TSP-based algorithms for DSP scheduling

Inki Hong and Miodrag Potkonjak

Computer Science Dept., University of California, Los Angeles, CA

ABSTRACT

Scheduling is probably the most often addressed optimization problem in DSP compilation, behavioral and system-level synthesis research. With a rapid pace of changes in modern DSP applications and implementation technologies, scheduling poses new types of optimization challenges. We present a new approach for addressing a large family of modern DSP scheduling problems. The approach is based on transforming a given scheduling problem into an instance of the traveling salesman problem (TSP). Although TSP is an NP-complete problem, it has been extensively studied and several exceptionally effective solution techniques are available. We leverage on these techniques to develop efficient and effective scheduling and assignment algorithms for modern DSP scheduling and assignment problems.

The approach is applied to two new DSP scheduling problems: block processing scheduling and disk scheduling. In the first case we address, simultaneously with block processing scheduling, algorithm and architecture selection for both single and multiple processor implementation platforms. The versions for optimization of throughput, cost, and power are developed. The DSP disk scheduling problem is solved using a TSP optimization algorithm. The effectiveness of all algorithms is demonstrated on numerous examples.

1.0 Introduction

1.1 Motivation: New Directions for Ancient Scheduling Problems

Scheduling in DSP synthesis and compilation has been one of the most popular research topics ever since the (r)evolution of implementation technologies enabled hardware sharing. Recently, DSP scheduling in its classical form lost some of its importance due to the following factors. First, extensive empirical studies indicate that different competing scheduling techniques have relatively small impact on the key design metrics of final implementation. Moreover, the recently developed estimation techniques indicate that there is little room for further improvement of scheduling algorithms. More importantly, there has been strong evidence that other behavioral synthesis tasks, in particular transformations, have significantly higher impact on the quality of the final implementation than scheduling.

However, the rapid pace of modern applications and implementation technologies trends, recently revitalized importance of scheduling and simultaneously imposed new sets of design issues, goals, and constraints which have to be considered. The trend in DSP behavioral synthesis and compilation 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. Both the size and clock frequency, and therefore the level of hardware sharing have been increasing rapidly. The average size of the industrial integrated circuits has been doubling every three years, while the clock period has been steadily decreasing at the similar rate. These trends imply that, in each new generation of technology, higher levels of hardware sharing are required, feasible, and economically desirable. Therefore, as we move toward systems-on-a-chip the future DSP scheduling will inevitably change its focus from the operation-level granularity to the task-level granularity.

Another pronounced trend in modern DSP application is focus shift from data manipulation to data management. This trend is exceptionally well illustrated by great popularity of www-based applications. Data management is inevitably connected to massive-memory storage. Therefore, as we move toward design of massive-memory DSP application specific systems, synthesis techniques for design and effective usage of disks in application specific systems will play an increasingly more important role.

In this paper we introduce a set of scheduling and assignment techniques for solving the optimization problems related to design of multitask and disk-based systems.

1.2 Block Processing Scheduling: How to use TSP for DSP

In this subsection, we discuss how to use the traveling salesman problem for the processor block-scheduling problem. The problem is related to generation of effective block-scheduling algorithms. Computations in many application domains are defined on semi-infinite or very long streams of data. For example, this is typical data manipulation scenario for many video, audio, radar, sonar, and multimedia applications [Lee95]. Simultaneous computation on several incoming samples of the stream is a widely used approach to satisfy throughput requirements through use of parallelism and pipelining. It has been used in many areas, ranging from scientific computing, DSP, and communications to database computations.

