Exploring the Processor and ISA Design for Wireless Sensor Network Applications

Shashidhar Mysore, Banit Agrawal, Frederic T. Chong, and Timothy Sherwood
Department of Computer Science
University of California, Santa Barbara
{shashimc, banit, chong, sherwood}@cs.ucsb.edu

Abstract

Power consumption, physical size, and architecture design of sensor node processors have been the focus of sensor network research in the architecture community. What lies at the foundation for these research is the hardware-level design which determines the boundaries for achievable utility and performance. Architecture design and evaluation, however, cannot be accomplished independent of the applications and software that run on these sensor nodes. On one hand, some researchers have proposed architectures that can cater to a variety of application classes while trading off on some performance improvements. On the other hand, a set of application-specific architectures have been proposed which perform certain operations extremely well but are not versatile enough to run a variety of applications.

This paper provides a design space exploration and optimizations platform to characterize the processor and ISA design tailored for a particular application or a class of applications. We collect a wide variety of sensor network applications to create a comprehensive benchmark suite called the WiSeNBench. We then present a careful profiling of these benchmark applications using an ARM simulator to identify some of the key characteristic behaviors. This also opens up avenue for a possible re-look at the classes of applications that could be supported on next-generation sensor networks and efficient architectural designs to enable these applications.

1 Introduction

Sensor network applications include environmental monitoring, structural sensing, battlefield communication, traffic, health, security monitoring, and other automation techniques. Consequently, sensor network research involves architecture, application optimization, communication protocol design, and developing efficient communication hardware. Small form factor, low-power budget, low-resource availability, and real-time requirements are some of the characterizing factors of sensor nodes. Each of the above is an additional constraint imposed on the designers of such networks. Given these various sites of improvement, in this paper we focus on sensor network applications, their characteristics, and explore ways in which they can influence the design of the underlying architecture.

Knowledge of the underlying hardware aids in efficient software development. A top-down approach towards software development may be well suited in scenarios where the processor architecture is already well defined and scope for optimization, if any, is just on the software front. With sensor applications and sensor network research, however, the scenario is quite different. Sensor applications require a special purpose hardware suitable to cater to a different set of requirement. Medical applications for example, need highly non-intrusive tiny sensors which are usually harmless to the human body as foreign-bodies . Whereas sensors spread on a military terrain have to be more tolerant to physical impacts and wide operating temperature range. These physical characteristics in which the sensors are placed and the difference in their utility makes it important to do research on sensor networks on a case by case basis.

Unique set of findings for a sensor application is good, but the sheer number of applications sensor networks are finding today rules out the feasibility of developing processors unique to each and every application. This brings us to a point where we will have to trade between the amount of customization available on a processor and the performance-cost benefits one would like to achieve. We believe that this aspect needs to reflect on the research focus in this area. The major contributions of this paper are a) To study some of the most important applications for sensor networks, including those in TinyBench [8] and SenseBench [27]. This includes a careful profiling of application behavior and its microarchitectural implications. b) To characterize the workload that is prominently visible among sensor network applications. c) To characterize the workload that is unique to a class of sensor network applications. d) To propose optimizations to existing sensor network architectures based on the observations made in (a,b,c).

In order for the characterization to be useful, the set of applications used for the purpose should be representative of the domain being analyzed. To this end, we present a thorough survey on sensor network applications, classify them based on the core functionality an application serves, and characterize each class independently.

Microarchitectural characteristics of programs serves as an important aspect in making architectural design decisions. The code size of an application influences critical decisions

made based on the footprint of a program. The execution pattern and the bottleneck region optimizations improve the response time. The memory access patterns (both spatial and temporal) provide avenues for various memory placement and memory design related optimizations. Studying the composition of the dynamic instruction stream aids in instruction set architecture optimization. Dynamic instruction execution sequences help architects not only understand the behavior of a program but also help in improving functional unit design to reduce the overall area or even increase transistor utility.

In this paper, we architecturally characterize all sensor network applications and compare the results for different class of sensor network applications. First, we collect all representative set of applications from TinyOS benchmarks [10], TinyBench [8], and from [28] and we build some of our own simple applications. We then classify them into various classes of applications to compare the architectural findings for different classes. Specifically, we study the following architectural characterizations and optimizations.

