

# Combined Functional Partitioning and Communication Speed Selection for Networked Voltage-Scalable Processors \*

Jinfeng Liu, Pai H. Chou, Nader Bagherzadeh  
Department of Electrical & Computer Engineering  
University of California, Irvine, CA 92697-2625, USA  
{jinfengl, chou, nader}@ece.uci.edu

## Categories and Subject Descriptors

C.3 [SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS]: Real-time and embedded systems

## General Terms

Design, Performance, Algorithms

## Keywords

functional partitioning, communication speed selection, communication/computation trade-offs, embedded multi-processor, low-power design

## ABSTRACT

This paper presents a new technique for global energy optimization through coordinated functional partitioning and speed selection for embedded processors interconnected by a high-speed serial bus. Many such serial interfaces are capable of operating at multiple speeds and can open up a new dimension of trade-offs to complement today's CPU-centric voltage scaling techniques for processors. We propose a multi-dimensional dynamic programming formulation for energy-optimal functional partitioning with CPU/communication speed selection for a class of data-regular applications under performance constraints. We demonstrate the effectiveness of our optimization techniques with an image processing application mapped onto a multi-processor architecture with a multi-speed Ethernet.

## 1. INTRODUCTION

A key trend in embedded systems is towards the use of high-speed serial busses for system-level interconnect. High-speed serial controllers such as Ethernet are now an integral part of many embedded processors. Newer protocols such as FireWire (IEEE

\*This research was sponsored by DARPA grant F33615-00-1-1719 and Printronix Fellowship.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISSS'02, October 2-4, 2002, Kyoto, Japan.

Copyright 2002 ACM 1-58113-576-9/02/0010 ...\$5.00.

1394) and USB are commonly used not only for peripheral devices but also for connecting embedded processors. Many have advocated high-speed, serial packet networks for systems-on-chip for their compelling advantages including modularity, composability, scalability, form factor, and power efficiency.

For power optimization, previous efforts focused on the processor for several reasons. The CPU was the main consumer of power, and it also offered the most options for power management, including voltage scaling. However, recent advances in both processors and communication interfaces are driving a shift in how power should be managed.

### *Low-power CPU, High-power Communication*

CPU-centric power management has given rise to a new generation of processors with dramatically improved power efficiency, and the CPU is now drawing a smaller percentage of the overall system power. The insatiable demand for bandwidth has also resulted in high-speed communication interfaces. Even though their power efficiency (i.e., energy per bit transmitted) has also been improved, communication power now matches or surpasses the CPU, and is thus a larger fraction of the system power. For instance, the Intel XScale processor consumes 1.6W at full speed, while a GigaBit Ethernet interface consumes 6W.

### *Multi-speed Communication Interfaces*

Many communication interfaces today support multiple data rates. However, the scaling effects tend to be the opposite those of voltage scalable CPUs. For CPUs, slower speed generally means lower power and lower energy per instruction; but for communication, faster speed means higher power but often less energy per bit. This is highly dependent on the specific controller. Few research works to date explored communication speed as a key parameter for power optimization.

### *Speed Selection and Functional Partitioning*

Speed selection cannot be performed for just communication or computation in isolation, 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 faster communication speed, even at a higher energy-per-bit, can save energy by creating opportunities for voltage scaling the processors. Greedily saving communication power 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.

## Approach

For a given workload on a networked architecture, our problem statement is to generate a functional partitioning scheme and to select the speeds of communication interfaces and processors, such that the total energy is minimized. In general, this problem is extremely difficult. Fortunately, for a class of systems with pipelined multiple processors under a latency constraint, efficient, exact solutions exist. We construct such a system model and formulate the energy consumed by the processors and communication interfaces with their power/speed scaling factors within their available time budget. In [8], we presented the schedulability conditions and the problem of communication speed selection and sketched solutions by exhaustive search. This paper combines communication speed selection with functional partitioning and presents 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 pipelined multi-processor architecture interconnected by a GigaBit Ethernet.

## 2. RELATED WORK

Previous works have explored communication synthesis and optimization in distributed multi-processor systems. [13] presents communication scheduling to work with rate-monotonic tasks, while [5] assumes the more deterministic time-triggered protocol (TTP). [10] distributes timing constraints on communication among segments through priority assignment on serial busses (such as control-area network) and customization of device drivers. While these assume a bus or a network protocol, LYCOS [7] integrates the ability to select among several communication protocols (with different delays, data sizes, burstiness) into the main partitioning loop. These techniques do not specifically optimize for energy by exploiting the processors' voltage scaling capabilities or the characteristics of the communication interfaces' power consumption.

