# Efficient Block Scheduling for Programmable Embedded Processors


## Abstract

Scheduling is probably the most often addressed optimization problem in DSP compilation, behavioral synthesis, and system-level synthesis research. With the rapid pace of changes in modern DSP applications requirements and implementation technologies, however, new types of scheduling challenges arise. This paper is concerned with the problem of scheduling blocks of computations in order to optimize the efficiency of their execution on programmable embedded systems under a realistic timing model of their processors.

We describe an effective scheme for scheduling the blocks of any computation on a given system architecture and with a specified algorithm implementing each block. We also present algorithmic techniques for performing optimal block scheduling simultaneously with optimal architecture and algorithm selection. Our techniques address the block scheduling problem for both single- and multiple-processor system platforms and for a variety of optimization objectives including throughput, cost, and power dissipation. We demonstrate the practical effectiveness of our techniques on numerous examples.

# 1 Introduction

## 1.1 New Directions for an Ancient Problem

Scheduling has been one of the most popular research topics in DSP synthesis and compilation ever since the (r)evolution of implementation technologies enabled hardware sharing. Over the years, however, DSP scheduling in its classical form lost some of its importance due to a variety of reasons. Extensive empirical studies have indicated that different competing scheduling techniques have relatively small impact on the key design metrics of the final implementation [6]. Recently developed estimation techniques have also indicated that there is little room for further improvement of scheduling algorithms [2] [21]. More importantly, there has been strong evidence that other behavioral synthesis tasks, in particular transformations, can have a significantly higher impact than scheduling on the quality of the final implementation [19].

The rapid pace of modern trends in DSP applications and implementation technologies has recently revitalized the importance of scheduling by simultaneously imposing new sets of design issues, goals, and constraints. The current trend in DSP compilation and behavioral synthesis is to consider higher levels of abstraction and larger designs. This trend continues at increasing rate due to both application and implementation factors. The size of average DSP commercial electronics applications has been doubling each year. At the same time, the clock frequency and the average size of industrial integrated circuits have been doubling every three years. Thus, higher levels of hardware sharing become feasible and economically desirable with each new generation of technology, while at the same time hardware sharing is becoming increasingly important for satisfying applications requirements.

As we move toward systems-on-a-chip, DSP scheduling is inevitably changing its focus from operation-level granularity to task-level granularity. Thus, new scheduling problems arise that seek to optimize the efficiency of the execution of tasks (or blocks) using realistic timing models for the embedded processor platforms. At this level, context switching time is a new degree of freedom, and its minimization is a prime optimization objective. In this paper, we present a collection of algorithmic techniques that generate optimal block schedules by minimizing the total context switch time. Our schemes yield optimal block schedules for any given platform and for any choice of algorithms for each task. Furthermore, our schemes can yield optimal block schedules while simultaneously enabling the selection of optimal architecture and algorithms for implementing the various tasks.

| Block | Processing Time | Switching Times Between Blocks | | | | |
|---|---|---|---|---|---|---|
| | | A | B | C | D | E |
| A | 30 | 0 | 18 | 20 | 15 | 10 |
| B | 15 | 8 | 0 | 15 | 8 | 7 |
| C | 20 | 12 | 8 | 0 | 22 | 15 |
| D | 10 | 10 | 10 | 18 | 0 | 32 |
| E | 25 | 19 | 8 | 12 | 10 | 0 |

Table 1: A simple instance of the block scheduling problem.

### 1.2 Block Scheduling Example

We introduce the block scheduling problem using the simple example described in Table 1. Given this table, we wish to find a schedule of blocks that maximizes throughput on a single-processor system. For the sake of simplicity, we assume that the iteration count for each block is 2, and that the blocks are independent. A block schedule $< A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2 >$ takes 431 cycles. The block schedule $< A_1, A_2, E_1, E_2, C_1, C_2, B_1, B_2, D_1, D_2 >$ takes 248 cycles, however. Since the order of execution does not affect processing times, only the switching times are to be considered during schedule optimization. Considering only the switching times, the former schedule takes 231 cycles while the latter one takes only 48.

