Profiling of Lossless-Compression Algorithms for a Novel Biomedical-Implant Architecture

Christos Strydis
christos@ce.et.tudelft.nl

Georgi N. Gaydadjiev
georgi@ce.et.tudelft.nl

Computer Engineering Laboratory, Electrical Engineering Dept.
Delft University of Technology
Postbus 5031, 2600 GA, Delft
The Netherlands

ABSTRACT

In view of a booming market for microelectronic implants, our ongoing research work is focusing on the specification and design of a novel biomedical microprocessor core targeting a large subset of existing and future biomedical applications. Towards this end, we have taken steps in identifying various tasks commonly required by such applications and profiling their behavior and requirements. A prominent family of such tasks is lossless data compression. In this work we profile a large collection of compression algorithms on suitably selected biomedical workloads. Compression ratio, average and peak power consumption, total energy budget, compression rate and program-code size metrics have been evaluated. Findings indicate the best-performing algorithms across most metrics to be *mlzo* (scores high in 5 out of 6 imposed metrics) and *fin* (present in 4 out of 6 metrics). Further *mlzo* profiling reveals the dominance of i) address-generation, load, branch and compare instructions, and ii) interdependent logical-logical and logical-compare instructions combinations.

Categories and Subject Descriptors

I.6.6 [**Simulation and modeling**]: [Simulation Output Analysis]; C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems—*Real-time and embedded systems*; E.4 [**Data**]: Coding and Information Theory—*Data compaction and compression*

General Terms

Performance, Measurement

Keywords

Implantable devices, ultra-low power, lossless compression, microarchitectural profiling

CODES+ISSS'08, October 19–24, 2008, Atlanta, Georgia, USA.

1. INTRODUCTION

Biomedical microelectronic implants have been around for more than 50 years. Their most popular instance, the implantable pacemaker, apart from saving lives, has acted as a catalyst on the general public closed-mindedness against biomedical implants. To illustrate, in the U.S. alone, a total number of 180,000 implantable pacemakers have been registered for the year 2005 (source: American Heart Association [5]). Nowadays, biomedical implants are being designed for a large, and constantly increasing, range of applications. Two prominent reasons for this boom are the rapid increase in healthcare costs and the population aging in advanced countries. A future where people are moving around performing their everyday tasks while tiny implants are monitoring or assisting their body may not be so far. Implants are expected to monitor and log biological data in-vivo and, depending on the application, to act on those readouts by regulating some physiological quantity in the body e.g. to release insulin to the blood stream when high blood-glucose levels are detected. In this context, an aspect of implants which has been largely overlooked so far is compression of sensed biological data. Given their highly resource-constrained nature (e.g. memory size), data compression is considered a crucial area of focus.

Our long-term work focuses on designing a novel, minimalistic, low-power processor suitable for a large subset of biomedical applications. Currently, various tasks related to implant functionality are being profiled. A study of encryption algorithms has already been performed [17]. In this paper, we profile various popular lossless-compression algorithms against suitable metrics. We, then, select the ones with the best characteristics for the targeted application domain and investigate their respective instruction frequencies and mixes. We, thus, offer insights on the design and implementation of the targeted processor. Concisely, the contributions of this work are:

- To identify compression algorithms achieving the highest compression ratios and rates on suitably selected biomedical workloads;
- To identify algorithms with the lowest average and peak power consumption when compressing biomedical workloads;
- To identify instruction mixes and frequencies of the best scoring algorithms for guiding microarchitectural

source name	size (Bytes)	samples (#)	duration (sec)	sample rate (Sml/sec)	sample rate (KB/sec)
Electromyogram II (EMGII)	1147/9605	144/1201	0,288/2,402	500/500	3,89/3,91
Electroencephalogram (EEGI)	984/9616	123/1202	0,615/6,010	200/200	1,56/1,56
Electrocardiogram (ECGI)	912/9615	114/1202	0,114/1,202	1000/1000	7,81/7,81
Respiratory Cycle I (RCI)	1192/9520	149/1191	1,490/11,910	100/100	0,78/0,78
Pulmonary Function I (PFI)	1184/9240	148/1155	1,480/11,550	100/100	0,78/0,78
Skin Temperature (AEP)	1120/9736	140/1217	0,700/6,085	200/200	1,56/1,56
Blood Pressure (BP)	1128/9545	141/1198	0,282/2,396	500/500	3,91/3,89

Table 1: 1-KB and 10-KB biomedical workloads. Double-precision (8-Byte) data samples are used.

and architectural optimizations in the envisioned implant processor.

