Communication-Aware Task Scheduling and Voltage Selection for Total Systems Energy Minimization†

Girish Varatkar Radu Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890
{gvv,radum}@ece.cmu.edu

Abstract: *In this paper, we present an interprocessor communication-aware task scheduling algorithm applicable to a multiprocessor system executing an application with dependent tasks. Our algorithm takes the application task graph and the architecture graph as inputs, assigns the tasks to processors and then schedules them. As main theoretical contribution, the algorithm we propose reduces the overall systems energy by (i) reducing the total interprocessor communication and (ii) executing certain cycles at a lower voltage level. Experimental results show that by tuning the parameter for communication awareness, a schedule using our algorithm can reduce upto 80% interprocessor communication in a complex video/audio application (compared to a schedule which is only voltage-selection aware) without losing much in the number of cycles executed at lower voltage.*

Keywords: low-power scheduling, dynamic voltage scaling

1. Introduction and objectives

Today communicating multiprocessor systems appear in a wide range of products. Modern embedded systems consist of more than one processing elements that *heavily communicate* with each other. In such a multiprocessor system, the processing elements can be general-purpose processors, application specific integrated circuits, or field-programmable gate arrays etc. These processing elements are networked using USB ports, ethernet, or high-speed serial busses [1] and they communicate with each other using point-to-point connections or on a shared bus or some such communication architecture. Such communicating multiprocessor systems are found in a wide range of products ranging from consumer electronics or peripheral devices attached to workstations, to automobiles. With the advent of systems-on-chip (SOCs), future embedded systems may have all the heterogeneous components (IPs) on a single chip. A new class of on-chip networks is also emerging as the interconnect of choice for connecting together these components [2][21]. Such an SOC will also have many IPs communicating with each other. Thus *communicating multiprocessor systems* appear not only in networked embedded systems but also in SOCs.

Since these embedded systems run on batteries, maximizing the battery-life has become one of the chief design drivers. Targeting *low-energy* (and low-power) *as early as possible* in the design process, at high levels of abstraction is extremely important. Dynamic voltage scaling [3] and power management [4] are the two main system-level techniques used to reduce energy consumption of the system. Dynamic voltage scaling is more effective in saving energy consumption than dynamic power management. As a result, several variable voltage processors appeared in the market (e.g. Intel's XScale, AMD's Mobile Athlon and Transmeta's Crusoe processor).

Most of the previous work in voltage scaling algorithms concentrates on saving energy of *independent* tasks running on a single processor or dependent tasks running on a single processor. However, most embedded systems today consist of *dependent tasks running on multiple processors*. Therefore we need scheduling algorithms which are able to save energy by giving maximum opportunity for voltage scaling in such multiprocessor embedded systems.

Any application running on a multiprocessor system can be modeled in the form of a task graph (Figure 1). These tasks have deadlines and two types of dependencies on each other: First, the *control dependency* indicating that one task can not start its execution until another task has finished. Second, the *data dependency* in which the output of one task is used as input by another task. The presence of data dependency implies these tasks communicate with each other. This application task graph needs to be implemented on a given set of variable voltage processors. The problem is then to find a system implementation that consumes the least amount of energy while satisfying the imposed deadlines. A practical system implementation has to solve three main issues: 1) task assignment to processors in the system, 2) task ordering, and 3) voltage selection.

Task assignment and task ordering, taken together, are referred to as *task scheduling*. The total systems energy consumption is dependent on the combined effect of the task schedule and the voltage selection step. Depending upon the task schedule, the system has different volume of total interprocessor communication and also different opportunities for lower voltage selection in the subsequent voltage selection step. Therefore the task scheduling and the voltage selection steps *can not* be treated as independent problems. Instead, they need to be combined in the effort to minimize the total systems energy; this is exactly the focus of the present paper.

1.1. Contribution of the paper

Total systems energy has two parts: communication energy and computation energy. The objective of this paper is to propose an algorithm for task scheduling and voltage selection, *foreseeing* the total interprocessor communication that the schedule will generate and trying to reducing it, while taking into account the energy savings that can be obtained in the subsequent voltage selection step.

Few previous voltage scaling approaches which discuss dependent tasks and multiple processors assume that the task assignment to processors is already done and then give an algorithm for voltage selection [7][12]. They try to distribute the slack, the maximum amount of time by which a task can be slowed down without violating the timing constraints, in different ways among the tasks. As pointed out in [8], the task scheduling algorithm has to be *voltage selection aware* and try to maximize the possibility of saving *computational energy* in the voltage selection step. The limitation of this approach, however, is that it ignores the interprocessor communication completely.

Tasks in real-world applications usually have data dependencies among them. The task assignment phase of the scheduling algorithm has to consider the penalty in total energy consumed by the system in assigning two heavily communicating tasks to two different processors. Real-time systems community has addressed the problem of minimizing the inter-task *communication energy* [16][5] for multiprocessor system, but their algorithms ignore the 'voltage selection' awareness, which requires maximizing the slack. This is because their

†This research was supported in part by NSF under grant CCR-00-93104 and MARCO/DARPA Gigascale Silicon Research Center (GSRC).

ICCAD'03, November 11-13, 2003, San Jose, California, USA.

work is mostly concerned with performance rather than power dissipation.

Our approach addresses the generalized scenario and tries to reduce the total energy consumed by the systems, which consists of the computation energy using *voltage selection awareness* and the interprocessor communication energy using *communication awareness* in the task assignment.

We emphasize that these two issues are *not* separate because if a system implementation minimizes the communication energy only, it will *not* necessarily produce the least total system energy solution because of lost opportunity in voltage scaling. At the same time, an implementation that minimizes the computation energy only may not be necessarily the least total systems energy solution because it might generate a lot of communication energy consumption. Therefore it is important to combine both these concerns in a single system implementation algorithm in order to minimize total energy consumption.