- Find the most frequently executed instructions
- Find the most frequently executed pair and triple of instructions
- Instruction-set and footprint optimization by combining frequently executed pair of instructions
- Memory behavior of the applications

2 Related Works

Wireless sensor network (WSN) has identified its applications in many disciplines including environmental engineering, military/security applications, and civil engineering. The main challenge is to make the sensor node a low-energy device so that it can scavenge energy from various sources. From an architectural perspective, the processor used inside should be designed to consume less energy while coordinating with other components in a manner which minimizes the total energy consumption. While initial microprocessors used in a Mote are of ATMEL (AVR) family, they were synchronous processors and not specifically designed for the sensor node applications. Past research [5, 9] have shown that an asynchronous processor design is the ideal choice for microprocessor to save energy, whereas in synchronous design energy could be wasted in clocking the synchronous processor and other components. ARM cores and variations of ARM cores such as StrongARM and XScale have also been used to see the energy-performance tradeoffs for sensor node applications.

Ekanayake et al. [5] have designed a low-energy asynchronous processor that only takes 24 pJ/instruction, whereas ATMel or ARM family processors takes energy in the order of nJ/instruction. They design a new ISA, new coprocessors which includes timers, radio units, and processor core for low energy design. But they do not provide any motivation or reasoning behind selecting this instruction set that could help the architecture community to understand the ISA design better in tandem with sensor network benchmarks. Similarly, Hemstead et al. [9] have also designed an event processor along with some hardware accelerators to improve the performance and energy consumption. Nazhandali et al. [28] have designed a sub-threshold sensor network processor that can run at very low voltage and hence at very low frequency. This sensor network processor is a CISC architecture and it consumes 1.6 pJ/instruction.

While these processors are very well optimized for all sensor network applications, some key insights on designing a particular ISA for these architectures would be very helpful. Some past benchmarks such as MediaBench [25] designed for multimedia applications, NetBench [26] , and MiBench [7] for embedded applications, do exhibit an architectural characterization and provide some insight/platform to build an optimized architecture. Ideally, ISA design or other components design should result from the characterization of all sensor network applications on a base processor. There are two benchmarks for sensor network applications TinyBench [8] and SenseBench [27] for this purpose. In TinyBench, all the applications are targeted for TinyOS and does not scale well for a general study of architecture exploration. While SenseBench provides a set of generalized benchmarks, it does not cover all the applications and the architectural characterization is limited to code size, energy per benchmark and real-time performance requirement. Instead we make our benchmark more comprehensive by extensively scanning research literature and also performing a large set of architectural characterization. Architectural characterization will also vary from different class of sensor node applications as we move from security/military applications to environmental/structural monitoring applications where the computational requirements are different. Therefore, we present a complete architectural characterization of the all sensor network applications (which we call *WiSeNBench*) and we then group them into different classes of applications and compare our findings for these different classes of applications.

3 WiSeNBench: Wireless Sensor Network Benchmark

In this section, we describe our benchmark suite *WiSeNBench* in detail. *WiSeNBench* consists of a large spectrum of sensor network applications and core algorithms that are mainly used inside sensor network applications. Identifying and collecting this set of applications required non-trivial efforts due to a plethora of wireless sensor network applications and many more applications which are not yet explored. To also make sure that these applications cover many different classes of applications, we had to rigorously scan the research literature in different domains of research in wireless sensor network. Specifically, we look for various cryptographic applications [24], security protocols [29], digital signal processing (DSP) applications [3], hashing techniques [36], message digest [31], random number generator [33], compression techniques [30], routing [4], applications related to computational geometry [23], some basic algorithms [27], and many pertinent survey papers [1, 22].

Based on this study, we identify the potential applications that will run on the sensor network processor and collect the optimized code for these applications. The main problem in the collection phase was to find the optimized code for one generic language instead of code written in a specialized language (such as nesC [6]) or targeted for a very specific architecture [8]. While we could find the optimized code

Table 1: WiSeNBench: Different classes of sensor network applications with its brief description and reference.

