned memory banks (also called X-Y memory or dual bank memory) along with parallel load/store instructions to achieve code density and/or performance. In order to effectively utilize the parallel load/store instructions, the compiler must partition the values to be loaded or stored into X or Y bank. This work develops a post-register allocation solution to merge the generated load/store instructions into their parallel counter-parts. Simultaneously, our framework performs allocation of values to X or Y memory banks.

We first remove as many load/stores and register-register moves through an excellent iterated coalescing based register allocator by Appel and George. Our goal is then to maximally parallelize the generated load/stores using a multi-pass approach with minimal growth in terms of memory requirements. The first phase of our approach attempts the merger of load stores without replication of values in memory. We model this problem in terms of a graph coloring problem in which each value is colored X or Y. We then construct a Motion Scheduling Graph (MSG) based on the range of motion for each load/store instruction. MSG reflects potential instructions which could be merged. We propose a notion of pseudo-fixed boundaries so that the load/store movement is minimally affected by register dependencies. We prove that the coloring problem for MSG is NP-complete. We then propose a heuristic solution, which minimally replicates load/stores on different control flow paths if necessary. Finally, the remaining load/stores are tackled by register rematerialization and local conflicts are eliminated. Registers are re-assigned to create motion ranges if opportunities are found for merger which are hindered by local assignment of registers. We show that our framework results in parallelization of a large number of load/stores on these emerging processors without much growth in data and code segments. Additional details of this technique can be found in [13].

5. Infrastructure Development

In order to carry out the compiler work we made use of the gcc compiler, both for superscalar and embedded research. The simulation infrastructure used in this work consisited of two systems. For the work on frequent values and superscalar we made use of the FAST simulator generation system. This system has been greatly enhanced as part of this project. In addition to implementing all of the newly developed techniques, we also for the first time incorporated power and energy estimation models into the system. For the work on embedded systems we started with a port of Simplescalar to ARM. We extended it by implementing the Thumb instruction set and we also modified it to make use of newlib which is the library of choice for embedded systems.

6. Other Contributions

In addition to developing low power techniques and the infrastructure to carry out low power research, this project has also made contributions in other ways. First, the work carried out under this project has been widely disseminated through publications in high quality conferences and journals [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]. Second, many students have actively contributed this project at each of the participating institutions. Some of these students have graduated and added to the much needed workforce in the area of low power computing. In particular, Youtao Zhang and Jun Yang, who have authored some of the papers related to this project, have completed their PhD dissertations.

References

1. S. Onder, ``Cost Effective Memory Dependence Prediction Using Speculation Levels and Color Sets,'' *The Eleventh International Conference on Parallel Architectures and Compilation Techniques* (PACT), Charlottesville, Virginia, September 2002.

2. S. Onder and R. Gupta, ``Instruction Wake-up in Wide Issue Superscalars,'' *European Conference on Parallel Computing* (Euro-Par), LNCS 2150, Springer Verlag, pages 418-427, Manchester, UK, August 2001.

3. S. Rele, S. Pande, S. Onder, and R. Gupta, ``Optimizing Static Power Dissipation by Functional Units in Superscalar Processors,'' *International Conference on Compiler Construction* (CC), LNCS 2304, Springer Verlag, pages 261-275, Grenoble, France, April 2002.

4. S. Onder and R. Gupta, ``Load and Store Reuse Using Register File Contents,'' *ACM 15th International Conference on Supercomputing* (ICS), pages 289-302, Sorrento, Naples, Italy, June 2001.

5. J. Yang and R. Gupta, ``Energy-Efficient Load and Store Reuse,'' *ACM/IEEE International Symposium on Low Power Electronics and Design* (ISLPED), pages 72-75, Huntington, CA, August 2001.

6. J. Yang and R. Gupta, ``Load Redundancy Removal through Instruction Reuse,'' *International Conference on Parallel Processing* (ICPP), pages 61-68, Toronto, Canada, August 2000.

7. J. Yang and R. Gupta, ``Frequent Value Locality and its Applications,'' *ACM Transactions on Embedded Computing Systems* (TECS), special inaugural issue on memory systems, Vol. 1, No. 1, pages 79-105, 2002.

8. Y. Zhang, J. Yang, and R. Gupta, ``Frequent Value Locality and Value-Centric Data Cache Design,'' *ACM 9th International Conference on Architectural Support for Programming Lanuguages and Operating Systems* (ASPLOS), pages 150-159, Cambridge, MA, November 2000.

9. J. Yang and R. Gupta, ``Energy Efficient Frequent Value Data Cache Design,'' *IEEE/ACM 35th International Symposium on Microarchitecture* (MICRO), Istanbul, Turkey, November 2002.

10. J. Yang, Y. Zhang and R. Gupta, ``Frequent Value Compression in Data Caches,'' *IEEE/ACM 33rd International Symposium on Microarchitecture* (MICRO), pages 258-265, Monterey, CA, December 2000.

11. Y. Zhang and R. Gupta, ``Data Compression Transformations for Dynamically Allocated Data Structures,'' *International Conference on Compiler Construction* (CC), LNCS 2304, Springer Verlag, pages 14-28, Grenoble, France, April 2002.

12. J. Yang and R. Gupta, ``FV Encoding for Low-Power Data I/O,'' *ACM/IEEE International Symposium on Low Power Electronics and Design* (ISLPED), pages 84-87, Huntington, CA, August 2001.

13. X. Zhuang, S. Pande and J. Greenland, ``A Framework for Parallelizing Load/Stores on Embedded Processors,'' *11th International ACM/IEEE Conference on Parallel Architectures and Compilation Techniques* (PACT), pages 68-79, Charlottesville, Virginia, September 2002.

14. A. Krishnaswamy and R. Gupta, ``Profile Guided Selection of ARM and Thumb Instructions,'' *ACM SIGPLAN Joint Conference on Languages Compilers and Tools for Embedded Systems & Software and Compilers for Embedded Systems* (LCTES/SCOPES), pages 55-63, Berlin, Germany, June 2002.

15. A. Krishnaswamy and R. Gupta, ``Mixed Width Instruction Sets,'' to appear in special issue on code compression in *Communications of the ACM* (CACM), August 2003.

16. B. Li and R. Gupta, ``Bit Section Instruction Set Extension of ARM for Embedded Applications,'' *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems* (CASES), pages 69-78, Grenoble, France, October 2002.

17. S. Tallam and R. Gupta, ``Bitwidth Aware Global Register Allocation,'' *30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (POPL), pages 85-96, New Orleans, LA, January 2003.

DISTRIBUTION LIST

DTIC/OCP
8725 John J. Kingman Rd, Suite 0944
Ft Belvoir, VA 22060-6218 1 cy

AFRL/VSIL
Kirtland AFB, NM 87117-5776 1 cy

AFRL/VSIH
Kirtland AFB, NM 87117-5776 1 cy

The University of Arizona
Dept. of Compute Science
Gould-Simpson Bldg., Rm. 246
Tuscon, AZ 85721 1 cy

Official Record Copy	
AFRL/VSSE/James Lyke	1 cy