1.2. Related work

In recent years, the use of voltage selection to reduce computational energy has received increased attention. In [14], the authors perform battery-aware task scheduling for multiprocessor systems by moving tasks to smoothen the power profile. In [18], the authors describe a dynamic programming technique for solving the multiple supply voltage scheduling problem but they assume that the tasks are already mapped to processors. Similarly, [11] considers a generalized task graph with intertask communication and suggests an algorithm for partitioning and mapping. That algorithm, however, does not try to minimize the total interprocessor communication volume. More recently, [8] describes the need for a scheduling algorithm to take into account the voltage selection awareness. All of the above mentioned algorithms completely ignore the impact of scheduling on the interprocessor communication volume.

From a different perspective, in the real-time community, the basic scheduling problem is to make sure that the tasks always meet their timing constraints [13]. Most of these algorithms try to balance the utilizations of the processors in the system and finish the tasks earlier than the deadline [17]. In [16], Hou et al try to minimize the interprocessor communication by scheduling periodic tasks on a set of communicating modules but they do not consider the voltage-selection-awareness in their algorithm.

In this paper, we combine both these concerns in a generalized task scheduling problem and propose an efficient heuristic to solve it. As we shall see later, this is very important to minimize the total systems energy.

1.3. Organization of the paper

Section 2 presents a motivational example for communication awareness in scheduling and its impact on energy. In Section 3, we present the detailed problem formulation. In Section 4, we describe the scheduling algorithm and in Section 5, we show our results using random, as well as real multimedia system taskgraphs. Finally, we conclude by summarizing our main contribution.

2. Motivational example

We use the task graph shown in Figure 1 to illustrate the intuition behind our contribution. Each node in Figure 1 represents a task of the application and the directed arrows indicate the control dependency. A directed edge from task T_i to task T_j indicates that T_j can not start before T_i is finished. The number attached to the edges indicates the quantity of communication traffic from task T_i to task T_j in terms of abstract communication unit tokens (e.g. t1 sends 10 units of data to t3). A deadline next to a task indicates that the task must finish before the deadline (e.g. task t4 in Figure 1 must finish before 19 time units). The number inside each task node in Figure 1 indicates the number of cycles taken by that task for computation. Note that this number is independent of the processor's voltage level of operation.

Without loss of generality, we can make the following assumptions about the system architecture:

- The system consists of two identical processors.
- For each processor, there exist two voltage levels of operation. At the higher voltage level V_h, we assume the cycle time is one time unit while at the lower voltage V_l, the cycle time is two time units.
- The computational energy consumed by a processor at V_h is four energy units per cycle while at V_l, it is only one energy unit per cycle. These energy consumption numbers are chosen close to the ratio of energy consumed by commercial processors [20] at high and low voltage levels.
- The communication energy between the two processors is assumed to be 0.1 energy units per unit quantity of communication (unit quantity of communication may be bit/byte/packet/token etc. for randomly generated taskgraphs). Depending upon the choice of this number, the ratio of communication energy to the total systems energy will vary.
- The time and energy overheads of voltage level switching are assumed to be negligible since the switching does not happen very frequently [6].
- The time taken for interprocessor communication is assumed to be negligible compared to the computational time.

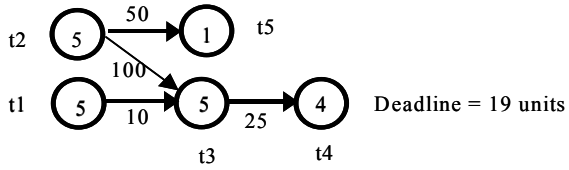


Figure 1. Taskgraph for the motivational example.

Under the above assumptions, Figure 2 shows three different system implementations of the task graph in Figure 1. For each time unit, the light color indicates the operation of the processor at the low voltage level while the shaded color indicates the operation at a high voltage level.

Figure 2 (a) shows the implementation using *only communication-aware* task scheduling algorithm followed by equal distribution of slack among the tasks in voltage selection phase. We point out that people in the real-time systems community who proposed the communication-aware scheduling [5] algorithm did *not* consider the voltage selection phase since their problem was in a different context (distributed systems). To make the algorithm applicable to the case of SOCs and embedded systems, we need to combine it with a voltage selection algorithm.

Figure 2 (b) shows the implementation using *only voltage selection aware* task scheduling and voltage level selection using the algorithm described in [8]. Finally Figure 2 (c) shows the task schedule obtained using our approach which is both *communication and voltage selection* aware and reduces the total energy of the system.

If we compare the energy consumed by the systems in Figure 2 (a) and Figure 2 (c), we can see that the computational energy consumed in the former is 56 units[1], while energy in the latter case is only 47 units. This clearly illustrates the shortcoming of the only communication-aware task scheduling algorithm to *foresee* that voltage scaling is going to follow in the next phase. So the schedule in figure 2 (a) does not maximize *slack,* and hence consumes higher computational energy. The schedule in 2(a) does well in limiting the communication energy of the system but the total energy consumed by the system is more in 2(a) than that in 2 (c).

Now comparing the energy consumed by the systems in Figure 2 (b) and Figure 2 (c), we notice that the computational energy con-

1. Number of cycles at V_h (shaded color) * 4 + Number of cycles at V_l (light color) * 1 = 12 * 4 + 8 * 1 = 56 units

sumed in both the schedules is the same but the communication energy consumed in the former is more than that in the latter (10 > 6 units). This is because the scheduling algorithm in 2(b) ignores the communication in the scheduling phase which is considered in 2 (c). So the total system energy consumption is lesser for 2 (c). This example illustrates the need for the new approach proposed in the paper.

3. Problem formulation

In this section, we formally state the *generalized task scheduling (GTS)* problem. Let us consider first a few useful definitions.

Definition 1: An *application task graph G(T,E)* is a directed graph, where each vertex represents a computational module of the application referred to as a task $T_i, T_i \in T$. Each task T_i belongs to a class CL_j ($1 \le j \le N_{CL}$), where N_{CL} is the maximum number of different classes of processors indicating its type depending upon the class of processor (general-purpose processor, DSP, etc.) on which it will run in the most efficient manner. For instance, a FFT task can be performed by different classes of processors, (e.g. DSP processors, general purpose processors or ASICs), but the number of cycles taken by the FFT task on each type of processor differs widely depending upon the actual class of processor used. Each task has its computational complexity associated with it in terms of the average *number of cycles* $NC_{i,CL}$ required to execute that task on different classes of processors. Note that the *number of cycles* taken by a processor to execute a task depends on the class of processor on which that task is executed while it is independent of the processor speed and its voltage level.