Class of applications	Application [reference]	Brief Description
Compression/ Hash/Digest	CRC [14]	Cyclic redundancy check (CRC) generates a checksum to correct errors for a block of data.
	RLE [15]	Run length encoding (RLE) is a simple data compression technique which scans the data and then stores the data with associated count.
	Hash algorithms [19]	A set of hash algorithms to produce a fix-length data for indexing and better search.
	Bloom filter [18]	Bloom filter consists of a set of hash algorithms and a hash table to resolve containment queries.
	MD5 [31]	Message digest algorithm 5 (MD5) is a powerful hash function to create a 128-bit key for integrity checking.
Routing/Radio	SMAC [37]	S-MAC is an energy-efficient medium access control (MAC) protocol.
	Ad-hoc routing [34]	A routing technique in a distributed multihop wireless network with a shared wireless channel.
	EnergyEff routing [32]	Its an unidirectional level-hop routing algorithm to assign each intermediate nodes a level to reach the sink node.
Cryptography/ Security	RC5 [16]	RC5 is a fast block cipher for RSA data security and has variable key size and rounds. We consider both encryption and decryption in this case.
	TEA [20]	Tiny encryption algorithm (TEA) is a block cipher which is very simple to design and code size is also very small.
	Crypto [13]	Crypto3 is a cryptographic technique to encrypt password in an Unix based system.
	RC6 [17]	RC6 is an advanced version of RC5 for data security.
	SPINS [29]	SPINS is a security protocol for sensor network which has two components (1) SNEP (2) TESLA
Computational geometry	Voronoi diagrams [12]	Voronoi diagrams is a special decompostion of metric space using a set of distinct points.
	Delaunay triangulation [11]	Delaunay triangulation for a set of points is the triangulation of points with some specific property.
	Localization [21]	Localization algorithms for sensor node to approximate its position.
Digital signal processing	FFT	Fast fourier transform (FFT) algorithm is a fast and efficient algorithm to calculate DFT and IDFT.
	FIR	Finite impulse response (FIR) is a type of digital filter and its response finally settles down to zero.
	IIR	Infinite impulse response (IIR) filter has non-zero response over a long period of time.
	Speech filtering	Some specialized filters such as Kalman filters for speech processing.
Basic core algorithms	sorting algos	This application contains 7 types of sorting algorithms with different runtime complexity and implementation.
	Fibonacci	Fibonacci numbers are special numbers from a well defined recurrence relation.
	MatrixMul	This is a simple matrix multiplication algorithm for small sizes of matrix.
	Binary search	Binary search algorithm is a typical search algorithm for a sorted list.
	Min-max finder	Finds the minimum and maximum values in a list of values.
	majority consenus	Finds the majority value in a list of values.
	Top10	Finds the top 10 values in a list of values.
	sum-array	Provides the sum of all values present in a list.

for many applications (about 70% of those in the suite), we also develop optimized code from scratch for various applications (about 30%) for which we could not find optimized code written in a generic language (such as C).

After the identification and collection phase, we accomplish a good representative set of a wide variety of sensor network applications. To characterize it better, we categorize these benchmarks into various classes: 1) Compression 2) Routing 3) Security 4) Computational geometry 5) DSP 6) Basic algorithms. Table 1 shows different classes of benchmarks with a brief description and the reference if applicable.

Although all the benchmarks can be run as a separate standalone binary, we preferred to combine all the applications into one single binary and create a unified framework. The motivation behind creating a this framework is to enable centralized control of inputs to these benchmarks, a simpler simulation platform, and easier statistics collection.

4 Experimental Setup

In this section, we explain our experimental setup which includes the compilation of the unified benchmark, simulation, and the statistics collection. We use the ARM SimpleScalar simulator [35], which was extended from the original SimpleScalar simulator [2]. We setup cross-gcc suites for ARM processor to compile the benchmark and make a single static binary as SimpleScalar ARM would not handle the dynamic binary file with shared object files. Since we are only interested in the architecture of a very simple RISC processor with no out-of-order execution, and no caches, we conservatively use sim-safe simulator for our experimentation. We modify the code of sim-safe simulator to extract the related statistics which is explained in detail below along with other implementation issues.