### 1.3 Paper Organization

The remainder of this paper has five sections. In Section 2 we present our computational and hardware model. We also briefly consider a simplified version of the general block scheduling problem in which no data dependencies are considered, and we argue that it is computationally intractable by a reduction from the well-known Traveling Salesman Problem (TSP). Section 3 discusses related research. In Section 4, we formulate, analyze, and solve the block scheduling problem with simultaneous architecture and algorithm selection and no data dependencies. Our results apply to both single- and multi-processor platforms. In Section 5, we address the general block scheduling problem with data dependencies on single-processor platforms. We summarize our results in Section 6.

## 2 Preliminaries

### 2.1 Computational and Hardware Model

This paper is concerned with the problem of scheduling computations in order to maximize the efficiency of their execution on programmable embedded systems. In this section we describe our computational model, our hardware model, and its associated cost functions.

We model a computation on a given system and a given algorithm implementation for each block as a directed edge-weighted, vertex-weighted graph $G = (V, E, d, p, s)$ which is called a *computation dataflow graph.* Each vertex $u \in V$ denotes a *computational block*, that is, a sequence of computations that are executed given an input. The edges in $G$ encode *data dependencies.* Specifically, each edge $(u, v) \in E$ denotes that block $v$ requires the output of block $u$ in its computation. The number of iterations by which an output of block $u$ must be delayed before it is presented to the inputs of block $v$ is given by the integer *delay count* $d(u, v)$. Each vertex-weight $p(u)$ denotes the processing time for the block corresponding to vertex $u$, and each edge-weight $s(u, v)$ denotes the time required to switch processing from block $u$ to block $v$.

The length $L(\mathcal{S})$ of a given schedule $\mathcal{S}$ is given by adding up the total processing time $p(\mathcal{S})$ and the total switching time $s(\mathcal{S})$. The total processing time is given by the sum

$$p(\mathcal{S}) = \sum_{u \in \mathcal{S}} p(u) \ ,$$

which does not depend on the execution order of the blocks. The total switching time is given by

$$s(\mathcal{S}) = \sum_{\{u_i, v_j\} \subseteq \mathcal{S}} s(u, v) \ ,$$

where the subset notation denotes that the blocks $u_i$ and $v_j$ are executed consecutively in $\mathcal{S}$. Thus, the order in which blocks are scheduled to execute can affect the switching cost for the implementation of a computation. Context switching costs are assumed to be high, and thus no preemption is allowed.

In the basic block scheduling problem, given a computation $G$ with an associated architecture and an algorithm for executing each block in $G$, we wish to find a schedule $\mathcal{S}$ of the blocks that satisfies all data dependencies in $G$ and minimizes $L(\mathcal{S})$. The basic block scheduling problem can be augmented to include architecture and algorithm selection. In the augmented problem, we wish to find an optimal schedule for an optimal selection of architecture and algorithms from a specified set of alternatives. The augmented block scheduling problem may have several optimization objectives such as cost of components or power dissipation. For example, designers may have a choice among platforms with different voltage supply characteristics, and thus different power dissipation characteristics, assuming that power consumption is given by $V_{dd}^2 \cdot C_{eff} \cdot \alpha$, where $V_{dd}$ is supply voltage, $C_{eff}$ is switching capacitance, and $\alpha$ is switching activity.

### 2.2 Equivalence of Block Scheduling Problem and Traveling Salesman Problem

When no data dependencies exist between blocks, the block scheduling problem is equivalent to the Traveling Salesman Problem (TSP). The TSP is formally stated as follows.

*Problem*: Traveling Salesman Problem

*Instance*: Given is a set of $n$ cities $C$, an $n \times n$ matrix $S$ of distances between cities in $C$, and a positive number $l$.

*Question*: Is there a tour of $C$ of length $l$ or less, i.e., a permutation $(\pi(1), \pi(2), \ldots, \pi(n))$, $\pi(i) \in C$, such that $l \geq s_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} s_{\pi(i)\pi(i+1)}$?

When the matrix $S$ is symmetric, i.e., $s_{ij} = s_{ji}$ for all $i, j$, then the TSP is called symmetric. When $S$ is not symmetric, the TSP is referred to as asymmetric.

The correspondence between the simplified block scheduling problem above and the asymmetric TSP is straightforward. Each block $u$ corresponds to a city, and the switching cost matrix $S$ corresponds to the distance matrix. Thus, the optimum block schedule corresponds to a minimal length tour, and vice-versa. In Section 4 we consider the augmented block scheduling problem with simultaneous algorithm and architecture selection and no data dependencies. The equivalence of block scheduling and TSP is central to our attack to this problem.