The rest of the paper is organized as follows: section 2 gives an overview of related works in the field. Section 3 outlines the framework onto which this profiling study has been built. Section 4 provides the details of our selected compression algorithms as well as the profiling testbed used. Section 5 contains, in detail, the findings of this work. Overall conclusions and future work are drawn in section 6.

2. RELATED WORK

Barr and Asanovic [1] have worked extensively towards the power trade-off between compression and Tx/Rx power of data on a testbed functionally similar to the popular Compaq iPAQ handheld. Their analysis reveals that with several typical compression algorithms, there actually is a net increase in energy. They propose the use of asymmetric compression, that is, use of a low-energy compression algorithm on the transmit side and a different algorithm for the receive side to cope with the problem.

In the area of wireless sensor networks (WSNs), Maniezzo et al. [11] work on surveillance sensor networks and seek to define an online energy trade-off mechanism between compressing image data in a sensor or forwarding (i.e. transmitting) them to the next sensor closer to the base station. Ferrigno et al. [6] attempt to balance between local and central data processing in an effort to minimize sensor energy consumption. They investigate various lossy image-compression algorithms and make an educated selection based on its performance and energy needs. Kimura and Latifi [10] perform a survey on data compression for WSNs and profile four compression algorithms specifically designed for WSNs.

The work presented here is original in that it targets a different class of low-power devices with particular idiosyncrasies. To the best of our knowledge, no similar effort has been reported so far in explicitly provisioning an implant processor with data compression. We have investigated other fields of highly resource-constrained systems such as WSNs, however implants present distinct traits. To exemplify, the energy efficiency of data decompression is not our priority in this work since the largest fraction of wirelessly transmitted data in implants is outbound traffic, i.e. telemetry of biomedical data to an ex-vivo monitoring system. Further issues applying to WSNs such as total energy cost for data hopping through a network of nodes do not apply in our case, too.

3. IMPLANT CHARACTERISTICS

This work primarily focuses on profiling a number of data compression algorithms for biomedical, microelectronic implants. The special nature of such devices has set the following parameters to our profiling experiment.

feature	value
ISA	32-bit ARMv5TE-compatible
Pipel. depth / Datap. width	7/8-stage, super-pipelined / 32-bit
RF size	16 registers
Issue policy/Instr.window	in-order/single-instruction
I/D-Cache, L1 (separ)	32B 1-entry, 1-cc hit/170-cc miss lat.
BTB/TLB	2-entry direct-mapped/1-entry
Branch Predictor	2-bit Bimodal
Write/Fill Buffer (separ)	2-entry/2-entry
Mem. bus width	1 Byte
INT/FP ALUs	1/1
Clock freq.	2 MHz
Implem. tech.	0.18 μm @ 1.5 Volt

Table 2: XTREM (modified) architecture details.

A large class of biomedical implants performs periodic, in-vivo measurements of physiological data through biosensors. The collected data need either to be stored inside the implant for later telemetry to an external monitoring device, e.g. a treating physician's office computer, or to be periodically transmitted to an external data-logging system such as a PDA, laptop computer etc.. This pattern of behavior indicates that outbound biological-data traffic almost always dominates inbound traffic. In effect, the most important aspect of the pair data compression-decompression is the former, thus this work deals only with compression. Further, the sensitive nature of biomedical signals dictates that, in the general case, no information can be afforded to be lost or altered during data acquisition, compression and transmission. We are, therefore, inclined to consider solely lossless compression to ensure information integrity.

Typical biomedical readouts are often highly periodic signals (e.g. heart beat) or stable signals (e.g. blood temperature) which can, under specific circumstances, display gradual or abrupt changes in value (e.g. a sudden muscle contortion). We have collected and used various representative workloads capturing both stable as well as rapidly changing patterns. The original data has been provided from the BIOPAC (R) Student Lab PRO v3.7 Software. Paper-size limitations do not allow for an extensive description of the various workloads; a concise overview of workload details is provided in Table 1. Reported literature and an extensive study [16] on implants have further revealed that typical data-memory sizes inside the implants range from 1 KB to 10 KB. Therefore, workloads of both sizes (1 KB and 10 KB) have been profiled.

