System-Level Synthesis of Low-Power Hard Real-Time Systems

Abstract

The imminent convergence of computer, communications, and consumer electronic products and market implies a need for optimization intensive techniques for synthesis of low power hard real-time systems. In this paper, we present a system-level approach for power minimization under a set of user-specified costs and timing constraints of hard real-time designs. The approach simultaneously optimizes all three degrees of freedom for power minimization, namely switching activity, effective capacity and voltage supply.

We first define two key associated optimization problems, processor allocation and task assignment, and establish their computational complexity. Next, we describe a novel meta-algorithmics algorithm development strategy. The strategy is used for evaluation and statistical validation of the key parameters that guide the heuristic assignment process. The statistical analysis of comprehensive experimental results and their comparison with the developed conservative and optimistic sharp lower bounds, clearly indicates the quality of the proposed optimization techniques.

1 Introduction

The application and technological factors behind the imminent convergence of computer, communications, and consumer electronic products and market impose a new set of design goals and constraints for the synthesis process of application specific systems. The short time-to-market and cost sensitivity of consumer markets imply a need for optimization intensive system-level synthesis. In addition, multiplicity of standards and a need for flexibility and customization favor programmable implementation platforms. The popularity of the portable nature of future personal digital assistants and communication devices is the principal motive for low-power design technique development [1]. The research presented in this paper answers the outlined design requirements by introducing a system of optimization intensive synthesis techniques and tools for design of hard real-time application specific systems implemented on a set of programmable processors.

The synthesis approach encompasses two fully modular tasks: hardware allocation and task assignment. All three degrees of freedom for power minimization, namely switching activity, effective capacity and voltage supply are considered. While the first two degrees are described by processor's power efficiency for a given task, the last one depends on both the run time of a task on a particular processor and balanced loads on all processors. In order to minimize the communication cost tasks are assigned atomically to individual processors.

Resource allocation and task assignment for power efficient hard real-time systems are difficult problems with several layers of conceptual and computational complexity. For example, if we analyze the assignment problem, where a set of tasks is assigned to a set of processors, the power can be minimized either by executing tasks on power efficient processors or scaling the operational voltage. However, voltage scaling is closely related to the time efficiency of particular task-to- processor assignments and balanced minimized processor workloads. From the optimization point of view the key novelty is the use of statistical meta-algorithmic techniques for optimization of parameters that dictate the assignment algorithm. From statistical and result evaluation points of view, the key novelties are related to several new techniques for establishing the quality of heuristic algorithms.

The key trade-offs and ideas used to develop the synthesis algorithms can be introduced using the following motivational example. For the sake of brevity we restrict our attention to the assignment phase of the synthesis approach.

A set of tasks $\{T_i$, i=1..6$\}$ has to be assigned to a given hardware configuration so that the power consumption is minimized. Each task T_i requires a certain number of cycles $T_{i,j}$ to execute on a particular processing element $\{PE_j$, j=1..3$\}$. The $C_{i,j}$ data are shown on the left side of Table 1. Each processor PE_j consumes a specific amount of energy E_j per operation. Thus, the energy $PE_{i,j}$ required

CC-Power	PE_1[CC]	PE_2[CC]	PE_3[CC]	PE_1[mJ]-200[μJ/CC]	PE_2[mJ]-20[μJ/CC]	PE_3[mJ]-5[μJ/CC]
T_1	100	250	900	20	5	4.5
T_2	120	250	800	24	5	4
T_3	140	300	800	28	6	4
T_4	140	200	400	28	4	2
T_5	140	400	500	28	8	2.5
T_6	130	800	800	26	16	4

Table 1: Task cycle and energy efficiencies on the allocated hardware.

for one execution of each task T_i can be computed as $P_{i,j}{=}PE_j \cdot C_{i,j}$. The right side of the Table 1. provides the data on the task execution energies $P_{i,j}$. The tasks are independent and have to be executed exactly once in each iteration. The iteration is 21μs long. A voltage supply of 3.3V is assumed. Finally, it is assumed that all processors, when driven by 3.3V power supply, can support a clock cycle delay of 5 ns, implying 200MHz operational frequency.

Power can be optimized by assigning tasks to processors which consume the least amount of power for their execution and by scaling the voltage of the power supply. The voltage is scaled with respect to the maximal cycle time required by the hardware configuration and its workload. Maximally scaled voltage corresponds to the cycle time allowed by the delay vs. voltage digital CMOS dependency.

Obviously, one attractive option for task assignment is to assign each task to a processor which consumes the least amount of power for its completion. In that case, all tasks have to be assigned to processor PE_3. The required number of cycles is equal to

$$T_{1,3} + T_{2,3} + T_{3,3} + T_{4,3} + T_{5,3} + T_{6,3} = 4200 \text{ cycles.}$$

Hence, the minimal frequency at which the circuit has to be clocked is $\frac{4200}{21\mu s} = 200MHz$. The power consumption of the configuration is equal to

$$P_{1,3} + P_{2,3} + P_{3,3} + P_{4,3} + P_{5,3} + P_{6,3} = 21\text{mJ.}$$

The second alternative is to assign each task to a processor that executes it in the least number of cycles. In which case, all the tasks will be assigned to processor PE_1. The required number of cycles in this cycle-greedy assignment is

$$T_{1,1} + T_{2,1} + T_{3,1} + T_{4,1} + T_{5,1} + T_{6,1} = 770 \text{ cycles.}$$

The minimal frequency at which the circuit can operate is equal to $\frac{770}{21\mu s} = 36.7MHz$. Lower operational frequency of the used hardware enables us to relax the cycle time $\frac{200MHz}{36.7MHz} = 5.45$ times.