## 3 Related Research

In this section we briefly summarize previous research on scheduling and on algorithm and architecture selection. We also outline the most relevant TSP references.

Scheduling has been widely studied in many areas, such as behavioral synthesis [16], parallel processing [5], and hard real-time systems [14]. The majority of the proposed DSP scheduling techniques has been presented in behavioral synthesis literature [16]. Initial scheduling and transformation techniques for block processing are presented in [1] [11] [22]. Also, algorithm and architecture selection has been recently recognized as an important system-level synthesis topic [18].

The Traveling Salesman Problem is probably the most widely studied problem in combinatorial optimization mostly due to its wide applicability as well as its representative nature of difficult optimization problems [12]. TSP is known to be NP-complete [4]. A number of efficient and effective solution techniques have been presented for the symmetric TSP that come within a few percentage points from the optimum [7]. According to Johnson's comprehensive empirical comparison of symmetric TSP heuristics [7], the most efficient and effective heuristics are the 3-Opt and the Lin-Kernighan [13] algorithms. Reinelt [20] has effectively reduced the running times of these schemes by considering only $O(n)$ candidate edges for the tour. Martin et al. [15] have given an effective way to improve the quality of the solutions obtained by these heuristics at the expense of increasing their running times.

Although a few solution methods have been proposed for the asymmetric TSP, including a heuristic by Kanellakis and Papadimitriou [9] which modifies the Lin-Kernighan symmetric TSP heuristic and a branch-and-bound approach by Miller and Pekny [17], there do not exist enough experimental data to judge their performance, and the heuristics have very high time complexity.

# 4 Block Scheduling with Independent Blocks

In this section we describe our algorithms for the augmented block scheduling problem with independent blocks and simultaneous architecture and algorithm selection. We first outline our key underlying assumptions. We then formulate the augmented block scheduling problem for a variety of objectives, including throughput, cost of components, and power optimization. Finally, we present effective TSP-based heuristics for solving these problems and present experimental results that support the effectiveness of the proposed schemes.

## 4.1 Problem Formulations

We assume that there are no data, control, or timing dependencies among blocks. In this case, only the blocks within a single iteration need to be scheduled, since any number of block iterations can be scheduled consecutively to generate the zero context switch time between the iterations of the block. For each block, we assume that there exist a number of algorithm choices. Also, there exist a number of processor types or implementation platforms. The switching time between the algorithms of any two blocks is considered as well as the running times for processing the blocks. Our schemes address two different design approaches: (i) only one processor is chosen for all blocks (ii) multiple processors (of possibly different types) are chosen, and each block is assigned to one them.

### 4.1.1 Throughput Optimization

The block scheduling problem for throughput optimization can be formally stated as follows.

*Problem*: Block Scheduling for Throughput Optimization with Simultaneous Algorithm and Architecture Selection.

*Instance*: We are given $n$ blocks, $T$ different processor types, and a positive constant $l$. Each block $i$ has a choice of $B_i$ different algorithms. For each processor type, we are given the processing times array $P$ for all the available algorithms and the switching times matrix $S$ between the algorithms.

*Question*: Is there a schedule of blocks, a selection of algorithms and a selection of processors such that the resulting schedule length is at most $l$?

### 4.1.2 Cost Optimization under Throughput Requirement

The block scheduling problem for minimum cost with simultaneous selection of algorithms and architectures is similar to the throughput optimization problem. The only difference is that the objective is to minimize the cost under a throughput constraint.

*Problem*: Block Scheduling for Cost Optimization under Throughput Requirement with Simultaneous Algorithm and Architecture Selection.

*Instance*: We are given $n$ blocks, $T$ processor types, a cost array $C$ for the processor types, and positive constants $l$ and $m$. Each block $i$ has a choice of $B_i$ different algorithms. For each processor type, we are given the processing times array $P$ for all the available algorithms, and the switching time matrix $S$ between the algorithms.

*Question*: Is there a schedule of blocks, a selection of algorithms and a selection of processors such that the resulting schedule length is at most $l$, and the cost of all the processors selected is at most $m$?