4. EXPERIMENTAL SETUP

Profiling has been based on XTREM [3], a modified version of SimpleScalar. The XTREM simulator is a cycle-accurate, microarchitectural, power- and performance- functional simulator for the Intel XScale core [8]. It models the effective switching node capacitance of various functional units inside the core, following a similar modeling methodology to the one found in Wattch [2]. XTREM has been selected for its straight-forward functionality but mostly for its high performance- and power-modeling precision. It ex-

compression algorithm	benchmark name	details
Static-Huffman Coding [13]	huff	Huffman coding with static symbol table
Adaptive-Huffman Coding [13]	ahuff	Huffman coding with adaptive symbol table
Arithmetic Coding, Order-0 [13]	arith	Simple arithmetic coding
Arithmetic Coding, Order-1 [13]	arith1	Order-1 arithmetic coding
Arithmetic Coding, Order-1e [13]	arith1e	Order-1 arithmetic coding with escape characters
LZSS (12-bit sliding window) [13]	lzss	Storer & Szymanski's slightly modified LZ77 version
LZW (fixed 12-bit) [13]	lzw12	LZW with fixed 12-bit symbols
LZW (variable up to 15-bit) [13]	lzw15v	LZW with variable-size symbols, up to 15 bits
Run-Length Encoding [7]	bclrle	Simple run-length encoding
Shannon-Fano [7]	bclsf	-
Finish [12, 4]	fin	LZ77-variant with 2-character memory window
Splay-Tree Compression [4, 9]	splay	Similar to Huffman encoding, locally adaptive
LZSS w/ Adaptive-Huff. Coding [4]	lzhuf_oku	LZSS with binary-tree symbol table
LZSS w/ Adaptive-Arith. Coding [4]	lzari_oku	-
Urban [12]	urban	High-order arithmetic coder working at the bit level
MiniLZO [14]	mlzo	Light-weight subset of the LZO library (LZ77-variant)
S-LZW [15]	slzw	Memory-constrained modification of LZW for Sensor-nodes

Table 3: Benchmark suite of lossless compression algorithms.

hibits an average performance error of only 6.5% and an average power error of only 4%.

Many of the XScale architectural features have been integrated into XTREM. XTREM allows monitoring of 14 different functional units of the Intel XScale core: Instruction Decoder (DEC), Branch-Target Buffer (BTB), Fill Buffer (FB), Write Buffer (WB), Pend Buffer (PB), Register File (REG), Instruction Cache (I$), Data Cache (D$), Arithmetic-Logic Unit (ALU), Shift Unit (SHF), Multiplier Accumulator (MAC), Internal Memory Bus (MEM), Memory Manager (MM) and Clock (CLK). To better much our application field, many of XTREM's architectural parameters have been cut down or disabled to better reflect the highly constrained implantable processors. The modified XTREM characteristics are summarized in Table 2. Performance/power figures have been checked and scale properly with the changes.

When putting together our benchmark suite of compression algorithms, we have made an effort to include sources adhering to the following principles: i) large range of lossless data compression techniques and styles, from high-performing to compact flavors; ii) mature implementation code base; iii) various algorithm complexities; iv) suitability: the XTREM simulator can only handle C and Java sources. Furthermore, in its current version it does not support an OS on top of the simulated hardware, thus prohibiting the use of compression sources - such as the excellent bzip2 algorithm - that require high-level, OS features; and v) availability: all collected benchmarks comprise utterly free, published or free under the GNU General Public License sources, readily available to the research community.

The implementation of a given compression algorithm plays as crucial a role for the performance and behavior of the algorithm as its underlying structure. While adhering to the above principles, in order to offer the best possible fairness in our selection process, we have attempted to include algorithms built with the same implementation philosophy (e.g. algorithm suite implemented by the same author(s)) and/or algorithms being top representatives in their category. Paper-size limitations do not allow for an extensive discussion; Table 3 summarizes the selected algorithms.