Related techniques that optimize for power consumption of processors typically assume a fixed communication data rate. [3] uses simulated heating search strategies to find low-power design points for voltage scalable embedded processors. [9] performs battery-aware task post-scheduling for distributed, voltage-scalable processors by moving tasks to smooth the power profile. [12, 11] propose partitioning the computation onto a multi-processor architecture that consumes significantly less power than a single processor. [4] reduces switching activities of both functional units and communication links by partitioning tasks onto a multi-chip architecture; while [6] maximizes the opportunity to shut down idle processors through functional partitioning. All these techniques focus on the computational aspect without exploring the speed/power scalability of the communication interfaces.

Existing techniques cannot be readily combined to explore many timing/power trade-offs between computation and communication. The quadratic voltage scaling properties for CPU's do not generalize to communication interfaces. Even if they do, these techniques have not considered the partitioning of power and timing budgets among computation/communication components across the network. Selecting communication attributes by only considering deadlines without power will lead to unexpected, often incorrect results at the system level.

## 3. SYSTEM MODEL

This section defines a system-level performance/energy model for both computation and communication components in a networked, multiple-processor embedded system. In this paper, such a system consists of  $M$  processing nodes  $N_i, i = 1, 2, \dots, M$  connected

by a shared communication medium. Each *processing node* (or *node* for short) consists of a processor, a local memory, and one or more communication interfaces that send and/or receive data from other nodes.

A *processing job* assigned to a node is modeled in terms of three *tasks*: *RECV*, *PROC*, and *SEND*, which must be executed serially in that order. *RECV* and *SEND* are communication tasks on the interfaces, and *PROC* is a computation task on the processor. For communication tasks *RECV* and *SEND*, workload  $W_r$  and  $W_s$  indicate the number of bits to be received and sent, respectively. For the computation task *PROC*, the workload  $W_p$  is the number of cycles. Let  $T_p, T_r, T_s$  denote the *delays* of tasks *PROC*, *RECV* and *SEND*, respectively. Let  $F_p$  denote the clock frequency of the processor, and  $F_r$  and  $F_s$  the respective data bit rates for receiving and sending. We have

$$T_p = \frac{W_p}{F_p}; \quad T_r = \frac{W_r}{F_r}; \quad T_s = \frac{W_s}{F_s} \quad (1)$$

(1) is reasonable for processors executing data-dominated programs, where the total cycles  $W_p$  can be analyzed and bounded statically.

To model non-ideal aspects of the medium, we introduce the *communication efficiency* terms,  $\rho_r$  and  $\rho_s$ , where  $0 \leq \rho_r, \rho_s \leq 1$ , such that  $T_r = \frac{W_r}{\rho_r F_r}$  and  $T_s = \frac{W_s}{\rho_s F_s}$ . Note that  $\rho_r$  and  $\rho_s$  need not be constants, but may be functions of communication speeds  $F_r, F_s$ . For brevity, our experimental results assume an ideal communication medium ( $\rho_r = \rho_s = 1$ ) without loss of generality.

$D$  is a *deadline* on each processing job, which requires  $T_r + T_p + T_s \leq D$  for the three serialized tasks. If any slack time exists, then we assume we can always slow down task *PROC* by voltage scaling to reduce energy, based on the capability of modern embedded processors. Therefore, we convert the inequality into an equality in the deadline equation. That is,

$$D = T_r + T_p + T_s \quad (2)$$

We assume a processor's voltage-scaling characteristics can be expressed by a scaling function  $Scale_p$  that maps the CPU's frequency to its power level. A communication interface also has scaling functions  $Scale_s$  and  $Scale_r$  for sending and receiving. (2) implies  $Scale_p$  is continuous, while communication interfaces support only a few discrete scaling points. Let  $P_p, P_r$ , and  $P_s$  denote the power for the processor, receiving, and sending, respectively. Then,

$$P_p = Scale_p(F_p); \quad P_r = Scale_r(F_r); \quad P_s = Scale_s(F_s) \quad (3)$$

Let  $P_{ovh}$  denote the power overhead associated with having an additional node into the system. It captures the power of the memory, minimum power of the CPU and communication interface, CPU's power during *RECV* and *SEND* (DMA), and communication interfaces' power during *PROC*.

The *energy consumption of a task* is the power-delay product. Let  $E_p, E_r, E_s$ , and  $E_{ovh}$  denote the energy consumption of tasks *PROC*, *RECV*, *SEND*, and overhead of a node, respectively. Let  $E_{N_i}$  denote the *total energy of node  $N_i$* . Finally, the *total energy of the system* is the sum of energy consumption on each node. To summarize,

$$E_p = P_p T_p; \quad E_r = P_r T_r; \quad E_s = P_s T_s; \quad E_{ovh} = P_{ovh} D \quad (4)$$

$$E_{N_i} = E_p + E_r + E_s + E_{ovh} \quad (5)$$

$$E_{sys} = \sum_{i=1}^M E_{N_i} \quad (6)$$

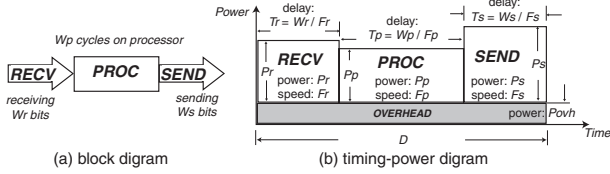


Figure 1: Timing and power properties of a processing node.