### 4.1.3 Power Optimization under Throughput Requirement

Power optimization under throughput requirement with simultaneous algorithm and architecture selection and voltage scaling can be formulated in a way similar to throughput optimization. The differences are that the optimization objective is the minimization of the power consumption under the throughput requirement, and that voltage scaling is performed in addition to algorithm and architecture selection.

*Problem*: Block Scheduling for Power Optimization under Throughput Requirement with Simultaneous Algorithm and Architecture Selection and Voltage Scaling.

*Instance*: We are given $n$ blocks, $T$ different processor types with cost array $C$, a clock-cycle time array $CCT$, an array of power dissipations $PD$, an initial voltage $V$ of each processor type, and positive constants $l$ and $m$. Each block $i$ has a choice of $B_i$ different algorithms. For each processor type, we are given the running times array $P$ for all the available algorithms and the switching times matrix $S$ between the algorithms.

*Question*: Is there a schedule of blocks, a selection of algorithms, a selection of processors and a new voltage $V'$ such that the resulting schedule length is at most $l$ and the power consumption is at most $m$?

### 4.2 TSP Implementation Issues

At the heart of all our schemes lies a basic block scheduler that is implemented efficiently using TSP heuristics. In this section we describe a few issues related to our implementations of these heuristics. All our schemes transform an asymmetric TSP into a symmetric one using the technique proposed by Jonker and Volgenant [8]. We are thus able to apply directly efficient and effective heuristics for the symmetric TSP.

The transformation into symmetric TSP is performed as follows. Let TSP($C, S$) denote an asymmetric TSP with a set of $n$ cities $C = 1, 2, ..., n$ on a distance matrix $S$. Let TSP($C', S'$) denote the transformed symmetric TSP problem. Each city $i$ in $C$ is converted into two cities, $i$ and $i + n$, in $C'$. If $|i - j| = n$, i.e., the two cities $i$ and $j$ in $C'$ are from the same city in $C$, $s'_{ij} = -M$, where $M$ is a very large positive number. If ($i \leq n$ and $j \leq n$) or ($i > n$ and $j > n$), $s'_{ij} = \infty$. Otherwise, $s'_{ij} = s_{ij}$ if $i > j$ and $s'_{ij} = s_{ji}$ if $i < j$. It is shown in [8] show that any optimal solution of the transformed symmetric TSP corresponds to an optimal solution of the original asymmetric TSP. Feuer and Koo [3] have successfully applied a similar transformation to the scan chain ordering problem.

Recent studies [7] suggest that the most efficient and effective heuristics for the symmetric TSP are 3-Opt and Lin-Kernighan [13] algorithms. These algorithms apply a greedy global optimization strategy which iteratively constructs an improved new solution $s'$ from the previous solution $s$ by making local changes to $s$. If no $s'$ can be found, the solution $s$ is returned as a local optimum. Martin et al. [15] have given an effective scheme, called the large-step markov chain (LSMC), that improves the quality of these solutions. This heuristic is now considered to be the "champion" for symmetric TSP. LSMC starts with an arbitrary initial local minimum and keeps improving on it by "perturbation and local optimization" cycles. In our software implementation, we have used the LSMC heuristic with Reinelt's fast, nearest-neighbor based implementation of the 3-Opt local optimization engine [20]. The quality of the TSP tour can be traded off with running time by adjusting the number of the 3-Opt local optimization algorithms. In all our runs, we used the LSMC heuristic with 10 iterations of 3-Opt local optimization algorithms.

## 4.3 Block Scheduling on Single Processor

### 4.3.1 Algorithms

We have developed an iterative improvement algorithm based on the efficient implementation of a wide variety of TSP heuristic. Our throughput optimization heuristic is given by the pseudo-code in Figure 1. Our heuristic for cost optimization under a throughput constraint is described in Figure 2. Our heuristic for power optimization under a throughput constraint is given by the pseudo-code in Figure 3. In all experiments, the threshold t was chosen to be 100.

```
Set the schedule length of the best-so-far solution to ∞.
For each processor type,
   Generate an initial random solution.
   While number of consecutive iterations with no improvement < threshold t
      Apply TSP heuristic on current solution to get a block schedule.
      If there is improvement, accept new solution.
      Change current solution by changing algorithms of a randomly chosen block.
   Keep the current solution as the best solution for the processor.
   If best solution is better than best-so-far, set the best-so-far solution to the best solution.
Return the best-so-far solution.
```

Figure 1: Heuristic for throughput optimization.

```
Set the best-so-far solution to have the cost ∞.
For each processor type,
   Apply throughput optimization heuristic.
   If schedule satisfies throughput requirement,
      If cost of processor type < the best-so-far cost,
         Set the best-so-far solution to the current solution.
Return the best-so-far solution.
```

Figure 2: Heuristic for cost optimization under throughput constraint.