Each directed edge $E_{ij} \in E$ starts from a task T_i and ends on a task T_j ($T_i, T_j \in T$). The direction from T_i to T_j indicates that the task T_j can not start before T_i is finished. Each edge E_{ij} has associated with it B_{ij}, the number of bytes transferred from T_i to T_j. If this number is greater that zero, then it means that T_j can start only after T_i has finished and transferred B_{ij} bytes to the processor implementing T_j. Some of the tasks T_i may also have deadlines dl_i associated with them. The Definition 1 thus characterizes the application task graph and defines the control and data dependencies among the tasks in the application.

Definition 2: The *architecture graph L(P,W)* is a directed graph, where each vertex represents a processor P_i ($P_i \in P$). Each processor has a class (CL_j) associated with it. The class represents the type of the processor, whether the processor is ASIC or DSP or general-purpose processor, etc. Each bidirectional edge $W_{ij} \in W$ indicates a communication link between the two processors P_i and P_j. Each communication link W_{ij} has associated with it a certain communication speed (SP_{ij}) and a certain communication energy cost per unit communication volume ($ECOM_{ij}$). The numeric value of $ECOM_{ij}$ depends upon the communication architecture of the system. The communication energy per unit volume $ECOM_{ij}$ has two components. First is the energy cost of sending signals over the communication link. Second is the energy cost of storing the data received on the communication link into a local buffer at the port. For embedded systems with processors P_i and P_j on board, the value of $ECOM_{ij}$ varies depending upon their positions as the lengths of the wires connecting them changes with position. For an SOC with multiple processors on a single chip, the value of $ECOM_{ij}$ is due to the energy consumed by the links (typically in the order of 10 pJ) and the energy consumed in buffering (typically in the order of 1 pJ/per bit).

Definition 2 characterizes the architecture of the system consisting of the processors and the communication links between them. Processors are divided into a finite number of classes, N_{CL}, depending upon their type as described in Definition 2. Now suppose that each processor class CL_i can have different voltage levels VL_j. We denote

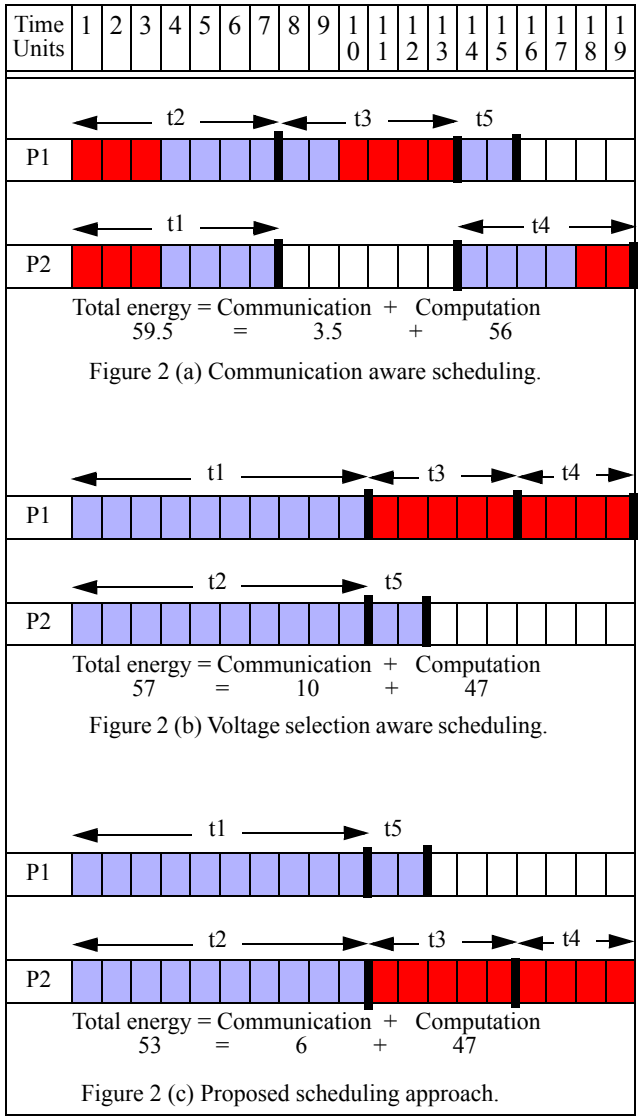


Figure 2 (a) Communication aware scheduling.

Figure 2 (b) Voltage selection aware scheduling.

Figure 2 (c) Proposed scheduling approach.

Figure 2. Different system implementations of taskgraph in Figure 1.

the computation energy per unit cycle for a processor of class CL_i operating at voltage level VL_j as $ECOP_{CLi,VLj}$. As we can see, this computation energy depends upon the class of processor and the voltage level. The time and energy overheads in changing the voltage level from VL_i to VL_j are denoted as DT_{ij} and DE_{ij} respectively [15]. A feasible schedule is defined as follows.

Definition 3: A *feasible schedule FS(G,L)* is defined as a mapping from the application task graph G to the architecture graph L such that

- Each $T_i \in T$ is mapped to a processor $P_j \in P$ such that P_j is in the set of classes of processors on which T_i can be executed. The tasks are assigned a start time s_i.
- All $T_j \in T$ such that there exists an edge $E_{ji} \in E$ starting from task T_j to task T_i, finish their execution and finish transferring B_{ji} bytes to P_i exactly at s_i or before s_i, the starting time of task T_i.
- All the tasks T_i for which deadlines dl_i are specified finish execution before their respective deadlines.

A feasible schedule characterizes the task assignment to different processors and the ordering in which the tasks are going to execute on the processors. It does not consider voltage selection which follows.

Definition 4: Given a *feasible schedule FS(G,L)*, a *system implementation I(G,L)* is a cycle accurate schedule of each task T_i, on processor P_j to which it is mapped to by the feasible schedule *FS*, after voltage selection is done for each cycle for all P_j, taking into account the time overhead for voltage level switching in satisfying all deadline constraints and energy overhead in voltage scaling. After *I(G,L)* is fixed, the $NC_{i,CLi,VLj}$ indicating the number of cycles for which task T_i is executed on the processor of class CL_i given by mapping *FS(G,L)* at voltage level VL_j gets fixed.