Now, due to the increased cycle time, the voltage can be scaled down to 1.46V. The total power consumption is equal to

$$(P_{1,1} + P_{2,1} + P_{3,1} + P_{4,1} + P_{5,1} + P_{6,1})\frac{V_1^2}{V_0^2}{=}154\text{mJ}\frac{1.46^2V^2}{3.3^2V^2}{=}30\text{mJ}.$$

Although each task in the previous assignment is executed in the processor that requires the minimal number of cycles for its completion, the operational voltage can be reduced further by balancing the workloads on all processors. It is important to notice that the previous two optimization approaches can be executed in linear time, while the problem of balancing the workloads on all processors is proven to be NP-hard. Exhaustive search has shown that the following task assignment achieves the minimal number of cycles when T_1,T_2,T_5,T_6 on P_1; T_3 on P_2; and T_4 on P_3;

In that case the minimal number of cycles per iteration required by the allocated hardware and balanced workload is $T_{1,1} + T_{2,1} + T_{3,2} + T_{4,3} + T_{5,1} + T_{6,1} = 490$ cycles. Now, the cycle time of the system's clock can be increased $\frac{200MHz}{\frac{490}{21\mu s}} = 8.57$ times and therefore, the voltage of the power supply can be reduced from 3.3V to 1.29V. The power consumption is equal to

$$(P_{1,1} + P_{2,1} + P_{3,2} + P_{4,3} + P_{5,1} + P_{6,1})\frac{V_1^2}{V_0^2}{=}106\text{mJ}\frac{1.29^2V^2}{3.3^2V^2}{=}16\text{mJ}.$$

Although the balance of workloads, i.e. maximally lowered operational voltage, appears to be the most important criteria considered for power optimization, proper analysis shows that the best solution can be obtained when the advantages of the previous three optimization criteria are combined. For example, consider the following assignment:T_1,T_2,T_3,T_4 on P_2; T_5, T_6 on P_3;

This requires 1300 cycles. The cycle time can be increased $\frac{200MHz}{\frac{1300}{21\mu s}} = 3.23$ times and the corresponding voltage for such cycle time is 1.70V. Thus, the total power consumption is

$$(P_{1,2} + P_{2,2} + P_{3,2} + P_{4,2} + P_{5,3} + P_{6,3})\frac{V_1^2}{V_0^2}{=}26.5\text{mJ}\frac{1.70^2V^2}{3.3^2V^2}{=}7\text{mJ}.$$

This is an improvement of 130%, 333% and 200% in comparison to the balanced, cycle-greedy and power-greedy solution respectively. The motivational example shows that high energy savings can be obtained even in small examples when optimization is treated as a combination rather than selection of the stated power optimization concepts. The solution to the motivational example is acquired using heuristics described in Section 7.

The rest of this paper is organized in the following way. In the next section we present related research. Section 3. outlines the preliminary material, including computational, task and power consumption models used for the research presented in this paper. Analysis of key design trade-offs and

Technique	Cycle-Greedy	Power-Greedy	Balance-Greedy	Power-Optimal
MaxFrequency	36.7MHz	200MHz	23.4MHz	62MHz
Voltage	1.46V	3.3V	1.29V	1.70V
Cycles	770	4200	490	1300
Energy (3.3V)	154mJ	21mJ	106mJ	265mJ
Scaled Energy	30.3mJ	21mJ	16.3mJ	7mJ
Ratio	reference point	44.3%	85.9%	333%

Table 2: Comparison of greedy optimization techniques to the optimal solution.

overall design flow are presented in Section 4. Two synthesis optimization problems, hardware allocation and assignment, are defined and their complexity is established in Section 5. Synthesis algorithms are described in detail in Section 6. and 7. Sections 8. and 9. contain a report on conducted experiments, and details evaluation and conclusion respectively.

2 Related Work

Directly related research efforts are ones conducted in system level synthesis, synthesis and compilation low power approaches, hard real-time scheduling and synthesis, and related optimization techniques. System level synthesis, including hardware-software codesign and partitioning, are premier design and CAD research topics [19], [2], [3], [8]. The most relevant system research subdomain is hardware-software partitioning, where a great variety of techniques are proposed [6], [7], [10]. Wolfe discussed the dependence of design freedom and key constraints in low-power system-level CAD [20].

A number of synthesis and compilation techniques for power optimization at all levels of abstractions during the design process have been proposed [16]. Although power optimization is often most effective when applied early in the design process at the algorithmic and architectural level, majority of power minimization techniques have been proposed for logic synthesis and physical design [16]. Potkonjak and Rabaey presented a CAD system-level synthesis approach which targets low-power [15]. However, the scope of this effort is restricted to a single task application specific system. Several system level optimizations have been proposed in [14].

Hard-real time scheduling efforts have been an important topic for research since early computing days. The initial research on scheduling a set of periodic tasks with strict timing constraints on periodicity, arrival and required time of each task, resulted in the popular rate-monotonic scheduling algorithm [13]. In the seventies and eighties, the rate-monotonic scheduling algorithm was being generalized in several directions by relaxing some of the initial constraints on the timing discipline [12]. A survey of the key results in hard real-time scheduling and assignment is given in [17].

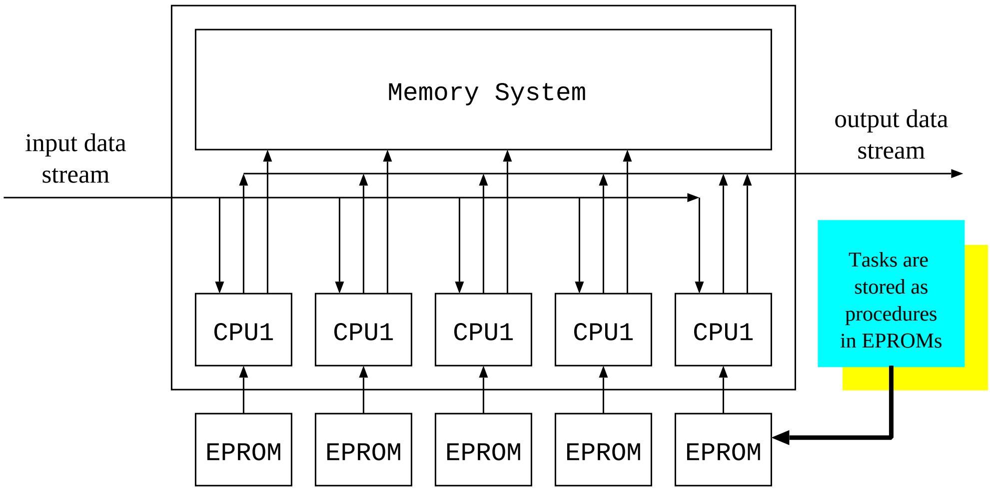

Figure 1: High-level architectural template.