5. PROFILING ANALYSIS

In this section, we present the findings of our profiling study. Due to the limited paper space, we report cumulative figures based on the averaged results across all profiled workloads. That is to say, we do not favor any of the workloads presented in Table 1. Further, all reported average values in fact are median values unless stated otherwise, since we cannot guarantee normal data distribution in the general case. Last, results have been grouped in two main categories of 1-KB and 10-KB data so as to capture also the variation in behavior when increasing the input size.

The first metric to discuss is **compression ratio** and findings are illustrated in Fig. 1. For the case of 1-KB data, our compression algorithms perform worse (−0.08% on average) that for the 10-KB data (10.14% on average). For the 1-KB case we actually see an expansion of data, on average. Given that workloads in this case are 10 times smaller, an overall approx. 100% poorer compression is performed. To put it simply, attempting to compress 10 consequent 1-KB readouts results in a compressed output double the size of a compressed, single, contiguous 10-KB readout. Clearly, compression of larger files is favored. This claim has to be backed also with energy-expenditure results in order to make it attractive for ultra-low-power (ULP) systems such as implants are. We will address this topic later.

The difference in compression ratios for different workload sizes is justified by the fact that for small inputs, many compression algorithms do not simply have sufficient context to become efficient; symbol tables may not have the time to be filled thus impacting compression efficiency. In short, it is a "cold start" problem. Overall, the most compression-efficient algorithms, as the figures indicate, are *lzari_oku*, *lzhuf_oku* and *mlzo*. *urban* and *arith1* are contesting with *lzss* and *fin* for the 4th and 5th positions, respectively.

Another interesting attribute of the compression algorithms is how fast they are able to pack data, i.e. their **compression rate**. In Fig. 2 average compression rates in KB/sec are reported. Overall, the average compression rate for 1-KB data is 0.051 KB/sec while for 10-KB data it is 0.095 KB/sec, or about double the speed. The reason for this difference is anticipated to be the fact that with 1-KB data, compression algorithms do not have the time to create and traverse excessively large data structures such as the symbol table. For instance, with a typical size of 256 $Bytes$ which is comparable to the input data size of 1 KB, the symbol table does not have the time to fill and become efficient. Of course, this has adverse effects on compression. Best-scoring algorithms for this metric are *bclrle*, *slzw*, *fin*, *mlzo* and *lzw12*. *bclrle* achieves by far the most impressive results due its simplistic design but does so at the cost of poor or no compression.

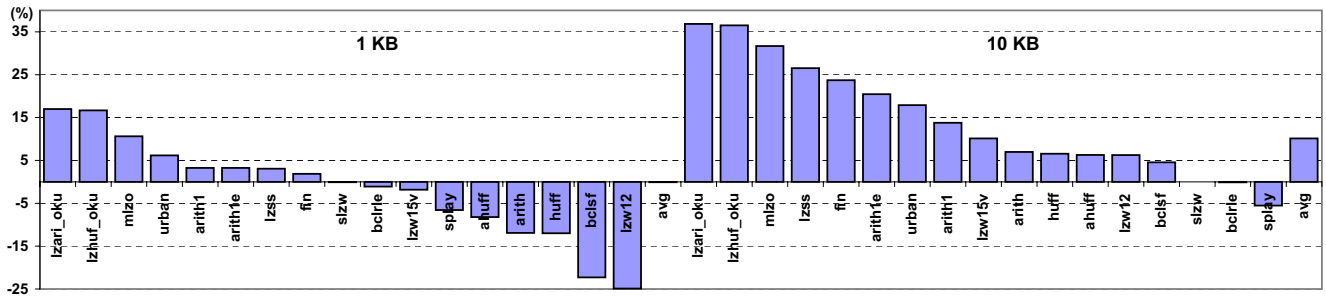

Figure 1: Averaged compression ratios for 1-KB and 10-KB datasets.

