In Proceedings of the Army Science Conference, Orlando, Florida, November 2006.

Compression Techniques for Minimum Energy Consumption

Sebastian Puthenpurayil
Department of Electrical and Computer Engineering
University of Maryland
College Park, MD, 20742, USA
Email: purayil@umd.edu

Ruirui Gu
Department of Electrical and Computer Engineering
University of Maryland
College Park, MD, 20742, USA
Email: rgu@umd.edu

Shuvra S Bhattacharyya
Department of Electrical and Computer Engineering
University of Maryland
College Park, MD, 20742, USA
Email: ssb@umd.edu

***Abstract*— Low power sensor nodes are integral parts of large-scale wireless sensor networks, which find extensive applications in domains such as military surveillance, transportation control, and environmental monitoring. This paper explores energy-consumption trade-offs associated with lossless data compression in low power sensor nodes. We focus specifically on compression of acoustic signals. The processing of acoustic signals in digital form and subsequent transmission of the signals across a network involve significant energy consumption by the transceiver of a sensor node. To reduce the volume of data, and the associated energy consumption of transmission, it is useful to apply data compression. The process of data compression will generally reduce the transmission energy at the expense of energy expended due to additional computation in the embedded processor. This paper focuses on this trade-off between computational and communication-oriented costs associated with using data compression in sensor nodes. Understanding this trade-off requires an integrated approach that considers the strength of compression, the energy expended by the embedded software in performing the compression, and optimization of the embedded software to further improve overall impact of compression. This paper develops a framework for design and experimentation to facilitate these goals, and reports preliminary experimental results based on this framework.**

I. Introduction

Sensor nodes deployed in remote locations for military applications or seismic recordings acquire signals from acoustic sources and transmit those signals to end users or servers. Signal processing techniques are extensively used in such sensor node applications. These processes, and the subsystems to which they interface, involve energy consumption through computation, transmission, and reception. It is well known that the major source of energy consumption in sensor nodes is in the transceiver portions, which contain analog circuitry such as amplifiers, modulators/demodulators, and filters. Some form of energy savings can be attained by keeping the transceiver in a standby mode and switching to the normal mode only when there is data for transmission. We can reduce the time when the transceiver is in the normal mode if the volume of data to be transmitted is reduced by compression. Reduction in data size for this purpose can give considerable savings in power. This savings leads to large advantages in increasing the lifetime of sensor node batteries, which often cannot be replaced, or are extremely difficult or inconvenient to replace.

In this paper, we address the use of lossless data compression in sensor nodes to improve energy efficiency, while preserving the integrity of the signals that are being analyzed. This process generally reduces transmission energy, as described above, at the expense of energy expended due to additional computation in the embedded processor of the sensor node. Since transceivers for sensor nodes typically consume much more power than the embedded processors that are employed, intensive computation effort can generally be expended on compression, while still yielding a favorable trade-off in terms of total energy consumed in the node.

In our design approach, we apply dataflow models to represent conventional compression algorithms, and map them into efficient embedded software realizations. This provides a formal link between algorithm representation and hardware/software optimization (e.g., see [8]) that is useful in maximizing overall impact of compression. We have modeled the fundamental data compression technique of *linear predictive coding (LPC)* based on the *autoregressive model (AR)*. We describe two versions of LPC implementation: a static version that is developed based on *synchronous dataflow (SDF) programming* [9]; and a dynamic version, which is more flexible in its operation and is based on *parameterized synchronous dataflow (PSDF) programming* [1]. The SDF form is best suited for intensive analysis and optimization; however, it is more restricted in the degree to which the compression approach can be adapted at run time. In contrast, the PSDF model is based on reconfigurable dataflow graphs, and provide for more tunable compression to be performed, while also allowing for a high degree of static analysis and optimization for the embedded software.

II. Background

Data compression shrinks raw data down to smaller volumes, which is desirable for data communication since the compressed data can require significantly less time and energy to transmit compared to the raw data. Previous research for data compression in communication mainly focuses on how to decrease delay or save required transmission bandwidth. Hans and Schafer [2] present an overview of lossless data compression in the context of audio data.

With the emergence of many severely energy- and power-constrained application domains, such as various kinds of sensor networks, the topic of energy efficiency is becoming increasingly important. There are some works that take data compression into account when dealing with energy efficiency. Zhang and Li [3] discuss the implementation of compression algorithms for seismic data. These works mainly estimate energy reduction due to the data reduction after compression, and consider the energy costs of communication alone or in isolation from the costs of computation.

