

Communication Speed Selection for Embedded Systems with 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

ABSTRACT

High-speed serial network interfaces are gaining wide use in connecting multiple processors and peripherals in modern embedded systems, thanks to their size advantage and power efficiency. Many such interfaces also support multiple data rates, and this ability is opening a new dimension in the power/performance trade-offs between communication and computation on voltage-scalable embedded processors. To minimize energy consumption in these networked architectures, designers must not only perform functional partitioning but also carefully balance the speeds between communication and computation, which compete for time and energy. Minimizing communication power without considering computation may actually lead to higher energy consumption at the system level due to elongated on-time as well as lost opportunities for dynamic voltage scaling on the processors. We propose a speed selection methodology for globally optimizing the energy consumption in embedded networked architectures. We formulate a multi-dimensional optimization problem by modeling communication dependencies between processors and their timing budgets. This enables engineers to systematically solve the problem of optimal speed selection for global energy reduction. We demonstrate the effectiveness of our speed selection approach with an image processing application mapped onto a multi-processor architecture with a multi-speed Ethernet.

1. INTRODUCTION

Networked embedded processors are fast becoming the mainstream in the architecture of embedded systems. Initially used in automobiles and other control-oriented systems, they are now found in everything from consumer electronics to peripheral devices attached to workstations. A key trend is towards the use of high-speed serial busses for system-level interconnect, including FireWire, USB, and Ethernet. These “busses” are actually more like networks, and the advent of systems-on-chip (SoC) is also giving rise to a new class of on-chip network protocols that offer compelling advantages as the interconnect of choice for complex

*This research was sponsored by DARPA under contract F33615-00-1-1719

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, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES'02, May 6-8, 2002, Estes Park, Colorado, USA.
Copyright 2002 ACM 1-58113-542-4/02/0005...\$5.00.

on-chip modules. In addition to better modularity, composability, scalability, and form factor, high-speed serial network protocols are also power efficient.

Power is an important objective in the optimization of many embedded systems. Previous efforts in power optimization focused on the processor, because not only was the CPU the main consumer of power, but it also offered the most flexible options for power management, including voltage and frequency scaling. By running the processor at the minimum speed without violating any deadline, its voltage can be reduced to a more energy-efficient, lower-power level that actually consumes less energy per instruction.

Processor power has been reduced dramatically and is now a smaller percentage of the overall system power, while communication interfaces are now consuming an equal if not larger share of the system power. For example, an Intel XScale processor consumes 1.6W at full speed, while a GigaBit Ethernet interface can consume as much as 6W. Many of today's communication interfaces support multiple data rates at different power levels, in ways analogous to voltage scaling on processors. However, few research works to date explore their use in power management in conjunction with the processors.

Previous co-design approaches to optimization of such systems focused on functional partitioning among the processors to minimize communication, and hence communication power. However, existing partitioning techniques lack the ability to explore the power consequences of processor and communication speeds. These resources can compose synergistically or competitively. That is, choosing a power efficient communication speed (respectively, processor speed) in one part of the network may either increase or decrease additional power-efficient opportunities for other resources on the network.

We consider a class of networked embedded systems where the processors are networked to execute task in a pipelined fashion and must satisfy an overall latency constraint. The processors cannot all be run at the slowest, most power-efficient speeds; instead, 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 more opportunities for subsystem shutdown or by freeing up more time for voltage scaling the processors. Conversely, a low-power, low energy-per-bit communication speed may actually result in higher overall energy due to increased time in keeping the interface on.

