Energy Optimization of Distributed Embedded Processors by Combined Data Compression and Functional Partitioning [*]

Jinfeng Liu, Pai H. Chou
Center for Embedded Computer Systems
University of California, Irvine, CA 92697-2625, USA
{jinfengl, phchou}@uci.edu

Abstract

Transmitting compressed data can reduce inter-processor communication traffic and create new opportunities for DVS (dynamic voltage scaling) in distributed embedded systems. However, data compression alone may not be effective unless coordinated with functional partitioning. This paper presents a dynamic programming technique that combines compression and functional partitioning to minimize energy on multiple voltage-scalable processors running pipelined data-regular applications under performance constraints. Our algorithm computes the optimal functional partitioning, CPU speed for each node, and their respective compression ratios. We validate the algorithm's effectiveness on a real distributed embedded system running an image processing algorithm.

1 Introduction

Dynamic voltage scaling (DVS) has been studied extensively as a power-saving technique for applications with slacks. By lowering the voltage and slowing down the processor to fill the slack, one can potentially achieve quadratic energy saving in CMOS technologies. However, if the application does not have much slack to begin with – that is, if the processor is always around its peak utilization – then DVS will not achieve any saving alone. Instead, it is well known that by increasing parallelism, one can afford to slow down the clock to enable more voltage scaling opportunities without performance loss. By partitioning the workload onto multiple processors, each processor is now responsible for only a fraction of the workload and can now afford to slow down by DVS to run at more power-efficient levels. This, of course, assumes that the application is parallelizable and that architectural overhead on the parallelism can be well amortized. In processor-based systems, having multiple processors means either shared memory or message passing communication. This paper assumes message passing communication for modularity and scalability reasons.

While distributed systems have many attractive properties, they pay a higher price for message-passing communication. Each node now must handle not only I/O with the external world, but also I/O on the internal network. Common communication interfaces such as RS-232 or BlueTooth are serial and are relatively slow. As a result, even if the actual data workload is not large on an absolute scale, it appears expensive relative to the computation performance that can be delivered by today's low-power embedded microprocessors. Since I/O transactions always appear on the critical paths in that they carry data dependencies between processors, they have become a limiting factor in exploiting DVS opportunities through parallelism.

Compression has been applied to saving energy and increasing effective bandwidth in many areas, ranging from telephone modems and faxes to caches and memories. By compression and decompression before and after communication transactions, it will be possible to save significant amounts of energy in communication. This may sound like an obvious idea, and in fact it has been used from modem standard to cache and memory. For the multi-processor, message-passing architecture studied in this paper, however, the trade-offs are not obvious and may even be counterintuitive. Compression can free up extra time budget by reducing the long communication delays in embedded systems. This extra time can be utilized towards either higher performance, or as additional DVS opportunities for energy savings. Different compression algorithms are available with different compression ratios, and even within an algorithm, it may be possible to set different target compression factors for both lossy and lossless algorithms. The compression algorithm chosen by a sender will not only dictate the receiver's decompression algorithm, but also determine the receiver's I/O delay and CPU speed. Thus, it can make a global impact on all communicating processors on their choices of compression algorithms and CPU clock rates with DVS. The design space becomes even larger if we also consider multi-speed communication interfaces.

The main challenge is that the selection of CPU speed, communication speed, and compression algorithms cannot be performed independently or greedily, because a local decision can have a global impact. The CPUs cannot all be run at the slowest, most power-efficient speeds, because they must compete for the available time and power with each other and with the communication interfaces. A high-ratio compression algorithm with time and power overhead may actually save energy by creating opportunities for voltage scaling the processors. Greedily saving power for communication or computation may actually result in higher overall energy. At the same time, functional partitioning must be an integral part of the optimization loop, because different partitioning schemes can dramatically alter the communication and computation workload for each node. For a given workload on a networked architecture, our problem statement is to generate a functional partitioning scheme, select the corresponding compression/decompression algorithms, and select the speeds of processors to perform computation tasks and compression/decompression, such that the total energy is minimized. In general, this is an extremely difficult optimization problem. Fortunately, for a class of systems with pipelined communication patterns under a latency constraint, efficient, exact solutions exist. This paper construct such a system model and formulate the energy consumed by communication, computation, compression and decompression within their available time budget. We present an efficient multi-dimensional dynamic programming solution to minimize system energy. We demonstrate the effectiveness of this technique with an image processing algorithm mapped onto a fully implemented distributed embedded system.

2 Related Work

Besides the well-known DVS techniques, previous studies also explored compression schemes for caches and memory busses to reduce energy in embedded processors. [8, 3] applied compression to

[*]This research was sponsored in part by DARPA contract F33615-00-1-1719 and by National Science Foundation under grant CCR-0205712

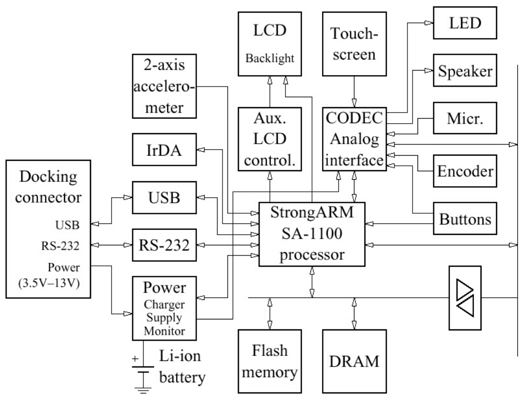


Figure 1: The block diagram of Itsy.

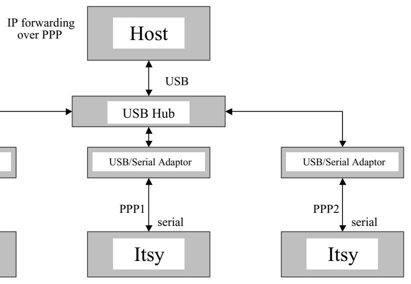


Figure 2: Networking Itsy nodes with a host computer.

reduce the code size and memory accesses for an SoC architecture. [4, 1] proposed bus encoding schemes to minimize the switching activities on the memory bus. These techniques often do not target inter-processor communication for multi-processor systems.

Many power management techniques including DVS have recently been extended to multi-processor systems. [9] extends DVS to inter-connection networks by tuning the data rate with various power levels to reduce communication energy. [11, 10] proposed partitioning the computation onto a voltage scalable multi-processor architecture that consumes significantly less power than a single processor. [5] reduces switching activities of both functional units and communication links by partitioning tasks onto a multi-chip architecture; while [7] maximizes the opportunity to shut down idle processors through functional partitioning. All these techniques primarily focus on either the computation or the communication aspect without exploring the interaction between them.