3 Preliminaries and Background Material

We assume that all tasks are periodic and defined as one-way infinite streams of data. Each task has its own period of appearance. This computational model is the standard abstraction for many video, digital signal processing, communication, and multimedia applications.

In the period equal to the Least Common Multiple (LCM) of all periods of all tasks, all instances of a single task have to be executed. However, their execution has to satisfy the deadline that is set by the LCM of all task periods. Tasks are independent which means that there is no cost for intertask communication. The assumption of task independence is actually done with no loss of generality whatsoever, because atomic independent tasks can be treated as a single composite task.

The targeted architecture template is shown in Figure 1. The system contains N programmable processors (CPUs). Each CPU has a reserved segment of data memory. The programs for the execution of tasks are stored in EPROMs. All processors are subject to the common voltage supply source.

In CMOS digital circuits the dynamic power consumption is given by the following formula

$$Energy \approx SwitchingActivity \cdot C_{eff} \cdot V_{dd}^2$$

Since the power consumption is proportional to the square of the voltage supply, its lowering is often the main concern in every power optimization technique. However, lowering the supply voltage causes increased delay of CMOS circuits according to the following formula [1]:

$$Delay = K \cdot \frac{V_{dd}}{(V_{dd}-V_t)^2}$$

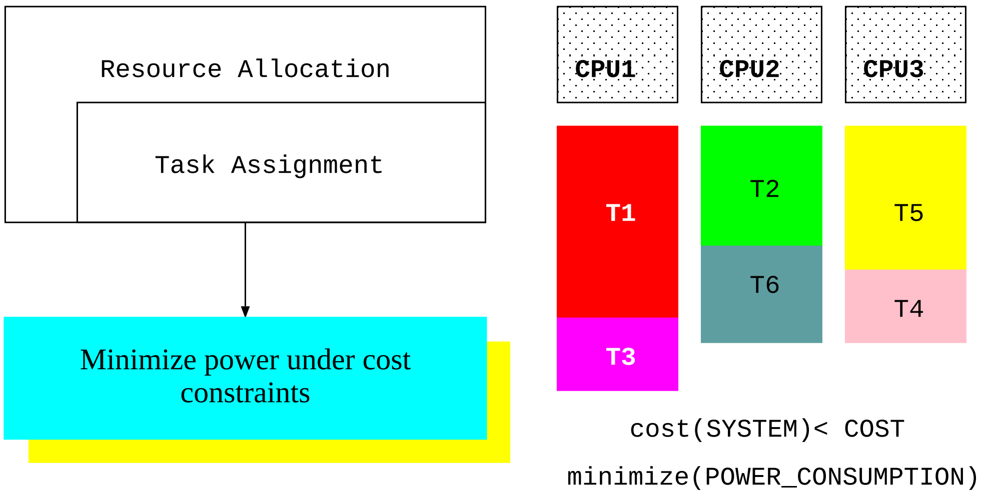

Figure 2: The novel synthesis approach.

where K and V_t are constants entirely dependent upon the implementation technology. This phenomena plays a great role in our optimization technique.

When assigning tasks to the allocated hardware we minimize and balance the workloads on all processors. The deadline, set by the periodic nature of the incoming data stream, imposes a boundary on the delay of a single clock cycle of the allocated resources

$$Deadline > Cycles \cdot Delay$$

and therefore, limits system designer's ability to lower the voltage of the power supply.

The other two parameters that impact power consumption, the effective capacity and switching activity, are optimized implicitly in two phases of our synthesis approach. First, for a given set of tasks we select hardware that has the least effective capacity and switching activity for the execution of the entire set of tasks. Then, during the task assignment phase, the processor of a minimal effective capacity and switching activity is given a certain preference.

4 The System-Level Synthesis Approach: Global Design Flow

The global design flow of the synthesis algorithm is given in Figure 2. The synthesis procedure starts with the resource allocation step. The allocation algorithm searches for the processor configuration which consumes the least power for a given set of tasks under a given cost constraint of the system. The allocation algorithm is based on a multistart gradient search. For the evaluation of the proposed solution, the allocation algorithm invokes the task assignment procedure. Once the allocation is completed, the final assignment starts. The assignment heuristic algorithm is developed by a new statistical meta-algorithm methodology explained in Section 7. The assignment procedure iteratively assigns one task at a time. The decision of selecting tasks and target processors is made globally; each time a task is considered for assignment, its impact on further assignment of remaining tasks is encountered.

The task is selected upon its relative size, and the most efficient power and cycle requirements for execution. Processor selection depends upon the processors' current workloads, and power and cycle efficiencies for the execution of the selected task. Once all tasks are assigned, the voltage of the power supply is scaled as dictated by the overall timing deadline and maximum cycle workload. Detailed explanations of the developed algorithms for each synthesis step are given in Sections 6. and 7.

5 Optimization Problems: Formulations and Computational Complexities

We now formulate the optimization problems associated with system- level synthesis of low power hard real-time multitask designs and establish their computational complexity. Our synthesis approach has two optimization intensive phases: allocation and assignment. The targeted synthesis subproblems can be defined in the following more formal way using the standard Garey-Johnson [5] format.

Problem: Allocation for synthesis of low power hard real-time application specific systems.

Instance: A set of K processors and a set of N independent periodic hard real-time tasks are given. Each processor has an associated cost and power consumption per cycle. Each task has its period and execution time for assignment to any of the processors. Two positive numbers C and P.

Question: Select a multisubset of processors (subset where some processors can be included more than once) such that each task is assigned to exactly one processor, the sum of costs of selected processors is at most C, and the power consumed by all tasks is at most P.

Problem: System-level Assignment of Hard Real-Tasks.