Our work differs from such related work in its focus on acoustic, sensor network signals, and in its emphasis on design and implementation considerations that relate compression algorithms with the underlying embedded software.

When implementing data compression algorithms, we are using dataflow modeling, which is widely used in signal processing. In a dataflow representation, a program is depicted as a set of tasks with data precedences. The tasks and precedences are represented as vertices and edges, respectively, in a directed graph, called a *dataflow graph*. Dataflow representations facilitate the optimization of hardware and software for signal processing systems (e.g., see [8]), and therefore provide a useful tool in the investigations targeted in this paper.

III. Static Version

As described above, our experimentation in this work is carried out in the context of the well-known LPC algorithm. The basic components of our targeted LPC system can be decomposed into three major computational blocks called framing, coefficient generator, and quantizer. Given a segment $s[l]$ of input data of size L, the framing block computes an optimal frame size for the input data. Each frame is then passed to a linear predictor, which computes the predictor coefficients. In the static version of the application, the frame size and model order of the predictor are constant and known beforehand. The output of the predictor is a set of predictor coefficients of size equal to the model order for the associated frame. Using the predictor coefficients and the input data, we compute the predicted value, and using the predicted value we can compute the error in prediction.

Building on the above notations, we can represent a linear predictor as follows: $s[l]$ represents the input data, $x[n]$ represents the data for a frame, and $a[k]$ represents the predictor coefficients of a frame. The predictor is modeled as an *autoregressive (AR)* process. The relation between the predictor coefficients and the input data is given by the *AR* model:

$$\text{Predicted}(i) = \sum_{i=1}^{k} (a_i)(x_{n-i}) \ . \tag{1}$$

The prediction error is given as:

$$\text{Error}(i) = \text{x[i]} - \text{Predicted(i)} \ . \tag{2}$$

We use backward error prediction in this model so that all computations can be done by each frame. The prediction error

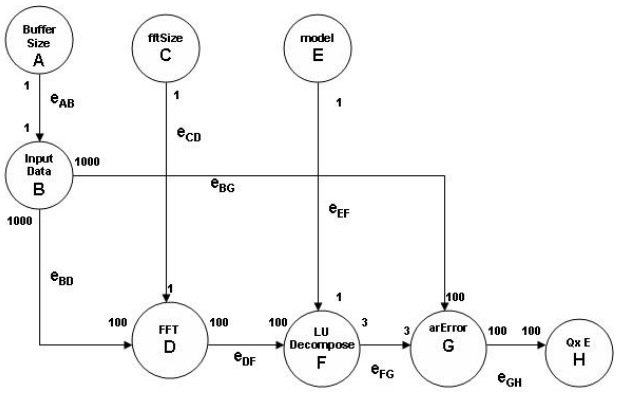

Fig. 1. Static configuration of LPC.

and its coefficients are quantized using the quantizer block and the quantizer output is then the compressed data. A detailed block diagram of the static LPC system is given in Figure 1.

This block diagram is modeled in terms of SDF semantics. In SDF, as in other forms of dataflow, an application is represented as a directed graph in which vertices (*actors*) represent computational tasks of arbitrary complexity, and edges specify data dependencies between actors. In SDF, the number of data values (*tokens*) produced and consumed by each actor is constant and known at compile time. In Figure 1, each input and port of each actor is annotated the corresponding rate of token production/consumption. For example, the FFT actor consumes and produces 100 tokens every time it is executed. In this application model, the size of an input data segment is 1000 and the frame size is 100.

Further details on the computational blocks used in this block diagram model are given in Section VI.

IV. Dynamic Version

The static LPC version presented in the previous section has the advantage of simplicity and more computational structure since all parameters remain fixed. However, the compression ratio of LPC can be improved by changing parameters such as the frame size and model order adaptively. Our dynamic version of LPC implementation is a dynamic model for achieving such adaptive operation. This implementation is derived from a block diagram model that is based on parameterized synchronous dataflow (PSDF) semantics. A subsystem ϕ in a PSDF model generally consists of three different graphs that cooperate to support dynamic changes to the computational structure. These cooperating graphs are called the *init graph* ϕ_i , *subint graph* ϕ_s, and *body graph* ϕ_b. The body graph usually models the core functional behavior of the subsystem, while the init and subint graphs control dynamic parameters for the body graph. In our dynamic version of LPC, the init graph implements the main function that initializes the segment size, and calls the subinit graph to determine the frame size. The frame size is then passed as a parameter to the body graph. Prior to each invocation of the body graph, the corresponding subinit graph is called to determine the frame size that is to