Set the power consumption of best-so-far solution to $\infty$.
**For** each processor type
  Generate a block schedule using throughput optimization heuristic.
  Scale the voltage so that the block schedule just satisfies the throughput requirement.
  Compute power consumption.
  **If** power consumption $<$ best-so-far, set best-so-far solution to the current solution.
Return the best-so-far solution.

Figure 3: Heuristic for power optimization under throughput constraint.

#### 4.3.2 Experimental Results

We applied our algorithms on randomly generated computation graphs with 10, 25, 50, 75, and 100 blocks. We assumed 5 processor types with randomly chosen parameters and 5 algorithm choices for each block. Table 2(a) gives the processor specifications. For each processor type, processing and switching times were chosen in the range from 1 to 100, cost was chosen between 100 and 200, clock-cycle time was chosen in the range from 5.0 to 10.0, initial voltages were chosen between 3.0 and 5.0, and power dissipation was between 5.0 and 10.0. The threshold voltage was assumed to be 0.7. For each processor type, in addition to its specification, we give a lower bound on the length of any schedule with a single iteration. This bound is derived by summing up the smallest processing times of the blocks and the smallest switching times of all block pairs.

Table 2(b)-(d) illustrates the effectiveness of our TSP-based heuristics on the 50-block computation, a representative example among the ones we experimented with. We compared the effectiveness of our algorithms with the best among 100 random solutions that were generated for the problem. For this specific example, the average improvement for throughput and power was a factor of 2.9 and 2.4, respectively. For cost optimization under throughput requirement, no random solution satisfied the upper bound on the schedule length.

Table 3 summarizes the experimental results for all randomly generated computations. For each block count, we generated 10 random computations. Note that if only the switching times were considered, then the improvements would be greater.

| | Processor Type | | | | |
|---|---|---|---|---|---|
| | # 1 | # 2 | # 3 | # 4 | # 5 |
| Cost | 171 | 126 | 165 | 119 | 155 |
| Clock-cycle time | 9.1 | 10.0 | 7.1 | 7.4 | 5.9 |
| Initial voltage | 4.2 | 5.0 | 4.3 | 4.3 | 4.2 |
| Power dissipation | 8.4 | 6.9 | 6.4 | 9.7 | 5.8 |
| Lower-bound on schedule length | 1701 | 1481 | 1311 | 1480 | 1657 |

(a) Processor specifications.

| | Random Solution | TSP-based heuristics |
|---|---|---|
| Processor Type | # 2 | # 3 |
| Optimal schedule length | 4304 | 1476 |

(b) Throughput optimization results.

| | | | | | |
|---|---|---|---|---|---|
| Schedule length constraint | 1900 | 1800 | 1700 | 1600 | 1500 |
| Optimal schedule length | 1653 | 1653 | 1653 | 1476 | 1476 |
| Cost | 119 | 119 | 119 | 165 | 165 |
| Processor type | # 4 | # 4 | # 4 | # 3 | # 3 |

(c) Results of cost optimization under throughput requirement. No random solution satisfied maximum schedule length constraint.

| | Random Solution | Optimized Solution |
|---|---|---|
| Processor type | # 5 | # 3 |
| Scaled clock-cycle time | 11.09 | 33.88 |
| Initial voltage | 4.2 | 4.3 |
| Scaled voltage | 2.77 | 1.75 |
| Scaled power dissipation | 2.53 | 1.06 |
| Total energy consumption | 126590.44 | 53124.29 |
| Optimal schedule length | 4508 | 1476 |

(d) Results of power optimization under throughput requirement. Sample period was 50,000 time units.

Table 2: Experimental results for a computation with 50 blocks on a single processor.

| Blocks | Throughput Improvement | Cost Improvement | Power Improvement |
|---|---|---|---|
| 10 | 2.8 | 2.9 | 2.1 |
| 25 | 3.1 | 3.5 | 2.4 |
| 50 | 2.9 | 3.4 | 2.3 |
| 75 | 2.7 | 2.9 | 2.2 |
| 100 | 3.2 | 3.7 | 2.6 |

Table 3: Average improvement factors on a single processor.