Using the gcc cross-compilation suites we first create the binary and then make sure that we collect statistics based on

various functions in the binary . Although Some benchmarks use multiple functions, we combined the results of all related functions. We use ARM disassembler in *binutils* toolset to disassemble the generated binary and feed this disassembled file to a PERL script which parses the disassembled file and generates a C header file containing a large structure with all the function initialization. Each function initialization mainly consists of following three entries: *<function-name,start-pc,end-pc>*. This header file is used with the sim-safe simulator. Since we intend to do architectural characterization based on each function, we identify every function range in the sim-safe execution loop and do relevant processing required to collect the statistics. At the end of simulation, the simulator prints all the related statistics to a file. Specifically, we consider the following statistics:

- Codesize - This is the footprint of the application and a crucial factor in the design of a resource constrained sensor node processor.
- Memory accesses - We characterize the memory access behavior by finding out number of load/stores instructions executed by a particular application.
- Loads - Percentage of loads in the memory accesses is another important factor in terms of energy consumption for sensor network device.
- Frequent instructions - To make some instructions power-conscious during their execution, it is important to understand the distribution of frequently executed instruction. Architects can optimize these instructions for energy savings and performance.
- Frequent pair of instructions - We also find the frequent pair of instructions while executing a specific function. This will give us an idea about the quantitative improvement we can achieve in energy savings and performance. This can also be done statically to improve the code size by combining frequent instruction pairs into a single instruction.

5 Results

Having described the complete benchmark suite in Section 3 and the experimental setup in Section 4, we now present some characterization results based on the parameters discussed in the previous section.

5.1 CodeSize

Codesize or the footprint of a program is a very important parameter as it directly signifies the amount of memory required for a particular application. We statically compute the codesize for ARM ISA and present the results in Figure 1. We can see that most of the DSP applications have much larger code size, whereas most of other applications have code size less than 500 bytes except MD5 and RC5.

5.2 Memory Accesses

A whole host of program optimization techniques are aided by the knowledge of memory access behavior of a program. We compute the total memory accesses to signify the memory intensive behavior of each benchmark. We present the results as percentage of memory accesses (load/stores) as a percentage of total executed instructions. Figure 2 shows the percentage of memory accesses for all of the benchmarks. We can see that most of applications have 40-60% memory accesses, while some of the applications such as DSP applications, Fibonacci numbers accesses memory through more than 60% of the executed instructions.

5.3 Load accesses

To characterize the memory accesses further, we calculate the percentage of loads in the memory accesses. As we can clearly see from the Figure 3, percentage of loads is larger than 50% for almost all applications which signifies that there are more loads than stores (which is also intuitive). We also find that basic algorithms have higher percentage of loads (specially sorting algorithms), whereas for DSP applications loads and stores are almost evenly distributed.

5.4 Frequent instructions

We find the frequent instructions for each application in *WiSeNBench* to get an idea if any particular instruction is suitable for optimization in terms of energy or performance. Although we gather results for all the benchmarks, due to space constraints we only present the results for two applications. These two applications represent a class of applications: 1) TEA - from cryptographic class 2) FIR - DSP class. We present the frequent instructions as the percentage of total instructions. For TEA, the results are shown in the graph on the left in Figure 4 and we find that almost 7 frequent instructions account for 95% of instructions, with load and add instructions at the top of the list. Similar results are presented for FIR application in graph on the right in Figure 4. We find that for FIR applications instructions are widely distributed with load and branch instructions at the top of the list with about 28% and 10% respectively of total instructions .

5.5 Frequent pairs of instructions

We also find the frequent pairs of instructions to further characterize the application behavior to seek any possible optimization of combining frequent pair of instructions. Although there is a tradeoff in combining the instructions, it can certainly result in lower code size, possibly lesser energy consumption and possibly improved performance. Once again, for this analysis we present the results for only TEA and FIR applications and they are shown in Figure 5. We see a very similar behavior found in frequent instructions analysis. For TEA applications, frequent pairs are attributed to a small number of pairs, whereas for FIR applications it is distributed to many pairs of instructions. Interestingly, the 2nd and 3rd frequent pairs $< mov, load >$ and $< load, mov >$ are same in both FIR and TEA applications. In TEA, we find that $< add, load >$ instruction pair is found to be more frequent (may be due to array accesses), whereas in FIR $< load, br >$ is found to be more frequent at about 9%.