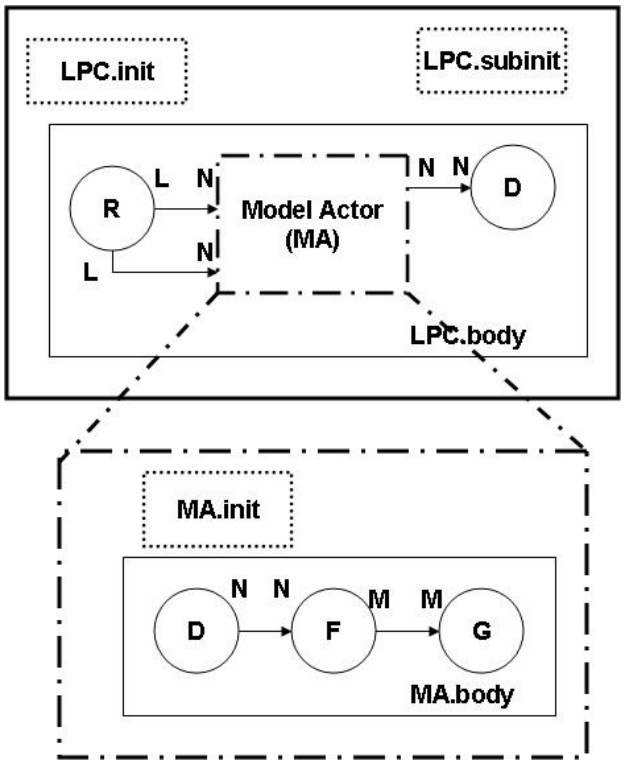

Fig. 2. Dynamic version of LPC.

be used for the new invocation. The overall block diagram of our dynamic application version is illustrated in Figure 2.

Here, the init graph is denoted as LPC.init and the subinit graph is denoted as LPC.subinit. The body graph, denoted as LPC.body, contains a (nested) hierarchical PSDF subsystem called which is abstracted in the higher level subsystem as a block called *Model Actor*. The internal representation of the Model Actor block has MA.init as its init graph and MA.body as its body graph.

PSDF can be viewed as a method for modeling dynamically reconfigurable dataflow graphs. Application models in terms of PSDF are useful for analyzing and optimizing the structure of software implementations when important characteristics of the actors or subsystems, including their dataflow properties, can vary dynamically or are otherwise not known at design time. The PSDF approach naturally accommodates adaptive applications such as our targeted framework of energy-adaptive compression, and allows variable amounts of data to be produced and consumed by functional blocks. Detailed background on PSDF modeling is developed in [1], [5].

V. Experimental Approach

We have developed embedded software implementations of the static and dynamic compression schemes discussed in this paper. Computational blocks (dataflow actors) are implemented in the C language, and designed in a modular way so that they can be used conveniently in other applications, such as other compression schemes that we consider in our future work. A *main* function is implemented to provide the overall schedule of the actors. The schedule provides for the sequencing of the computational blocks according to their relative rates of data production and consumption, and their data precedence constraints.

For interfacing purposes, an *ActorInput* block transfers successive data values from the signal source, and injects corresponding tokens into the dataflow graph for processing. The input data injected in this way is divided into frames using the *ActorFrame* block. In the dynamic version, the frame size to employ for compression is calculated based on first-order statistics, where the variance is minimized. The model order is calculated based on *Akaike Information Criteria*. The linear predictor coefficients are determined by generating the autocorrelation matrix of the difference equation of the *AR* process, and solving the matrix using *LU* decomposition.

For our experiments, we use development tools associated with the Texas Instruments fixed-point processor models 6000 and 5000, which are the target processors employed in our experiments. Specifically, we use the 64xx simulator and the 5509 DSK board.

VI. Descriptions of Functional Modules

This section provides details of the lower level computational modules used in our implementations.