## 4.4 Block Scheduling on Multiple Processors

### 4.4.1 Algorithms

We developed an exhaustive-enumeration algorithm for cost optimization under throughput requirement that was based on TSP heuristics and lower-bound estimations. Similarly to the lower-bound estimation technique for block scheduling on a single processor, we designed a simple and effective lower-bound estimation technique for throughput optimization on multiple processors. For each block, we choose an algorithm with the smallest sum of processing and switching times over all processor types. A lower bound on the length of a schedule for the multiple-processor block scheduling problem can be obtained by dividing the sum of the processing and switching times for the best algorithms by the number of processors. Using this lower-bound, we can prune infeasible branches of the search. The heuristics for throughput optimization and for cost optimization under throughput requirement are given in Figures 5 and 4, respectively. The threshold $t$ was chosen to be 10, and the value of $k$ was set to 5 to allow escaping from local minima.

Randomly assign the same number of blocks to each of the allocated processors.
**For** each processor in the processor set,
  Apply the heuristic for throughput optimization on the processor to get a block schedule.
**While** the number of consecutive iterations with no improvement $<$ a threshold $t$
  Move a randomly chosen block from the processor with the maximum schedule length
  to another processor which is chosen by the proportional probability
  depending on the schedule length.
  **If** the resulting maximum schedule length is not larger than k% more of the best-so-far
  maximum schedule length, the current assignment of blocks is updated with the change.
  **If** it is smaller than the best-so-far schedule and assignment solution is updated.
Return the best-so-far solution.

Figure 4: Heuristic for throughput optimization on multiple processors.

| |
|---|
| The current processor set is empty.<br>**Repeat**<br>Choose the cheapest processor set as the new current processor set<br>which costs more than or equal to the current processor set.<br>Compute the lower-bound for the current processor set.<br>**If** the lower-bound satisfies the throughput requirement, apply the heuristic to optimize throughput<br>on multiple processors described in Figure 4 for the current processor set.<br>**Until** the current processor set satisfies the throughput requirement.<br>Return the current processor set and block schedules. |

Figure 5: Heuristic for cost optimization under throughput requirement on multiprocessors.

### 4.4.2 Experimental Results

We generated random computations with 10, 25, 50, 75, and 100 blocks. For each block count, we generated 10 computations graphs. As in the single-processor experiments, we assumed 5 processor types and 5 algorithm choices for each block. Running and switching times were randomly chosen in the range from 1 to 100, and the costs of the processors were chosen between 100 and 200. Table 4 gives cost optimization results under a throughput requirement for the example computation with 50 blocks. The results for all examples are illustrated in Table 5. Random solutions were generated by picking the best among 100 random schedules of blocks to processors which were allocated by exhaustively enumerating all allocation possibilities in increasing cost order. Under various throughput requirements, the average improvement obtained over all examples was a factor of 3.1.

| | Cost | | | Schedule Length | |
|---|---|---|---|---|---|
| Sample Period | Random | Opt | Improvement Factor | Random | Opt |
| 500 | 1408 | 416 | 3.38 | 458 | 486 |
| 600 | 1040 | 312 | 3.33 | 577 | 595 |
| 750 | 766 | 208 | 3.68 | 738 | 749 |
| 850 | 684 | 208 | 3.29 | 845 | 749 |
| 1000 | 520 | 208 | 2.5 | 970 | 749 |
| 1500 | 312 | 104 | 3.0 | 1472 | 1389 |

Table 4: Results of cost minimization for 50-blocks computation on multiple processors.

## 5 Block Scheduling with Dependent Blocks

In this section we address the basic block scheduling problem for throughput optimization with data dependencies. We focus on single processor platforms without considering algorithm and architecture

| Blocks | Cost Improvement |
|---|---|
| 10 | 2.6 |
| 25 | 2.8 |
| 50 | 3.3 |
| 75 | 3.1 |
| 100 | 3.5 |

Table 5: Average cost improvement factors on multiple processors over all examples generated.

selection.

### 5.1 Problem Formulation

The block scheduling problem for throughput optimization can be formally stated as follows.