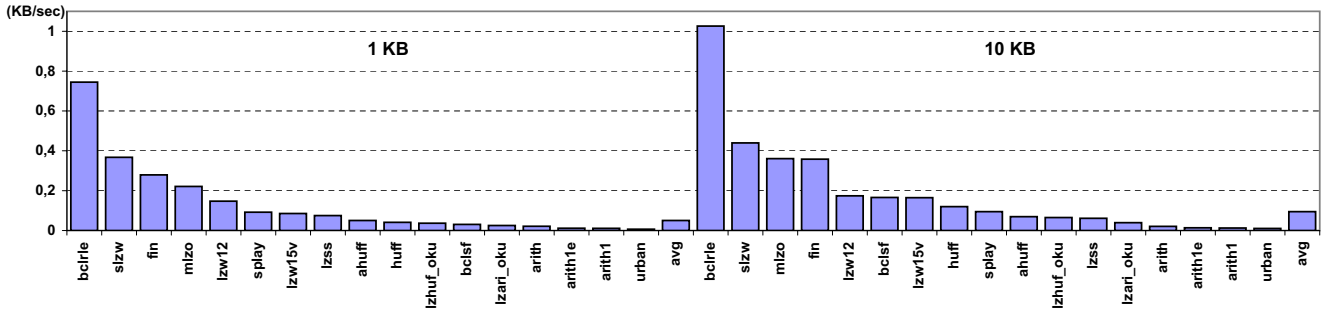

Figure 2: Averaged, average compression rates for 1-KB and 10-KB datasets.

Average power consumption is an important metric of an algorithm's performance. It reveals the average rate at which the executed algorithm draws energy from the system. An implant battery may have sufficient charge to support a whole compression operation, however, it might not be able to sustain the energy rate needed by the compression algorithm. Another interesting metric in this context is **peak power consumption**. A battery able to support a compression algorithm with a given average power consumption may be unable to deliver the required output at a given point in time if the algorithm sporadically presents peak power values which are largely deviating from its average power needs. To address both aspects of the profiled algorithms, we have plotted Fig. 3. The algorithms are ordered in order of increasing average-power profiles. Bars indicate average (overall and per-processor-component) power while black dots indicate peak power.

We can readily see that the memory-manager unit (MM) is the power-hungriest component with a rough 94% fraction of overall power consumed throughout both workload groups. The MM unit is activated each time the core is stalled because of a main-memory instruction or data fetch. A high power consumption in the MM is expected for resource-constrained devices with small or totally absent I/D-caches as the ones we consider here. Next follows the clock structure (CLK) consuming about 5% of the overall power.

From the figure we can further observe that average power consumption increases marginally with workload size. In effect, the algorithms' power needs are unaffected by the workload size they operate on. We can also see that most algorithms converge to a consumption threshold of roughly 95 mW. We have performed some further tests whereby some of the processor's characteristics have been enhanced, e.g. cache sizes have been increased. In that case, a large variation among the power profiles of the various algorithms has resulted. This indicates that the constrained version of the processor we currently use essentially "chokes" the performance of many algorithms forcing them to slow their execution down and, thus, demand less power from the underlying machine. This is a crucial observation since it excludes from selection those algorithms whose performance enhancements will not bring any benefit to a highly resource-constrained, implant processor. Outright best performing algorithms in terms of average power consumption are *mlzo*, *arith*, *arith1e*, *arith1* and *urban*. When peak power consumption is considered, the ranking changes with *lzw15v*, *lzss*, *fin*, *mlzo* and *slzw* scoring best, indicating large deviations between average and peak power figures.

An interesting point to make here is that implantable systems would greatly benefit from power-aware compression techniques. In effect, compression algorithms that dynamically adapt their actual compression speed and/or ratio depending on the amount of energy they spend in a given time interval. When this amount surpasses a preset (or dynamically set) threshold value, they lower their performance to make it back to the threshold. Of course, this presumes a way for the algorithm (thus, software) of tapping into processor (thus, hardware) power figures at run-time. None of the profiled algorithms in this study has such capabilities, yet it would be a crucial adaptation for future ULP systems.

Knowing the **overall energy budget** needed for completing a single compression task is important for battery-operated implants. It directly tells us how much stored energy the given task needs in order to execute and, in effect, what stored-energy amount will be deduced from the battery. It also tells us if the compression computation is worth the effort compared to simply transmitting the data uncompressed over the air. Accordingly, in Fig. 4 averaged, overall energy expenditures for both workload sizes have been plotted. *urban* and *arith1* display very large energy costs and have, thus, been omitted from the plots to give better resolution for the rest of the algorithms.