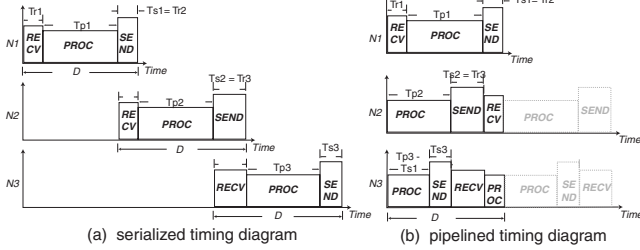


Figure 2: A three-node pipeline.

Fig. 1 shows the timing and power breakdown of the tasks on a node. The gray bar represents the overhead, while the white bars represent tasks *RECV*, *PROC* and *SEND*. The area of a bar represents the energy consumption by the corresponding task or overhead.

This paper considers a special case called an  $M$ -node pipeline. It consists of identical nodes  $N_i, i = 1, 2, \dots, M$  as characterized by  $Scale_p, Scale_r, Scale_s, E_{ovh}$ . Each node  $N_i$  receives  $W_{r_i}$  bits of data from the previous node  $N_{i-1}$  (except  $N_1$ ), processes the data in  $W_{p_i}$  cycles, and sends the  $W_{s_i}$ -bit result to the next node  $N_{i+1}$  (except  $N_M$ ). Each  $SEND_i \rightarrow RECV_{i+1}$  communication pair sends and receives same amount of data at the same communication speed, with the same communication delay, and we assume they start and finish at the same time. That is,  $W_{s_i} = W_{r_{i+1}}, F_{s_i} = F_{r_{i+1}}, T_{s_i} = T_{r_{i+1}}$ . All nodes have the same deadline  $D$ , and each node acts as a pipeline stage with delay  $D$ . Fig. 2 shows an example of a three-node pipeline. For brevity, the overhead is not shown. Fig. 2(b) shows the pipelined timing diagram by folding the tasks in Fig. 2(a) into a common interval with duration  $D$ , which is the delay of each pipeline stage. [8] presented the schedulability conditions for an  $M$ -node pipeline based on collision and utilization of the shared communication medium.

An  $M$ -node pipeline can be partitioned and mapped onto an  $M'$ -node pipeline ( $M' \leq M$ ) by merging adjacent nodes  $N_i, N_{i+1}, \dots, N_j$  ( $j \geq i$ ) into a new node  $N'_k$ . The new node  $N'_k$  combines all computation workload, receives  $W_{r_i}$  bits of data, and sends  $W_{s_j}$  bits of data. Communication within a node become local data accesses. That is,  $W'_{p_k} = \sum_{l=i}^j W_{p_l}$ , and  $W'_{r_k} = W_{r_i}, W'_{s_k} = W_{s_j}$ . The new  $M'$ -node pipeline is called a *partitioning* of the initial  $M$ -node pipeline.

## 4. MOTIVATING EXAMPLE

We use an automatic target recognition (ATR) algorithm (Fig. 3) as our motivating example. Originally it is a serial algorithm. We reconstructed a parallel version and mapped it onto pipelined multiple processors. Pipelining allows each processor to run at a much slower speed with a lower voltage level to reduce overall computation energy, while parallelism compensates for the performance. Of course, having extra processors costs energy overhead for inter-processor communication, memory, etc.

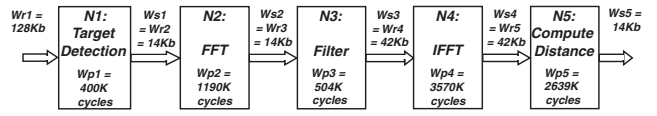


Figure 3: Stages of the ATR algorithm.