Benefits of block processing go far beyond advantages pertinent to parallel processing platforms. Actually, they are often equally high or even higher on single processor platforms. Block processing enhances both regularity and locality in computations, two important properties of computation which greatly facilitate efficient implementation on many hardware platforms [Kun88]. Block-processing also reduces software-switching overhead which often dominates run-time. For example, in modern workstation the time required to bring a single piece of data in working memory is usually more than 100 cycles (109 for Sun Workstation). Block processing improves the effectiveness of methods for code size reduction due to its ability to improve locality. Block-processing enables efficient use of pipelines, and in particular processor which employs data-chaining and reconfigurability. Finally, block processing acts as enabling transformations for many other transformations, and in particular algebraic laws and redundancy elimination techniques. For example, Srivastava [Sri95] recently showed the procedure which enables the reduction of the number of operations in linear computations by almost a factor of 2. To conclude, minimization of context switch time is prime degree of optimization freedom at task-level

Block	Processing Time	Switching Time Matrix Between Blocks				
		1	2	3	4	5
1	30	-	18	20	15	10
2	15	8	-	15	8	7
3	20	12	8	-	22	15
4	10	10	10	18	-	32
5	25	19	8	12	10	-

Table 1: A simple instance of the processor block-scheduling problem

granularity DSP scheduling

We introduce the processor block-scheduling problem using the following example, which shows its simplified version. Table 1 shows a simple instance of the processor block-scheduling problem for throughput optimization. Only one processor type is available. Each block has only one algorithmic option. A block-schedule <1, 2, 3, 4, 5> takes 206 cycles. A different block-schedule <1, 5, 3, 2, 4> takes 148. Because the order of execution doesn't change the processing times, only the switching times are to be considered to minimize the schedule length. Considering only the switching times, the former block-schedule takes 106 while the latter takes only 48. This optimization problem can be formulated as the traveling salesman problem(TSP). The TSP is formally stated as follows:

Problem: Traveling Salesman Problem (TSP)

Instance: Given a set of n cities C, an $n \times n$ matrix D of distances between cities in C, and a positive number B

Question: Is there a tour of C of length B or less, i.e., a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ such that $B \ge D[\pi(n)][\pi(1)] + \sum_{i=1}^{n-1} D[\pi(i)][\pi(i+1)]$.

When the matrix D is symmetric, i.e., $D[i][j] = D[j][i]$, the TSP is called the symmetric TSP. When D is not symmetric, the TSP is referred to as the asymmetric TSP.

The mapping from the original problem to the asymmetric TSP is straightforward. For each block, a city is introduced. The switching time matrix becomes a distance matrix. Then, the minimum tour for the TSP is the best schedule for the processor block-scheduling problem. In Section 4, the more generalized problem which considers the block-scheduling, algorithm and architecture selection simultaneously is studied, where the proposed transformation plays a major role to attack the problem.

1.3 Disk Scheduling: TSP Approach

It is well known that a dominating bottleneck of current methodologies is disk. At the same time, disk is an indispensable element of computation systems due to its exceptional memory capacity/cost ratio. The speed of data transfer from disk to datapath is mainly limited by a seek time, which is in the first approximation proportional to distance which disk head has to travel between reading two pieces of data. We now introduce an approach which minimizes disk head movement in the following system design scenario. A set of tasks is given. Each task has a statically assigned set of data blocks to access for its execution. The assignment of data blocks to tracks is given. The goal is to find the best schedule (an order in which tasks will be executed) of tasks so that the head movement is minimized. It is assumed that the schedule is semi-infinite periodic. This assumption is proper computational model for majority of DSP and video computations [Lee95].

Task	Disk Block Access Sequence
A	5, 4
B	1
C	1, 2
D	3, 5
E	5

Disk Block Assignment
1, 2, 3, 4, 5

Disk Head Movement Distance Between Tasks

$$
\begin{array}{c|ccccc|}
 & A & B & C & D & E \\
\hline
A & - & 3 & 3 & 1 & 1 \\
B & 4 & - & 0 & 2 & 4 \\
C & 3 & 1 & - & 1 & 3 \\
D & 0 & 4 & 4 & - & 0 \\
E & 0 & 4 & 4 & 2 & - \\
\end{array}
$$