Our objective is to minimize total energy for a given workload on a networked architecture, where both the communication speeds and processor speeds are selectable. Our proposed approach is to formulate the energy consumed by the processors and communication interfaces with their power/speed scaling factors within their available time budget. This enables us to solve the optimal speeds systematically. We envision that this technique be integrated into

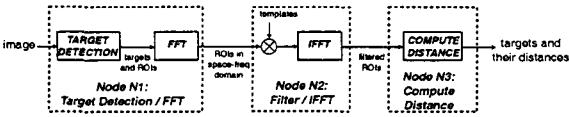


Figure 1: Block diagram of the ATR algorithm.

the partitioning loop of an existing co-design framework. In this paper, we demonstrate the effectiveness of our approach with an image processing algorithm mapped onto a multi-processor architecture interconnected by a wired GigaBit Ethernet.

2. RELATED WORK

Previous works in co-design have explored distributed multi-processor systems, communication protocol and speed selection, and power management techniques. [11] presents communication scheduling to work with rate-monotonic tasks, while [5] assumes the more deterministic time-triggered protocol (TTP). [8] 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 [6] integrates the ability to select among several communication protocols (with different delays, data sizes, burstiness) into the main partitioning loop. However, these and many other works do not specifically optimize for energy minimization by exploiting the processors' voltage scaling capabilities.

Related techniques that optimize for power apply voltage scaling to the processors while assuming a fixed communication data rate. [3] uses simulated heating search strategies to find low-power design points for voltage scalable embedded processors. [7] performs battery-aware task post-scheduling for distributed, voltage-scalable processors by moving tasks to smooth the power profile. [10] proposes a multi-processor architecture that consumes significantly less power than a single processor on the same task by partitioning an image processing task onto four slower processors whose workload is only 1/4. However, 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 budgets and timing among components across the network. Selecting communication attributes by considering deadlines without considering power will lead to unexpected, often incorrect results at the system level.

3. MOTIVATING EXAMPLE

We use an automatic target recognition (ATR) algorithm [9] as our motivating example. Its block diagram is shown in Fig. 1. The algorithm first detects a few targets on the input image. For each target, a region of interest is extracted and filtered by several pre-defined templates. Finally, the distance of each target is computed. A deadline is imposed on processing each frame.

The original ATR algorithm is a serial algorithm. We reconstructed a parallel version and mapped it onto multiple processors in a pipelined organization. In this paper, we partition the algorithm onto three processors, shown as blocks with dashed borders in Fig. 1. In general, parallelizing an algorithm has the effect of increasing performance, reducing power, or both. Even though more processors are involved, each processor now has reduced workload

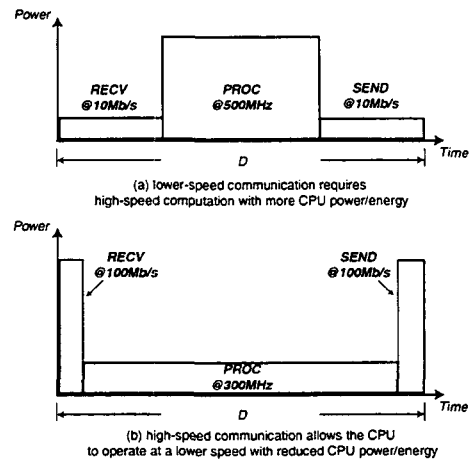


Figure 2: Choices for communication and computation speeds.

and can afford to run at a much slower speed and voltage level, where it consumes significantly less energy per instruction. Meanwhile, parallelism compensates for the reduced performance on individual processors, although inter-processor communication consumes both time and energy.

Fig. 2 gives such an example. We assume that each process is modeled by a computation task *PROC* and two communication tasks *RECV* and *SEND*. *RECV* receives data from either external input or from another processor. The data is processed by *PROC*, and the results are transmitted by communication task *SEND* to the next stage or output. Clearly *RECV*, *PROC*, and *SEND* must be fully serialized for a given stage. In addition, *SEND* must finish before its deadline *D*. *PROC* is mapped onto a voltage-scalable processor, while the corresponding *SEND* and *RECV* tasks are mapped onto a communication interface with multiple data rates and different power levels. In this case, a node consists of an Intel XScale and a 10/100/1000Base-T Ethernet interface.