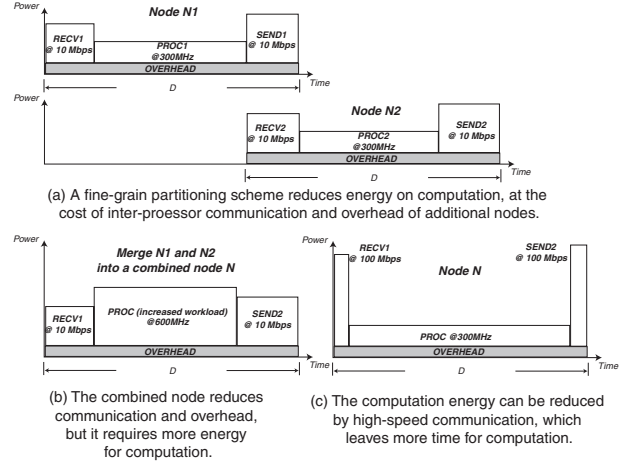


Figure 4: The impact of different partitioning schemes and communication speed settings.

### Task to Node Mapping

Given the decomposition into five stages of the ATR algorithm, several partitioning schemes are possible for mapping them onto a number of pipelined nodes. Fig. 4 shows an example by considering how they map the first two stages onto (a) two nodes and (b) one node. In Fig. 4(a), mapping onto two nodes  $N1$  and  $N2$  enables both processors to operate at a reduced speed (300MHz) for computation. The two nodes together consume lower computation energy than one node at a faster speed but must pay the price of communication energy for  $SEND1 \rightarrow RECV2$ . In Fig. 4(b), even though merging the two stages onto one node eliminates the  $SEND1 \rightarrow RECV2$  communication, the CPU must execute the combined computation workload at a faster clock rate (600MHz), a less energy-efficient level.

Zooming out, many partitioning schemes are possible, even when limited to a pipelined organization. For example, one partitioning  $[N1, N2][N3, N4, N5]$  may be optimal for nodes  $N1$  and  $N2$ ; but it will preclude another solution  $[N1], [N2, N3], [N4, N5]$  that may lead to less energy for the whole system.

### Speed Selection for CPU and Communication

The selection of communication speed is an equally critical issue. For example, a 10/100/1000 Base-T Ethernet interface can consume more power than a CPU at high (100/1000Mbps) speeds, but less power at the slower, 10Mbps data rate. In Fig. 4(b), the processor must operate at a high clock rate due to the low-speed communication at 10Mbps. Because of the deadline  $D$ , communication and computation compete for this budget. Low-speed communication leaves less time for computation, thereby forcing the processor to run faster to meet the deadline. Conversely, high-speed communication could free up more time budget for computation, as shown in Fig. 4(c), where the CPU's clock rate is dropped to 300MHz. Although extra energy could be allocated to high-speed communication, if the energy saving on the CPU could compensate for this cost, then (c) would be more energy-efficient than (b).

The communication-computation interaction becomes more intricate in a multi-processor environment. Any data dependency between different nodes must involve their communication interfaces. The communication speed of a sender will not only determine the receiver's communication speed but also influence the choice of the receiver's computation speed. The communication speed on the first node of the pipeline will have a chain effect on all other nodes in the system. A locally optimal speed for the first node will not necessarily lead to a globally optimal solution.

### Combining Partitioning and Speed Selection

Given a fixed partitioning scheme, the designers can always find the corresponding optimal speed setting that minimizes energy for that scheme. However, energy-optimal speed selection for a partitioning is not necessarily optimal over all partitionings. Instead, partitioning and speed selection are mutually enabling. In this paper, we take a multi-dimensional optimization approach that considers performance requirements, schedulability, load balancing, communication-computation trade-offs, and multi-processor overhead in a system-level context.

## 5. PROBLEM FORMULATION

Given an  $M$ -node pipeline, choices of partitioning and communication speed settings will lead to different levels of energy consumption at the system level. This section formulates three energy minimization problems: by partitioning, by communication speed selection, and by both. In the first two problems, the optimal solution can be obtained by dynamic programming, and the combined optimization problem can be solved by multi-dimensional dynamic programming.

### Problem 1 (Optimal Partitioning)

Given (a)  $M$  pipelined nodes  $N_i$  with workload  $W_{p_i}, W_{r_i}, W_{s_i}, i = 1, 2, \dots, M$ , (b) a deadline  $D$  for all nodes, and (c) the constraint that the speed settings of all communication instance must match:  $F_{r_i}, F_{s_i} = F_{r_{i+1}}, F_{s_{i+1}}$ , for  $i = 1, 2, \dots, M - 1$ , find a partitioning scheme that minimizes energy  $E_{\text{sys}}$ .

To avoid exhaustive enumeration in the  $O(2^{M-1})$  solution space, we construct a series of optimal solutions to sub-problems by mapping the original  $M$  nodes one by one onto new sub-partitionings. We compute the optimal cost function in terms of the minimum energy consumption over the sub-partitionings. Upon mapping each node, the new optimal sub-solution can be computed from past optimal sub-solutions. Therefore, a dynamic programming approach is applicable.