Given the application task graph *G* and the architecture graph *L*, we need to find a system implementation *I(G,L)* such that the total energy *TE* consumed by the system is minimized. The total energy consumed by the system incorporates the *computation energy* plus the *communication energy* and the *energy overhead* for switching voltage levels.

Problem formulation: Using the notations defined above, the problem can be formulated as to find a *system implementation I(G,L)* such that:

$$\min \{TE = TE_{COM} + TE_{COP} + TE_D\} \quad (1)$$

where TE_{COM} is the total communication energy given by

$$TE_{COM} = \sum_{i,j} B_{ij} \cdot ECOM_{lm} \text{ for } (i, j \text{ such that } T_i T_j \in T \text{ and } T_i, T_j \text{ are mapped to } P_l, P_m, \text{ respectively, s.t. } l \neq m \text{ in } I(G,L)) \quad (2)$$

TE_{COP} is the total computation energy given by

$$TE_{COP} = \sum_{i} NC_{i,\,CLi,\,VLj} \cdot ECOP_{CLi,\,VLj} \text{ where } (i \text{ s.t. } T_i \in T) \quad (3)$$

and TE_D is the total overhead energy during voltage level switching.

$$TE_D = \sum_{i,\,i} DE_{ij} \text{ whenever voltage changes from } VL_i \text{ to } VL_j \text{ in } I(G,L). \quad (4)$$

The *GTS* problem is NP-hard as the number of different possible schedules increases exponentially with the increasing number tasks. Hence we propose a heuristic algorithm described in the next section.

4. Task scheduling and voltage selection

Since finding an optimum *system implementation I(G,L)* consuming the minimum energy is known to be NP-hard [5] [13], we propose a heuristic scheduling algorithm which will (1) increase the total slack and create an opportunity for energy savings by voltage selection and, at the same time, (2) be interprocessor communication aware and try to save on interprocessor communication energy.

4.1. Input information

We consider real-time embedded systems with a few processors interacting with each other to execute a certain application (e.g. MPEG codec, MP3 codec, etc.) We consider that the application is divided into tasks (e.g. FFT, Variable Length Encoding, etc.) which can be executed by a certain processor once it receives the necessary input data and all its parent tasks are finished.

Without loss of generality, we assume all the processors are general purpose processors belonging to the same class (i.e. $N_{CL} = 1$). This means that for each task T_i the number of cycles NC_i is independent of the class of the processor on which the task is executed. Also the number of bytes, B_{ij}, transferred from T_i to T_j is known from profiling the application. This is a standard assumption and such task-graphs have been used previously in many related works [11][7][5]. The deadlines for the tasks are known from the application requirements. The deadlines can be hard deadlines which can not be violated or soft deadlines which can be stretched within a range. In this paper, we consider only hard deadlines in the experiments since multimedia applications have certain hard deadlines. Thus, the relevant information for the *application task graph G(T,E)* defined in Section 3, is given to the system designer.

We consider that the designer has already selected the set of processors and fixed their location and the interconnection between them for communication. We assume that the processors have point-to-point connections between them for communication as one example of a communication architecture. We discuss the impact of choosing another communication architecture, e.g. a bus-based solution in Section 5.3. Since the switching between voltage levels takes place not very frequently, we can neglect the time overhead and energy overhead for voltage level switching. Thus the *architecture graph L(P,W)* defined in Section 3, is assumed to be provided.

4.2. Design flow overview

Given the *application task graph G(T,E)* and the *architecture graph L(P,W),* our algorithm starts by ignoring interprocessor communication by setting *K*, the *communication ignorance parameter*, equal to 10. The algorithm first performs task scheduling by mapping the tasks onto processors and deciding their order of execution. Then, in the next step (Figure 3), voltage selection is performed for each cycle of all the scheduled tasks by formulating the deadline constraint equations and solving the ILP problem in order to minimize the total energy consumption.

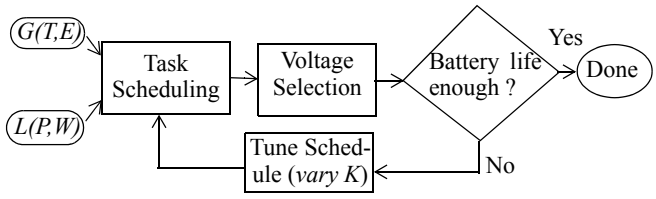


Figure 3. The system implementation design flow.

After the voltage selection step, the total energy consumed by the system can be computed. If the total energy consumption does not satisfy the battery-life constraint, the value of the parameter *K* is changed and the task scheduling and voltage selection steps are iterated until the battery life constraint is satisfied (Figure 3).The tuning of *K* can be done by creating an outer loop which varies the value of *K* in a suitable range (e.g. 0.1 to 10 in steps of 0.1). Since the *optimum* value of *K* that gives the *minimum* total systems energy is not known theoretically, the above mentioned outer loop gives the experimentally found value of *K* close to the optimum value.

4.3. Task scheduling and voltage selection algorithm

The task scheduling algorithm takes the application task graph and architecture graph as inputs. The output is a mapping of tasks to different processors and the order in which the tasks will be executed on these processors so that all the deadlines can be met. We use the basic framework of the multiprocessor task scheduling algorithm described in [8] to form an underlying voltage selection aware task scheduling algorithm. However, we modify the task assignment step to gain communication awareness and reduce interprocessor communication.

The algorithm we propose uses a priority-based ordering and best fit processor selection as shown in the pseudo-code in Figure 4. The task T_i's ready time r_i is the time when all its predecessors finish execution. The priority for each task T_i is calculated from the latest finish time, the earliest start time of the task (earliest start time of T_i denoted as es_i is the time when T_i is ready and a processor is available).

Suppose there are *N* processors in the system. The available time of processor P_j is denoted as a_{Pj} and it is updated regularly along with r_i, es_i during the scheduling step. The actual starting time of the task T_i is denoted as s_i. At each step, the algorithm calculates the priorities of tasks remaining to be scheduled and decides which task to schedule next. Then it uses a combination of voltage scheduling and communication-aware criterion to decide the processor to which the selected

Table 1. Trend in communication volume and number of cycles executed at low voltage for random taskgraphs.