Instance: A set of K processors and a set of N independent periodic hard real- time tasks are given. Each task has its period and execution time (number of cycles) for each of the available processors.

Question: Assign each task to one of the processors in such a way that all tasks can be scheduled within their timing constraints and that the power consumed by all tasks is at most P.

The allocation and assignment subproblems of the problem of system-level synthesis for low power hard real-time multitask designs are NP-complete. The proof is based on the classical Karp's polynomial time reduction technique and the equal-subset problem as the starting point [5].

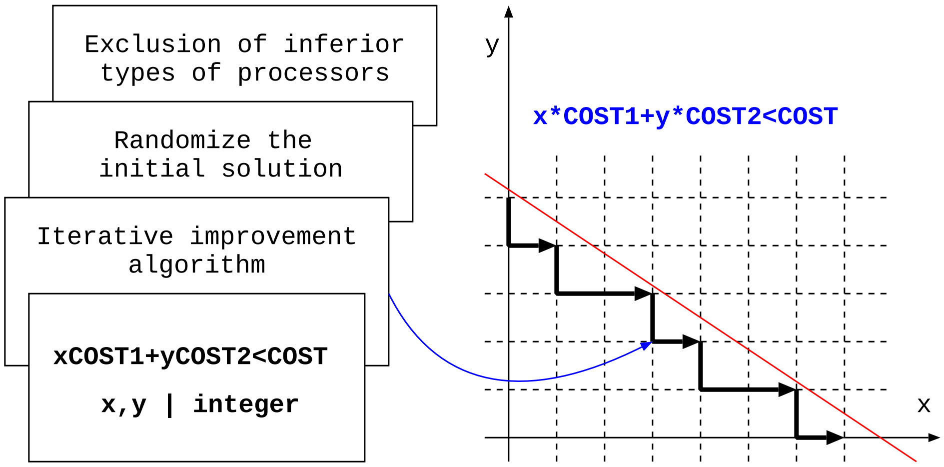

Figure 3: Resource allocation algorithm (example for two types of processors)

6 Resource Allocation

The first synthesis step is resource allocation. The goal is to select a multiset from the set of available processors, so that the power consumption of the hard real-time system is minimized under the maximum cost constraint. The inputs to the resource allocation program are costs and power consumptions per cycle of all available processing units, denoted by C_i and PE_i respectively, and the total user specified system cost C. Also, the information about the execution time of each task on each type of processor is available. This information can be obtained either by executing tasks on a given processor or by using simulation and estimation techniques. Since the resource allocation is computationally intractable problem, we opted to use heuristic search which provides a favorable run-time vs. quality trade-off. The synthesis algorithm starts with a preprocessing step, which eliminates all types of processors dominated by other types of processors. Processor PE_x is inferior to processor type PE_y, if PE_x is more expensive, performs all tasks in more clock cycles and requires more energy for the execution of each task. The minimum set of non- inferior processors is selected using the standard 3-D dominance algorithm [4].

Once all inferior processing types are excluded from the hardware menu, the heuristic that searches for the most power-efficient configuration starts. The heuristic defines a random hardware configuration HC_0 which is used as a starting point for an iterative improvement heuristic. This configuration is constructed in such a way that it is not possible to add a single processor of any type to HC_0 so that it stays within the range of the system cost C. The initial configuration is established by randomly adding processing units, as long as the cost of the system is lower than C.

The iterative heuristic at each step tries all the options for addition or removal of any of the consid-

```
Remove all inferior processors from the hardware menu
While (number of attempts ¡ K)
   Repeat
      Generate initial random feasible solution
      For all types of processors add or delete a processor from
      the current configuration HC_i, and calculate the OPC as well
      as the objective function OF_i = OPT_i + αMC_i , where MC_i is
      the marginal cost of HC_i.
      If C_i > C then MC_i = 0 else MC_i = C_i − C.
      Depending on the values of the objective function for all
      types of processors, delete or add a processor of the winning
      type and action.
      Increase α .
   Until there is improvement and the cost of the configuration
   satisfies the system requirements.
```

Figure 4: Resource allocation pseudo-code.

ered types of resources. The heuristic is gradient-driven, i.e. the configuration is upgraded by following the most beneficial action. The objective function is a weighted sum of the optimized power consumed (OPC) by the current configuration and the marginal cost of the system. The power is calculated by invoking a simplified task assignment procedure. If the difference is positive, the marginal cost of the system is defined as the excess of the cost of the current configuration over the user specified system cost C. If the configuration cost satisfies the cost requirement, the marginal cost is equal to zero. The weighted sum of the two components is time dependent. As the optimization process progresses, the importance of the marginal cost over power increases, since it is more important to produce feasible solutions. Thus, at the beginning of the heuristic search, we impose relatively non-strict cost enforcement, while at the end, we strongly favor solutions that satisfy the cost constraint.

Finally, when no further improvements cannot be obtained by adding or deleting a processor of any type, we are likely to be stuck in one of the local minima of the space of all possible solutions. Therefore, we terminate our search and memorize the power- optimal hardware configuration. The initial random-constructive and iterative improvement heuristic are repeated k times in order to find a local minimum as close as possible to the global minimum. An alternative is the automatic option, used in the experiments reported in this paper, where the core of the allocation procedure was repeated as long as no improvement was measured in last three instances of the allocation procedure.

The resource allocation procedure can be summarized by using the pseudo-code on Figure 4.

```
While (there is at least one non-assigned task)
   T = Select Task()
   P = Select Processor()
   Assign (T,P)
```

Figure 5: Task assignment pseudo-code. [m denotes the number of types of processors for which more than one processor is selected in the allocation phase].

7 Task Assignment

Whenever the allocation algorithm generates a hardware configuration it invokes a simplified task assignment heuristic in order to estimate the power consumption of the hardware configuration. The task assignment procedure has three components organized according to the pseudo-code on Figure 6.

Therefore, the algorithm iteratively selects a task for assignment and then assigns the selected task to the allocated hardware. The post-processing step aims at additional load balancing. The three key steps of the algorithm are described in the remainder of this section.