From Fig. 4, we can readily observe that the energy budget does not scale linearly with workload size. The cost

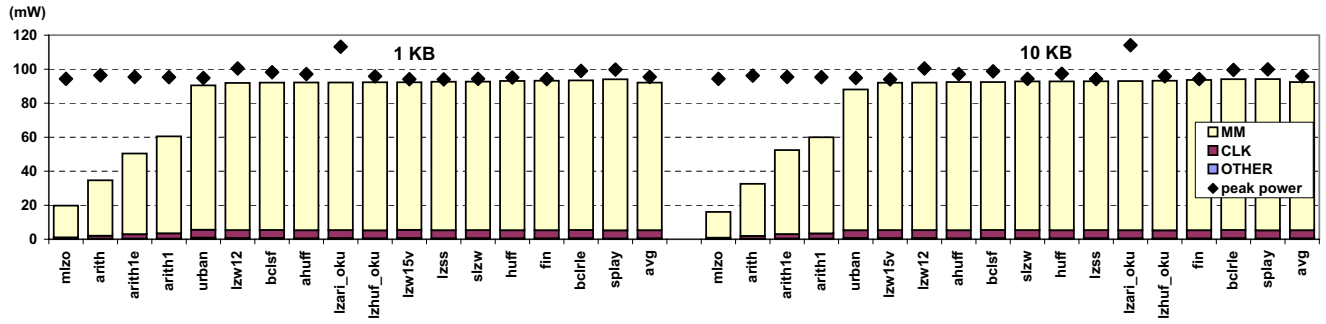

Figure 3: Averaged, average and peak power consumption for 1-KB and 10-KB datasets.

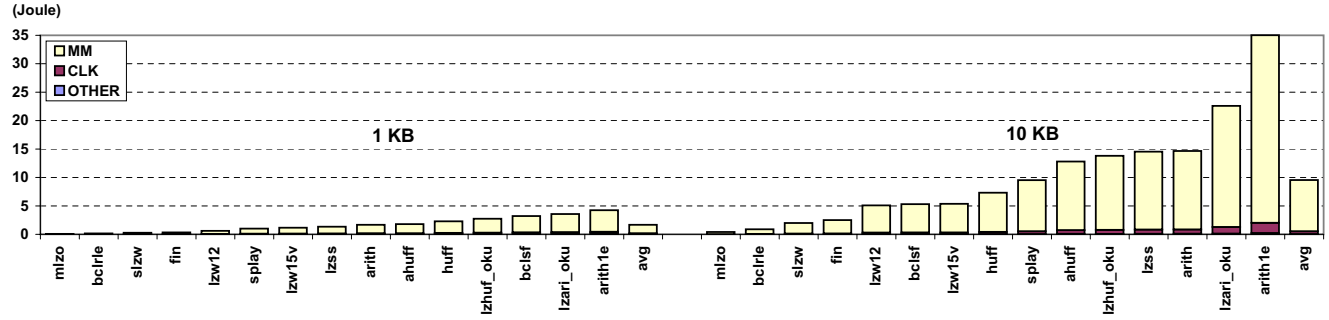

Figure 4: Averaged, total energy expenditure for 1-KB and 10-KB datasets.

of compressing one 10-KB workload (9.541 J) as opposed to that of successively compressing 10 1-KB workloads (1.684 J for one) is about 55% smaller. This agrees also with our compression-ratio results; that is, rarer compression of larger input data is energy- and compression-wise preferable to frequent compression of smaller input data. Agreeing with the previous discussion on power, we can further see that the MM and CLK components indeed are the overall most energy-consuming parts of the processor. The best performing algorithms in this case are *mlzo*, *bclrle*, *slzw*, *fin* and *lzw12* and they preserve their ranking for both workload sizes. Interestingly, with the exception of *mlzo*, these are not the same algorithms as the top-ranking ones in terms of average power consumption, as one might expect. The reason for this difference lies in the actual algorithm execution times. An algorithm might consume little power on average but might do so for a disproportionately large amount of time, thus canceling all benefits of its low-power nature. For instance, *arith* consumes only 32.58 mW on average while compressing a 10-KB workload but it completes its task in 456.63 *sec* on average while the overall average compression time for 10-KB workloads is only 86.28 *sec*. Hence, its excessive energy budget and resulting poor ranking.