Figure 1: A Simple Instance of the Disk Scheduling Problem

Figure 1 shows a small instance of disk scheduling problem to minimize the disk head movement. One disk contains all the blocks. A task schedule <E, D, C, A, B> incurs the disk head movement distance of 20 tracks. A different task schedule <A, B, C, D, E> makes the disk head to move the distance of 8 tracks. Because the order of the task schedule doesn't change the distance of disk head movement within each task, we need to optimize only the distance of disk head movement between tasks. Considering only the head movement distance between tasks, the former schedule takes 16 tracks while the latter takes only 4 tracks. The mapping from the disk scheduling problem to the TSP is straightforward. For each task, a city is introduced. The disk head movement distance matrix between tasks becomes the distance matrix for the TSP. Then, the minimum tour of the TSP is the best schedule of the disk scheduling problem.

1.4 Paper Organization

The rest of the paper is organized in the following way. We first survey the related research and summarize the background material to make the paper self-contained. In Section 4, we formulate, analyze, and solve using the TSP algorithmic methods processor block-scheduling problem for both single and multiple processor cases. Section 5 introduces the hard real-time disk scheduling problem and optimization algorithm. Finally we make conclusions in Section 6.

2.0 Related Research

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

Scheduling has been widely studied in many areas, such as behavioral synthesis [McF90], parallel processing [Gon77], and hard real-time systems [Liu73]. Majority of DSP scheduling techniques have been presented in behavioral synthesis literature [McF90]. Initial scheduling and transformation techniques for block processing are presented in [Ziv94, Bha95]. Disk scheduling recently attracted a great deal of interests. While initially disk scheduling algorithms were studied mainly in connection with time-sharing general purpose computing [Teo72], more recently focus has been on disk-based multimedia scheduling algorithms [Dai94]. Also, recently, algorithm and architecture selection has been recognized as an important system-level synthesis topic [Pot94].

The traveling salesman problem(TSP), which is known to be NP-complete [Gar79], 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 [Law85]. Through the extensive studies of the past on the symmetric TSP, a number of efficient and effective solution techniques for the symmetric TSP that achieve solutions within a few percentage points of optimal in practice have been developed [Joh90]. According to the comprehensive empirical comparison of heuristics for the symmetric TSP by Johnson [Joh90], the most efficient and effective heuristics are 3-Opt and Lin-Kernighan [Lin73] algorithms. Reinelt [Rei92] has effectively reduced the running times of the above successful algorithms by considering only $O(n)$ candidate edges for the tour. Martin et al. [Mar92] have given an effective way to improve the quality of the solutions on the earlier successful heuristics while increasing running times.

Although a few solution methods for the asymmetric TSP exist, which includes a heuristic by Kanellakis and Papadimitriou [Kan80], which modifies the Lin-Kernighan symmetric TSP heuristic and a branch-and-bound approach by Miller and Pekny [Mil91], there do not exist enough experimental data to judge their performance and the heuristics have very high time complexity.

3.0 Background Material

3.1 Computation and Hardware Models

Our computational model for block processing scheduling is task level synchronous dataflow [Lee95]. No preemption is allowed, due to high context switching cost. An algorithm from a variety of algorithmic choices is to be chosen for each block to implement its specified functionality. The selection of an implementation platform among available implementation platforms for the chosen algorithm should be also made. For each implementation platform available, the processing times for the execution of blocks and the context switching times between the algorithms of any two blocks are assumed to be provided. Once algorithm and architecture selections are made and the schedule of blocks is given, the time to take for the periodic schedule per period can be computed easily from the given processing times and context switching times for the selected algorithms and implementation platforms.

Low power consumption has been established as an important metric for VLSI design. The capacitive power dissipation, which is the dominant component of power consumption in CMOS design is equal to $V_{dd}^2 * C_{eff} * \alpha$, where V_{dd} is supply voltage, C_{eff} is switching capacitance and α is activity. For programmable processors, extensive empirical studies show that power consumption is proportional to the number of operations. Another optimization degree of freedom is supply voltage. There is a quadratic dependence between power and supply voltage. In order to reduce voltage, one has to accept operation at low speed. The exact delay-V_{dd} curve is presented in [Cha95].