3 Motivating Example

Our experimental platform consists of multiple Itsy pocket computers as distributed processing nodes. Itsy was developed by Compaq Western Research Lab [2, 6]. It supports DVS on the StrongARM SA-1100 processor with 11 different frequency levels from 59–206.4MHz. Its block diagram is shown in Fig. 1. We mapped an automatic target recognition (ATR) image processing algorithm onto this distributed architecture with multiple Itsy Pocket Computers to evaluate the impact of data compression with different performance vs. power trade-offs.

We set up a separate host computer with multiple serial ports to to connect the Itsy nodes with each other through PPP connections. The host computer also provides IP forwarding service to allow Itsy nodes to communicate with each other transparently on the same TCP/IP network. The network configuration is shown in Fig. 2.

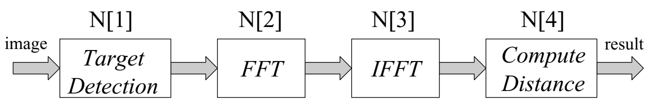


Figure 3: The ATR algorithm.

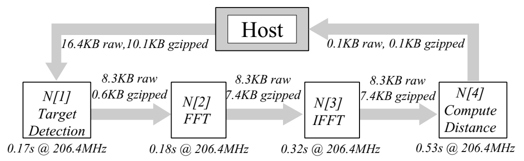


Figure 4: Performance profile of ATR on Itsy.

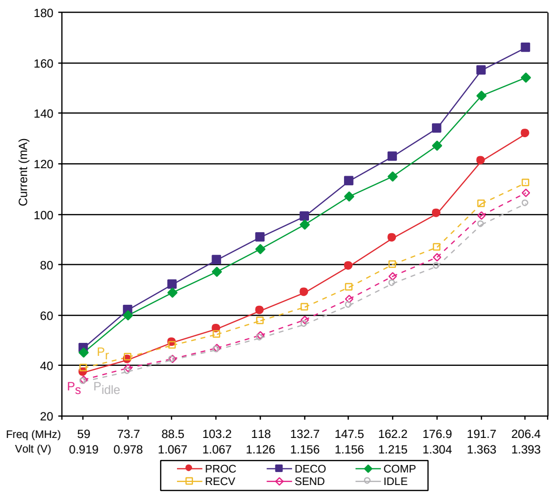


Figure 5: Power profile of ATR on Itsy.

3.1 Running ATR on Itsy

The structure of the ATR algorithm is shown in Fig. 3 and Fig. 4. It performs four sequential processing stages to an image frame. We constructed a parallel version of the algorithm such that it can be mapped onto 1, 2, 3, or 4 Itsy nodes with pipelined communication patterns. Given a *frame delay D* as the performance constraint, the host computer provides one image and collects one result in every *D* seconds. Pipelining allows each Itsy node to run at a lower frequency while maintaining the same throughput. However, communication between adjacent nodes costs additional time and power. Fig. 4 also summarizes the performance profile of ATR on Itsy. The performance degrades proportionally with the CPU clock frequency. The maximum data rate of the serial port is 115.2Kbps, though our measured data rate is 70–80Kbps over TCP/IP. Therefore, even though the raw data size is not large, the communication still takes long delays (e.g., 0.85s for 8K bytes). To reduce long communication delays, we compress the data before transmission and decompress after using `gzip` on the host computer (source and sink of images) and on each of the Itsy nodes (image processing stages). Compression and decompression take less than 10ms. For brevity they are omitted in Fig. 4.

Fig. 5 shows the measurement results of the current draw (in mA) over different speeds of an Itsy node running different tasks. The horizontal axis represents the frequency and voltage levels. Itsy has a 4V lithium-ion battery supply. Therefore, the curves refer to the actual power consumption ranging from 100mW to 700mW. We refine the tasks of this multi-node ATR system as follows:

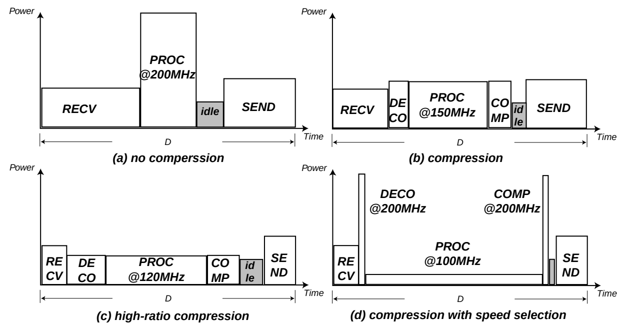


Figure 6: The impact of data compression to one node.

Name of task	Description
SEND	sending communication transaction
RECV	receiving communication transaction
PROC	execution of the ATR algorithm
COMP	data compression before *SEND*
DECO	data decompression after *RECV*

Tasks *DECO* and *COMP* dominate the power consumption. However, since their execution delays are short, task *PROC* is the primary energy consumer. *SEND* and *RECV* also take long delays, but their power levels are relatively low. We allow *PROC*, *DECO*, and *COMP* to operate at any CPU clock rate enabled by DVS. However, during communication (that is, *SEND* and *RECV*), we set the CPU speed to the lowest power state (0.919V at 59MHz), since there is no performance benefit to running the CPU faster during serial communication. When a node idles, we also set its CPU frequency to 59MHz. In addition, to avoid extra power draw from other components, we completely shut down unnecessary peripherals, including the LCD screen and the speaker during all experiments.

3.2 Data Compression for One Node

Fig. 6 illustrates the trade-offs between compression and DVS for a single node in a distributed system. The node performs tasks *RECV*, *DECO*, *PROC*, *COMP*, *SEND* in a sequential order. The areas of the bars represent the energy consumption. The delay D represents the performance constraint.

Compressed Data vs. Raw Data

Fig. 6(a) shows a node without data compression. Due to the long communication delays, the processor must run at $\geq$200MHz to finish all tasks by time D. In Fig. 6(b), data compression reduces the communication load at the cost of additional computation workload. If the reduced communication delay exceeds the extra compression/decompression delays, then this new slack can be applied towards DVS at a much lower power level (150MHz). Compression/decompression could also allow the node to deliver higher performance with a reduced delay on its critical path.