A last metric we evaluate is the **binary size** of the algorithms' executables, as a measure of program-memory needs. Executables have been built with the GNU ARM-GCC v4.1.2 cross-compiler and optimization level O2. Furthermore, executables have been statically linked (this is an ARM requirement) and, therefore, are expected to be somewhat larger in size than their dynamically linked counterparts. In Table 4, the code complexities of the selected compression algorithms are shown in ascending order. Obviously, results shown in the table are heavily implementation-dependent and should be considered with caution. However, as we have mentioned also in section 4, many different algorithms have been based on the same infrastructure (or

alg.	size (KB)	alg.	size (KB)	alg.	size (KB)
fin	10.4	bclrle	15.7	arith1	17.1
splay	12.5	bclsf	15.7	arith1e	17.1
urban	13.5	huff	16.2	lzhuf_oku	17.4
lzw12	13.8	mlzo	16.3	arith	17.4
slzw	14.0	lzw15v	16.7	ahuff	21.5
lzss	14.6	lzari_oku	17.0		

Table 4: Compression algorithms' program sizes.

suite), built by the same author(s). Therefore, the difference in sizes (rather than the actual sizes), can give an indication of the program-memory needs, regardless of the underlying implementations. Best scoring algorithms in this case are *fin*, *splay*, *urban*, *lzw12* and *slzw*.

To summarize our analysis results, we present in Table 5 the 5 best-performing algorithms on each one of our profiled metrics, for both workload sizes. The undisputed winner is *mlzo*, followed by *fin* and *slzw*. Accordingly, we take a closer look at the underlying instruction mix of *mlzo*.

By design, the XTREM simulator internally breaks up executed ARM instructions to "uops". This quirk in fact is useful to us since it allows us to capture microarchitectural details at the smallest granularity possible. In Table 6, the on average most frequent ($> 5\%$) uops for both workload sizes are listed. The *address-generation* ("agen") uop is by far the most common and, although it is specific to ARM-based microarchitectures, it reveals the importance of implementing an efficient address-generation mechanism in the envisioned processor. *Loads* ("ldp") follow in frequency, justifying the previously observed large power component of the MM unit and hinting towards a power-efficient MM design. *Branch/jump* ("b") and *compare* ("cmp") instructions follow and expectedly have similar occurrence frequencies. They indicate that even small optimizations in the compare-and-branch mechanism will improve power and performance significantly.

ratio	avg. rate	avg. power	peak power	total energy	code size
lzari_oku	bclrle	mlzo	lzss	mlzo	fin
lzhuf_oku	slzw	arith	lzw15v	bclrle	splay
mlzo	fin	arith1e	fin	slzw	urban
urban	mlzo	arith1	mlzo	fin	lzw12
arith1	lzw12	urban	slzw	lzw12	slzw
lzari_oku	bclrle	mlzo	lzw15v	mlzo	fin
lzhuf_oku	slzw	arith	lzss	bclrle	splay
mlzo	mlzo	arith1e	fin	slzw	urban
lzss	fin	arith1	mlzo	fin	lzw12
fin	lzw12	urban	slzw	lzw12	slzw

Table 5: Best-performing compression algorithms in descending order (top: 1-KB, bottom: 10-KB).

uop	avg(1KB)	uop	avg(10KB)
agen	30.00%	agen	26.88%
ldp	19.89%	ldp	20.57%
b	9.94%	b	12.18%
cmp	8.53%	cmp	9.65%
stp	8.29%	add	7.39%
mov	5.90%	stp	5.91%
add	5.66%	eor	5.77%

Table 6: Popular *mlzo* uop frequencies.

instr.	pairs	/ triplets	avg(1KB)	avg(10KB)
and	eor	-	14%	17%
eor	eor	-	8%	5%
and	eor	and	-	7%
eor	cmp	-	-	6%
beq	add	add	-	6%

Table 7: Popular *mlzo* instruction pairs/triplets.

