Task Scheduling under Performance Constraints for Reducing the Energy Consumption of the GALS Multi-Processor SoC

Ryo Watanabe[†] Masaaki Kondo[†] Masashi Imai[‡] Hiroshi Nakamura[†] Takashi Nanya[†]
[†]Research Center of Advanced Science and Technology (RCAST), The University of Tokyo
[‡]Komaba Open Laboratory (KOL), The University of Tokyo
4-6-1 Komaba, Meguro-City, Tokyo, Japan
{Watanabe, Kondo, Miyabi, Nakamura, Nanya}@hal.rcast.u-tokyo.ac.jp

Abstract

The present paper focuses on applications that are periodic and have both latency and throughput constraints. For these applications, pipeline scheduling is effective for reducing energy consumption. Thus, the present paper proposes a pipelined task scheduling method for minimizing the energy consumption of GALS MP-SoC under latency and throughput constraints. First, we model target GALS MP-SoC architecture and application tasks. We then show that the energy optimization problem under this model belongs to the class of Mixed-Integer Linear Programming. Next, we propose a new scheduling method based on simulated annealing for the purpose of solving this problem quickly. Finally, experimental results demonstrate that the proposed method achieves a significant energy reduction on a real application under a practical architecture.

1. Introduction

Increasing energy consumption and heat dissipation are among the most significant problems in present-day computer systems. A number of studies have examined energy reduction using Dynamic Voltage Scaling (DVS)[5][6]. Energy consumption can be reduced by lowering its clock frequency and supply voltage while satisfying the given performance constraints. Ishihara et al. [11] investigated a method for selecting the optimal voltage setting under a constraint.

The Multi-Processor SoC (MP-SoC) is also an attractive solution to the energy problem because the energy consumed by a CMOS circuit is approximately proportional to the square of its computing speed, so that increasing the number of processors leads to a quadratic decrease in the energy consumption required to achieve the same performance level [4]. Application tasks are distributed on each processor in the MP-SoC. Thus, the DVS is not necessarily effective if the frequencies and voltages of all of the cores are the same, because the workload of each core is different.

A Globally Asynchronous Locally Synchronous (GALS) design [16] works very well in this situation. In the GALS design, a chip is partitioned into several Locally-Synchronous modules and each module operates at its own clock frequency and supply voltage. Thus, the GALS MP-SoC has a great potential for energy reduction by selecting the appropriate frequency and voltage level for each processor.

Task and voltage scheduling methods for reducing energy consumption can be divided broadly into two categories: static scheduling and dynamic scheduling. In static scheduling, the allocation of tasks onto processors and selection of the frequency/voltage level are performed prior to runtime, whereas in dynamic scheduling, the dynamic behavior of the system is observed. The advantages and disadvantages of these methods, as well as the differences in their targets, are detailed in [4].

In the present paper, we propose a method by which to statically schedule tasks on GALS MP-SoC for optimizing its energy consumption under given performance constraints. The optimal solution from static scheduling indicates the potential of DVS for reducing energy consumption and is useful as a guide for constructing dynamic scheduling algorithms.

Many applications are periodic and have both latency and throughput constraints. For a movie player, for example, the throughput is the frame rate, whereas the latency is the response time from the moment the ”PLAY” button is pushed until the file begins to play. In such a case, execution of tasks should be pipelined for energy optimization. By setting the pipeline cycle time and pipeline depth according to the throughput and latency constraints, significant energy reduction becomes available, flexibly adapting to these constraints.

The number of cores integrated on a chip is expected to increase in the future as technology scales. Thus, pipelined scheduling will become increasingly attractive for exploiting parallelism on abundant hardware resources. Therefore, we propose a pipelined scheduling on GALS MP-SoC for reducing energy consumption.

1.1. Contribution of the present study

The contributions of the present study are as follows.

First, to the authors' knowledge, this is the first work to investigate the optimization of energy consumption on pipelined task scheduling when both the throughput and the latency constraints are given independently. We show conventional task scheduling methods cannot fully utilize opportunities for reducing energy consumption in such a situation, and that the pipelined task scheduling is promising for further energy reduction.

Second, both energy and performance overheads due to inter-processor communication are taken into account. Since these overheads are expected to increase as the technology scales, it is important to consider these overheads.