Description of magnetic disks can be found in [Hen90] and [Jor96]. Each disk is divided in concentric circles, tracks. There are typically several thousand tracks per disk. A seek moves the disk head (arm) from track to track. Several techniques have been proposed for analytic and empirical modeling of access data [Rue94, Wor95]. The common denominator in all of them is that longer distance which arm has to travel corresponds to larger time overhead. We assume linear dependency between seek time and travel distance [Hen90].

3.2 The Traveling Salesman Problem

As shown in the Sections 1.2 and 1.3, the scheduling problems we address can be transformed into an asymmetric TSP. Due to the current status of the available asymmetric TSP heuristics, we transform an asymmetric TSP into a symmetric one by the technique proposed by Jonker and Volgenant [Jon83], which would allow us to apply the efficient and effective symmetric TSP heuristics directly.

Let TSP(C,D) denote an asymmetric TSP with a set of n cities $C = \{1, 2, ..., n\}$ on a distance matrix D. Let TSP(C', D') 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, $D'[i][j] = -M$, where M is a very large positive number. If $(i \le n \text{ and } j \le n)$ or $(i > n \text{ and } j > n)$, $D'[i][j] = \infty$. Otherwise, $D'[i][j] = D[i][j]$ if $i > j$ D' and $[i][j] = D[j][i]$ *if* $i < j$. They show that any optimal solution of the transformed symmetric TSP corresponds to an optimal solution of the original asymmetric TSP. Feuer and Koo [Feu83] have successfully applied the similar transformation to the scan chain ordering problem.

Recent studies [Joh90] suggest that the most efficient and effective heuristics for the symmetric TSP are 3-Opt and Lin-Kernighan [Lin73] algorithms. These algorithms apply the global optimization strategy of iterative greed 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. Reinelt [Rei92] has reduced the running times of the above successful algorithms with little loss of the solution quality by considering only the k nearest neighbors of each city with k between 5 and 20 for the tour. Martin et al. [Mar92] have given an effective way to improve the quality of the solutions on the earlier successful heuristics, which is called as the large-step markov chain(LSMC) heuristic and now considered to be the "champion" heuristic for symmetric TSP. LSMC starts with an arbitrary initial local minimum and keeps improving on it by "perturbation + local optimization" cycles. LSMC can be seen as a variant of simulated annealing which moves among local minima only. In this paper, the LSMC heuristic with the Reinelt's nearest neighbor-based fast implementation ($k = 25$ was chosen to be more conservative) of the 3-Opt local optimization engine has been used. The quality of the TSP tour can be traded off with the running time by adjusting the number of the 3-Opt local optimization algorithms applied. LSMC heuristic with 10 and 100 iterations of 3-Opt local optimization algorithms has been used for all the examples of processor block-scheduling and disk scheduling, respectively.

4.0 Processor Block-Scheduling

First, we briefly outline key underlying assumptions. After that, we formulate the problems of throughput, cost and power optimization using block-scheduling, algorithm and architecture selection. Finally, we present effective TSP-based heuristics for the problems and provide experimental results.

4.1 Problems Formulation

We assume that there are no data, control, or timing dependencies among blocks. For each block, a number of algorithm choices exist. There exist a number of processor types or implementation platforms. As discussed in the earlier sections, the switching time between the algorithms of any two blocks is considered as well as the running times for the processing of blocks. We consider the two different design methods; (i) only one processor is chosen for all the blocks. (ii) multiple processors which can be different processor types are chosen, where each block is assigned to one of the processors.