The Ethernet interface is a prime target for power management, because it consumes 4 times the power as the CPU at peak speed. If the communication interface were voltage scalable in the same way as the CPU, then the designer would choose the low-power, slow data rate for reducing power and energy at the system level. Fig. 2(a) shows this design point, where the data rate is 10 Mbps. However, this turns out not to be energy efficient. Because of the deadline *D*, communication (*RECV*, *SEND*) and computation (*PROC*) compete for this time budget. By slowing down communication, it leaves less time budget for computation, thereby forcing the processor to operate at 500MHz or faster in order to meet the deadline. Conversely, if a low-power CPU speed were selected first, then it would cut the time budget for the communication, forcing a higher-power, faster data rate. This is an example where communication must not be considered as fixed timing and power overhead.

Fig. 2(b) shows a solution that uses high-speed communication (100 Mbps) to leave more time for computation, such that the processor can operate at 300MHz with reduced power and energy. Although extra energy could be allocated to communication, if the energy saving on the processor could compensate for this cost, then (b) would be a more energy-efficient design point. There is no simple correlation between the communication speed and the power or energy at the system level. System-level energy consumption is de-

terminated by not only the processors and communication interfaces, but also the hardware configuration and the dependencies imposed by the application. Therefore, an energy-efficient design must consider all these aspects as an entire system.

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. For example, in the system-pipelined version of the ATR algorithm, the communication speed on the first node will have a chain effect on all other nodes in the system. Even if this speed setting were locally optimal for the first node, it would still lead to an inferior system design.

4. PROBLEM FORMULATION

This section formulates the problems of minimizing energy consumption in a networked multi-processor embedded system. Section 4.1 defines such a system-level performance/energy model for both computation and communication components. Section 4.2 formulates the energy minimization problems and discusses their solutions. Section 4.3 outlines the scheduling technique for avoiding data collision on the shared communication medium and presents the schedulability conditions for such a multi-processor system.

4.1 Basic definitions

In the scope of this paper, a *multi-processor system* is a network of M processing nodes $N_i, i = 1, 2, \dots, M$ connected by a shared communication medium. Each *processing node* consists of the following components: a processor, a local memory, and one or more communication interfaces that send and/or receive data from other processing nodes. We assume on each processing node, only the processor and the communication interfaces are power-manageable. The memory draws a fixed amount of power and is not considered in the energy optimization problem. The power consumption by the communication medium is interpreted to be the total power consumed by all active communication interfaces. Therefore, we focus only on the processors and communication interfaces.

We assume the processors are voltage-scalable, and their voltage-scaling characteristics can be expressed by a scaling function $Scale_p$ that maps the CPU frequency to its power level. We also assume that each communication interface has a scaling function that characterizes the power levels at different communication speeds (data rates). Moreover, we define two separate scaling functions for sending and receiving. They are

$$P_p = Scale_p(F_p); \quad P_r = Scale_r(F_r); \quad P_s = Scale_s(F_s) \quad (1)$$

where P_p and F_p denote the power level and the clock frequency of the processor, P_r and F_r denote the power and the bit rate for receiving, and P_s and F_s denote those for sending, respectively. Most communication interfaces can support only a few discrete combinations of power vs. speed, e.g., 10/100/1000 Mbps Ethernet.

A *processing job* assigned to a processing node has three tasks: *RECV*, *PROC*, and *SEND*, which must be executed serially in that order. *RECV* and *SEND* are communication tasks that receive and send data on the communication interfaces, respectively, and *PROC* is a computation task that processes the received data on the processor.