Third, we model GALS MP-SoC and periodic applications, and show that the energy optimization problem belongs to the class of Mixed-Integer Linear Programming (MILP). We show that this problem belongs to a simpler class of optimization problem than was claimed in a previous study, although we use a more realistic assumption whereby the processors can use a limited number of frequency/voltage levels. The optimal task scheduling is derived by solving a MILP formulation.

Fourth, we propose a simulated annealing-based (SA-based) task scheduling method to very quickly obtain a near optimal task scheduling. In solving a MILP formulation, the most time consuming part is the branch-and-bound search of integer variables. In the proposed method, this is replaced by SA.

Fifth, the experimental results show that proposed method can quickly derive a task scheduling for which the energy consumption is approximately the same as that of the optimal scheduling when the method is applied to a real application under practical architectural assumptions.

1.2. Previous work

Several studies have examined energy saving techniques for MP-SoCs, or more generally System-on-Chips with multiple processor cores. Gruian et al.[9] proposed a task scheduling method by which to optimize the energy consumption under the assumption that the assignment of application tasks to processors is predetermined. Luo et al. [15] proposed a method by which to increase battery lifetime by changing the execution order of tasks and smoothing the power profile. Zhang et al. [21] reported that the opportunity of lowering the supply voltage is affected by the task allocation and ordering and introduced a two-phase method that integrates the task allocation and the voltage scheduling. Leung et al.[14] presented a Mixed-Integer Non-Linear Programming (MINLP) formulation in which the task allocation and the voltage selection problems are solved simultaneously. Varatkar et al. [20] proposed a heuristic algorithm that optimizes the energy consumption of a system while taking the inter-processor communication into account. Hu et al. [10] studied the case in which the system is structured as a tile-based NoC with a 2D-mesh network. Chen et al. [7] proposed a scheduling method for independent tasks with the consideration of power consumption due to leakage current.

The main consideration of these methods is to find the optimal scheduling that minimizes energy consumption under the latency constraint for given applications. However, these methods either do not take into account the throughput constraint, or, the throughput constraint is implicitly assumed to be equal to the inverse of the latency constraint. Therefore, these methods cannot optimize energy savings if there are different constraints for latency and throughput.

Pipeline scheduling is a well-known method in the field of signal processing in which the throughput is a major performance metric. Parazzari et al. [17] presented a method by which to schedule both periodic tasks with a throughput constraint and non-periodic interrupting tasks with a latency constraint. Ruggiero et al. [18] proposed a pipeline scheduling method by which to minimize the amount of inter-processor communication by combining Linear Programming (LP) and Constrained Programming (CP) approaches.

The goal of the proposed method is the minimization of energy consumption by solving the task and voltage scheduling problem while considering both the latency and throughput constraints. On this point, the present study differs from previous studies.

2. Problem modeling

Task scheduling is a procedure to map each task (a small unit of a program, e.g., a basic block or function) to a processor and to decide the start time and clock frequency/supply voltage for each task. We call the first step ”task allocation” and the second step ”voltage selection”.

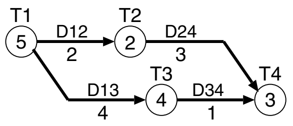

Figure 1. An example of a task graph.

We consider periodic applications that are executed periodically for different input data. Periodically released instances of the application are called jobs, which are represented using task graphs.

The application has two parameters, (PR, DL), where PR is the period of the job release and DL is the relative deadline of the job. Although many of the previous studies assumes $PR = DL$ in their scheduling model, we do not make this assumption and show that pipeline scheduling can work well in such a situation.

In the present paper, the task, the hardware (system), and the performance constraints are defined as follows.

Definition 1 (task graph): A task graph is a directed acyclic graph (DAG) $G = (\mathbf{T}, \mathbf{D}, w, q)$. Vertices $T_i \in \mathbf{T}$ represent tasks, and the amount of computation for task T_i is expressed as $w(T_i)$. Directed edges $D_{ij} \in \mathbf{D}$ indicate that T_j is dependent on T_i. This means that the execution of T_j cannot start before the execution of T_i and the data transfer from T_i to T_j are finished. Here, the amount of data to be transferred from T_i to T_j is expressed as $q(D_{ij})$. Subscripts i, j, i', and j' represent indices for tasks.