*Problem*: Throughput Optimization

*Instance*: We are given $I$ iteration counts, $n$ blocks with dependencies described by a computation dataflow graph $G$, the processing times array $P$ for all the blocks, the switching times matrix $S$ between the blocks, and a positive constant $l$.

*Question*: Is there a schedule of blocks such that data dependencies are satisfied and the schedule length is at most $l$?

The throughput optimization problem is clearly intractable, since its special case with no dependency is already intractable.

### 5.2 Verifying Valid Schedules

In this subsection we show how to verify whether a schedule is *valid*, that is, whether it correctly executes a given computation. We first give two necessary and sufficient conditions for a schedule to be valid. We then describe a simple scheme that checks these conditions.

Given a computation $G = (V, E, d, p, s)$ and schedule $\mathcal{S}$, the following two conditions are necessary and sufficient for $\mathcal{S}$ to be valid.

(C1) For every computational block $u \in V$, and for $i = 1, 2, \ldots, I$, we must have that $u_i$ is in the schedule.

(C2) For every pair $u, v$ in $V$ such that there exists a path from $u$ to $v$ in $G$, and for every integer $i, j$ such that $1 \leq i \leq j$ we must have that $u_i$ precedes $v_j$ in the schedule.

The first condition ensures that every computational block computes an output for every iteration of the computation. The second condition ensures that data dependencies are not violated.

If the schedule $\mathcal{S}$ is finite, conditions C1 and C2 can be verified in $O(|\mathcal{S}|^2)$ time, where $|\mathcal{S}|$ is the number of blocks in the schedule. Verifying condition C1 requires a single scan of the schedule, which can be performed in linear time. In order to verify condition C2, we first compute the transitive closure on $G$ to determine reachability. We then perform a pairwise check of all blocks in the schedule to determine whether the precedence constraint C2 is satisfied between blocks that are reachable.

If the schedule $\mathcal{S}$ is periodic, the two conditions can be verified within $O(T^2)$ steps, where $T$ is the number of blocks in each period of $\mathcal{S}$.

### 5.3 Optimization Algorithm

For the solution of the throughput optimization problem, we have applied a general combinatorial optimization technique known as simulated annealing (SA) [10]. The cost function is the length of the schedule. The initial schedule presented to the SA algorithm is obtained by a topological sort of the computation dataflow graph. Subsequent solutions are generated by randomly swapping two blocks in the schedule until we obtain a valid new schedule. At any time during the execution of the algorithm, only valid schedules are allowed. SA escapes local minima by allowing worse solutions to be accepted as new starting solutions with an initially high probability that gradually decreases as the optimization process continues. The details of the implementation for SA will be presented in the journal version of the paper.

| | Iteration Count | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 2 | | | 3 | | | 5 | | |
| # Blocks | Random | Opt | Impr | Random | Opt | Impr | Random | Opt | Impr |
| 5 | 529 | 292 | 1.81 | 696 | 394 | 1.77 | 982 | 577 | 1.70 |
| 8 | 824 | 439 | 1.88 | 1138 | 633 | 1.80 | 1489 | 851 | 1.75 |
| 10 | 1056 | 569 | 1.86 | 1388 | 795 | 1.75 | 1991 | 1150 | 1.73 |
| 15 | 1521 | 800 | 1.90 | 1912 | 1062 | 1.80 | 2844 | 1663 | 1.71 |
| 20 | 2174 | 1187 | 1.83 | 2731 | 1566 | 1.74 | 3993 | 2264 | 1.76 |

Table 6: Results of throughput optimization with data dependencies.

### 5.4 Experimental Results

We randomly generated computations with 5, 8, 10, 15, and 20 blocks. We considered iteration counts of 2, 3, and 5. The number of edges in the computation dataflow graph for a single iteration was $\frac{\#\ blocks}{2}$, and the number of edges with a single delay was $\frac{\#\ blocks}{5}$. The processing times and switching times were randomly chosen in the range between 1 and 100. The results of our experiments are given in Table 6. The best of 100 random solutions are reported and compared with the heuristic solution. Observe that the improvement factor decreases as the number of iteration count increases, since the percentage of the processing time over the total execution time becomes higher. When only switching times are considered, the improvement factor increases as the number of iteration count increases, as one would intuitively expect.