The *workload* for each task is defined as follows. 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. They can all be calculated as the workload divided by the speed of the component on which the task is executed. That is,

$$T_p = \frac{W_p}{F_p}; \quad T_r = \frac{W_r}{F_r}; \quad T_s = \frac{W_s}{F_s} \quad (2)$$

(2) is reasonable for processors executing data-dominated programs, where the total cycles W_p can be analyzed and bounded statically. However, it does not hold true in general if the effective data rate can be reduced by collisions and errors on the shared communication medium. Section 4.3 proposes a scheduling technique for avoiding collision on the shared medium. To model the non-ideal aspect of the medium, we introduce the *communication efficiency* terms, ρ_r and ρ_s , $0 \leq \rho_r, \rho_s \leq 1$, such that $T_r = \frac{W_r}{\rho_r F_r}$ and $T_s = \frac{W_s}{\rho_s F_s}$.

Note that ρ_r and ρ_s need not be constants, but may be functions of communication speeds F_r, F_s . Different communication speeds may incur different error rates. For brevity, our experimental results assume an ideal communication medium ($\rho_r = \rho_s = 1$) without loss of generality. A more practical communication model can be directly applied, since ρ_r and ρ_s can be very well bounded for a collision-free medium.

There is a *deadline* D on each processing job. Since each job includes three serialized tasks, $T_r + T_p + T_s \leq D$, if there is any slack time available, it can always be used to slow down task *PROC* by voltage scaling to reduce energy. Therefore, we assume the last task *SEND* finishes just before the deadline. That is,

$$D = T_r + T_p + T_s \quad (3)$$

The *energy consumption of a task* is the power-delay product.

$$E_p = P_p T_p; \quad E_r = P_r T_r; \quad E_s = P_s T_s \quad (4)$$

where E_p, E_r, E_s are the energy consumption for tasks *PROC*, *RECV*, and *SEND*, respectively. For one node N_i with tasks *PROC* _{i} , *RECV* _{i} , and *SEND* _{i} , the *total energy of node i* is

$$E_i = E_{p_i} + E_{r_i} + E_{s_i} \quad (5)$$

In (5) we ignore the power level of the processor during the execution of communication tasks *RECV* and *SEND*. The processor can remain in a low power mode during DMA cycles. Similarly, the communication interface can be set to a low power mode during *PROC*. We leave out these terms for brevity without loss of generality, and they can always be appended to (5) for a more realistic model. Finally, the *total energy of the system* is the sum of energy consumption on each node:

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

4.2 Problem statements

We start our problem statements from a single node (Fig. 3); then we extend the problem to the entire system with multiple nodes.

Problem 1 (Single Node) Given a processing node N (characterized by $Scale_r, Scale_p, Scale_s$), the workload W_r, W_p, W_s , and deadline D , find the communication speeds F_r, F_s and the processor speed F_p that minimize the energy consumption E for this node.

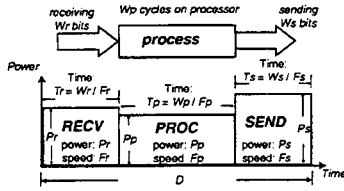


Figure 3: Single processing node.

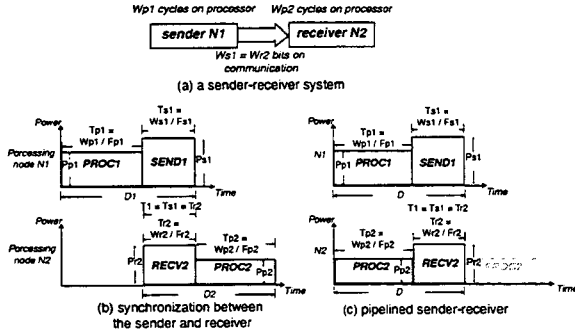


Figure 4: A two-node sender-receiver system.

Solution By (1)–(5), E can be expressed in terms of T_r, T_s ,

$$E = Scale_r \left(\frac{W_r}{T_r} \right) T_r + Scale_s \left(\frac{W_s}{T_s} \right) T_s + Scale_p \left(\frac{W_p}{D - T_r - T_s} \right) (D - T_r - T_s)$$

The problem can be solved mathematically by examining $\frac{d^2 E}{dT_r dT_s}$, if all scaling functions are continuous on second-order derivatives. The speeds F_r, F_s , and F_p can be derived from T_r, T_s .