Definition 2 (system model): Target MP-SoC $S = (\mathbf{P}, \mathbf{V}, L, c_P, e_P, c_L, e_L)$ consists of identical processors $P_x \in \mathbf{P}$. Hereinafter, the subscript x represents the index for processors. Each processor is defined as a GALS locally-synchronous region. Therefore, the frequency/voltage level can be chosen independently for each processor. $V_k \in \mathbf{V}$ indicates the available frequency/voltage levels for processors. Hereinafter, the subscript k represents the index for frequency/voltage levels. The processing time for a unit of computation on a processor that operates at the level V_k is represented as $c_P(V_k)$, while $e_P(V_k)$ is the energy consumption for this computation. All processors are connected along a shared communication link L and, c_L and e_L indicate the transfer time (equal to $1/bandwidth$) and energy consumption, respectively, of L for a unit of interprocessor communication. Unlike processors, the bandwidth and energy consumption of L are supposed to be fixed. In addition, processors are not involved in the control of interprocessor communication because they have dedicated hardware for this function. Thus, computation and communication can be processed in parallel.

Definition 3 (performance constraint): The performance constraint is given by two parameters: the pipeline cycle time C_p and the number of pipeline stages L_p. This means that the system must finish executing a job every C_p time units (throughput constraint, $C_p = 1/throughput$), and the time for executing a job cannot exceed $C_p\ L_p$ ($= latency$: latency constraint). C_p must be the same as the job release interval PR, and C_pL_p must be the same as the relative deadline DL.

Next, we consider an example using the task graph shown in Figure 1. Each digit on T_i and D_{ij} represents $w(T_i)$ and $q(D_{ij})$, respectively. Suppose this application is scheduled on a system described by following parameters: $\mathbf{P} = (P_1, P_2)$, $\mathbf{V} = (V_{lo}, V_{hi})$, $(c_P(V_{lo}), e_P(V_{lo})) = (2, 1)$, $(c_P(V_{hi}), e_P(V_{hi})) = (1, 2)$, and $(c_L, e_L) = (1, 1)$. The throughput constraint is assumed to be $C_p = 15$, while the latency constraint is assumed to be $C_pL_p = 30$ ($L_p = 2$).

The non-pipelined scheduling result is shown in Figure 2(A). When scheduling jobs in a non-pipelined manner under the condition that latency and throughput constraints are given independently, the throughput must be the inverse of the latency. There-

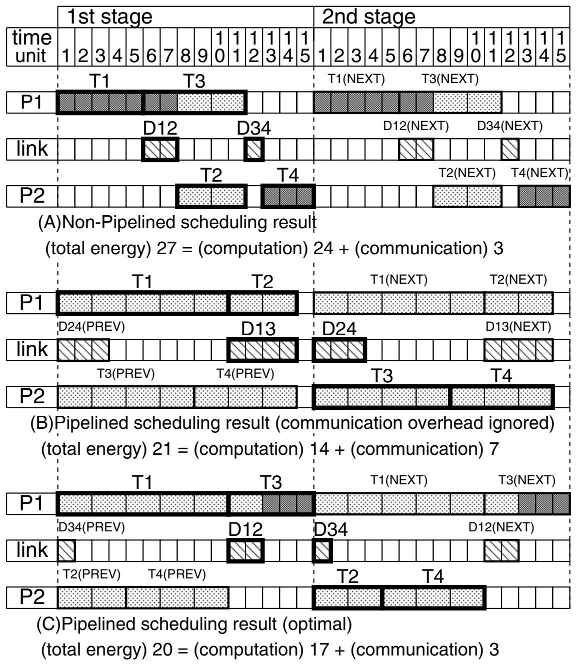
1st stage
2nd stage
time unit
P1
link
P2
T1
T3
T2
T4
D12
D34
D13
D24
(A)Non-Pipelined scheduling result
(total energy) 27 = (computation) 24 + (communication) 3
(B)Pipelined scheduling result (communication overhead ignored)
(total energy) 21 = (computation) 14 + (communication) 7
(C)Pipelined scheduling result (optimal)
(total energy) 20 = (computation) 17 + (communication) 3

Figure 2. Task scheduling results obtained from three different methods.

fore, the smaller value between C_p and C_pL_p, that is, C_p, is chosen for both the throughput and latency constraints.

Each colored grid in the figure indicates that the processor and communication link processes one unit of computation/communication. The light color indicates that the processor operates at V_{lo}, and the dark color indicates that the processor operates at V_{hi}. "(PREV)" indicates that the task belongs to the previous job, and "(NEXT) indicates that the task belongs to the next job. This scheduling exploits task level parallelism using two processors to obtain opportunities to use a lower frequency/voltage level.