The Impact of Different Compression Algorithms

Different compression algorithms can achieve different compression ratios over a given piece of raw data. Unlike Fig. 6(b) which uses only one compression algorithm, Fig. 6(c) applies alternative algorithms with higher compression ratios to further reduce communication delays. This creates DVS opportunities for reducing the CPU clock rate to 120MHz. Algorithms with higher compression ratios typically require more energy with more CPU cycles, but if this overhead can be more than compensated by aggressive DVS, then (c) will consume less energy than (b).

Compression and CPU Speed Selection

In Fig. 6(c), the idle period cannot be further utilized for DVS. Many DVS studies indicated the three tasks *DECO*, *PROC* and *COMP* must operate at the same CPU speed to achieve minimum energy, under an assumption that the CPU clock rate can be scaled continuously to fully utilize the slack time (idle period). However it is not true in reality when the processor can operate only at discrete frequency levels. If the processor further reduces its frequency to the next level, e.g., 100MHz, it will fail to meet the timing constraint. The idle period represents the wasted (or, fragmented) time budget, when DVS can be performed on only a few discrete frequencies. Fig. 6(d) shows an alternative solution. It runs *DECO* and *COMP* at higher frequencies to allocate more time budget for *PROC*. As a result, task *PROC* can be run with a reduced clock rate at 100MHz to make better use of the idle time. Although (d) spends more energy on *DECO* and *COMP* than (c) does, (d) can still be a better solution if it can save more energy on *PROC*. Deciding whether this is possible requires simultaneous selection of the CPU speeds and compression algorithms. A different compression algorithm can significantly alter the communication delay and the compression/decompression time, which would result in a different slack available for DVS.

3.3 Data Compression for Pipelined Nodes

Next, we map the ATR algorithm onto multiple pipelined nodes. Fig. 7 shows a two-node pipeline in which the whole computation workload is partitioned onto two nodes $N'[1], N'[2]$. Having two nodes requires node $N'[1]$ to transmit its output to $N'[2]$. This is an extra communication transaction not in the single-node case and is denoted as $\mathit{SEND}[1] \rightarrow \mathit{RECV}[2]$. All stages of the pipeline must have the same deadline D such that if node $N'[1]$ is fed one image frame in every D seconds, node $N'[2]$ must always produce one result in every D seconds.

The trade-offs between communication and computation with data compression discussed earlier for the single node are generally applicable to pipelined multiple nodes, too. With multiple nodes, network contention tends to have a greater impact on the entire system and therefore must be avoided through the selection of compression algorithms and partitioning schemes.

Compression Algorithm Selection in the Pipeline

Having a choice of compression algorithms adds a new dimension of communication-computation trade-offs in multiple processors. By selecting a compression algorithm for a sender, it forces the receiver to choose the corresponding decompression algorithm, thereby affecting not only the receiver's communication delay but also the receiving node's CPU speed. Then, the choice of the receiver's CPU speed could further affect the receiver's compression algorithm and the subsequent nodes in a chain effect. A locally optimal choice for the first node will not necessarily lead to a globally optimal solution.

Partitioning with Compression

Data compression also affects the choices of partitioning schemes. This is primarily because different data do not compress equally well even by the same algorithm. As an example, Fig. 7(a) shows the the optimal partitioning scheme without data compression for two nodes with the minimum internal communication payload (8.3KB). However, Fig. 7(a) is no longer optimal with data compression, because the internal data from $N[2]$ to $N[3]$ cannot be effectively compressed (8.3KB down to 7.4KB), and the relatively long communication delay limits DVS opportunities. The optimal partitioning scheme is shown in Fig. 7(b) by remapping task $N[2]$ to node $N'[2]$. Although the raw data size from $N[1]$ to $N[2]$ is also 8.3KB, the data can be compressed very well (8.3KB down to

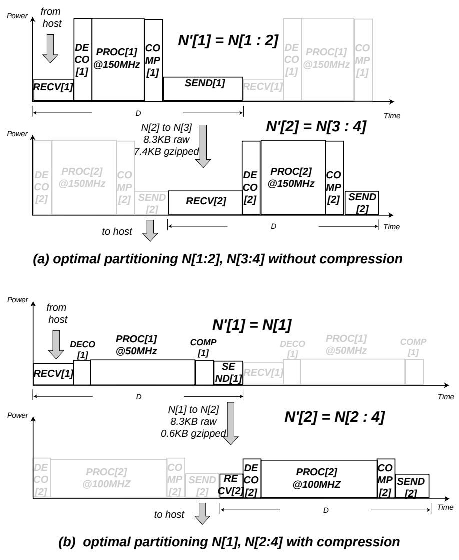


Figure 7: The impact of data compression with partitioning.

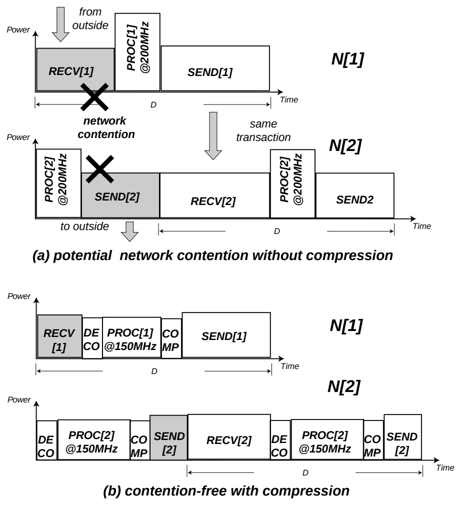


Figure 8: Data compression eliminates network contention.

0.6KB) to effectively reduce the communication delay. As a result, both nodes are able to operate at much lower power levels with more energy savings, although the computation loads on the two nodes are more imbalanced compared to Fig. 7(a).

Compression to Reduce Network Contention

Given the assumption of a shared communication medium, all communication transactions should be scheduled into different time slots. Since a transaction consists of a pair of send and receive tasks on neighboring nodes, they should be scheduled together. As an example in Fig. 8(a), *SEND*[1] and *RECV*[2] should always occupy the same time slot. In the case of long communication delays, two different transactions such as *RECV*[1] and *SEND*[2] might overlap in time slots, causing network contention. If the network utilization is over-saturated, one way to eliminate network contention is to increase the stage delay *D*, but this causes performance degradation. Alternatively, data compression can reduce network utilization and eliminate the network contention while maintaining the same performance, as shown in Fig. 8(b).

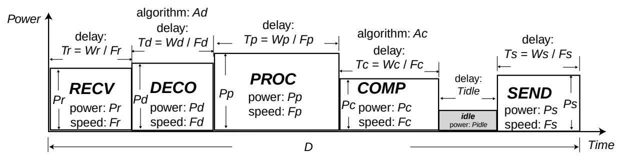


Figure 9: Timing and power diagram of a processing node.