6 Conclusion

Sensor networks, though classified under one umbrella, have varied requirements and utilities. To efficiently design protocols, architectures, and applications, it is important to characterize the applications and categorize them based on their effects at the microarchitectural-level. We present a new set of comprehensive benchmarks called the WiSeNBench in a unified framework. We show that WiSeNBench effectively

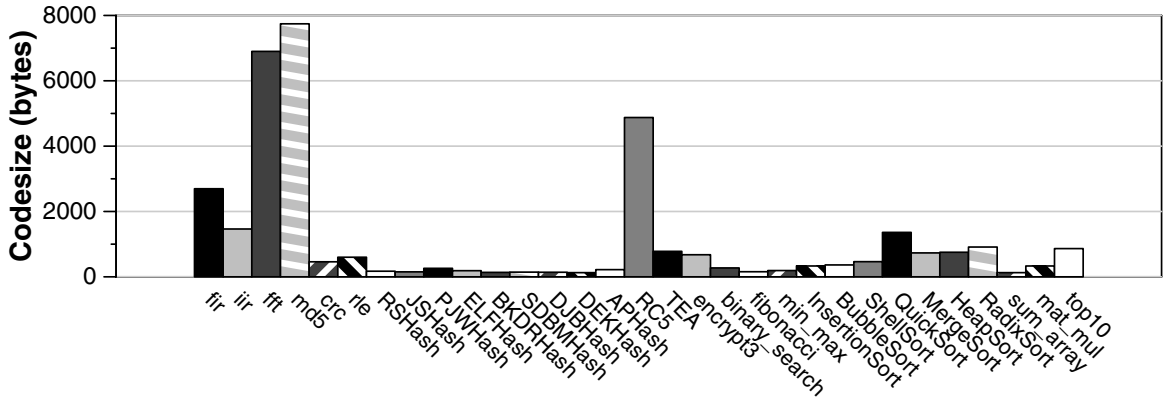

Figure 1: Footprint of all application in our benchmark. The y-axis shows the code size in bytes for each benchmark on x-axis.

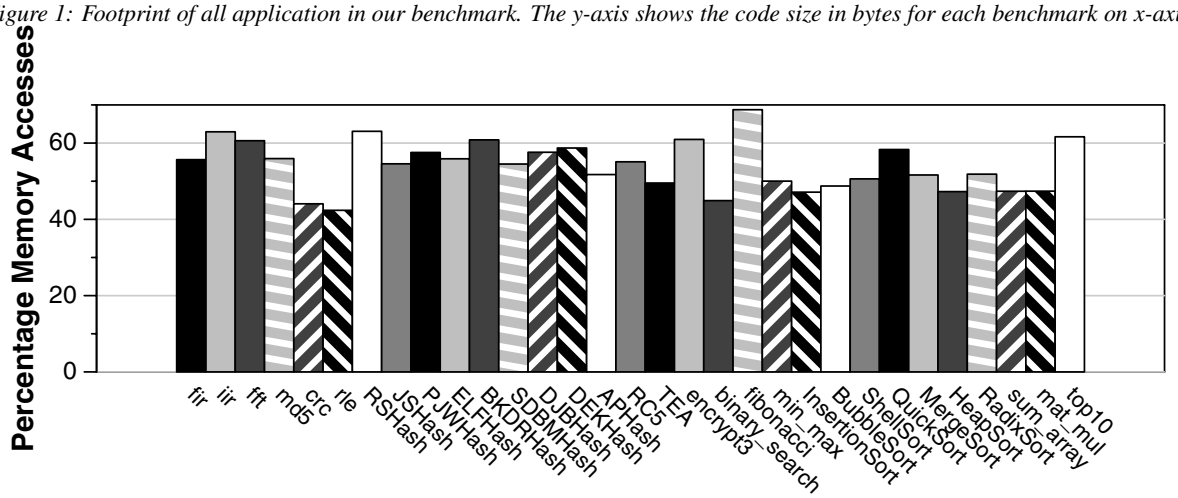

Figure 2: Memory access behavior of all applications in WiSeNBench. The y-axis shows the memory accesses as a percentage of total instructions for each benchmark on x-axis.

characterizes the myriad application set sensor network often deal with and provides insights into the behavior of each of these. Architectural characterization was performed using ARM SimpleScalar Simulator. We find that the code size of MD5, RC5 and DSP applications is larger compared to other categories. On the instruction stream composition front, we find that the set of basic algorithms execute larger percentage of loads and that the DSP applications. Also, we believe that there is a potential for further research on ISA design based on the results presented on frequent instructions and frequent pairs of instructions.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. 2002.