Figure 2(B) is the result of pipelined task scheduling. The opportunity to use a lower frequency/voltage increases by not only exploitting task level parallelism but also parallelizing the tasks in other jobs. Pipelining is a method by which to multiply the throughput of a system, even if the performances of the processing elements do not improve at all. However, from another point of view, the voltage and frequency of each processor can be lowered using a pipelining without decreasing the throughput of the system. When latency and throughput are given independently as constraints, pipelining allows the performance of the system to be fit to constraints and increases the chance for reducing energy consumption.

Figure 2(C) shows another pipelined scheduling result. While the chance of using a lower frequency/voltage is smaller, it achieves a smaller energy consumption by optimizing costs in interprocessor communication. Balancing energy consumption in computation on a processor and communication on a communication link is also important for energy optimization of the MP-SoC.

These results motivate us to develop an effective task scheduling method.

3. MILP formulation

In this section, using the model described in Section 2, we show that the energy optimization problem in pipelined task scheduling under a performance constraint belongs to the Mixed-Integer Linear Programming (MILP) problem class. The MILP formulation of this problem has not yet been presented.

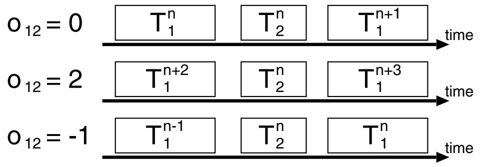

Figure 3. Definition of the execution order o_{ij}. T_i^n denotes T_i in the n^{th} job.

3.1. Notations

Real variables

s_i	Time to start executing T_i
r_{ik}	Proportion of computation executed at frequency/voltage level V_k to $w(T_i)$
cs_{ij}	Time to start transferring data of D_{ij}

Integer variables

$$o_{ij} = \begin{cases} n & \text{(if } T_j \text{ in the } m^{\mathrm{th}} \text{ job begins its execution after } T_i \text{ in the } (m+n)^{\mathrm{th}} \text{ job is completed and before } T_i \text{ in the } (m+n+1)^{\mathrm{th}} \text{ job begins (shown in Figure 3))} \end{cases}$$

$$co_{iji'j'} = \begin{cases} n & \text{(Represents the execution order of communication tasks } D_{ij} \text{ and } D_{i'j'}. \text{ Defined in the same manner as } o_{ij}.) \end{cases}$$

$$m_{ix} = \begin{cases} 1 & \text{(if } T_i \text{ is allocated on } P_x) \\ 0 & \text{(otherwise)} \end{cases}$$

3.2. Constraint inequalities and equations

All tasks must be completed by the time defined by the latency constraint.

$$\forall i \ (s_i \geq 0, \ s_i + t_i \leq C_p \, L_p) \tag{1}$$

t_i in (1) is the execution time length of T_i, as described in the following formula:

$$t_i = \sum_k w(T_i) \, r_{ik} \, c_P(V_k) \tag{2}$$

r_{ik} represents the ratio of how much V_k is selected within task T_i, Therefore, the following constraints must be satisfied:

$$\forall i \ \left(\sum_k r_{ik} = 1 \right) \tag{3}$$

$$\forall i, k \, (0 \leq r_{ik} \leq 1) \tag{4}$$

The throughput constraint is described as follows. The execution time of each task must not exceed C_p, otherwise the tasks cannot be executed every C_p time units on a processor.

$$\forall i \ (t_i \leq C_p) \tag{5}$$

The order of task execution is constrained if the task depends on another task. If dependent tasks are allocated on the same processor, dependence on the other task cannot begin earlier than the completion of its preceding task. Otherwise, data transfer begins after the preceding task is completed and a succeeding task can begin execution after data transfer has been completed. These criteria are indicated by the following inequalities:

Table 1. Combination of values of the variables defined in (10).