For dynamic programming, we use an *energy matrix*  $E$  to store the cost function. Each entry  $E[i, j]$  indicates the minimum energy of a sub-problem that maps the first  $j$  original nodes  $N_1, N_2, \dots, N_j$  onto a new sub-partitioning with  $i$  nodes  $N'_1, N'_2, \dots, N'_i$ . Matrix  $E$  is initialized to  $\infty$ .

$$E[i, j] = \begin{cases} 0 & \text{for } i = j = 0 \\ \min_{i-1 \leq l \leq j-1} \begin{bmatrix} E[i-1, l] + \\ E_{N'_i} \end{bmatrix} & \text{for } \begin{matrix} 1 \leq i \leq \\ j \leq M \end{matrix} \end{cases} \quad (7)$$

(7) indicates that the optimal  $i$ -node sub-partitioning that maps the first  $j$  original nodes must be a combination of the followings: (a) a sub-partitioning that maps the first  $l$  original nodes  $N_1, N_2, \dots, N_l$  to  $i-1$  new nodes, and (b) the  $i^{\text{th}}$  new node  $N'_i$  that combines the original nodes  $N_{l+1}, \dots, N_j$ . The sub-partitioning (a) must be optimal with minimum energy  $E[i-1, l]$ . (b) only has one

node  $N'_i$ . Its energy is denoted as  $E_{N'_i}$ . Since  $E[i, j]$  is the optimal energy for the sub-problem, it must be the minimum value of (7) among all possible choices of  $l$ . The dynamic programming algorithm can iterate (7) from  $i = j = 0$  until  $i = j = M$ . Each optimal sub-solution  $E[i, j]$  can be derived from previously computed  $E[i-1, l]$ . Finally, the minimum energy is  $\min(E[i, M]), \forall i = 1, 2, \dots, M$ . We omit the algorithm for brevity. Its time complexity is  $O(M^3)$ .

**Problem 2 (Optimal Communication Speed Selection)** Given (a) a fixed partitioning scheme with  $M$  pipelined nodes  $N_i$  with workload  $W_{p_i}, W_{r_i}, W_{s_i}, i = 1, 2, \dots, M$ , (b) a deadline  $D$  for all nodes, and (c) the available choices for communication speed settings  $F_{c_k}, k = 1, 2, \dots, C$ , find all processor speeds  $F_{p_i}$  and communication speeds  $F_{r_i}, F_{s_i}$  that minimize energy  $E_{\text{sys}}$ .

We also perform dynamic programming as opposed to exhaustive search in  $O(C^{M+1})$  solution space. During step  $i$  when processing node  $N_i$ , we only select communication speeds  $F_{r_i}, F_{s_i}$  of  $N_i$ , because they determine  $F_{p_i}$ , and the previous speed settings of the sub-problems have already been selected to optimal. For each choice of  $F_{r_i}, F_{s_i}$ , we compute the energy of node  $N_i$ , plus the optimal energy of a sub-problem computed by step  $i-1$  with  $F_{s_{i-1}} = F_{r_i}$  to find the optimal energy of the new sub-problem in step  $i$ .

Each element  $E[i, k]$  in the *energy matrix*  $E$  indicates the minimum energy of a sub-problem. It has  $i$  nodes  $N_1, N_2, \dots, N_i$  with the last node  $N_i$ 's sending speed selected to be the  $k^{\text{th}}$  speed choice  $F_{c_k}$ .  $E$  is initialized to  $\infty$ .

$$E[i, k] = \begin{cases} 0 & \text{for } i = 0, \\ & 1 \leq k \leq C \\ \min_{1 \leq m \leq C} \begin{bmatrix} E[i-1, m] + \\ E_{N_i}(F_r = F_{c_m}, F_s = F_{c_k}) \end{bmatrix} & \text{for } \begin{matrix} 1 \leq i \leq M, \\ 1 \leq k \leq C \end{matrix} \end{cases} \quad (8)$$

(8) indicates that the optimal speed setting for the sub-problem up to node  $N_i$  whose sending speed  $F_{s_i} = F_{c_k}$  is determined by: (a) a previous optimal sub-solution where node  $N_{i-1}$ 's sending speed  $F_{s_{i-1}} = F_{c_m}$ , plus (b) node  $N_i$  whose receiving speed  $F_{r_i} = F_{c_m}$ , sending speed  $F_{s_i} = F_{c_k}$ . (a) includes  $i-1$  nodes  $N_1, N_2, \dots, N_{i-1}$  and communicates with (b) at speed  $F_{c_m}$ . The optimal energy of sub-problem (a) is  $E[i-1, m]$ . (b) has only one node  $N_i$  that receives data from (a) through speed  $F_{c_m}$ ; and its sending speed is  $F_{c_k}$ . Its energy is denoted as  $E_{N_i}(F_r = F_{c_m}, F_s = F_{c_k})$ . Since  $E[i, k]$  is optimal, it must be the minimum value among all possible speed settings  $F_{c_m}$  in (8). The algorithm is omitted for brevity. It iterates (8) until  $i = M, k = C$ . Each  $E[i, k]$  can be derived from previously computed  $E[i-1, m]$ . The global minimum energy is  $\min(E[M, k]), \forall k = 1, 2, \dots, C$ . The time complexity of the algorithm is  $O(MC^2)$ .