To summarize, this paper exploits communication-computation trade-offs in the context of a distributed embedded architecture. These trade-offs include timing budget for both communication and computation, compression algorithm selection with DVS fragmentation, and compression algorithm selection with functional partitioning. We next formulate a multi-dimensional optimization approach to effectively minimize energy consumption for both communication and computation on all nodes.

4 System Model

This section defines a system-level performance/energy model of a distributed embedded system running an application with a natural pipelined organization. We first define the process-to-architecture mapping followed by the associated cost functions.

4.1 Node

A *node* is a computer in our system. It consists of a processor, local memory, one or more communication interfaces, and optional compression and decompression units. A *processing job* assigned to a node is modeled in terms of five *tasks*: *RECV*, *DECO*, *PROC*, *COMP* and *SEND* that must be executed serially in this order. A node receives data by *RECV*, decompresses the data by *DECO* if necessary. Then task *PROC* produces the result that can be compressed by *COMP* if necessary. Finally, the result is sent to the next node by *SEND*. Fig. 9 shows the timing vs. power diagram of a node. The total area of these five tasks plus the idle period represents the energy consumption of the corresponding node.

Each task has its *workload* W. For the computation tasks *PROC*, *DECO* and *COMP*, their workload W_p, W_d and W_c refer to the number of cycles. For communication tasks *RECV* and *SEND*, workload W_r and W_s indicate the communication payloads in the number of bits.

Let T_p, T_r, T_s, T_d, T_c denote the *execution times* of tasks *PROC*, *RECV*, *SEND*, *DECO* and *COMP*, respectively. The performance constraint is a *delay* D to finish all tasks, that is $T_r + T_d + T_p + T_c + T_s \leq D$ for the five serialized tasks. There could be an idle period T_{idle} during which the node is not performing any of the five tasks.

$$T_r + T_d + T_p + T_c + T_s + T_{idle} = D \tag{1}$$

Let F_p denote the CPU clock frequency to perform task *PROC*, F_r and F_s the respective bandwidths for receiving and sending, and let F_d and F_c be the processing speeds of decompression and compression, performed by the processor or other hardware units. Let P_p, P_r, P_s, P_d, and P_c denote the *power level of tasks*, and E_p, E_r, E_s, E_d and E_c be the *energy consumption of tasks*, P_{idle} and E_{idle} be the power level and energy consumption of the idle period. Finally, let E_N denote the energy consumption of a node. We have

$$T_p = \frac{W_p}{F_p}; \;\; T_r = \frac{W_r}{F_r}; \;\; T_s = \frac{W_s}{F_s}; \;\; T_d = \frac{W_d}{F_d}; \;\; T_c = \frac{W_c}{F_c} \tag{2}$$

$$\begin{aligned} E_p = P_p T_p; \quad E_r = P_r T_r; \quad E_s = P_s T_s; \\ E_d = P_d T_d; \quad E_c = P_c T_c; \quad E_{idle} = P_{idle} T_{idle}; \end{aligned} \tag{3}$$

$$E_N = E_p + E_r + E_s + E_d + E_c + E_{idle} \tag{4}$$

(2) is a reasonable estimate for processing units executing data-dominated tasks, including *PROC*, *DECO* and *COMP*, where the total cycles W can be analyzed and bounded statically. The communication bandwidth is normally less than the rated maximum data rate and can be measured or profiled.

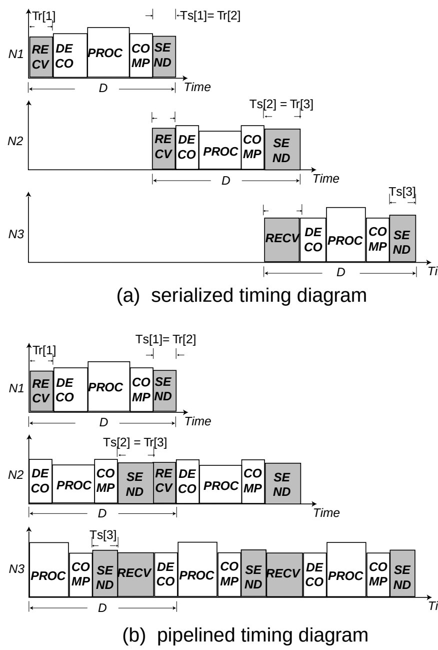
A node can choose from a set $\alpha_c[1:C]$ of compression algorithms. The corresponding set of decompression algorithms $\alpha_d[1:C]$ must be used by the receiver to correctly recover the raw data. We denote a decompression and a compression algorithm as A_d, A_c, which are members of two ordered sets of algorithms, $A_d \in \alpha_d$ and $A_c \in \alpha_c$, respectively. If the raw data is compressed by the j^{th} compression algorithm $\alpha_c[j]$, it must be decompressed by the j^{th} decompression algorithm $\alpha_d[j]$ on the receiver. Decompression or compression are not necessary if the node receives or sends uncompressed data. Without loss of generality, we order the two sets of algorithms such that $\alpha_c[1]$ and $\alpha_d[1]$ perform no compression and decompression, and their execution time is zero. That is, if $A_d = \alpha_d[1]$ or $A_c = \alpha_c[1]$, then $T_d = 0$ or $T_c = 0$.

It must be noted that some parameters are functions of other parameters rather than constant values. For example, communication workload W_r, W_s and delay T_r, T_s are functions of A_d and A_c regarding different compression algorithms and compression ratios, which are dependent on not only the raw data sizes $W_{r_{raw}}$ and $W_{s_{raw}}$, but also the "data nature" (how compressible the data is with respect to the compression algorithms) of the raw data. To be precise, workload W_d, W_c, W_r, W_s should be denoted as functions of A_d, A_c and the raw data r_{raw}, s_{raw}, e.g., $W_s(A_c, s_{raw})$ indicating the data to be sent is a function of the compression algorithm A_c and the raw data s_{raw}. These functions can usually be analyzed or profiled as lookup tables. For example, in the ATR algorithm, each compression algorithm's delay, power and compression ratio for different background scenes can be profiled, and the results can be organized as lookup tables.