Taskset	Number of Tasks	Number of Arcs	K = 0.1		K = 1		K = 10	
			Total inter-processor traffic	Number of slowed cycles	Total inter-processor traffic	Number of slowed cycles	Total inter-processor traffic	Number of slowed cycles
R1	11	12	33196	46	33196	46	44583	46
R2	15	21	14540	270	35441	522	109582	584
R3	17	25	34104	328	90926	452	151918	736
R4	21	32	41372	85	99529	339	243693	760
R5	24	36	62753	97	84934	373	255506	856

communication awareness increases

least priority task will be mapped to. The main idea behind the first step of task ordering is to schedule a task with smaller lft_i, latest finish time, before another one with greater lft_i. This is reflected in the priority PRI_i definition in the form of lft_i term in it as can be seen in Figure 4. Now, in the second step, in order to maximize energy savings opportunity in the voltage selection step, we must maximize the overall slack of the system. The main idea behind this is to minimize the slack given to some processors so that the slack on all other remaining processors is maximized. Therefore in the mapping step, the lowest priority task is assigned to a processor that was busy just before that task was released.

We add the communication awareness in the second step of mapping of the lowest priority task to the appropriate processor in the form of *communication criterion* (Step 2 in Figure 4). We set initially the communication ignorance parameter K to a high value (*e.g.* K=10) and then iteratively tune this parameter as shown in Figure 3. In the task ordering phase, whenever a task T_i is being mapped, we are sure that all its *parent* tasks, T_j s.t. $E_{ji} \in E$ are already mapped to processors and they have finished their execution. Therefore, we can calculate the total increase in the communication traffic that the mapping of T_i to P_j will generate by adding the traffic between parents of T_i that are mapped on a different processor than P_j and T_i (see Figure 5). We also know the number of edges which will contribute to this increase in traffic. So we can calculate the increase in the communication traffic per edge and compare it with K multiplied by the average traffic per edge for the whole application task graph.

For example in Figure 1, the average traffic per edge is ((100+50+25+10)/4 = 185/4 = 46.25) communication units (which can be bits/bytes/packets/tokens etc.) for the task graph. In Figure 2(c), when P_1 is finished with executing T_1 and at the same time, P_2 is finished with executing T_2, the communication criterion takes into account the fact that scheduling T_3 on P_1 will add 100 units per edge of communication traffic (since there is single edge between P_1 and P_2) to the system implementation. This is greater than the average traffic per edge (46.25 units) multiplied by K (assuming K=1). Therefore the communication criterion changes its mapping to P_2. The communication aware criterion can be written in the form of pseudocode as shown in Figure 5.

K is the tunable parameter; the lower the value of K, the more is the communication awareness. As shown in Figure 3, we can vary the value of K in an outer tuning loop to find the system implementation with lower total systems energy. When the value of K is high, the system implementation is only voltage selection aware and results in a schedule as given by algorithm discussed in [8]. As we start reducing the value of K, the implementation becomes more and more communication aware and reduces both the communication and computation energy. The optimal value of K varies for different taskgraph inputs and the outer loop can be used to give the system implementation with lower total energy consumption. We do not claim this to be the absolutely optimal solution with lowest possible total energy consumption, but it is certainly better than the previously proposed approaches. It remains an open research direction to find the truly optimum K.

Index of symbols:
lft_i = latest finish time of task T_i, dl_i = deadline time of T_i,
NC_i = number of cycles of T_i es_i = earliest start time of T_i,
r_i = release time of T_i, a_{Pi} = available time of processor P_i,
PRI_i = priority of task T_i, s_i = start time of T_i,

Step 1: // *Task order selection*: For each task T_i,
Calculate $lft_i = \min (dl_i, lft_j - NC_i$ s.t. $\forall i$ and $E_{ij} \in E)$
Calculate $es_i = max\ (r_i, \min (a_{Pj}$ s.t. $\forall j < N+1))$
Calculate $PRI_i = lft_i + es_i$
Select T_i with the smallest PRI_i
Step 2:// *Best-fit processor selection* (Voltage scheduling aware)
1. if $a_{Pj} = r_i$, select P_j, map T_i *to* P_j
if (*communication criterion*) // see Figure 5
$s_i = r_i = a_{Pj}$, $a_{Pj} = s_i + NC_i$
else
Find-new-mapping // see Figure 6
2. else Map T_i *to* P_j, *such that* $a_{Pj} < r_i$ *and* $a_{Pj} > a_{Pk}$ for every $a_{Pk} < r_i$, $(k \neq j)$ // P_j *is the processor available at the latest time just before* r_i
if (*communication criterion*) // map T_i *to* P_j
$s_i = r_i = a_{Pj}$,
$a_{Pj} = s_i + NC_i$
else
Find-new-mapping
3. else // all processors are busy at release time of task T_i
Map T_i *to* P_j, if $a_{Pj} \leq a_{Pk}$, $\forall (k \neq j)$ //P_j is the earliest available processor after r_i
if (*communication criterion*) // map T_i *to* P_j
$s_i = a_{Pj}$, $a_{Pj} = s_i + NC_i$
else
Find-new-mapping

Figure 4. The proposed task scheduling algorithm.

If communication criterion described in Figure 5 detects that the particular mapping of T_i to P_j is causing lot of interprocessor communication, we change the mapping of T_i to P_k. P_k is selected to be the processor to which the heaviest communicating parent of T_i was mapped to. In the case of figure 2(c), there is only one other processor P_2. This is the maximum that we can reduce the interprocessor communication and at the same time keep the algorithm to find new mapping simple. We take care to check that the deadline constraints are satisfied in the new mapping. The algorithm for finding the new mapping is described in Figure 6.

Map T_i to P_j
// Communication criterion starts
count = 0
$\forall k$ s.t. $E_{ki} \in E$ and $map(T_i) \neq map(T_k)$
    Add_traffic += B_{ki}
    count += 1

Add_traffic_per_edge = *Add_traffic* / *count*
Avg_traffic_per_edge = $\sum B_{ik}$ / Total number of edges
if (*Add_traffic_per_edge* > *K* * *Avg_traffic_per_edge*)
    return *false*
else
    return *true*

Figure 5. The proposed communication-awareness criterion.