[2] Douglas C. Burger and Todd M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-1997-1342, 1997.

[3] C. Chiasserini and R. Rao. The concept of distributed digital signal processing in wireless sensor networks, 2002.

[4] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 134–146, New York, NY, USA, 2003. ACM Press.

[5] Virantha Ekanayake, IV Clinton Kelly, and Rajit Manohar. An ultra low-power processor for sensor networks. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 27–36, New York, NY, USA, 2004. ACM Press.

[6] David Gay, Philip Levis, J. Robert von Behren, Matt Welsh, Eric A. Brewer, and David E. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI*, pages 1–11. ACM, 2003.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B.Brown. Mibench: A free, commercially representative embedded benchmark suite. In *In IEEE Workshop in workload characterization*, 2001.

[8] Mark Hempstead, David Brooks, and Matt Welsh. TinyBench: The Case For A Standardized Benchmark Suite for TinyOS Based Wireless Sensor Network Devices. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors(EmNets'04)*, November 2004.

[9] Mark Hempstead, Nikhil Tripathi, Patrick Mauro, Gu-Yeon Wei, and David Brooks. An ultra low power system architecture for sensor network applications. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 208–219, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[11] http://mathworld.wolfram.com/DelaunayTriangulation.html. Delaunay triangulation, 1999.

[12] http://mathworld.wolfram.com/VoronoiDiagram.html. Voronoi diagrams, 1999.

[13] http://michael.dipperstein.com/. Crypto3: Unix password cryptography algorithm, 2004.

[14] http://michael.dipperstein.com/. Cyclic redunancy check (crc) implementation, 2004.

[15] http://michael.dipperstein.com/rle/index.html. Run length encoding (rle) discussion and implementation, 2004.

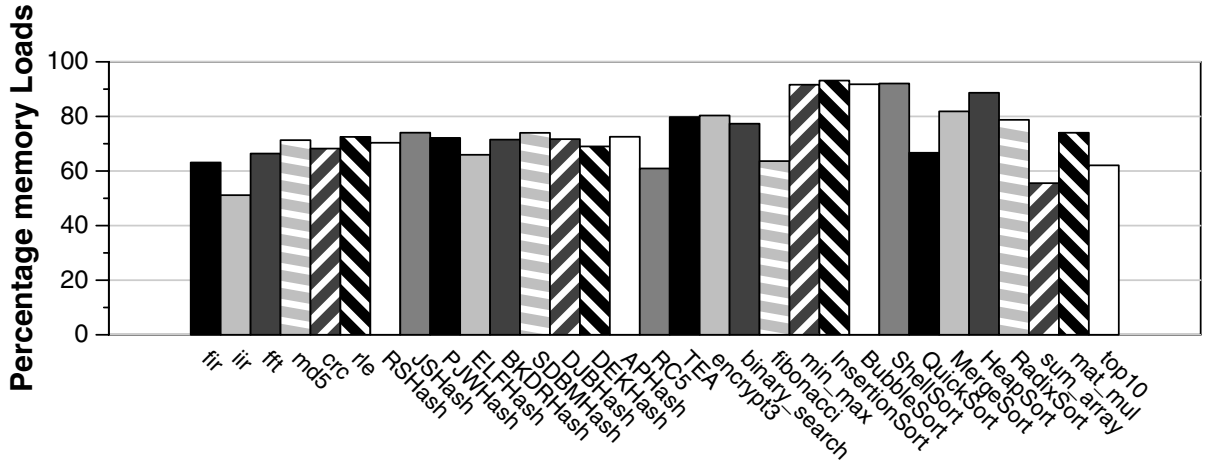

Figure 3: Percentage of loads in the memory accesses is shown on y-axis for each application in WiSeNBench. This gives an idea about the load/store contribution and scope of read optimization if percentage of loads is higher.

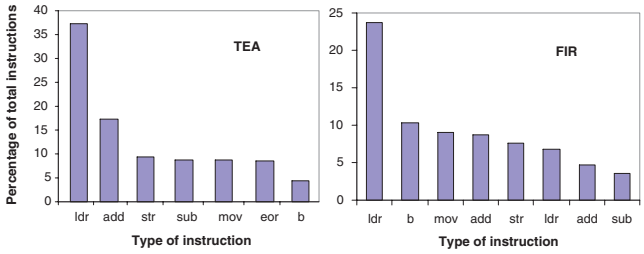

Figure 4: Frequent instructions for the benchmarks is shown on the x-axis. The frequency of these instructions is shown as the percentage of the total instruction executed on the y-axis.