If $Scale_r$ and $Scale_s$ are discrete functions, then all combinations of $(P_r, F_r) \times (P_s, F_s)$ must be enumerated by

$$E = P_r \frac{W_r}{F_r} + Scale_p \left(\frac{W_p}{D - \frac{W_r}{F_r} - \frac{W_s}{F_s}} \right) \left(D - \frac{W_r}{F_r} - \frac{W_s}{F_s} \right) + P_s \frac{W_s}{F_s}$$

This two-dimensional optimization problem can be reduced to one-dimensional if the node does not receive data ($W_r = 0$), or it does not send data ($W_s = 0$). If both W_r and $W_s = 0$ are zero, then the problem is reduced to the well-known voltage scaling problem where the slowest speed on the processor yields the minimum energy. The other variation is that when $F_r = F_s$, meaning that $RECV$ and $SEND$ use the same communication speed, the problem is also reduced to a one-dimensional problem.

The simplest multi-processor system consists of two nodes: one sender and one receiver. It is shown in Fig. 4.

Problem 2 (Dual-Node Sender-Receiver) Given a system consisting of two nodes: sender N_1 ($Scale_{p_1}, Scale_{r_1}, Scale_{s_1}$) and receiver N_2 ($Scale_{p_2}, Scale_{r_2}, Scale_{s_2}$), workload $W_{r_1} = 0, W_{p_1}, W_{s_1}$ and $W_{r_2}, W_{p_2}, W_{s_2} = 0$, and deadlines D_1, D_2 , find processor speeds F_{p_1}, F_{p_2}

and communication speeds F_{s_1}, F_{r_2} that minimizes energy E_{sys} for the system.

Solution In the $SEND-RECV$ pair, $W_{s_1} = W_{r_2}$. We also assume the sender and receiver have same communication speed, $F_{s_1} = F_{r_2}$, therefore, $T_{s_1} = T_{r_2}$. In practice, $F_{s_1} \leq F_{r_2}$, but we assume $F_{s_1} = F_{r_2}$ to make the problem one-dimensional. Let $T_1 = T_{s_1} = T_{r_2}$. The total energy then is a function of T_1 ,

$$E_{sys} = Scale_{p_1} \left(\frac{W_{p_1}}{D_1 - T_1} \right) (D_1 - T_1) + \left[Scale_{s_1} \left(\frac{W_{s_1}}{T_1} \right) + Scale_{r_2} \left(\frac{W_{r_2}}{T_1} \right) \right] T_1 + Scale_{p_2} \left(\frac{W_{p_2}}{D_2 - T_1} \right) (D_2 - T_1)$$

To enumerate combinations of $(P_{s_1}, F_{s_1}) \times (P_{r_2}, F_{r_2})$, since $F_{s_1} = F_{r_2}$, the combination becomes $(P_{s_1}, P_{r_2}, F_{s_1})$, which is one dimensional. The enumeration function is omitted.

When $D_1 = D_2 = D$, the serialized tasks on the sender N_1 and the receiver N_2 can be pipelined as shown in Fig. 4(c). The system becomes a *two-node pipeline*.

An M -node pipeline consists of nodes $N_i, i = 1, 2, \dots, M$, among which each node N_i receives data from the previous node N_{i-1} (except the first node N_1), and sends the results to the next node N_{i+1} (except the last node N_M). Fig. 5 shows an example of a three-node pipeline.

Problem 3 (M -node Pipeline) Given an M -node pipeline consisting of M nodes $N_i: (Scale_{p_i}, Scale_{r_i}, Scale_{s_i}), i = 1, 2, \dots, M$ with workload $W_{p_i}, W_{r_i}, W_{s_i}$, and a deadline D for all nodes, find all processor speeds F_{p_i} , and communication speeds F_{r_i}, F_{s_i} that minimize energy E_{sys} for the system.