Map T_i to P_j
// Communication criterion returns false, Find-new-mapping starts
Find *k'* such that $B_{k'i}$ = max (B_{ki}) and $map(T_i) \neq map(T_k)$
if ($a_{P(map(Tk'))} > r_i$ *and* $a_{P(map(Tk'))} + NC_i < lft_i$)
    map (T_i) = map($T_{k'}$)
else if $a_{P(map(Tk'))} \leq r_i$
    map (T_i) = map($T_{k'}$)

Figure 6. Find-new-mapping.

After task scheduling is done, the tasks are mapped to processors and the order in which they will execute is determined. There are different ways by which the slack can be distributed among the tasks [12][8]. We choose the approach described in [8] which is known to work better. Using this approach, we select the voltage levels for the cycles in the tasks by formulating the ILP problem to minimize the total energy. We use IBM optimization solutions and library OSLSSLV available from [9] to solve the ILP problem.

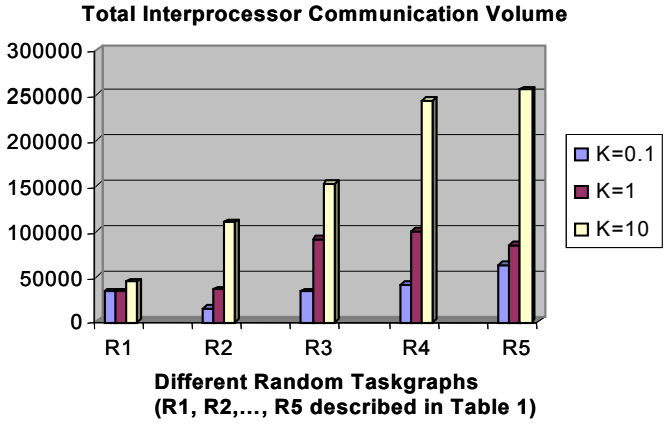
5. Experimental results and discussion

In order to verify the impact of interprocessor communication energy on scheduling, we implement the above framework and perform experiments on random tasksets generated using TGFF [10]. We also evaluate the framework on different architecture configurations by varying the number of processors in the system. To evaluate the potential of our algorithm in real applications, we apply this algorithm to a generic MultiMedia System (MMS) and discuss the results of the experiments.

5.1. Experiments on random taskgraphs

First we perform experiments on a system with only two processors and two different voltage levels, a high voltage at V_h and a low voltage at V_l. For simplicity, we assume that the cycle time at V_h is 1 time unit while at V_l it is only 2 time units. In these taskgraphs, the number of cycles taken by each task as well as intertask communication traffic volume are randomly generated. We vary the communication awareness parameter *K* in Figure 5 and observe the effect on *total interprocessor communication volume* and *number of slowed down cycles executed at low voltage level*. The scheduling algorithm described in Figure 4 takes a few seconds to run to completion for all the task graphs we considered in our experiments. The time taken by the algorithm increases linearly as the number of tasks and the number of arcs in the graph increase. The results are tabulated in Table 1.

We expect that when *K* is high (*K* =10), the communication awareness criterion in Figure 5 almost always returns true and the scheduling algorithm works only on minimizing the energy from voltage selection awareness perspective by maximizing the total slack. Therefore this column has maximum interprocessor communication volume due to lowest communication awareness and at the same time maximum number of cycles executed at low voltage. As we decrease *K* (*K*=1 in Table 1), the scheduling algorithm becomes more interprocessor communication energy aware and so, the mapping is done in such a way that the interprocessor communication volume decreases compared to the case with higher *K*. Therefore we expect the interprocessor communication volume to decrease in this case. If we set *K* very low (*K* =0.1), the scheduling tries to minimize the communication volume further more limiting the slack considerations. Therefore the communication volume is the minimum among the three cases, but it also has fewer cycles executed at low voltage level. This confirms to the trade-off between the interprocessor communication energy and the processor computation energy for different schedules. We can clearly see the expected trend in the *communication volume* and *number of slowed down cycles* as discussed above in the results for the random taskgraphs generated using TGFF listed in Table 1.



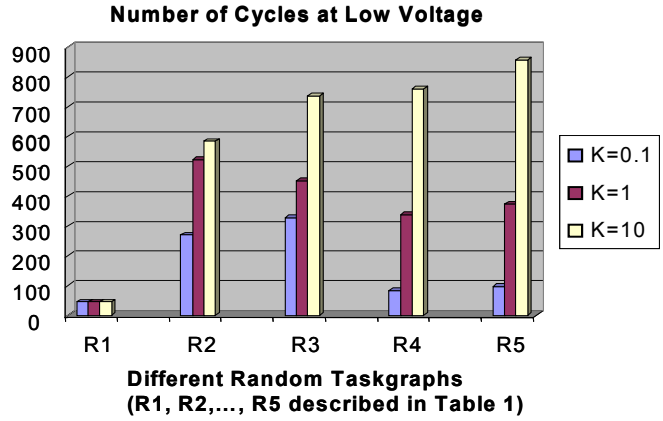


Figure 7. Trends in total interprocessor communication volume and number of cycles executed at lower voltage for random graphs.

The graphs shown in Figure 7 show the expected trend in total interprocessor communication volume (in number of units from the randomly generated taskgraphs) and the total number of cycles executed at low voltage. We can see from the interprocessor communication volume graph, that as value of *K* is reduced, the total communication volume decreases. The reduction is much more as the number of tasks increases as is the case of R5 compared to R1. At the same time, the number of cycles executed at lower voltage also decreases.

In order to evaluate the effectiveness of the algorithm in different system architecture configurations, we evaluate the communication volume and the number of slowed down cycles by varying number of processors. The results of these experiments are tabulated in Table 2. (R6-R8 are benchmarks of similar complexity as R1-R5 in Table 1.)

Table 3. Trend in communication volume and number of cycles executed at low voltage for MMS taskgraphs.