m_{ix}	0	0	1	1
m_{jx}	0	1	0	1
a_{ijx}	0	1	1	0
b_{ijx}	0	1	1	1

$\forall i, j \ s.t. \ D_{ij} \in \mathbf{D}$

$$s_i + t_i \leq s_j + C_p \, L_p \, d_{ij} \tag{6}$$

$$s_i + t_i \leq cs_{ij} + C_p \, L_p \ (1 - d_{ij}) \tag{7}$$

$$cs_{ij} + q(D_{ij}) \, c_L \leq s_j + C_p \, L_p \ (1 - d_{ij}) \tag{8}$$

Constraints that are valid only under specific conditions are indicated using binary variables, as proposed in [14]. The binary variable d_{ij} in (6), (7), and (8) is equal to 0 if T_i and T_j are allocated on the same processor, and is equal to 1 otherwise. This definition is given in following formulas:

$$\forall i \ \left(\sum_x m_{ix} = 1 \right) \tag{9}$$

$$\forall i, j, x \ (m_{ix} + m_{jx} + a_{ijx} - 2b_{ijx} = 0) \tag{10}$$

$$\forall i, j \ \left(d_{ij} = \frac{1}{2} \sum_x a_{ijx} \right) \tag{11}$$

Here, (9) is the definition of task allocation, i.e., a task is executed on only one processor, and (10) is a constraint for obtaining two binary variables a_{ijx} and b_{ijx} used for calculating d_{ij}. The relationship among variables a_{ijx}, b_{ijx}, m_{ix}, and m_{jx} is shown in Table 1. As shown in the table, the value of d_{ij} is determined according to (11).

Processors and communication links can process only one task or one data transfer at a time. Constraints for computation tasks are placed on all pairs of tasks on the same processors and are given as follows:

$\forall i, j$

$$s_i + t_i + C_p \, o_{ij} \leq s_j + C_p \, L_p \, d_{ij} \tag{12}$$

$$s_j + t_j \leq s_i + C_p \, (o_{ij} + 1) + C_p \, L_p \, d_{ij} \tag{13}$$

Similarly, constraints for communication tasks are placed on all pairs of communication tasks.

$\forall i, j, i', j' \ s.t. \ D_{ij} \in \mathbf{D} \cap D_{i'j'} \in \mathbf{D}$

$$cs_{ij} + q(D_{ij}) \, c_L + C_p \, co_{iji'j'} \leq cs_{i'j'} + C_p \, L_p \left(2 - d_{ij} - d_{i'j'}\right) \tag{14}$$

$$cs_{i'j'} + q(D_{i'j'}) \, C_p \leq cs_{ij} + C_p \, (co_{iji'j'} + 1) + C_p \, L_p \left(2 - d_{ij} - d_{i'j'}\right) \tag{15}$$

3.3. Objective function

The objective function is the energy consumption of the system.

$$\sum_i en(T_i) + \sum_{D_{ij} \in \mathbf{D}} en(D_{ij}) \tag{16}$$

In (16), $en(T_i)$ indicates the energy consumption of the processor for executing T_i, while $en(D_{ij})$ represents the energy consumed on the communication link transferring D_{ij}, as given in (17) and (18).

$$en(T_i) = \sum_k w(T_i) \, r_{ik} \, e_P(V_k) \tag{17}$$

$$en(D_{ij}) = \sum_x e_L \, q(D_{ij}) \, d_{ij} \tag{18}$$

Then, energy-optimal scheduling is obtained by solving the above formulas with minimizing objective function. Note that the above formulas include the interprocessor communication overheads, and that this formulation is based on a hardware model with limited number of available frequency / voltage levels, while [14] and [7] assumed systems with continuous frequency / voltage levels.

4. SA-based scheduling method

Although the optimal solution can be derived by solving the MILP formulas in Section 3, its computational complexity is a serious problem when applied to real systems and applications. MILP solvers use the branch and bound algorithm to obtain possible values of integer variables. In this task scheduling problem, allocation of tasks is represented by integer variables m_{ix}, o_{ij}, and $co_{iji'j'}$. The size of the search space is proportional to the number of combination patterns of integer variables, and therefore increases exponentially as number of tasks, processors, and pipeline stages increases.

Therefore, we propose a SA-based scheduling method by which to obtain a near optimal solution. This method replaces the branch and bound algorithm used to search integer variables with simulated annealing. The algorithm is shown as pseudo-code in Figure 4. In this algorithm, SA searches a set of integer variables, which represents task allocation. Whenever SA generates a new allocation of tasks, the value of energy consumption is required in order to evaluate the objective value. In this situation, integer variables are replaced by the values obtained from task allocation. Then, voltage selection is evaluated by solving Linear Programming. Note that the voltage selection is no longer MILP because all of the integer variables are substituted by specific values in task allocation. This is the reason why the proposed SA-based solution is very fast.