7.1 Task Selection

The first step during each iteration of the assignment procedure is to select a task which will be assigned to one of the processors. The selection is conducted using the following objective function:

$$Select_{task_i} = \alpha_1(T_{2,i} - T_{1,i}) + \alpha_2(P_{2,i} - P_{1,i}) + \alpha_3 \frac{T_{2,i}+T_{1,i}}{\sum_{j=1}^{tasks}(T_{2,j}+T_{1,j})} + \alpha_4 + \alpha_5 + \alpha_6$$

where $T_{1,i}$ denotes the minimal number of cycles required to execute the task T_i, $T_{2,i}$ denotes the second minimal number of cycles required to execute the task T_i, $P_{1,i}$ is the most power-efficient processor for executing T_i, and $P_{2,i}$ is the second most power- efficient processor for executing T_i.

The selection function components represent the key trade-offs among the considered mechanisms for power minimization while the weight parameters dictate their relative importance. The first component of the objective function favors assigning a task to a processor where the task has minimal execution time. Since short execution times enable voltage scaling, the difference $T_{2,i} - T_{1,i}$ encountered in the objective function includes the lower bound for not following the most run-time efficient option. Similarly, the second component favors assigning a task to a processor which consumes the minimal amount of energy at the nominal voltage. The difference $P_{2,i} - P_{1,i}$ captures the minimal penalty for not following the most power-efficient choice. Long tasks often cause workload balance problems if not scheduled early. A task T_i is termed long compared with the length of all other tasks if

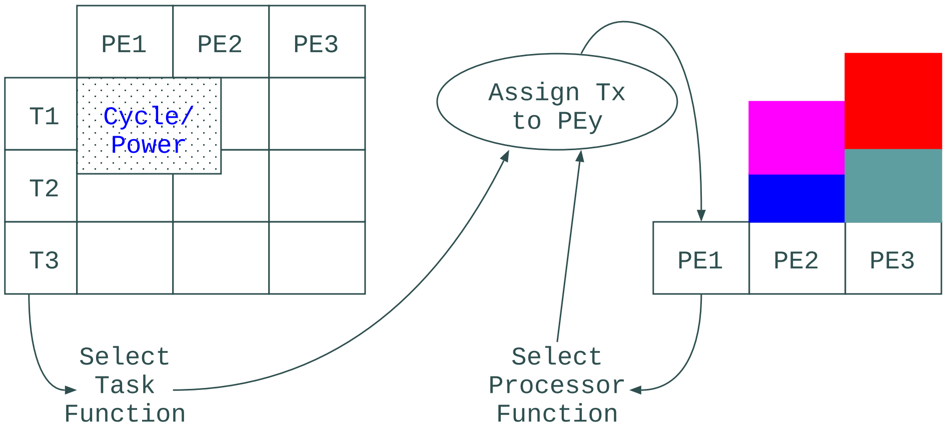

Figure 6: Task assignment algorithm.

$$\alpha_3 \frac{T_{2,i}+T_{1,i}}{\sum_{j=1}^{tasks}(T_{2,j}+T_{1,j})}$$

is much greater than the same expression calculated for all other tasks. Parameters $\alpha_4, \alpha_5, \alpha_6$ are values added to the selection function if the most power-efficient and the most cycle-efficient processor are the same (α_4), if the most power- efficient processor is the same as the second most cycle-efficient processor or if the most cycle-efficient processor is the same as the second most power-efficient processor (α_5), or if the second most power- and cycle-efficient processors are the same (α_6). Those three parameters capture the relative difficulty of making an optimal assignment choice for the task. For example, it is relatively easy to make a decision where to assign the task if both the execution run-time and power requirements favor the same processor. However, if those two parameters point to two different processors, the decision process is more difficult. The task that has the highest value of its selection function is the one selected for assignment.

7.2 Processor Selection

Processor selection is based on the execution characteristics for the selected task on each processor. The selection function used in our approach is

$$Select_{processor_j,task_i} = \beta_1 T_{1,processor_j,task_i} + \beta_2 P_{1,processor_j,task_i} + \beta_3 workload_{offset}(processor_j)$$

where $workload_{offset}(processor_j)$ is equal to zero, if the workload of $processor_j$ does not exceed the current maximum workload in the system. Otherwise it is equal to the difference of the current maximum workload and the workload after the task is assigned to this processor.

As in the task selection function, here we also deal with several trade-offs. The selected processor is the one with the smallest value of the selection objective function. The first and the second component favor the processor that performs the selected task in the most time- and power efficient way respectively. The third component facilitates voltage scaling by load balancing. The quality of the final solution is a function of parameters α and β. In the next Section we describe how parameters for this function are obtained by using meta-algorithmics and statistical techniques.

The assignment algorithm is illustrated by using the motivational example from Section 1. First, all the tasks are sorted by using the meta-algorithmic task selection function. Then, the heuristic assigns each task to a processor that has the minimal objective function value for its selection function at that particular moment in the assignment process. The meta-algorithmic set of parameters of the objective function corresponds to the following set of values: α_1=0.520656 α_2=0.381958 α_3=0.0431519 α_4=0.160583 α_5=0.522339 α_6=0.696564 β_1=0.0970764 β_2=0.400818 β_3=0.773407.

Task	Select	T_2 - T_1	P_2 - P_1	T_1 + T_2	sequence
T_1	234.056	78.0984	133.685	21.576	(6)
T_2	252.858	67.6853	141.324	43.1519	(4)
T_3	338.367	83.305	168.062	86.3038	(3)
T_4	248.105	31.2394	129.866	86.3038	(5)
T_5	579.66	135.371	206.257	237.335	(2)
T_6	1222.58	348.84	355.221	517.823	(1)

From the table we can see that tasks that perform significantly better on one processor in comparison with the other choices tend to have relatively high values for their selection functions. Those values result in their early assignment. In particular, task T_6 is assigned to processor PE_3. After each task assignment the processor selection function is recalculated.