4.1.1Throughput Optimization

The processor block-scheduling problem for throughput optimization can be formally stated as follows:

Problem: Throughput Optimization Using Block Scheduling, Algorithm and Architecture Selection

Instance: N blocks and P different processor types are given. Block i has a choice of B_i different algorithms. For each processor type, given are the running time array R for all the available algorithms and 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?

4.1.2Cost Optimization Under Throughput Requirement

The Cost Optimization Under Throughput Requirement Using Block Scheduling, Algorithm and Architecture Selection can be formulated in the same way as the throughput optimization problem. The only difference is that the targeted goal is the minimization of the cost under the throughput requirement.

Problem: Cost Optimization Under Throughput Requirement Using Block Scheduling, Algorithm and Architecture Selection

Instance: N blocks and P processor types are given. The cost array C of the processor types is given. Block i has a choice of B_i different algorithms. For each processor type, given are the running time array R for all the available algorithms and switching time matrix S between the algorithms. A positive constant L is given.

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

The Power Optimization Under Throughput Requirement Using Block Scheduling, Algorithm and Architecture Selection and Voltage Scaling can be formulated in the same way as the throughput optimization problem. The differences are that the goal of the optimization is the minimization of the power consumption under the throughput requirement and the voltage scaling is performed in addition.

Problem: Power Optimization Under Throughput Requirement Using Block- Scheduling, Algorithm and Architecture Selection and Voltage Scaling

Instance: N blocks and P different processor types are given. The cost array C, clock cycle time array CCT, array of power dissipation per unit time PD and an initial voltage V of each processor type are given. Block i has a choice of B_i different algorithms. For each processor type, given are the running time array R for all the available algorithms and switching time matrix S between the algorithms. A positive constant L is given.

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-based Block-Scheduling on Single Processor

4.2.1 Algorithm

We proved that the problem is NP-complete by using Karp's polynomial transformation technique form TSP. An iterative improvement algorithm based on TSP heuristic has been developed. First, the heuristic for throughput optimization is given by the following pseudo-code, where the threshold t is chosen to be 100 for the performed experiments.

A heuristic for throughput optimization:

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

Next, the heuristic for cost optimization under throughput requirement is built by separately applying the heuristic for throughput optimization to each processor type and then choose the cheapest processor type which satisfies the throughput requirement.

Lastly, the heuristic for power optimization under throughput requirement is given by the following pseudo-code.

A heuristic for power optimization under throughput requirement:

```
Set the best_so_far solution to have the power consumption ∞.
For each processor type,
  Using the heuristic for throughput optimization on the processor
  type, a block-schedule is generated.
  Scale the voltage so that the block-schedule just satisfies the
  throughput requirement.
  Compute power consumption.
  If the power consumption is less than the best-so-far, the current
  solution becomes the best-so-far.
Return the best-so-far solution.
```

4.2.2Experimental Results

For all the randomly generated examples, we assume that there are 5 available processor types, each block can have between 1 and 5 algorithmic choices, and the running times and switching times are randomly chosen from the numbers between 1 and 100. The costs of the processors are between 100 and 200. The clock cycle times range between 5.0 and 10.0. The power dissipations per unit time are drawn from 5.0 to 10.0. The initial voltages for the processor types are chosen between 3.0 and 5.0. The threshold voltage is assumed to be 0.7. All the random solutions are generated from 100 random solutions for each example. The best of the 100 random solutions is reported and compared to the heuristic solution. For each processor type, a lower bound for clock cycles is computed in the following way. An algorithm which has the smallest processing + switching times on the processor type is chosen for each block. The sum of the processing and switching times for all the blocks with the best algorithms is a lower-bound for the processor type.

Processor cost = <171, 126, 165, 119, 155>
Initial voltage = <4.2, 5.0, 4.3, 4.3, 4.2>
Power dissipation per unit time = <8.4, 6.9, 6.4, 9.7, 5.8>
Clock cycle time = <9.1, 10.0, 7.1, 7.4, 5.9>
Lower-bound for clock cycles = <1701, 1481, 1311, 1480, 1657>

(1) Throughput Optimization

	Random Solution	Optimized Solution
Processor	2	3
Schedule length	4304	1476

(2) Cost Optimization Under Throughput Requirement
No random solution satisfies even the maximum sample period.