MMS Taskset *(video/audio)* clip	Number of Tasks	Number of Arcs	$K = 0.1$		$K = 1$		$K = 10$	
			Total inter-processor traffic	Number of slowed cycles	Total inter-processor traffic	Number of slowed cycles	Total inter-processor traffic	Number of slowed cycles
ENC_*akiyo/wawa*	24	27	87600	19215	103790	19215	333190	21740
ENC_*foreman/wawa*	24	27	38190	22913	51712	28314	219212	25033
ENC_*toybox/wawa*	24	27	38190	13237	55954	12795	209000	20710
DEC_*akiyo/beyond*	16	17	724	11687	14854	17687	82123	20544
DEC_*foreman/beyond*	16	17	724	9092	14854	15107	80795	15274
DEC_*toybox/beyond*	16	17	14774	3226	14774	3226	80715	5864

communication awareness increases

Task set	NP=2		NP=3		NP=4		NP=5	
	Total communication	Slowed cycles	Total communication	Slowed cycles	Total communication	Slowed cycles	Total communication	Slowed cycles
R6	135875	849	143145	1007	143145	1021	143145	1021
R7	136785	942	151323	999	158593	999	158593	999
R8	72270	701	64518	925	81783	925	81783	925

Table 2. The impact of varying the number of processors

We notice that as the number of processors (NP in Table 2) increases, the total interprocessor communication volume does not necessarily show a monotonic trend. The number of cycles executed at the lower voltage level also does not increase much because the schedule does not distribute the tasks equally among processors and so some processors are not executing any task most of the time. We can observe this trend in the results tabulated in Table 2.

5.2. Experiments on MultiMedia System taskgraphs

In subsection 5.1, we have observed the effectiveness of our proposed technique in reducing the total interprocessor communication volume by tuning the parameter *K* in the algorithm. Now we need to assess the impact of reducing the interprocessor communication volume on the total systems energy. The number of computation cycles and the number of bytes transferred in a random taskgraph may not be useful to calculate the computation and communication energies, respectively. This is because the proportion of the number of cycles and number of bytes of communication in a random taskgraph may never correspond to that in any real application taskgraph. For this reason, we need to compare the communication energy with the computation energy for a real system taskgraph rather than a random taskgraph.

We apply our algorithm to schedule the tasks in a generic MMS. The MMS we consider is an integrated video/audio system consisting of a H263 video/MP3 audio encoder pair (ENC in Table 3) and an H263 video/MP3 audio decoder pair (DEC). We partition these applications into 40 distinct tasks. We insert monitors in the C++ code and profile the intertask communication, as well as the number of cycles taken by each task. We experiment with three different video clips (*akiyo, foreman, toybox*) and two different MP3 clips (*wawa* and *beyond*). Using the profiled information in the application task graph, we schedule the task graph on a system with two ARM 11 processors. The interprocessor communication energy for these processors is 20 pJ/bit assuming these processors are placed adjacent to each other. The computation energy is 40 pJ/cycle at higher voltage and 13.3 pJ/cycle at lower voltage [19]. The interprocessor communication energy is estimated to be 20 pJ/bit[1]. We assume that the data communicated over a link will be stored in local registers at the port of the processor for a few cycles (we assume 10 cycles in general) before it is stored in a local memory bank of the processor or used by the computation unit for computation. We evaluated the register energy consumption metrics using Spice. The register buffer energy for storing the data is found to be 0.75 pJ/bit.

The *interprocessor communication volume* and *the number of slowed down cycles executed at low voltage* are tabulated in Table 3. From the table, we can see that for encoding say *akiyo* video clip using H263 encoder together with encoding *wawa* audio clip using MP3 audio encoder (row ENC_*akiyo/wawa* in Table 3), as we decrease *K* in our algorithm from 10 to 0.1, the task scheduling on the processors changes in such a way that the interprocessor communication volume decreases considerably from 333190 to 87600 (almost 70% decrease!). Of course, this decreases the interprocessor communication energy. The side-effect of the varying schedule is that the slack exploitation for lowering voltage levels and saving energy decreases. As we can see, the number of cycles executed at low voltage decreases (by almost 10%) and causes increase in the computation energy.

The impact of these variations on the total system energy can be seen in Figure 8. We can see that for *K*=10, the communication is totally ignored in making the scheduling decisions. For example, for the *akiyo* video clip encoded using the MMS encoder, the interprocessor communication energy is approximately 60% of the total systems energy for *K*=10. This results in higher total systems energy for *K*=10 than other schedules for smaller values of *K*. For *K*=0.1, the interprocessor communication energy is reduced to only 30% of the total systems energy and this schedule reduces the total systems energy. The optimal value of *K* for which the systems energy is minimized is found by tuning *K* as shown in the design flow in Figure 3. Thus we can conclude that the proposed system implementation approach reduces the total interprocessor energy.

5.3. Practical system implementation considerations

In products which implement such complex multimedia systems, the software code for the application is given to the system designer. The better the algorithm implemented in the software in terms of accounting for the hardware features of the processor on which it is going to implement, the lesser is the number of cycles it will take. The amount of *computational energy* savings that can be achieved by the voltage selection, is limited by the software. The *interprocessor communication energy* that can be saved depends upon the interprocessor communication volume as well as the communication energy cost per unit communication volume for the communication link W_{lm} denoted as $ECOM_{lm}$. The total communication energy is given by $TE_{COM} = \sum_{i,j} B_{ij} \cdot ECOM_{lm}$ (recall (2) from Section 3). We try to reduce the interprocessor communication volume which corresponds to reducing $\sum_{i,j} B_{ij}$. So TE_{COM} can be further reduced by placing the processors

1. Communication energy per bit is estimated approximately from ARM documentation [19] and using formula E=1/2 * V^2 * (Width of metal wire) * (Length of metal wire) * X(capacitance parameter from TSMC data sheet) = 20 pJ/bit.