A. Static Version Actors

- A (Buffer Size): sets the segment size L of the input data.
- B (Input Data): reads a segment of input data and stores it in a buffer.
- D (FFT): implements a Fast Fourier Transform (FFT) operation on the input segments.
- C (FFT Size): calculates the size of the required FFT operation based on the frame size.
- E (Model order): determines the fixed model order for the frames.
- F (LU Decompose): performs LU decomposition for determining the predictor coefficients.
- G (arError): generates the error on segments using the AR model.
- H (QxE): implements Huffman coding on the error segments.

B. Dynamic Version Actors

Descriptions of the actors of this adaptive model are as follows

- LPC.init: sets the segment size L.
- LPC.subinit: sets the frame size N.
- LPC.body: implements the body graph — i.e., the core, parameterized computation of the compression system.
- R: reads a segment of input data and stores it in a buffer.

The actors D, F, G and H have the same functionality as described in Section VI-A.

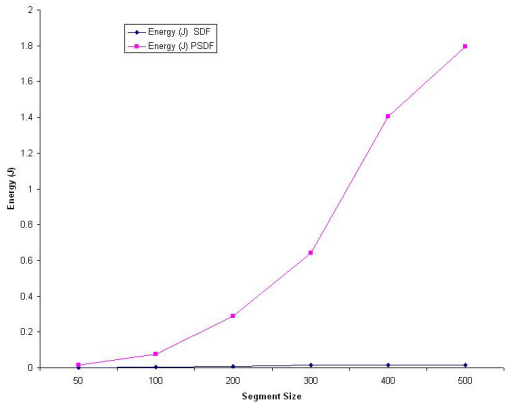

Fig. 3. Energy consumption comparison.

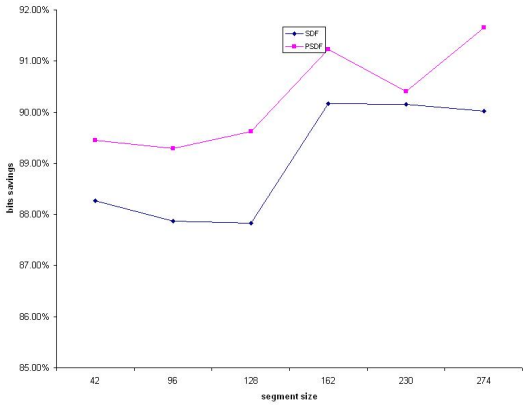

Fig. 4. Comparison in depth of compression.

TABLE I
CPU CYCLES FROM TWO DIFFERENT MODELS ON C64XX

Segment Size	SDF	PSDF
50	2,519,928	22,999,430
100	5,166,166	113,232,300
200	10,589,544	426,023,069
300	21,314,321	941,318,880
400	22,118,403	2,065,488,259
500	2,293,163,2	2,637,102,688

TABLE II
CPU CYCLES ON THE C5509A DSK.

Segment Size	SDF	PSDF
50	7,428,668	8,779,214
100	15,331,785	20,323,870
150	28,475,036	31,775,540
200	31,407,796	44,890,908

VII. RESULTS

Experiments were carried out to compare the static and dynamic versions of LPC-based compression, and measure their performance. C implementations, derived by hand-coded transformations from the dataflow representations, of the two versions were run on the Texas Instruments Code Composer Studio with the target simulator set as the fixed point DSP simulator C64xx. The implementations were also run on a C5509A DSK DSP processor.

The input data we used was a representative segment of seismic and magnetic vehicle data in *wav* format from a recent test in Yuma, Arizona. This data was collected directly from one of the sensors at the Yuma Proving Grounds. In these tests, a *wav* file reader removes the header and converts the data from *wav* format into raw data. The framing, linear prediction and quantization are performed on the raw data and from these operations, we obtain binary, compressed data as the output.

Initialy our experiments were conducted on the Texas Instruments Code Composer simulator with the TI 64XX as the target processor. The CPU frequency was set at 25 MHz. Figure 3 shows the comparison of the the two alternative dataflow models for segment sizes ranging from 50 to 500. This graph gives a plot of the energy consumption for the two different models.

Table I gives the numbers of CPU cycles required for the two different models.

Figure 4 shows a comparison of the two models with respect to the depth of compression that is achieved. From figure 4, we see that the savings for our dynamic (PSDF-based) version is better than the static (SDF-based) version of LPC. For experimental purposes, we have assumed that each sample of raw input data will fit within 14 bits. Values for samples fit within a range of +/- 10000.