Solution Between each $SEND-RECV$ pair, $W_{s_i} = W_{r_{i+1}}$, and the sender and receiver have same communication speed, $F_{s_i} = F_{r_{i+1}}$, therefore, $T_{s_i} = T_{r_{i+1}}$. Let

$$T_i = \begin{cases} T_{r_1} & \text{for } i = 0 \\ T_{s_i} = T_{r_{i+1}} & \text{for } i = 1, 2, \dots, M-1 \\ T_{s_M} & \text{for } i = M \end{cases}$$

indicating the delays of $M+1$ instances of data communication. The expression of total energy is

$$E_{sys} = \sum_{i=1}^M \left(Scale_{r_i} \left(\frac{W_{r_i}}{T_{i-1}} \right) T_{i-1} + Scale_{s_i} \left(\frac{W_{s_i}}{T_i} \right) T_i + Scale_{p_i} \left(\frac{W_{p_i}}{D - T_{i-1} - T_i} \right) (D - T_{i-1} - T_i) \right)$$

assuming no collision on the shared communication medium (discussed later in Section 4.3). This is an $(M+1)$ -dimension optimization on variables T_0, T_1, \dots, T_M . The enumeration space is also $(M+1)$ -dimensional.

4.3 Communication scheduling

Fig. 5(c) shows a pipelined view of the three-node pipeline by folding the tasks on different nodes to a common time interval with a length D . Note that there seem to be two instances of task $PROC$ on node N_3 . This does not mean that task $PROC$ on node N_3 is

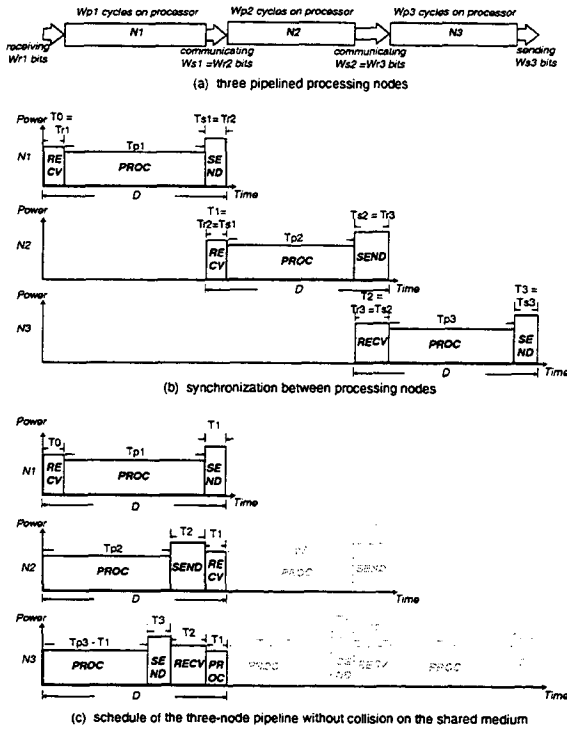


Figure 5: A 3-node communication pipeline.

preempted. In fact, each instance is a part of an integrated task *PROC* across the iteration boundary. In this particular view of the pipeline, the iteration boundary resides in the middle during the execution of task *PROC*. Such a technique is called *wrapping* and has been well-studied [4].

Fig. 5(c) shows that communication activities are shifted away from each other, such that at any given time, there is at most one active communication instance. This is especially meaningful if all nodes share the communication medium such as Ethernet, PCI bus, etc. If data collision will not occur, then our estimation on both performance and energy of the whole system can be well bounded. Collision is always undesirable because retransmission costs both time and energy. Communication activities should be scheduled such that the system is collision-free.