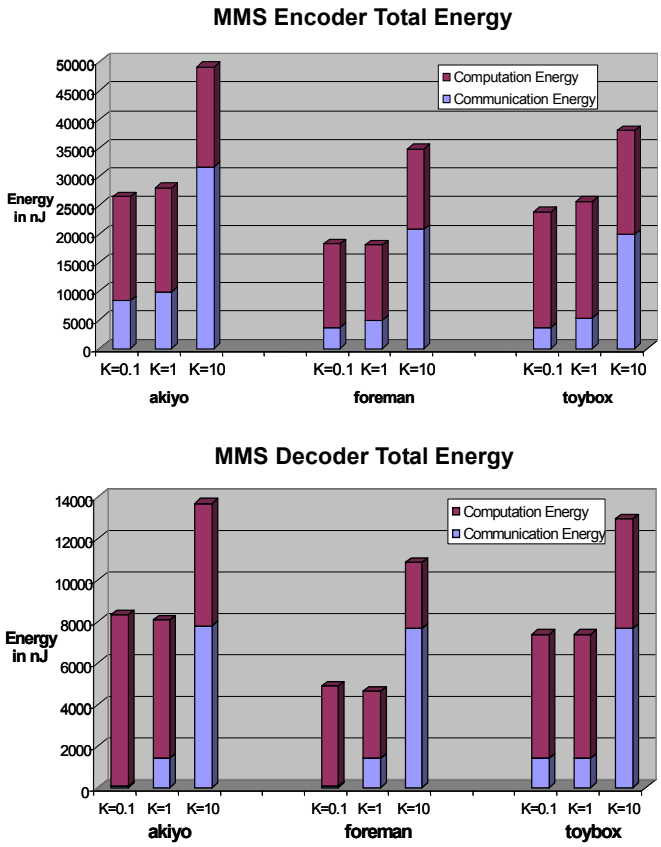


Figure 8. Total systems energy trends for MMS Encoder and MMS Decoder for three different clips.

communicating the most close to each other so that $ECOM_{lm}$ corresponding to the large B_{ij} will also be reduced.

The architecture graph *L(P,W)* shown in Section 3 corresponds to the communication architecture for the processors. In our experiments we consider the point-to-point communication architecture. This can also be a network-on-chip type of architecture in which the adjacent processors have a direct link for communication between them. Instead, if we have a bus-based communication architecture, E_{lm} will be the same for all processors. This is because when any two processors P_l and P_m are communicating over a bus, irrespective of their physical proximity, the whole bus is switched. In that case, reducing TE_{COM} amounts to reducing total interprocessor communication volume. But in the case of bus-based communication, the time overlap of communication also comes into picture and then, the feasible schedule has to consider the impact of bus arbitration scheme on the scheduling and subsequently on the total system performance and energy. This will lead to additional trade-offs in performance and total system energy. This can be the future work for the *communication aware scheduling* which points out the importance of reducing the interprocessor communication volume in the task scheduling step itself and evaluates it in comparison with the *computation energy* reduction.

6. Conclusion

In this paper, we first motivate the need to take into account the interprocessor communication volume in order to minimize the total system energy during the task scheduling phase. We then formally state the *generalized task scheduling* problem and suggest a heuristic approach which tries to reduce the interprocessor communication volume and, at the same time, increase the total available slack for voltage scaling. Experimental results with random and real multimedia system taskgraphs show that by tuning the parameter for communication awareness, we can vary the impact of task scheduling on total interprocessor communication volume. We can further extend this framework to integrate processor placement and communication architecture exploration.

References

[1] J. Liu et al, "Communication Speed Selection for Embedded Systems with Networked Voltage-Scalable Processors," *Proc.CODES* 2002, USA.

[2] W. Dally, B. Towles, "Route Packets, Not Wires: On-chip Interconnection Networks," *Proc. DAC,* Las Vegas, NV, June 2001.

[3] A. Chandrakasan, R. Broderson, “Low Power Digital CMOS Design,” Kluwer Academic Publishers, 1995.

[4] L. Benini et al, "A Survey of Design Techniques for System-level Dynamic Power Management," IEEE Trans. on VLSI Systems, June 2000.

[5] C. M. Krishna, K. G. Shin, “Real-time Systems,” WCB/McGraw-Hill, 1997.

[6] N. Namgoong, M. Yu, and T. Meng, "A High-efficiency Variable-voltage CMOS Dynamic DC-DC Switching Regulator," *Proc. ISSCC,* 1997.

[7] J. Luo, N. Jha, "Power-conscious Joint Scheduling of Periodic Task graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems," *Proc. ICCAD,* San Jose, CA, Nov. 2000.

[8] Y. Zhang, X. Hu, and D. Chen, "Task Scheduling and Voltage Selection for Energy Minimization ," *Proc. DAC,* New Orleans, LA, June 2002.

[9] http://www-3.ibm.com/software/data/bi/osl/features/lp-sol.html

[10] http://helsinki.ee.Princeton.Edu/dickrp/tgff

[11] J-M. Chang, M. Pedram, ”Codex-dp: Co-design of Communicating Systems Using Dynamic Programming,” IEEE Trans. on CAD, July 2000.

[12] F. Gruian, K. Kuchcinsky, “LEneS: task scheduling for low-energy systems using variable supply voltage processors,” *Proc. ASP-DAC,* 2001.

[13] H. El-Rewini et al, ”Task Scheduling in Multiprocessor Systems,” IEEE Computer, Dec.1995.

[14] J. Luo, N. K. Jha, “Battery-aware static scheduling for distributed real-time embedded systems,” *Proc. DAC,* Las Vegas, NV, June 2001.

[15] B. C. Mochocki, X. Hu, ”A Realistic Variable Voltage Scheduling Model for Real-Time Applications,” *Proc. ICCAD,* San Jose, CA, Nov. 2002.

[16] C. J. Hou et al, ”Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems,” IEEE Trans. on Computers, Dec. 1997.

[17] P. D. Hong et al, ”Scheduling of DSP programs onto multiprocessors for maximum throughput,” IEEE Trans. on Signal Processing, June 1993.

[18] J. M. Chang, M. Pedram, ”Energy minimization using multiple supply voltages,” IEEE Trans. on VLSI Systems, Dec. 1997.

[19] http://www.arm.com/armtech/ARM11

[20] http://www.transmeta.com/technology/specifications/index.html

[21] A. Jantsch and H. Tenhunen (Eds.), ”Networks on Chip,” Kluwer Academic Publishers, 2003.