Task	PE_3	PE_2	PE_1	assigned-to
T_6	**1327.69**	2492.6	2676.64	PE_3
T_5	**829.803**	936.938	2774.2	PE_3
T_3	1327.69	**702.704**	2774.25	PE_2
T_2	1327.69	**585.587**	2377.93	PE_2
T_4	663.843	**468.469**	2774.25	PE_2
T_1	1493.65	**585.587**	1981.6	PE_2

7.3 Meta-Algorithms

In the previous subsection, we discussed the key trade-offs and optimization mechanisms which impact the decision process and the quality of the final design produced by the iterative task selection and assignment process. Common engineering strategy is to start with ad-hoc values for the parameters using insights obtained on small manually done examples. In the second phase, a majority of traditional

```
Set all parameters P_i to random values.
For each parameter P_i
   Calculate TPC_i^+ when P_i = P_i · (1 + w)
   and the TPC_i^- when P_i = P_i · (1 − w).
Let TPC_k^@ be the minimal TPC, where @ is
the sign of the variation of P_k.
Set P_k to P_k · (1@w).
Repeat
   Set P_k to P_k · (1@w).
until TPC of the current set of parameters is greater than the TPC of the
previous set of parameters.
Memorize the previous three values of P_k, P_k1, P_k2 and the last P_k3.
Repeat
   Calculate P_k as the "golden ratio" P_kg between values P_k1 and P_k3.
   If the TPC_kg^- is greater than the TPC_k2 then P_k3 = P_kg.
   else P_k1 = P_k2 and P_k2 = P_kg.
until TPC_kg^- − TPC_k2 > ε .
```

approaches usually use the feedback from executed benchmarks to tune the values of the parameters and therefore to determine their relative importance.

However, the traditional approaches usually leave a lot to be desired. First, they most often do not provide any formal, statistical, or even intuitive guarantees of the optimality of the achieved result. There is not even any guarantees that all of the proposed and used parameters are even relevant for the optimization and synthesis process. Both theoretical and practical dangers of overtuning the optimization algorithm to the available benchmark are high. Finally, the tuning process is time consuming, there is no guarantee of convergence, and its formal or statistical analysis is impossible.

In this subsection we outline a new meta-algorithmic approach for statistical parameter determination (evaluation) and validation and demonstrate its application on our assignment procedure for system level synthesis of hard real-time tasks. The key steps of the new design methodology for CAD algorithm development are described using the following pseudo-code.

```
Meta-Algorithmic Procedure(){
generate_learning_examples();
generate_optimization_mechanism();
parameter_evaluation();}
```

As a first step to this strategy, we generated sets of statistically identical benchmark examples using the set of available real-life examples [See Section 8]. We prepared a set of K statistically random sets

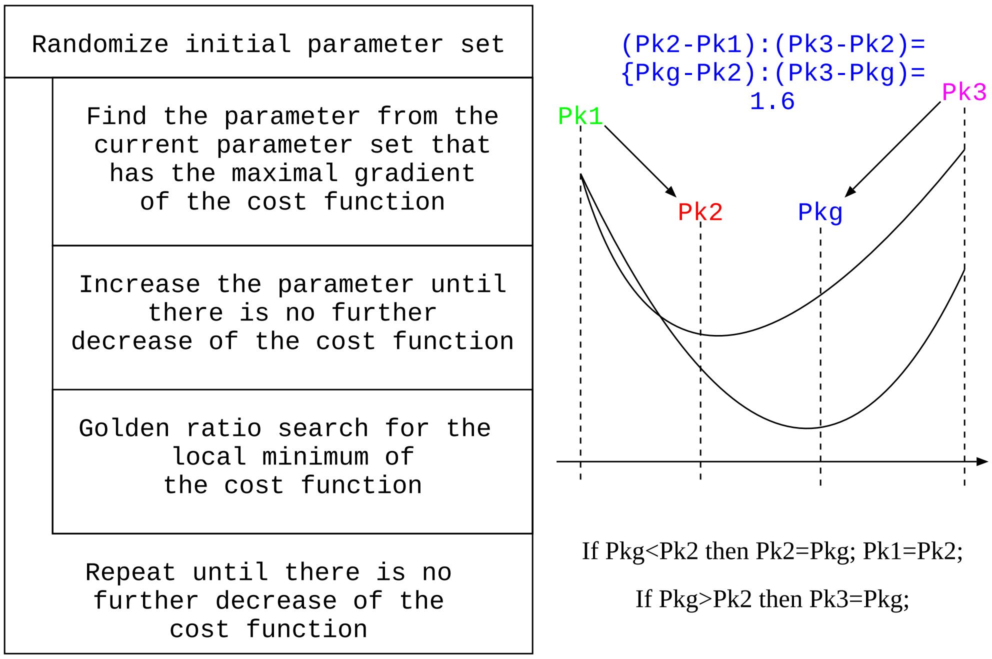

Figure 7: Evaluation of parameters for the meta-algorithm and golden ratio search.

of M tasks and N different hardware options. For the results presented in this and next section, we used the following set of values: K = 150, M = 20 - 320, and N = 5.

The second step, the generation of optimization mechanisms is explained in Section 7. The core of the meta-algorithmics approach is the parameter evaluation procedure given in Figure 7.

The core procedure performs multistart gradient search. One iteration of the program for parameter evaluation first sets the parameters to random values. Then it checks the total power consumptions (TPC) when each parameter is positively and negatively offset by k% of its value. In our experimentations we used k = 2. Other parameters in range {0.01-10} produced very similar results. The parameter that has the smallest TPC is then offset to the value that caused this improvement, following the deepest descent paradigm. Then the procedure continues exponentially increasing or decreasing the selected parameter using the golden ratio search approach [18]. The one iteration of the golden ratio search is terminated once the improvement is smaller than a user specified value ϵ (we used value 0.1%).

Once the golden ratio search is completed, the list of the parameters is updated to this new value and the procedure is iteratively repeated as long as local minimum is not reached using our stopping criteria. Then we start a search for the minimum value of TPC by adjusting the parameter P_k. The one iteration of the search is terminated once when no improvement along directions indicated by any of the parameters is not detected.