**Problem 3 (Optimal Partitioning and Speed Selection)** Given (a)  $M$  pipelined nodes  $N_i$  with workload  $W_{p_i}, W_{r_i}, W_{s_i}, i = 1, 2, \dots, M$ , (b) a deadline  $D$  for all nodes, and (c) the available choices for communication speed settings  $F_{c_k}, k = 1, 2, \dots, C$ , find a partitioning scheme and corresponding communication speed settings that minimize energy  $E_{\text{sys}}$ .

Due to the inter-dependency between speed setting and partitioning, the optimal solution cannot be achieved by solving two previous problems individually. Exhaustively enumerating over one

```

partitioning-speedselection( $W_r[1 : M], W_s[1 : M], W_p[1 : M],$ 
 $F_c[1 : C], Scale_r, Scale_s, Scale_p, D, P_{ovh}$ )
for  $i := 0$  to  $M$  do
  for  $j := i$  to  $M$  do
    for  $k := 1$  to  $C$  do
       $E[i, j, k] := U[i, j, k] := P[i, j, k] := S[i, j, k] := \infty$ 
for  $k := 1$  to  $C$  do
   $E[0, 0, k] := 0$ 
   $U[0, 0, k] := W_r[1]/F_c[k]/D$ 
for  $i := 1$  to  $M$  do
  for  $j := i$  to  $M$  do
    for  $k := 1$  to  $C$  do
      for  $l := i - 1$  to  $j - 1$  do
        for  $m := 1$  to  $C$  do
           $e := E[i - 1, l, m] + E_{N'_i}(F_r = F_c[m], F_s = F_c[k])$ 
           $u := U[i - 1, l, m] + W_s[j]/F_c[k]/D$ 
          if  $u \leq 1$  and  $e < E[i, j, k]$  then
             $E[i, j, k] := e$ 
             $U[i, j, k] := u$ 
             $P[i, j, k] := l$ 
             $S[i, j, k] := m$ 
 $E_{opt}, P_{opt}, S_{opt} :=$  retrieve from matrices  $E, U, P, S$ 
return  $E_{opt}, P_{opt}, S_{opt}$ 

```

Figure 5: Combined partitioning with speed selection.

dimension and dynamic programming over the other is quite expensive with the time complexity as either  $O(2^{M-1}MC^2)$  or  $O(C^{M+1}M^3)$ . We propose a multi-dimensional dynamic programming algorithm given the fact that the previous two problems can be solved by dynamic programming independently. Based on the previous two dynamic programming approaches, the *energy matrix*  $E$  for the combined problem is defined as follows: each element  $E[i, j, k]$  stores the minimum energy of a sub-problem that maps the first  $j$  original nodes  $N_1, N_2, \dots, N_j$  onto a new  $i$ -node sub-partitioning, whose last node  $N'_i$  has sending speed  $F_{s'_i} = F_{c_k}$ .

$$E[i, j, k] = \begin{cases} 0 & \text{for } i = j = 0, \\ & 1 \leq k \leq C \\ \min_{\substack{i-1 \leq l \leq j-1, \\ 1 \leq m \leq C}} \begin{bmatrix} E[i-1, l, m] + \\ E_{N'_i}(F_r = F_{c_m}, \\ F_s = F_{c_k}) \end{bmatrix} & \text{for } 1 \leq i \\ & \leq j \leq M, \\ & 1 \leq k \leq C \end{cases} \quad (9)$$

The optimal energy  $E[i, j, k]$  is derived from: (a)  $E[i-1, l, m]$  of a previous optimal sub-solution, which maps  $l$  original nodes  $N_1, \dots, N_l$  onto  $i-1$  new nodes  $N'_1, \dots, N'_{i-1}$  with the last node  $N'_{i-1}$ 's sending speed selected to be  $F_{c_m}$ , plus (b) the new node  $N'_i$  that combines original nodes  $N_{l+1}, \dots, N_j$  with receiving speed  $F_{c_m}$  and sending speed  $F_{c_k}$ . The sub-solution (a) has the optimal energy  $E[i-1, l, m]$ . Note that (b) has only one node  $N'_i$ , and its energy is denoted as  $E_{N'_i}(F_r = F_{c_m}, F_s = F_{c_k})$ .  $E[i, j, k]$  must be derived from all possible pairs of  $(l, m)$  to achieve the minimum value of (9).