We assume the CPU frequency can be chosen from a discrete ordered set $\phi[1:S]$, that is $F_p \in \phi[1:S]$. For example, a voltage scalable processor can operate at a few discrete clock rates. If decompression and compression are performed as software routines on the processors, their speeds are also chosen from the same set, $F_d, F_c \in \phi[1:S]$. The power levels of tasks *PROC*, *DECO*, *COMP* are directly related to the CPU frequencies. In addition, the tasks may consume different power levels even if they run on the same processor with the same clock rate. Therefore, the power levels P_p, P_d and P_c are also functions (lookup tables) of F_p, F_d and F_c, rather than constant values. For example, the ATR algorithm's power profile on Itsy (Fig. 5) consists of multiple lookup tables. In this paper we omit the details of lookup tables to keep the notation concise.

4.2 M-node Pipeline

We consider a specialized organization, called an *M-node pipeline*, of such a distributed embedded system. It consists of M pipelined nodes $N[1:M]$. Each node $N[i]$ receives data from the previous node $N[i-1]$ (except the first node $N[1]$ that receives from an outside source), followed by decompression (if necessary), processing, compression (if necessary), and finally sends the result to the next node $N[i+1]$ (except the last node $N[M]$ that sends the result to an external destination). Each pair of tasks $SEND[i] \rightarrow RECV[i+1]$ refers to the same communication transaction with the same data on both ends. We assume $SEND[i]$ and $RECV[i+1]$ take the same communication delay, and they start and finish at the same time. That is, $W_s[i] = W_r[i+1], F_s[i] = F_r[i+1], T_s[i] = T_r[i+1]$. In each pair of tasks $COMP[i] \rightarrow DECO[i+1]$, the decompression side must choose the appropriate algorithm to correctly recover the data. That is, if $A_c[i] = \alpha_c[j], A_d[i+1] = \alpha_d[j]$. All nodes have the same delay D, and each node acts as a pipeline stage with delay D. Fig. 10 shows an example of a three-node pipeline. Fig. 10(b) shows the pipelined timing diagram by folding the tasks in Fig. 10(a) into a common in-



(a) serialized timing diagram

(b) pipelined timing diagram

Figure 10: A three-node pipeline.

terval with duration D, which is the delay of each pipeline stage. During each time interval with a duration D, the first node of the pipeline will be fed with one set of incoming data; meanwhile one set of resulting data will be produced by the last node. The pipeline timing diagram can easily identify conditions of network contention on the shared communication media.

Let $E[i]$ be the energy consumption of a node $N[i]$ (4). The *total energy consumption of a pipeline E_{sys}* is defined as follows,

$$E_{sys} = \sum_{i=1}^{M} E[i] \qquad (5)$$

An M-node pipeline $N[1:M]$ can be partitioned into M' segments and mapped onto an M'-node pipeline $N'[1:M'](M' \leq M)$ by merging adjacent nodes $N[i:j](i \leq j)$ into a new node $N'[k]$. The new node $N'[k]$ combines all computation workloads. Communication transactions within a node become local data accesses, and the corresponding compression/decompression tasks are eliminated. That is, $W'_p[k] = \sum_{l=i}^{j} W_p[l]$, $W'_r[k] = W_r[i], W'_s[k] = W_s[j]$, and $W'_d[k] = W_d[i], W'_c[k] = W_c[j]$. The new M'-node pipeline is called a *partitioning* of the initial M-node pipeline.

5 Problem Formulation

In this section we formulate three energy minimization problems by: (1) compression algorithm and CPU speed selection for one node, (2) compression algorithm and CPU speed selection for a pipeline with a fixed partitioning scheme, and (3) combined compression algorithm and CPU speed selection with functional partitioning for the pipeline. For all three problems, we assume the delay, power and compression ratios of all corresponding tasks are known as functions or look-up tables and the details are omitted.

Problem 1. Optimal Compression Algorithm and CPU Speed Selection for One Node

Given

(a) a node N with processing load W_p, communication payload $W_{r_{raw}}, W_{s_{raw}}$ in raw data r_{raw}, s_{raw}, and

```
OPT-1($W_{rraw}, W_{sraw}, W_p, \alpha_d[1:C], \alpha_c[1:C], \phi[1:S], D$)
1  $E_{opt} \leftarrow \infty$
2  for $i \leftarrow 1$ to $C$ do
3      compute $E_r$ with $A_d = \alpha_d[i]$
4      for $j \leftarrow i$ to $C$ do
5          compute $E_s$ with $A_c = \alpha_c[j]$
6          for $k \leftarrow 1$ to $S$ do
7              compute $E_d$ with $A_d = \alpha_d[i]$, $F_d = \phi[k]$
8              for $l \leftarrow 1$ to $S$ do
9                  compute $E_c$ with $A_c = \alpha_c[j]$, $F_c = \phi[l]$
10                 derive $E_p, E_{idle}$
11                 compute $E_{node}$
12                 if $E_{node} < E_{opt}$ then $E_{opt} \leftarrow E_{node}$
13 return $E_{opt}$
```

Figure 11: Algorithm OPT-1: selecting optimal compression and decompression algorithms and CPU speeds for one node.

(b) C compression algorithms $\alpha_c[1:C]$, and the corresponding decompression algorithms $\alpha_d[1:C]$,

(c) S CPU frequencies $\phi[1:S]$,

(d) the delay D to finish all tasks,

Find

(1) the optimal decompression algorithms $A_d \in \alpha_d$, compression algorithms $A_c \in \alpha_c$, with

(2) CPU speeds $F_p, F_d, F_c \in \phi$ for tasks *PROC*, *DECO*, and *COMP* to minimize the total energy E_N.

The incoming data r_{raw} and the choice of the decompression algorithm A_d (assuming the sender will also agree to compress the data with the corresponding compression algorithm) determine the energy consumption E_r for *RECV*. Similarly, the outgoing data s_{raw} and the choice of the compression algorithm A_c decide the energy E_s for *SEND*. For *DECO* and *COMP*, the decompression algorithm A_d, the incoming data r_{raw}, and the CPU speed F_d decide E_d; and A_c, s_{raw}, F_c decide E_c. For *PROC*, E_p depends only on F_p. The choices of A_d and A_c are independent for one node. So are the choices of F_d and F_c, but together they decide F_p due to the timing constraint D. Therefore, we must enumerate over C choices of both A_d and A_c, and S choices of both F_d and F_c for the minimum energy consumption.

The algorithm shown in Fig. 11 has a runtime complexity of $O(C^2S^2)$. It selects the optimal compression/decompression algorithms, combined with the optimal CPU speed settings to overcome the DVS fragmentation problem. In reality C and S are usually small integers ranging from 3 to 10. Therefore the runtime complexity of this algorithm is close to a constant.