Lemma 1 (Collision-free Condition) An M -node pipeline does not have collision on the shared communication medium iff $\exists T_i, i = 0, 1, \dots, M$ (where T_i corresponds to the speed settings on all nodes), such that the *utilization* of the shared medium is less than or equal to 1. That is,

$$U = \sum_{i=0}^M \frac{T_i}{D} \leq 1 \quad (7)$$

If Lemma 1 cannot be satisfied, then the communication tasks involved in collision will incur extra delay and energy, and thus will force the processors to run faster with even more energy. Given that the extra communication delay is unbounded due to collision, the whole set of assumptions to support our analysis will be invalidated.

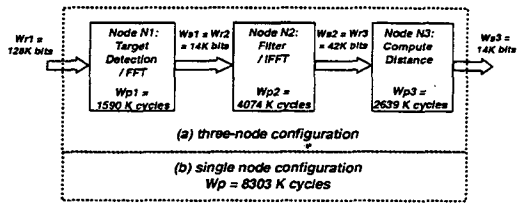


Figure 6: ATR mapped to three-node or single node.

Lemma 2 (Necessary Condition for Schedulability) An M -node pipeline is schedulable, i.e., able to meet deadline D , only if \forall nodes $N_i, i = 1, 2, \dots, M$,

$$\min(T_r) + \min(T_p) + \min(T_s) \leq D \quad (8)$$

Lemma 2 states the *overload* condition of a single node: if its workload cannot be completed before the deadline by operating at the maximum speeds (leading to the minimum execution delays, accordingly) for both communication and computation, then this node will fail to meet the deadline and thus the whole pipeline will be malfunctioning. The minimum execution delays can be derived from (2) by applying the maximum speeds, assuming ideal communication medium. A more realistic communication model can be also applied to calculate the minimum communication delays.

Lemma 3 (Sufficient Condition for Schedulability) An M -node pipeline is schedulable to meet a deadline D , if
 (1) There is no collision on the shared medium (Lemma 1), and
 (2) No node is overloaded (Lemma 2).

Lemma 3 says that the system is schedulable when conditions in Lemma 1 and Lemma 2 are satisfied. However, when the system has collision but no overloaded node, the schedulability is unknown.

5. ANALYTICAL RESULTS

To evaluate our method, we experimented with mapping the ATR algorithm onto two architectures: a three-node pipeline and a single processor. The three-node mapping is shown in Fig. 6 as solid bars. The input data size is 128K bits, and the output is 14K bits per frame. In addition, the internal communication requires 14K bits and 42K bits per frame, respectively. The one-processor mapping is shown as the surrounding dashed bar. The input and output data rates are the same as before, and there is no internal communication. We assume the deadline $D = 20ms$ for both architectures.

Each of our hardware nodes consists of an XScale processor and an LXT-1000 Ethernet interface from Intel. The $Scale_p$ and $Scale_s$ (same as $Scale_r$) functions, which indicate the power vs. performance characteristics of a node, are extracted from their data sheets [1, 2] and are shown in Fig. 7 and 8.

Fig. 9 shows the energy consumption for processing one image frame at different communication speeds on the one-node and three-node architectures. In both cases, the low-power, 10 Mbps communication speed results in the highest energy, while the highest-power, fastest 1000-Mbps communication speed achieves minimum energy. The three-node pipeline consumes lower energy than the single processor case, and it is conceivable that additional processors can achieve even more energy-efficient design points.

In this particular example, low-power communication at 10 Mbps is undesirable for both the 1-CPU and 3-CPU configurations. For the single node case, the low-speed communication left so little

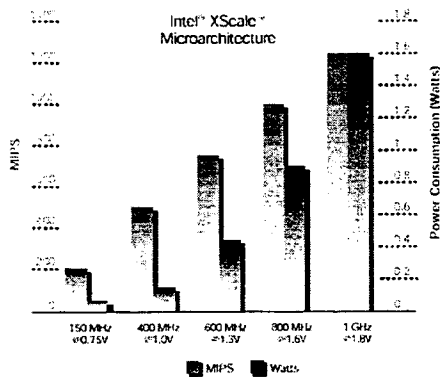


Figure 7: Power vs. performance of the XScale processor.