The intuition behind the evaluated parameters is the following: the most important is to early select a task for which there is a cleanly preferable assignment as indicated by parameters $\alpha_1, ..., \alpha_6$. During the assignment it is relatively important to follow balancing, unless there are high timing and power advantages. The evaluated parameters were used in the next section to conduct experiments that proved our assumptions about the efficiency of our synthesis paradigm.

8 Experimental Results

In this section we describe our benchmark examples, experimental results evaluation strategy, and present and analyze the experimental results. Our goal is to evaluate the effectiveness of proposed algorithms on task mixes that approximate the expected task mixes of application specific space program computers, wireless portable personal communicators, and module application specific servers. Initially we assembled 40 typical DSP, video, and communication tasks. The tasks include the low pass 5th order elliptical wave digital filter, Winograd Convolution, several Winograd's FFTs, 1D and 2D DCT algorithms, VHS transform, NTSC coder and decoder, an LMS adaptive filter, Hilbert Transform, 2nd order Volterra filter, 3rd order truncated Volterra filter, 28th, 50th, 100th, 126th and 133rd order FIR filters, PCM DA converter, image histogram, NEC digital to analog converter, several modems, and several error-correction codes such as Goppa, Fire, Hamming and BCH code. These set of tasks provide a solid approximation of expected loads for emerging DSP, video, and broadband servers.

The execution times of tasks on a variety of processors are either obtained through direct measurements on the available hardware platforms (e.g. SUN Sparcstation) or by using estimation formulas developed in [9] for a number of DSP processors. The power consumption data is obtained by combining data from manufacturers product descriptions and the well known fact that power consumption of an arbitrary task on majority of modern processors is linearly proportional to its execution time [11]. The assembled mix of 40 tasks was used as test benchmarks for two smallest design instances reported in Table 3. In order to test the performances of the proposed algorithms on large instances of input data, we used the following statistical technique to generate synthetic mixes of tasks with identical statistical properties as the original test set.

First, histograms of run-times for each processor and all available tasks are constructed. The run times of additional tasks are generated using a random number generator that follows the distribution dictated by the histogram. All statistically generated examples are divided into two groups: learning and testing benchmarks. The learning examples are used to evaluate parameters of the assignment heuristics, the testing examples along with the original set are used for evaluation purposes.

In order to properly evaluate our synthesis approach we developed the following two-phase approach. In the first phase the assignment algorithm is evaluated. First, we defined a lower bound solution. Then, we recorded the best results achieved for any proposed set of parameters during the application of the

Task(cpu)	Lower-bound	MetaBound	Random(best)	Random(med)	Our Algorithm	Our Alg. vs. Random
20(5)	217	247	285	452	255	1.0585
40(5)	480	533	635	862	565	1.1239
40(10)	390	616	842	1431	632	1.3323
80(5)	901	997	1186	1464	1030	1.1515
80(10)	767	1130	1591	2262	1155	1.3775
80(20)	698	925	1779	3038	943	1.8865
160(5)	1816	2004	2384	2818	2067	1.1534
160(10)	1543	2255	3096	3966	2296	1.3484
160(20)	1449	1827	3270	4768	1839	1.7781
160(40)	1399	1684	4160	7137	1736	2.3963
320(5)	3629	3984	4833	5359	4110	1.1759
320(10)	3038	4389	6009	7141	4414	1.3614
320(20)	2998	3774	6297	8090	3807	1.6541
320(40)	2828	3311	7259	10746	3388	2.1426
320(80)	3156	3746	12838	20319	3783	3.3936

Table 3: Power consumptions for 15 classes of sets of tasks (assignment; resources allocated).

meta-algorithmic technique. We also generated a large set of random solutions and recorded the best and median random solutions. Finally, we applied the new assignment algorithm on the test examples.

The provably conservative lower bound used for the evaluation of our synthesis algorithms is developed in the following way. For each task the minimal amount of energy required to execute the task on any of the available processors is selected and added up to the overall power consumption of the entire set of tasks. The resulting power consumption is further reduced by scaling the voltage of the power supply according to timing requirements of the fastest perfectly balanced solution. In order to obtain a perfectly balanced solution, for each task the minimal number of cycles required to execute the task is selected, added up and divided by the total number of processors in the configuration.

Note that meta-algorithmic approach by itself can be considered as an assignment algorithm. While the search for optimal assignment parameters progresses, the meta- algorithmics generates numerous assignments. We run the meta-algorithmics approach on all test examples and record the best solutions (resulting in minimal power consumption) regardless of the used parameters. The main disadvantage of the solution developed by the meta-algorithmics itself is the execution time of the program.

In the second phase the allocation algorithm is evaluated. To obtain the lower bound solution we relax several constraints imposed by the proper integral solution requirement: we assume that fractional number of processors can be allocated, that best power and timing characteristics of processors can be combined, and that perfect balancing is achievable. These approximations transform the problem which has several layers of NP-hard subproblems into one which can be solved rapidly in polynomial time. This is achieved in the following way.

Task(cpu)	Lower-bound	Random(best)	Random(med)	Our Algorithm	Our Algorithm vs. Random(best)
20($_1$)	148	442	737	255	1.73307
40($_1$)	327	1263	2077	565	2.23557
40($_2$)	270	3602	6120	632	5.69143
80($_1$)	589	1799	3024	1030	1.74529
80($_2$)	510	6573	11710	1155	5.68698
80($_3$)	476	12488	21535	943	13.2289
160($_1$)	1268	4452	7813	2067	2.15394
160($_2$)	1012	12939	21589	2296	5.63557
160($_3$)	970	18720	31887	1839	10.1773
160($_4$)	942	37940	66339	1736	21.8469
320($_1$)	2574	9524	16785	4110	2.31734
320($_2$)	2102	25331	43416	4414	5.73777
320($_3$)	2016	26581	43834	3807	6.98174
320($_4$)	1986	46620	78965	3388	13.7595
320($_5$)	2249	188588	311387	3783	49.8481

Table 4: Power consumptions for 15 sets of tasks (allocation and assignment).