## 6 Conclusion

This paper addresses the problem of scheduling blocks of computations in order to optimize the efficiency of their execution on programmable embedded systems under a realistic timing model of the processor. We leveraged on our approach to the basic block scheduling problem to address simultaneous algorithm and architecture selection for both single and multiple processor implementation platforms. The optimization objectives we considered were throughput, cost, and power. We demonstrated the effectiveness of our schemes on numerous examples.

## References


[1] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Optimal Parenthesization of Lexical Orderings for DSP Block Diagrams", VLSI SP Workshop, pp. 177-186, 1995

[2] S. Chaudhuri, R.A. Walker, "Computing Lower Bounds on Functional Units Before Scheduling", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 4, No. 2, pp. 273-9, 1996.

[3] M. Feuer and C. C. Koo, "Method for Rechaining Shift Register Latches Which Contain More Than One Physical Book", IBM Technical Disclosure Bulletin, Vol. 25, No. 9, pp. 4818-4820, 1983.

[4] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. H. Freeman and Co., 1979.

[5] M.J. Gonzales, "Deterministic Processor Scheduling", ACM Computing Surveys, Vol. 9, No. 3, pp. 173-204, 1977.

[6] R. Jain et al., "Empirical Evaluation of Some High-level Synthesis Scheduling Heuristics", Design Automation Conference, pp. 686-689, 1991.

[7] D. S. Johnson, "Local Optimization and the Traveling Salesman Problem", In Proceedings of the 17th Intl. Colloquium on Automata, Languages and Programming, pp. 446-460, 1990.

[8] R. Jonker and T. Volgenant, "Transforming Asymmetric into Symmetric Traveling Salesman Problems", Operations Research Letters, Vol. 2, No. 4, 1983.

[9] P.-C. Kanellakis and C. H. Papadimitriou, "Local Search for the Asymmetric Traveling Salesman Problem", Operations Research, Vol. 28, pp. 1086-1099, 1980.

[10] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by Simulated Annealing," Science, Vol. 220, No. 4598, pp. 671-680, 1983.

[11] K.N. Lalgudi, M.C. Papaefthymiou, M. Potkonjak, "Optimizing Systems for Effective Block-processing: the $k$-delay Problem", Design Automation Conference, pp. 714-719, 1996.

[12] E. L. Lawler, J. K. Lenstra, A. Rinnooy-Kan, and D. Shmoys, "The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization", Wiley, Chichester, 1985.

[13] S. Lin and B. W. Kernighan, "An effective Heuristics Algorithm for the Traveling-Salesman Problem", Operations Research, Vol. 31, pp. 498-516, 1973.

[14] C.L. Liu, J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment", Journal of ACM, Vol. 20, No. 1, pp. 46-61, 1973.

[15] O. Martin, S. W. Otto and E. W. Felten, "Large-Step Markov Chains for the TSP Incorporating Local Search Heuristics", Operations Research Letters, Vol. 11, No. 4, pp. 219-224, 1992.

[16] M.C. McFarland, A.C. Parker, R. Camposano: "The High-Level Synthesis of Digital Systems", Proceedings of the IEEE, Vol. 78, No. 2, pp. 301-317, 1990.

[17] D. L. Miller and J. K. Pekny, "Exact Solution of Large Asymmetric Traveling Salesman Problems", Science, Vol. 251, pp. 754-761, 1991.

[18] M. Potkonjak, J. Rabaey, "Algorithm Selection: A Quantitative Computation-Intensive Optimization Approach", International Conference on Computer-Aided Design, pp. 90-95, 1994.

[19] M. Potkonjak, J. Rabaey, "Optimizing Resource Utilization Using Transformations", IEEE Transactions on CAD, Vol. 13, No. 3, pp. 277-292, 1994.

[20] G. Reinelt, "Fast Heuristics for Large Geometric Traveling Salesman Problems", ORSA Journal on Computing, Vol. 4, No. 2, pp. 206-217, 1992.

[21] A. Sharma, R. Jain, "Estimating Architectural Resources and Performance for High-level Synthesis Applications", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 1, No. 2, pp. 175-90, 1993.

[22] V. Zivojnovic, S. Ritz, H. Meyr, "Retiming of DSP programs for Optimum Vectorization", International Conference on Acoustic, Speech, and Signal Processing, Vol. 2, pp. 465-468, 1994.