Problem 2. Optimal Compression Algorithm and CPU Speed Selection for Pipelined Nodes with a Fixed Partitioning Scheme

Given

(a) a fixed M-node pipeline $N[1:M]$ with processing load $W_p[1:M]$, communication payload $W_{r_{raw}}[1:M], W_{s_{raw}}[1:M]$ in raw data, and,

(b) C compression algorithms $\alpha_c[1:C]$, and the corresponding decompression algorithms $\alpha_d[1:C]$,

(c) S CPU frequencies $\phi[1:S]$,

(d) the same single-stage delay D for all nodes,

Find

(1) compression algorithms $A_c[1:M] \in \alpha_c^M$, and the corresponding decompression algorithms $A_d[1:M] \in \alpha_d^M$, subject to the following constraint: if $A_c[i] = \alpha_c[x]$ and $A_d[i+1] = \alpha_d[y]$, then $x = y$ (i.e., sender and receiver agree on the choice of compression/decompression algorithms)

(2) CPU speeds $F_p[1:M] \in \phi^M$, $F_d[1:M] \in \phi^M$, and $F_c[1:M] \in \phi^M$ to minimize energy E_{sys}.

In an M-node pipeline, there are $M+1$ communication transactions that require a combination of $M+1$ pairs of independent compression/decompression algorithms. The CPU speed selection is an $O(S^2)$ procedure to be performed on M nodes. Therefore, the overall enumeration space is $O(C^{M+1}S^2M)$. Problem 1 becomes a special case when $M = 1$. We propose a dynamic programming solution to eliminate exhaustive enumeration. We construct a series of optimal solutions to the sub-problems by selecting the compression algorithm for one node at a time. We compute the optimal cost function in terms of the minimum energy consumption over the sub-problems. Upon selecting a compression algorithm for each node, the new optimal sub-solution can be computed from past optimal sub-solutions. Therefore, dynamic programming is applicable.

We define an *energy matrix* $E[0:M, 1:C]$. Each entry $E[i,j]$ indicates the minimum energy of a sub-problem that selects the compression algorithms for the first i nodes, with the i^{th} node $N[i]$ using algorithm $A_c[i] = \alpha_c[j]$ for compression. All entries of E are initialized to ∞ except $E[0,j] = 0, \forall 1 \le j \le C$.

$$E[i,j] = \begin{cases} 0 & \text{for } i = 0, 1 \le j \le C \\ \min_{1 \le l \le C} \left[\begin{array}{l} E[i-1,l] + \\ E_N[i](\alpha_d[l], \alpha_c[j]) \end{array} \right] & \text{for } 1 \le i \le M, 1 \le j \le C \text{, if network is contention-free} \end{cases} \quad (6)$$

(6) indicates that the optimal solution for the first i nodes with $A_c[i] = \alpha_c[j]$ must be a combination of the following: (a) the minimum energy of the first $i-1$ nodes with the $(i-1)^{th}$ node's compression algorithm $A_c[i-1] = \alpha_c[l]$ for an $l \le C$; and (b) the optimal energy of the i^{th} node $N[i]$ with the corresponding l^{th} decompression algorithm $A_d[i] = \alpha_d[l]$ and $A_c[i] = \alpha_c[j]$. To compute (b), we can use algorithm OPT-1 (Fig. 11) to optimize the single node $N[i]$, with $A_d[i] \in \alpha_d[l:l]$ and $A_c[i] \in \alpha_c[j:j]$. Matrix E can be updated only if the network is contention-free. The dynamic programming algorithm can iterate (6) from $i = 1, j = 1$ until $i = M, j = C$. Finally, the minimum energy is $\min_j(E[M,j]), \forall j = 1, 2, \ldots, C$. The algorithm can be derived from a new algorithm to be presented in the next section (Fig. 12) as a special case. Therefore, we omit it for brevity. Its time complexity is $O(C^2S^2M)$, which is practically linear with M.

Problem 3. Optimal Compression Algorithm and CPU Speed Selection with Functional Partitioning for Pipelined Nodes

Given

(a) M pipelined nodes $N[1:M]$ with workload $W_p[1:M], W_{r_{raw}}[1:M], W_{s_{raw}}[1:M]$,

(b) C compression algorithms $\alpha_c[1:C]$, and the corresponding decompression algorithms $\alpha_d[1:C]$,

(c) S CPU frequencies $\phi[1:S]$,

(d) the same single-stage delay D for all nodes,

Find

(1) the optimal partitioning $N'[1:M']$, with

(2) compression algorithms $A'_c[1:M'] \in \alpha_c^{M'}$, and the corresponding decompression algorithms $A'_d[1:M'] \in \alpha_d^{M'}$, and

(3) CPU speeds $F'_p[1:M'] \in \phi^{M'}$, $F'_d[1:M'] \in \phi^{M'}$ and $F'_c[1:M'] \in \phi^{M'}$ to minimize energy E_{sys}.

We propose a two-dimensional dynamic programming algorithm shown in Fig. 12 to solve this more complex problem, whose solution space is exponential with M. [1] We define a three-dimensional *energy matrix* $E[0:M,1:C,0:M]$ as follows: each element $E[i,j,k]$ stores the minimum energy consumption of a sub-problem, which maps the first k original nodes $N[1:k]$ onto a new i-node sub-partitioning $N'[1:i]$, whose last node $N'[i]$'s compression algorithm is selected to be $A'_c[i]=\alpha_c[j]$. Matrix E is initialized to ∞, except $E[0,j,0]=0, \forall 1 \le j \le C$.

The optimal energy $E[i,j,k]$ is the summation of two portions. (a) $E[i-1,l,m]$ of a previous optimal sub-solution, which maps m original nodes $N[1:m]$ onto $i-1$ new nodes $N'[1:i-1]$, with node $N'[i-1]$'s compression algorithm selected as $A'_c[i-1]=\alpha_c[l]$. Plus, (b) the last new node $N'[i]$ that combines original nodes $N[m+1:k]$ with decompression algorithm $A'_d[i]=\alpha_d[l]$ and compression algorithm $A'_c[i]=\alpha_c[j]$. The sub-solution (a) has the optimal energy $E[i-1,l,m]$. (b) also must have the optimal energy for the only node $N'[i]$, and its optimal energy is denoted as $E_{N'[i]}(\alpha_d[l],\alpha_c[j])$. $E[i,j,k]=\min_{l,m}(E[i-1,l,m]+E_{N'[i]}(\alpha_d[l],\alpha_c[j])), \forall 1 \le l \le C, i-1 \le m \le k-1$. Matrix E can be updated by (7) only if the network is contention-free.