A MILP solver with the branch and bound algorithm is used only to give the initial state for SA. The proposed method terminates the MILP solver once any feasible solution is found. Thus, the computational complexity becomes less problematic.

5. Evaluation

5.1. Assumptions

In order to evaluate the effectiveness of our SA-based scheduling method on the reduction of energy consumption, we herein perform a number of experiments.

We evaluate 16 randomly generated task graphs, each of which has eight tasks, and the values of $w(T_i)$ and $q(D_{ij})$ are $100 \sim 800$ Kcycle, $100 \sim 800$ Kbit, respectively. These graphs are generated so that $\sum_i w(T_i)$ is equal to 3Mcycle. These graphs are evaluated under two settings of the throughput constraint: $C_p = 3$ ms and $C_p = 4$ ms .

We also evaluate a task graph derived from the Rijndael cipher[8] encoder. The Rijndael encoding algorithm for $128-$bit blocks and $128-$bit keys is divided into 40 tasks. Values of $w(T_i)$ and $q(D_{ij})$ are estimated from the algorithm. Because the task graph of 40 tasks cannot be scheduled in a reasonable time, even in the SA-based method, part of the task graph that has 32 tasks is used for the experiment. We assume that the throughput constraint C_p is 4 μs.

We assume that the evaluated MP-SoC consists of three ARM11 processors for evaluating randomly generated tasks and four ARM11 processors for Rijndael encoding. Each processor consumes 450 pJ/cycle at 500 MHz [2] and is assumed to have

```
Input
  MILP formulas described in Section 3
Output
  $S_{best}$: task scheduling result (set of all scheduling variables)
{
  /* initialize SA */
  Start MILP solver, which uses branch and bound;
  Terminate MILP solver once any feasible solution $S$
    that satisfies all constraints of MILP is found;
  Let $E$ be the energy consumption on the scheduling $S$;
  Let $P$ be the set of integer variables in $S$;
  Let $T$ be the starting temperature, as described in [19];
  $E_{best} = E$; $P_{best} = P$; $S_{best} = S$
  /* start SA */
  while (terminating condition[19] is not satisfied) {
    /* monte carlo cycle at temperature $T$ */
    for (i=0;i<num_iter;i++) {
      Make a new variable set $P'$, which is in the vicinity of $P$;
      Make LP substituting variables in $P'$ into MILP;
      Solve LP to obtain scheduling result $S'$;
      Let $E'$ be the energy consumption on the scheduling $S'$;
      if ($E' < E$) {
        $E = E'$; $P = P'$; /* unconditionally accept */
        if ($E' < E_{best}$) {
          $E_{best} = E'$; $P_{best} = P'$; $S_{best} = S'$
        }
      } else {
        $k$ = random(between 0 and 1);
        if ($k < exp(E - E')/T$) {
          $E = E'$; $P = P'$;
        }
      }
    } /* end for */
    $T = r \times T$ /* lower the temperature */
  } /* end while */
} /* end scheduling */
```

Figure 4. SA-based scheduling method.

Table 2. Setting of frequency / voltage levels.

frequency [MHz]	500	400	300	200	100
energy [pJ/cycle]	450.0	349.2	261.5	186.3	123.8

five frequency/voltage levels, as shown in Table 2, which is derived from the linear approximation relationship between the frequency and voltage settings of a Pentium M processor[12]. We also assume that the energy consumption per cycle is quadratically proportional to the voltage.

The Shared communication link used in the evaluation is assumed to be based on AMBA AHB[1], which operates at 200 MHz and has a $32-$bit data bus. Its transferring speed and energy consumption are assumed to be 0.16 ns/bit and 0.5 pJ/bit, respectively[13].

Under the above assumption, task scheduling is performed using the proposed method, while varying the latency constraint. Note that the XPRESS Solver Engine[3] is used to solve the MILP.

5.2. Results

Figure 5 shows the energy consumption and time required to solve the scheduling problem for each method. Each figure corresponds to the case of $C_p = 3$ ms and $C_p = 4$ ms . The values of energy consumption are normalized to 1.0 in the case of non-pipelined ($L_p = 1$) scheduling using MILP. The values in the figures are the averages for the results of 16 task graphs.

Previously, using non-pipeline scheduling in which the latency was equal to the inverse of throughput, the energy consumption could only reach a level equal to the case of $L_p = 1$, regardless of the relationship between the latency and throughput constraints. On the other hand, the results shown in Figure 5 indicate that pipeline scheduling can reduce the energy consumption by adapting to the constraints. The proposed pipeline scheduling can be applied to the situation in which hL_p is not an integer (the latency constraint is not a multiple of C_p). Thus, it can be flexibly applied in various situations.