The algorithm is shown in Fig. 5. It combines two previous algorithms by two-dimensional dynamic programming. There are three additional matrices. The *utilization matrix*  $U$  tracks the schedulability condition [8] and guards each optimal sub-solution to guarantee its schedulability. The *partitioning matrix*  $P$  and *speed matrix*  $S$  are used to record the intermediate solutions and for retrieving the optimal partitioning  $P_{opt}$  and optimal speed setting  $S_{opt}$  when the algorithm terminates. The global minimum energy is

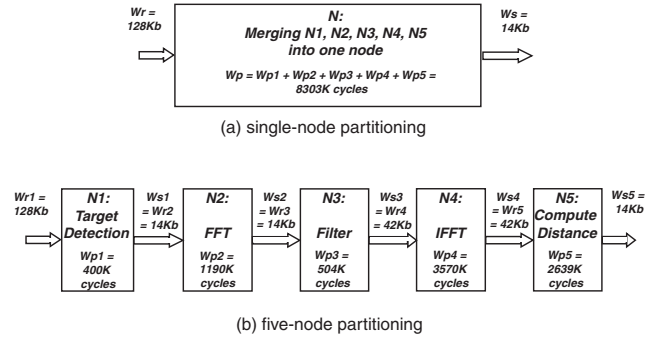


Figure 6: Two fixed partitioning schemes of ATR.

$\min(E[i, M, k]), \forall i = 1, 2, \dots, M, \forall k = 1, 2, \dots, C$ . The time complexity of the algorithm is  $O(M^3C^2)$ .

## 6. EXPERIMENTAL RESULTS

To evaluate our energy optimization techniques, we experiment with mapping the ATR algorithm onto two fixed partitioning schemes: (a) a single-node that combines all blocks, and (b) a five-node pipeline that maps each block onto an individual node (Fig. 6). The input data size is 128K bits, and the output is 14K bits per frame. In scheme (a), the single node combines all the workload of five nodes in (b); and it eliminates all internal communication instances between nodes in (b). (a) and (b) are two extremes representing serial vs. parallel schemes. For both (a) and (b) we apply optimal speed selection. We also find the optimal partitioning with speed selection as (c) and compare its energy consumption per image frame with (a) and (b) under two types of performance requirements: (1) high performance,  $D = 10ms$ , (2) moderate performance,  $D = 15ms$ .

Each node consists of an Intel XScale processor [2] whose power vs. performance level ranges from 50mW@150MHz to 1.6W@1GHz (Fig. 7), and an LXT-1000 Ethernet interface [1] with power levels of 0.8W@10Mbps, 1.5W@100Mbps, and 6W@1000Mbps (Fig. 8). We assume each node has a constant power draw  $P_{ovh} = 100mW$ .

The results are presented in Fig. 9. In all cases, 1000Mbps is always the optimal speed setting for communication. The low-power, 10Mbps communication speed results in the highest energy. This is because it leaves so little time for computation such that the processors must run faster with more energy to meet the deadline, and it has the highest energy-per-bit rating. The low-speed communication also tends to violate the schedulability conditions [8]. Given properties of this particular Ethernet interface, 1000Mbps communication will always lead to the lowest energy consumption since it requires the least amount of energy per bit and leaves the maximum amount of time budget for reducing CPU energy. However, in cases where the energy-per-bit rating does not decrease monotonically with the communication speed, the optimal speed setting may involve some combinations of low-speed and high-speed settings between different nodes. For example, the node  $N_i$  may communicate with  $N_{i-1}$  at 1000Mbps and with  $N_{i+1}$  at 100Mbps.

Fig. 9(1) shows the energy consumption of all three partitioning schemes under a tight performance constraint. The single-node (a) is heavily loaded with computation. Therefore it is desirable to reduce CPU energy by pipelining. As a result, the five-node pipeline (b) is more energy-efficient at the cost of additional communication and overhead. However, the optimal partitioning is (c) with three nodes:  $[N1, N2], [N3, N4], [N5]$ . It consumes more CPU energy than (b), but overall it is optimal with less energy on communication and overhead.

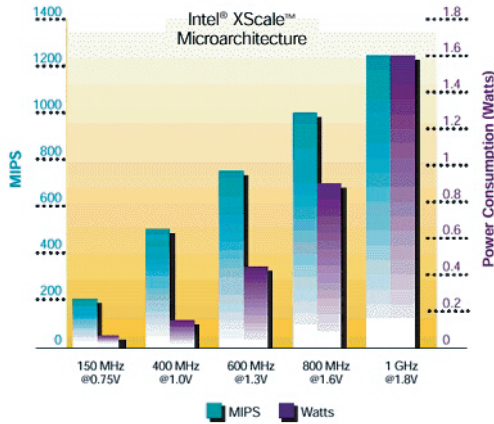


Figure 7: Power vs. performance of the XScale processor.