The algorithm is shown in Fig. 12. The global minimum energy is $\min_{i,j}(E[i,j,M]), \forall 1 \le i \le M, 1 \le j \le C$. Computing $E_{N'[i]}(\alpha_d[l],\alpha_c[j])$ by calling algorithm OPT-1 (Fig. 11) requires $O(S^2)$ time with both arrays $\alpha_d[1:C], \alpha_c[1:C]$ having only one element as $\alpha_d[l:l], \alpha_c[j:j]$. The runtime complexity of the algorithm is $O(C^2S^2M^3)$.

$$E[i,j,k]=\begin{cases} 0 & \text{for } i=k=0, 1 \le j \le C \\ \min\limits_{\substack{1 \le l \le C, i-1 \le \\ m \le k-1}} \left[\begin{array}{l} E[i-1,l,m]+ \\ E_{N'}[i](\alpha_d[l],\alpha_c[j]) \end{array}\right] & \text{for } 1 \le i \le k \le M,\ 1 \le j \le C, \text{ if network is contention-free} \end{cases} \tag{7}$$

If we let $k=i$ in this algorithm, the new partitioning algorithm is fixed to be the same as the original one, and the two loops over k and m (at line 9 and 11) will be eliminated. Then, the same algorithm can solve the previous Problem 2 on a fixed partitioning.

In reality, algorithm OPT-M and OPT-1 should also compute the optimal partitioning, decompression and compression algorithms, and CPU speeds for all nodes. Since it is not difficult to derive the optimal values of these parameters, we omit the details for brevity.

6 Experimental Results

We experiment with the ATR algorithm mapped onto one and two Itsy nodes. The delay D for each frame is used as the performance metric. We repeat executing the ATR algorithm until the battery is fully discharged. We define I to be the processed image count per node and use it as a measure of energy efficiency.

I. Experiments with One Node

With the one-node configuration, we perform three experiments:

(I.A) The baseline configuration is a single Itsy node to run the entire ATR algorithm at the maximum CPU speed of 206.4MHz, without data compression. Its peak performance is D = 2.9s for each frame and the node can process I = 10.1K images before the battery is exhausted.

[1]There are $\begin{pmatrix} i-1 \\ M-1 \end{pmatrix}$ $i-$node partitionings, each having $i+1$ compression/decompression instances with C^{i+1} choices, combined with S^2 CPU speed enumerations on i nodes. The total number of solutions is $\sum_{i=1}^{M}\begin{pmatrix} i-1 \\ M-1 \end{pmatrix} iS^2C^{i+1} = C^2S^2(C+1)^{M-2}(CM+1)$

```
OPT-M($W_{rraw}[1:M], W_{sraw}[1:M], W_p[1:M], \alpha_c[1:C], \alpha_d[1:C], \phi[1:S], D$)
1  for $i \leftarrow 1$ to $M$ do
2      for $j \leftarrow 1$ to $C$ do
3          for $k \leftarrow i$ to $M$ do
4              $E[i,j,k] \leftarrow \infty$
5  for $j \leftarrow 1$ to $C$ do
6      $E[0,j,0] \leftarrow 0$
7  for $i \leftarrow 1$ to $M$ do
8      for $j \leftarrow 1$ to $C$ do
9          for $k \leftarrow i$ to $M$ do
10             for $l \leftarrow 1$ to $C$ do
11                 for $m \leftarrow (i-1)$ to $(k-1)$ do
12                     if network is contention-free then
13                         $W'_p \leftarrow \sum_{q=m+1}^{k} W_p[q]$
14                         $E' \leftarrow$ OPT-1($W_{rraw}[m+1], W_{sraw}[k], W'_p$,
15                                 $\alpha_d[l:l], \alpha_c[j:j], \phi[1:S], D$)
16                         $e \leftarrow E[i-1,l,m] + E'$
17                         if $e < E[i,j,k]$ then $E[i,j,k] \leftarrow e$
18 $E_{opt} \leftarrow$ retrieve from matrix $E$
19 return $E_{opt}$
```

Figure 12: Algorithm OPT-M: selecting optimal compression and decompression algorithms and CPU speeds for pipelined nodes, combined with functional partitioning.

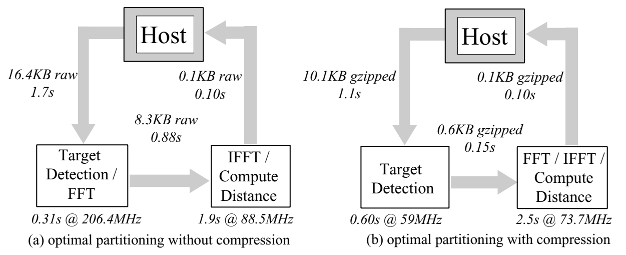


Figure 13: Partitioning schemes for two nodes.

(I.B) Same as (I.A) D = 2.9s but with data compression. Even though the entire ATR algorithm is mapped onto one node and internal communication is completely eliminated, compression still applies to communication with the external source (host computer). As a result, the processor can reduce its clock rate to 132.7MHz. Meanwhile the node can process I = 15.1K images with a 50% improvement in energy efficiency.

(I.C) Same as (I.B) except it maximizes performance. With compression, this achieves a higher peak performance at D = 2.3s while processing I = 11.5K images. That is, it can speed up the performance by 26% and increase the energy efficiency by 14% at the same time.

II. Experiments with Two Nodes

We also perform three similar experiments for the two-node pipeline.

(II.A) We use $D=2.9s$ from the the baseline configuration (I.A) for two nodes, without data compression.