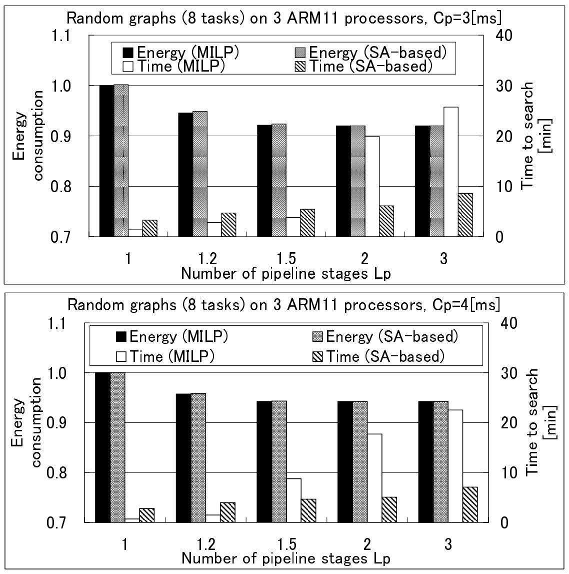

Figure 5. Experimental results using randomly generated task graphs.

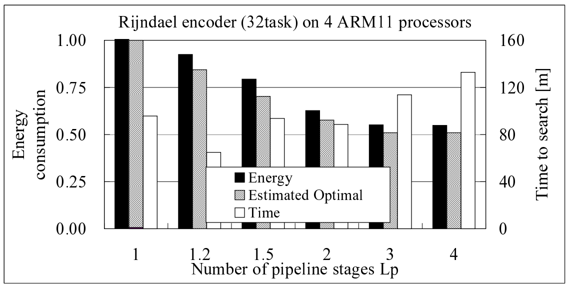

Figure 6. Experimental results using task graphs derived from the Rijndael encoder.

The results obtained using the proposed SA-based method shows almost the same effect of energy reduction as the optimal solution obtained by the MILP solver. In addition, the search time of the SA-based method does not increase dramatically compared to that of the MILP solver, even if the search space expands as the value of L_p increases. Thus, the SA-based method is expected to be more suitable for larger systems and applications.

Next, we discuss the results of the Rijndael tasks shown in Figure 6. Since the MILP approach requires a very long time to find a scheduling candidate, we estimated the energy consumption of the optimal solution from the average processor workload and the critical path length of the task graph. As shown in Figure 6, the proposed method can find a near-optimal solution for a larger task graph in real application.

Figure 7 shows average processor utilization of the schedule given by the proposed method. Here, utilization means the fraction

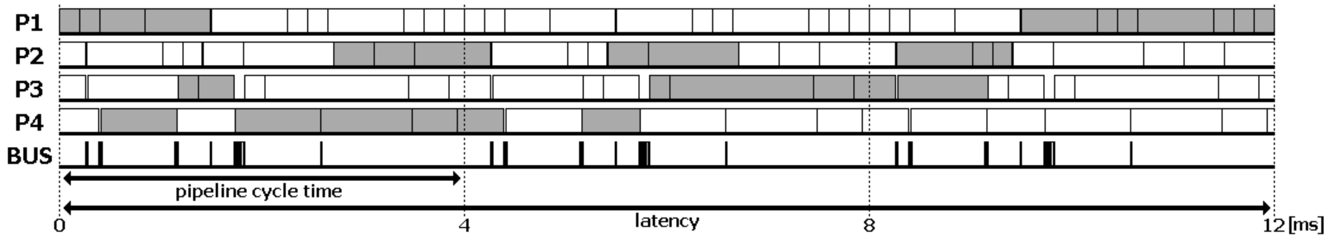

Figure 8. Scheduling result of the Rijndael encoder ($C_p = 4\,\mathrm{ms}$, $L_p = 3$).