Mode	Power consumption
10M bps	800 mW
100M bps	1.5W
1000M bps	6W

Figure 8: Power modes of the Ethernet interface.

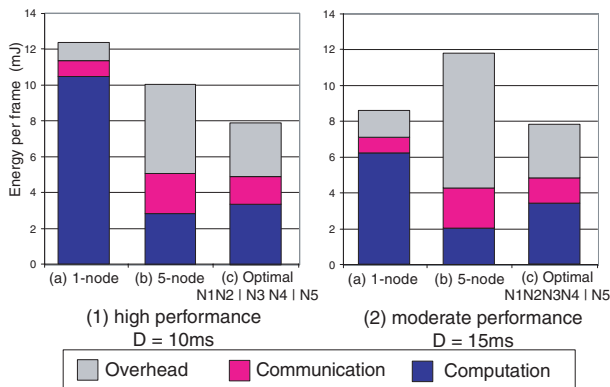


Figure 9: Energy consumption of three partitioning schemes.

In case of the moderate performance constraint (Fig. 9(2)), (a) is still dominated by computation but it is not heavily loaded due to the relaxed deadline. The reduction of CPU energy by (b) cannot compensate for the added overhead of new nodes and communication. Therefore (a) is better than (b) and pipelining seems inefficient. However, the optimal partitioning (c) is still a pipelined solution. It combines  $N1, N2, N3, N4$  into one node and maps  $N5$  to another node. (c) achieves minimum energy by appropriately balancing computation, communication with pipelining overhead. If the performance constraint is further relaxed, the serial solution (a) will become optimal.

## 7. CONCLUSION

We present an energy optimization technique for networked embedded processors and emerging system-on-chip architectures with high-speed on-chip networks. We exploit with the multi-speed feature of modern high-speed communication interfaces as an effective way to complement and enhance today's CPU-centric power opti-

mization approaches. In such systems, communication and computation compete over opportunities for operating at the most energy-efficient points. It is critical to not only balance the load among processors by functional partitioning, but also to balance the speeds between communication and computation on each node and across the whole system. Our multi-dimensional dynamic programming formulation is exact and produces the energy-optimal solution as defined by a partitioning scheme and the speed selections for all computation and communication tasks. We expect this technique to be applicable to a large class of data dominated systems that can be structured in a pipelined organization.

## 8. REFERENCES

- [1] INTEL ethernet PHYs/transceivers. [http://developer.intel.com/design/network/products/ethernet/linecard\\_ept.htm](http://developer.intel.com/design/network/products/ethernet/linecard_ept.htm).
- [2] INTEL XScale microarchitecture. <http://developer.intel.com/design/intelxscale/>.
- [3] N. K. Bambha, S. S. Bhattacharyya, J. Teich, and E. Zitzler. Hybrid global/local search strategies for dynamic voltage scaling in embedded multiprocessors. In *Proc. International Symposium on Hardware/Software Codesign*, pages 243–248, 2001.
- [4] R. Cherabuddi, M. Bayoumi, and H. Krishnamurthy. A low power based system partitioning and binding technique for multi-chip module architectures. In *Proc. Great Lakes Symposium on VLSI*, pages 156–162, 1997.
- [5] P. Eles, A. Doboli, P. Pop, and Z. Peng. Scheduling with bus access optimization for distributed embedded systems. *IEEE Transactions on VLSI Systems*, 8(5):472–491, 2000.
- [6] E. Huwang, F. Vahid, and Y.-C. Hsu. FSMDF functional partitioning for low power. In *Proc. Design, Automation and Test in Europe*, pages 22–28, 1999.
- [7] P. V. Knudsen and J. Madsen. Integrating communication protocol selection with hardware/software codesign. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(8):1077–1095, August 1999.
- [8] J. Liu, P. H. Chou, and N. Bagherzadeh. Communication speed selection for embedded systems with networked voltage-scalable processors. In *Proc. International Symposium on Hardware/Software Codesign*, pages 169–174, April 2002.
- [9] J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *Proc. Design Automation Conference*, pages 444–449, June 2001.
- [10] R. Ortega and G. Borriello. Communication synthesis for distributed embedded systems. In *Proc. International Conference on Computer-Aided Design*, pages 437–444, 1998.
- [11] 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.
- [12] 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.
- [13] W. Wolf. An architectural co-synthesis algorithm for distributed embedded computing systems. *IEEE Transactions on VLSI Systems*, pages 218–229, June 1997.