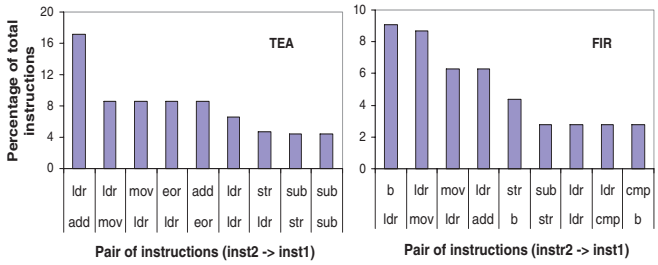

Figure 5: Frequent pairs of instructions is shown on the x-axis. The frequency of these pairs of instructions is shown as the percentage of the total instruction executed on the y-axis.

[16] http://www.bearcave.com/cae/chdl/rc5.html. Rc5 algorithm implementation, 1995.

[17] http://www.codeproject.com/cpp/hexenc.asp. Rc6 encryption and decryption, 2002.

[18] http://www.partow.net/programming/hashfunctions/index.html. Bloom filters implementation, 2006.

[19] http://www.partow.net/programming/hashfunctions/index.html. General purpose hash function algorithms, 2006.

[20] http://www.simonshepherd.supanet.com/tea.htm. Tiny encryption algorithm (TEA), 2006.

[21] Lingxuan Hu and David Evans. Localization for mobile sensor networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 45–57, New York, NY, USA, 2004. ACM Press.

[22] Holger Karl and Andreas Willig. A short survey of wireless sensor networks, 2003.

[23] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, New York, NY, USA, 2000. ACM Press.

[24] Y. W. Law, J. M. Doumen, and P. H. Hartel. Benchmarking block ciphers for wireless sensor networks. Technical report, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, February 2005. Imported from DIES.

[25] C. Lee, M. Potkonjak, and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture (Micro-30)*, pages 330–335, 1997.

[26] Gokhan Memik, William H. Mangione-Smith, and Wendong Hu. Netbench: a benchmarking suite for network processors. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 39–42, Piscataway, NJ, USA, 2001. IEEE Press.

[27] L. Nazhandali, M. Minuth, and T. Austin. Sensebench: toward an accurate evaluation of sensor network processors. In *In Proceedings of the IEEE International Workload Characterization Symposium*, October 2005.

[28] Leyla Nazhandali, Bo Zhai, Javin Olson, Anna Reeves, Michael Minuth, Ryan Helfand, Sanjay Pant, Todd M. Austin, and David Blaauw. Energy optimization of subthreshold-voltage sensor network processors. In *ISCA*, pages 197–207. IEEE Computer Society, 2005.

[29] Adrian Perrig, Robert Szewczyk, Victor Wen, David E. Culler, and J. D. Tygar. SPINS: security protocols for sensor netowrks. In *Mobile Computing and Networking*, pages 189–199, 2001.

[30] D. Petrovic, R.C. Shah, K. Ramchandran, and J. Rabaey. Data funneling: Routing with aggregation and compression for wireless sensor networks. In *Sensor Network Protocols and Applications (SNPA'03)*, May 2003.

[31] R. Rivest. The MD5 Message-Digest algorithm, April 1992. RFC 1321.

[32] C. Schurgers and M. Srivastava. Energy efficient routing in wireless sensor networks. 2001.

[33] Deva Seetharam and Sokwoo Rhee. An efficient pseudo random number generator for low-power sensor networks. In *LCN*, pages 560–562. IEEE Computer Society, 2004.

[34] R. Shah and J. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *In Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, Orlando, FL, 2002.

[35] The SimpleScalar-Arm Power Modeling Project. www.eecs.umich.edu/ tnm/power/, 2000.

[36] Volker Turau and Christoph Weyer. Location-aware in-network monitoring in wireless sensor networks. In *Proceedings of the Sensor Networks Workshop at Informatik 2004*, Ulm, Germany, September 2004.

[37] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOMM*, 2002.