First, for each task the most time and power efficient processors are identified. For each task the nominal power consumption at the initial operational voltage is accepted as the most power efficient option. Next, the cost of cycle per dollar for each task on its most time efficient processor is calculated. This cost is multiplied by the ratio of the number of cycles vs. the iteration period for each task, and added up for all tasks. By dividing the total specified cost by this sum we obtain the factor by which the cycle time can be prolonged as well as the bound on how much we can reduce the supply voltage.

Similarly to the case of assignment evaluation, we generated the best and median random solution by using lower bound assignments in 300 random allocation runs. The relative performance of the new allocation algorithm versus the described three comparison options are given in Table 5.

The experimental results for the task assignment heuristic are shown in Table 3. The table shows power consumption for 15 different classes of sets of tasks and processors. Each class is represented by 10 different sets of constant number of tasks and processors. The first column provides information about the number of tasks and processors. The first number in the first column corresponds to the number of tasks followed by the corresponding number of processors. The next five columns provide power consumption data per iteration for lower bound, meta-algorithmics bound, best and median random in 1,500 runs, and the proposed new approach respectively. From Table 3. it can be noticed that in average our synthesis approach performed close to the described lower-bound (29%) and the meta-algorithmics assignment parameter evaluation (2.4%).

Typical execution times for the meta-algorithmics solution and the new assignment algorithm are given in Table 5. Table 4. provides information about the effectiveness of the allocation algorithm.

Number of tasks	Number of processors	Execution time
150 sets of tasks (20 - 320) [parameter evaluation]	5 configurations (5 - 80)	3184:41 min:sec
320	80 cpus (10 types)	0:14.64 min:sec
640	160 cpus (20 types)	0:44.52 min:sec
2560	640 cpus (80 types)	37:11.58 min:sec

Table 5: Meta-algorithmic optimal parameter search and task assignment run-times.

Number of tasks	Number of types of processors	Execution time
160	5	5:05.43 min:sec
320	10	56:51 min:sec
640	20	1465:40 min:sec

Table 6: Execution times for the allocation search.

The first column contains information about the number of tasks and the number of processors used by our allocation algorithm. The next four columns provide power consumption produced by the lower bound algorithm, the best and median random solution, and our solution. Table 4. shows that our allocation algorithm produced from 73% to almost 4900% better results than the exhaustive random search. Typical run times of the allocation algorithms are given in Table 6.

9 Conclusion

We developed a system-level approach for power minimization of cost- constrained hard real-time designs. The approach simultaneously optimizes all three degrees of freedom for power minimization, namely switching activity, effective capacity and supply voltage. Novel meta-algorithmics approach strategy is used for computer-aided statistical optimization of the assignment algorithm. The comprehensive experimental results demonstrate the quality of the proposed optimization techniques.

References

[1] A.P. Chandrakasan, et. al., "Low-power CMOS digital design", IEEE Journal of Solid-State Circuits, vol.27, (no.4):473-484, April 1992.

[2] M. Chiodo, et. al. "Hardware-software codesign of embedded systems", IEEE Micro, vol.14, (no.4):26-36, August 1994.

[3] M. Chiodo, et. al. "A case study in computer-aided co-design of embedded controllers", Design Automation for Embedded Systems, vol.1, (no.1-2):51-67, January 1996.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, “Introduction to algorithms”, MIT Press, Cambridge, MA, 1990.

[5] M.R. Garey, D.S. Johnson, “Computers and intractability : a guide to the theory of NP-completeness”, W. H. Freeman, San Francisco, CA, 1979.

[6] D. D. Gajski, F. Vahid, and S. Narayan, “A System-Design Methodology: Executable Specification Refinement”, Euro-DAC ‘94, pp. 458-463, 1994.

[7] R. K. Gupta and G. De Micheli, “Hardware-software cosynthesis for digital systems”, IEEE Design and Test of Computers, vol.10, (no.7):29-41, September 1993.

[8] R.K. Gupta, “Analysis of operation delay and execution rate constraints for embedded systems”, Design Automation Conference, pp. 601-604, 1996.

[9] P.D. Hoang, J.M. Rabaey, “Scheduling of DSP programs onto multiprocessors for maximum throughput”, IEEE Transactions on Signal Processing, vol.41, (no.6):2225-2235, June 1993.

[10] T. B. Ismail, K. O’Brien, and A. Jerraya, “Interactive System-Level Partitioning with PARTIF”, Euro-DAC ‘94, pp. 464-468, 1994.

[11] M.T.-C. Lee, et. al., “Power analysis and low-power scheduling techniques for embedded DSP software”, Fujitsu Scientific and Technical Journal, vol.31, (no.2):215-229, 1995.

[12] J.P. Lehoczky et. al., “The Rate Monotonic Scheduling Algorithms - Exact Characterization and Average Case Behavior”, IEEE Real-Time System Symp., pp. 181-191, 1986.

[13] C.L. Liu, J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment”, Journal of ACM, vol. 20, (no.1):46-61, 1973.

[14] S. Molloy, R. Jain, “System and architecture optimizations for low power MPEG-1 video decoding”, Symposium on Low Power Electronics, pp. 26-27, October 1994.

[15] M. Potkonjak, W.H. Wolf, “Cost Optimization in ASIC implementation of Periodic Hard-Real Time Systems using Behavioral Synthesis Techniques”, ICCAD95, pp. 446-451, 1995.

[16] D. Singh et al., “Power Conscious CAD Tools and Methodologies”, Proc. of the IEEE, vol.83, (no.4):570-594, 1995.

[17] J.A. Stankovic, M. Spuri, M. Di Natale, G.C. Buttazzo, “Implications of Classical Scheduling Results for Real-Time Systems”, IEEE Computer, vol.28, (no.6):16-25, June 1995.

[18] R.A.Thisted, “Elements of statistical computing”, Chapman and Hall, New York, NY 1988.

[19] W.H. Wolf: “Hardware-Software Co-Design of Embedded Systems”, Proceedings of the IEEE, vol.82, (no.7):967-989, July 1994.

[20] A. Wolfe, “Opportunities and obstacles in low-power system-level CAD”, Design Automation Conference, pp. 15-20, June 1996.