We have conducted similar experiments on the TMS320VC5509A DSK, which also provides a fixed point signal processor. This is a hardware emulator board. For our measurements, the project is loaded onto the board, and executed, and the clock cycles and energy consumption are recorded. Table II gives the CPU cycles obtained from these experiments.

Table III gives the number of bits after compression for the different application versions.

The results above quantifies how on the hardware environment of the C5509A DSK, the dynamic version consumes more energy on computation compared to the static version, but the dynamic version also reduces the data required for

TABLE III
NUMBER OF BIT AFTER COMPRESSION WITH DIFFERENT MODELS.

Segment Size	SDF	PSDF
50	260	152
100	557	485
150	842	723
200	1131	1033

TABLE IV

RELEVANT PARAMETERS FOR THE C64XX AND C5509A PROCESSORS.

Parameter	C64xx	C5509A
cycle time	40ns	9.26ns
memory	2MB	128KB

TABLE V

CPU CYCLES FROM TWO DIFFERENT PROCESSORS FOR STATIC VERSION.

Segment Size	C64xx	C5509A
50	2,519,928	7,428,668
100	5,166,166	15,331,785
150	10,208,819	28,475,036
200	10,589,544	31,407,796

transmission to a smaller size. Overall, the total energy consumption for the dynamic version is lower, and it therefore achieves a more favorable trade-off between computation and communication.

In addition, the specific processor that is used is also a factor in determining overall energy efficiency. In the experiments, we make use of both the C64xx simulator and the C5509A DSK board as platforms to evaluate system performance. Note that the C64xx is more powerful processor compared to the C5509A. Parameters related to the performance are listed in Table IV. Only internal memory is used in our experiments.

Table V shows differences in required clock cycles between different processors for the static version of compression. In comparison, for the same segment size, the C64xx uses less clock cycles than the C5509A to perform data compression.

Note that in projects on the C64xx, parameters of Huffman coding are less accurate than for C5509A, and this results in less required clock cycles and less bits after entropy coding.

VIII. CONCLUSIONS

Energy efficiency is a critical consideration for sensor network applications, such as those used in surveillance and monitoring. In this paper, we have explored energy consumption trade-offs in sensor nodes between computation (embedded processing) and communication (transceiver operation). We have focused specifically on energy consumption trade-offs associated with data compression, which generally reduces transceiver energy consumption and the expense of increased computational energy. We have implemented data compression algorithms using different application models on different processors, and evaluated the performance and energy consumption. The experimental results quantify how total energy consumption can be decreased by increasing the computational load allocated for data compression. Our approach demonstrates a methodology for integrating algorithmic considerations of data compression along with application modeling, and software implementation for optimizing overall energy consumption. In our future work, we will build on this methodology to systematically tailor the communicaiton/computation trade-off associated with compression to targeted sensor node platforms.

IX. ACKNOWLEDGEMENTS

This research is sponsored by the Advanced Sensors Collaborative Technology Alliance.

REFERENCES

[1] B. Bhattacharya, S. S. Bhattacharyya,"Parameterized dataflow modeling for DSP systems," *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2408–2421, October 2001.

[2] M. Hans, R. W. Schafer "Lossless compression of digital audio," *IEEE Signal Processing Magazine*, vol. 18, no. 4, pp. 21–32, July 2001.

[3] Y. Zhang and J. Li, "Efficient seismic response data storage and transmission using ARX model-based sensor data compression algorithm," *Earthquake Engineering and Structural Dynamics*, vol. 35, pp. 781–788, 2006.

[4] D. Ko and S. S. Bhattacharyya, "Modeling of block-based DSP systems," *Proceedings of the IEEE Worksop on Signal Processing Systems*, vol. 40, no. 2, pp. 381–386, August 2003.

[5] D. Ko and S. S. Bhattacharyya, "Dynamic configuration of dataflow graph topology for DSP system design," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. , no. , pp. 69–72, March 2005.

[6] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous dataflow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, no. 2, February 1982.

[7] E. A. Lee and J. C. Bier, "Architectures for statically scheduled dataflow," *Journal of Parallel and Distributed Computing*, vol. 10, pp. 333–348, December 1990.

[8] S. S. Bhattacharyya,"Hardware/software co-synthesis of DSP systems," *Programmable Digital Signal Processors: Architecture, Programming, and Applications*, Y. H. Hu, Ed. Marcel Dekker, Inc., 2002, pp. 333–378.

[9] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.