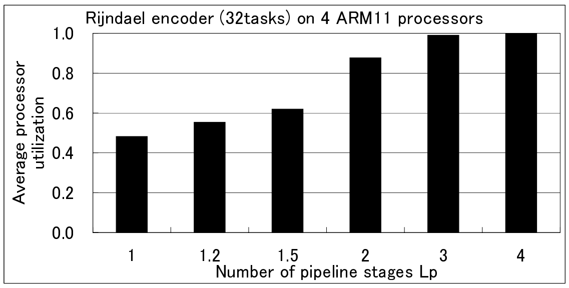

Figure 7. Processor utilization of the task schedule given by the proposed method.

of time which the processor is assigned to execute any task. Higher utilization means that the tasks take longer time to be executed by using lower frequency and voltage, which leads to lower energy consumption. It is shown in the figure that the pipelined task scheculing can utilize processors more efficiently than non-pipelined scheduling, because pipeline scheduling for large L_p can exploit the inter-job parallelism while non-pipelined scheduling can exploit only task-level parallelism.

A gantt chart of the scheduling result is shown in Figure 8. In this graph, each square represents a time slot assigned for a task, and the colored part shows time slots used for executing tasks in the same job. It shows that the processors are fully utilized for increasing chance for lowering frequency and voltage. Although applications with less task-level parallelism was thought to not be suitable for the MP-SoC platform, the proposed pipelined task scheduling enables such an application to be executed efficiently on the MP-SoC with abundant hardware resources.

6. Conclusion

In the present study, we proposed a pipelined task scheduling method of GALS MP-SoC to reduce the energy consumption under latency and throughput constraints. First, we expressed the energy optimization problem using MILP formulas. We then proposed a simulated annealing-based method to solve the MILP formulas very quickly. The proposed method was applied to synthetic task graphs and the Rijndael cipher encoder. The experimental results revealed that the proposed method can successfully derive task schedulings that consume approximately the same energy as the optimal schedulings. In addition, with respect to the Rijndael encoder, pipelined scheduling was revealed to be very effective on GALS MP-SoC when both the latency and throughput constraints are given. Based on these results, we concluded that the proposed method is very effective for reducing energy consumption on GALS MP-SoC.

References

[1] Amba home page. *http://www.arm.com/products/solutions/AMBAHomePage.html.*

[2] Arm11 family. *http://www.arm.com/products/CPUs/families/ARM11Family.html.*

[3] Xpress solver engine. *http://www.solver.com/xlsxpresseng.htm.*

[4] J. H. Anderson and S. K. Baruah. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. *Proc. of ICDCS*, 2004.

[5] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE J. of Solid-State Circuits*, Nov. 2000.

[6] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power cmos digital design. *IEEE J. of Solid-State Circuits*, Apr. 1992.

[7] J. J. Chen, H. R. Hsu, and T. W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. *Proc. of RTAS*, 2006.

[8] J. Daemen and V. Rijmen. The rijndael block cipher: Aes proposal. *http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf*, 1999.

[9] F. Gruian and K. Kuchcinski. Lenes: Task scheduling for low-energy systems using variable supply voltage processors. *Proc. of ASP-DAC*, 2001.

[10] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. *Proc. of DATE*, 2004.

[11] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. *Proc. of ISLPED*, 1998.

[12] K. Krewell. Pentium m hits the street. *Microprocessor Report*, Mar. 2003.

[13] K. Lahiri and A. Raghunathan. Power analysis of system-level on-chip communication architectures. *Proc. of CODES+ISSS*, 2004.

[14] L. F. Leung, C. Y. Tsui, and W. H. Ki. Minimizing energy consumption of multiple-processors-core systems with simultaneous task allocation, scheduling and voltage assignment. *Proc. of ASP-DAC*, 2004.

[15] J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. *Proc. of DAC*, 2001.

[16] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner. Globally-asynchronous locally-synchronous architectures to simplifu the design of on-chip systems. *Proc. of ASIC/SOC*, 1999.

[17] P. Palazzari, L. Baldini, and M. Coli. Synthesis of pipelined systems for the contemporaneous execution of periodic and aperiodic tasks with hard real-time constraints. *Proc. of IPDPS*, 2004.

[18] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. *Proc. of DATE*, 2006.

[19] S. M. Sait and H. Youssef. *Iterative Computer Algorithms With Applications in Engineering: Solving Combinatorial Optimization Problems*. IEEE Computer Society, 2000.

[20] G. Varatkar and R. Marculescu. Communication-aware task scheduling and voltage selection for total systems energy minimization. *Proc. of ICCAD*, 2003.

[21] Y. Zhang, X. Hu, and D. Chen. Task scheduling and voltage selection for energy minimization. *Proc. of DAC*, 2002.