(II.B) $D=2.9s$ with compression for energy efficiency

(II.C) With compression and maximum performance

For (II.A), the best partitioning scheme $N[1:2], N[3:4]$ without compression is shown in Fig. 13(a). With more parallelism, the CPU speeds may be reduced on both processors. However, the first node must still run at the fastest speed of 206.4MHz to achieve the

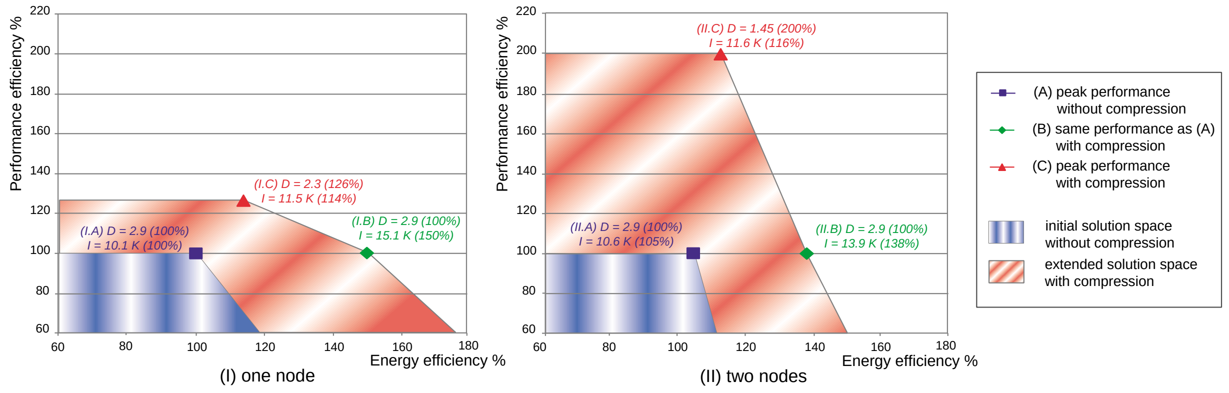


Figure 14: Extended solution space with data compression.

same performance of D = 2.9s, due to the long communication delays. The second node can operate at 88.5MHz with a much lower power level. As a result, the two-node pipeline can process 21.2K frames with two batteries. Therefore, I = 10.6K. Compared with (I.A), The energy efficiency is improved by 5%. The increased parallelism with two nodes cannot further improve performance due to imbalanced workload, where the first node must run at the highest speed.

(II.B) shows that data compression unveils a new optimal partitioning as $N[1], N[2:4]$ (Fig. 13(b)), because the raw data between the two new partitionings can be very well compressed (8.3KB down to 0.6KB). The saved time budget allows both nodes to reduce their CPU clock rates to 59MHz and 73.7MHz, respectively. While not perfectly balanced because the second node must now process more workload, this is a much better solution. The battery efficiency is increased by 38% with 27.8K images being processed by two nodes (I = 13.9 K).

(II.C) Data compression also allows a 100% speedup with D = 1.45s and a 16% improvement in the energy efficiency with I = 11.6K. Without data compression, it would be impossible to deliver higher performance with two nodes in (II.A).

In summary, Fig. 14 presents the Pareto views of solution spaces of our experiments. We sketch the boundaries of the solution spaces with straight lines for brevity. The performance efficiency and energy efficiency are normalized to the baseline configuration (I.A). (I.A) and (II.A) represent the peak performance levels without compression. (II.A) and (II.B) extend the energy efficiency through compression while delivering the same performance as (I.A) and (I.B). Finally, (I.C) and (II.C) improve both performance and energy efficiency at the same time. The curves along (I.B) – (I.C) and (II.B) – (II.C) represent many new solutions that were not possible without data compression. They strictly dominate (I.A) and (II.A) with both higher performance levels and lower energy consumption. It should be noted that in experiment (II.B) and (II.C), data compression achieves a much wider range of energy vs. performance trade-offs. This finding validates the concept that multiple processors can support both high-performance and low-power applications. However, as indicated by (II.A), increasing parallelism alone may not be effective unless it is explored synergistically with other trade-offs by a joint effort. These important trade-offs include selecting compression algorithms, CPU speeds and partitioning schemes that are discussed in this paper.

7 Conclusion

We present an energy optimization technique for distributed embedded systems. In such systems, communication and computation compete over time and power budgets for operating at the most energy-efficient states. It is critical to balance the time and power budget for both communication and computation on each node and across the whole system. With data compression, the system can be tuned towards either high performance with shortened critical delays, or low power with extra DVS opportunities. We present an exact multi-dimensional dynamic programming formulation that produces the energy-optimal solution as defined by a partitioning scheme with compression algorithm selections for all tasks. This technique is applicable to a whole class of data-oriented systems that can be structured in a pipelined organization.

Acknowledgment

Special thanks to HP Western Research Lab for providing Itsy. This research was sponsored by DARPA F33615-00-1-1719 and Natinal Science Foundation CCR-0205712.

References

[1] Y. Aghaghiri, F. Fallah, and M. Pedram. Reducing transitions on memory buses using sector-based encoding technique. In *Proc. International Symposium on Low Power Electronics and Design*, pages 190–195, August 2002.

[2] J. F. Bartlett, L. S. Brakmo, K. I. Farkas, W. R. Hamburgen, T. Mann, M. A. Viredaz, C. A. Waldspurger, and D. A. Wallach. The itsy pocket computer. Technical Report 2000/6, COMPAQ Western Research Laboratory, 2000.

[3] L. Benini, A. Macii, and A. Nannarelli. Cached-code compression for energy minimization in embedded processors. In *Proc. International Symposium on Low Power Electronics and Design*, pages 322–327, August 2001.

[4] W.-C. Cheng and M. Pedram. Power-optimal encoding for DRAM address bus. In *Proc. International Symposium on Low Power Electronics and Design*, pages 250–252, July 2001.

[5] R. Cherabuddi, M. Bayoumi, and H. Krishnamurthy. A low power based system partitioning and binding technique for multi-chip module architectures. In *Proc. Proc. Great Lakes Symposium on VLSI*, pages 156–162, 1997.

[6] W. R. Hamburgen, D. A. Wallach, M. A. Viredaz, L. S. Brakmo, C. A. Waldspurger, J. F. Bartlett, T. Mann, and K. I. Farkas. Itsy: stretching the bounds of mobile computing. *IEEE COMPUTER*, 34(4):28–36, April 2001.

[7] E. Huwang, F. Vahid, and Y.-C. Hsu. FSMD functional partitioning for low power. In *Proc. Design, Automation and Test in Europe*, pages 22–28, 1999.

[8] H. Lekatsas, J. Henkel, and W. Wolf. Code compression for low power embedded system design. In *Proc. Design Automation Conference*, pages 294–299, June 2000.

[9] L. Shang, L.-S. Peh, and N. K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proc. International Symposium on High-Performance Computer Architecture*, pages 91–102, February 2003.

[10] A. Wang and A. Chandrakasan. Energy efficient system partitioning for distributed wireless sensor networks. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 905–908, May 2001.

[11] E. F. Weglarz, K. K. Saluja, and M. H. Lipasti. Minimizing energy consumption for high-performance processing. In *Proc. Asian and South Pacific Design Automation Conference*, pages 199–204, 2002.