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

Figure 8: Power modes of the Ethernet interface.

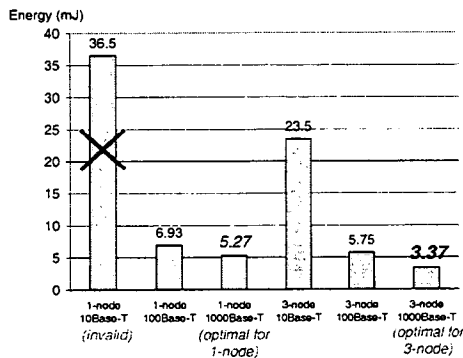


Figure 9: Analytical results.

time for computation, that the processor would have to operate at 1.5GHz in order to meet the deadline. This is an invalid solution, because it exceeds the 1GHz maximum clock rate. We say that the single node is overloaded and cannot meet the deadline according to Lemma 2. Even if the processor could support a higher clock rate at 1.5GHz, this solution would still demand much more energy.

In the three-node configuration, 10 Mbps communication not only requires more energy, but it is also dangerously close to violating the schedulability condition by Lemma 3. By (7), the utilization $U = 0.97$. Given that Ethernet is not an ideal communication medium, in practice it will be overloaded ($U > 1$) when overhead is taken into account. By Lemma 1, when collision occurs, the system will spend even more (potentially unbounded) time and energy than estimated. By Lemma 3, the system's ability to meet the deadline is undecidable.

It is notable that given properties of the Ethernet interface in Fig. 8, 1000 Mbps communication will always lead to lower energy consumption, even though it has the highest power rating.

This is due to the fact that at the 1000 Mbps data rate, the interface consumes the least energy per bit. Thus, it consumes the least energy for a fixed amount of data while leaving the most opportunity (time) for reducing CPU energy. However, a communication speed with the lowest energy-per-bit will not necessarily lead to the minimum energy at the system level, if it is not the highest speed. In a general case, optimal speed selection is based on many factors, including the processor, the communication interfaces, the hardware configuration (number of nodes), the application, and its mapping to the architecture (partitioning). The optimal speed selection may involve a mixture of heterogeneous speed settings between different nodes. For example, the node N_2 may communicate with N_1 at 1000 Mbps and with N_3 at 100 Mbps.

6. CONCLUSION

This paper presented a communication speed selection technique for minimizing energy consumption in networked embedded processors. Communication and computation represent two classes of resources that compete over opportunities for operating at the most power-efficient speeds, but the intricate dependencies between them require that speed selection for optimal energy be performed at the system level. Our formulation captures the data/precedence dependencies and their relations to the speed between all pairs of computing/communicating nodes and is generally applicable to networks of embedded processors, as well as single processor architectures that must perform I/O. Our communication speed selection optimizes for a particular functional partitioning scheme but we envision that it will be easily incorporated into most existing functional partitioning co-design loops for enhancing the exploration of power/performance trade-offs in networked embedded processors and systems-on-chip architectures.

REFERENCES

- [1] INTEL ethernet PHYs/transceivers. 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] L.-F. Chao, A. LaPough, and E. H.-M. Sha. Rotation scheduling: A loop pipelining algorithm. *IEEE Transactions on Computer Aided Design*, 16(3):229–239, March 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] 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.
- [7] 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.
- [8] R. Ortega and G. Borriello. Communication synthesis for distributed embedded systems. In *Proc. International Conference on Computer-Aided Design*, pages 437–444, 1998.
- [9] R. Sims. Signal to clutter measurement and ATR performance. *Proc. of the SPIE - The International Society for Optical Engineering*, 3371(1):13–17, April 1998.
- [10] 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.
- [11] W. Wolf. An architectural co-synthesis algorithm for distributed embedded computing systems. *IEEE Transactions on VLSI Systems*, pages 218–229, June 1997.