We finally report Table 7 which lists popular dynamic instruction pairs/triplets during *mlzo* execution for both workload sizes. Instruction pairs or triplets are consecutive instructions whereby data generated by the first instruction is consumed by the second and/or third instruction; i.e. whereby data dependencies occur. The table reveals that by far the most popular pair is "and-eor" (eor: exclusive or) followed by "eor-eor". We, thus, get a clear indication that data-forwarding in the logical-operation part of the ALU, interlock-collapsing-ALU techniques [18] or other (micro)architectural optimizations will significantly benefit the implant processor. Further, the "and-eor-and" triplet falls in the above category of optimizations. However, the "eor-cmp" and "beq-add-add" combinations relate also to the previous discussion on optimizing the compare-and-branch subsystem of the processor. Last but not least, all above observations on instruction/uop frequencies can give clear directions as to which instructions should be explicitly implemented in hardware and which ones can be afforded to be implemented in software (compiler-side conversion).

6. CONCLUSIONS

Based on a number of suitably selected metrics, in this work, we have offered quantitative insights on the relative behavior of various general-purpose, lossless-compression algorithms running on a highly resource-constrained processor simulator, compressing a variety of biological workloads. Based on this behavior, we have selected best-performing algorithms and we have further profiled the microarchitectural characteristics of the top-ranking one, *mlzo*. Along with our previous profiling studies, the current work is the first stage in a two-step research effort towards the design of a novel digital processor for microelectronic implants. The presented work gives clear directions for (micro)architectural features and optimizations of such a processor. Future work concerns drawing the detailed specifications for the processor and developing a first, proof-of-concept design.

7. ACKNOWLEDGEMENTS

This work has been partially supported by the ICT Delft Research Centre (DRC-ICT) of the Delft University of Technology. Many thanks are due to Christopher Sadler for the interesting SLZW compression algorithm.

8. REFERENCES

[1] Barr, K., and Asanovic, K. Energy-aware lossless data compression. *ACM TCS 24*, 3 (2006), 250–291.

[2] Brooks, D., et al. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *ISCA'00* (2000), pp. 83–94.

[3] Contreras, G., et al. XTREM: A Power Simulator for the Intel XScale Core. In *LCTES'04* (2004), pp. 115–125.

[4] de Vries, N. Lossless Data-Compression Kit, LDS v1.3. `http://www.nicodevries.com/nico/lds13.zip`.

[5] Ector, H., and Vardas, P. Heart disease and stroke statistics - 2008 Update. *AHA* (2008).

[6] Ferrigno, L., et al. Balancing computational and transmission power consumption in wireless image sensor networks. In *VECIMS'05* (2005), pp. 61–66.

[7] Geelnard, M. Basic Compression Library, BCL v1.2.0. `http://bcl.comli.eu/`.

[8] Intel Corp. *Intel XScale Microarchitecture for the PXA255 Processor: User's Manual*, March 2003.

[9] Jones, D. Application of splay trees to data com-pression. *Communications of the ACM 31*, 8 (1988), 996–1007.

[10] Kimura, N., and Latifi, S. A survey on data compression in wireless sensor networks. In *ITCC'05* (2005), pp. 8–13.

[11] Maniezzo, D., et al. Energetic trade-off between computing and communication ressource in multimedia surveillance sensor network. In *4th IWMWCN* (2002), pp. 373–376.

[12] Nelson, M. DDJ Data Compression Contest results. *Dr. Dobb's Journal 16*, 11 (Nov. 1991), 62–64.

[13] Nelson, M., and Gailly, J.-L. *The Data Compression Book, 2nd Ed.* M&T Brooks, 1995.

[14] Oberhumer, M. "LZO v2.0.2". `http://www.oberhumer.com/opensource/lzo/`.

[15] Sadler, C., and Martonosi, M. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *SenSys'06* (2006), pp. 265–278.

[16] Strydis, C., et al. Implantable microelectronic devices: A comprehensive review. CE-TR-2006-01, Computer Engineering, TU Delft, Dec. 2006.

[17] Strydis, C., Zhu, D., and Gaydadjiev, G. Profiling of symmetric encryption algorithms for a novel biomedical-implant architecture. In *ACM CF'08* (2008), pp. 231–240.

[18] Vassiliadis, S., et al. Interlock collapsing ALU's. *IEEE Transactions on Computers 42*, 7 (Jul 1993), 825–839.