Sample period in clock cycles	1900	1800	1700	1600	1500
Schedule length	1653	1653	1653	1476	1476
Processor	4	4	4	3	3
Cost	119	119	119	165	165

(3) Power Optimization Under Throughput Requirement
Sample period in unit time = 50000

	Random Solution	Optimized Solution
Processor	5	3
Initial voltage	4.2	4.3
Scaled voltage	2.77	1.75
Clock cycles	4508	1476
Power dissipation per unit time	2.53	1.06
Clock cycle time	11.09	33.88
Power consumption	126590.44	53124.29

Figure 2: Experimental Results for the Example of 50 Blocks on Single Processor

Figure 2 illustrates the effectiveness of the TSP-based heuristic for the optimization of the three important metrics in the design. For the example shown, the average improvement is by a factor of 2.9 and 2.4 for throughput and power respectively. For the cost optimization under throughput requirement, no random solution satisfies even the maximum sample period tried. Note that if only the switching times are considered, then the improvement would be greater.

4.3 TSP-based Block Scheduling on Multiple Processors

4.3.1 Algorithm

An exhaustive-enumeration algorithm based on TSP heuristic and lower-bound estimations has been developed for cost optimization under throughput requirement. Similar to the lower-bound estimation technique for a single processor block-scheduling problem, a simple and effective lower-bound estimation technique for throughput optimization on multiple processors has been derived. The lower-bound is computed given the allocation of processors. An algorithm which has the smallest sum of processing and switching times over all the processor types in the allocation is chosen for each block. The sum of the processing and switching times for all the blocks with the best algorithms divided by the number of processors allocated is a lower-bound for the multiple processor block-scheduling problem given the allocation of processors. Using this lower-bound, many infeasible allocations are pruned. The heuristic for cost optimization under throughput requirement is given by the following pseudo-code.

A heuristic for cost optimization under throughput requirement:

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

A heuristic for throughput optimization on multiple processors:

```
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 is less
than 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.
```

The threshold t is chosen to be 10 and the value of k is set to 5 to allow escaping from the local minima.

4.3.2Experimental Results

For all the randomly generated examples, we assume that there are 5 available processor types, each block can have between 1 and 5 algorithmic choices, and the running and switching times are randomly chosen from the numbers between 1 and 100. The costs of the processors are between 100 and 200. Random solutions are generated by 100 random assignments and schedules of blocks to processors which are allocated by the exhaustive enumeration of all the allocation possibilities in the order of increasing cost. The average improvement obtained is by a factor of 3.18 for the examples.

	Sample Period									
	1500		1000		750		600		500	
	Random	Optimized	Random	Optimized	Random	Optimized	Random	Optimized	Random	Optimized
Cost	312	104	520	208	766	208	1040	312	1408	416
Schedule Length	1472	1389	970	791	738	749	577	595	458	486

Table 2: Experimental Results of the Cost Minimization for the Examples of 50 Blocks on Multiple Processors.

5.0 Disk Scheduling

In this section we present disk scheduling problem, discuss its complexity, and present optimization algorithms and experimental results.

5.1 Problem Formulation and Complexity

We assume that there are no data, control, or timing dependencies among tasks, and that no preemption of tasks is allowed. The distance which the disk head should travel from block i to block j is assumed to be the number of the tracks between the center of block i and the center of block j. In this section we present the formulation of the disk scheduling problem for the disk-based application specific systems to minimize the disk head movement and establish its computational complexity. First, we define the disk scheduling problem.

Problem: The Disk Scheduling Problem

Instance: A set of M tasks described by the disk block access sequence, and a disk block assignment in disks is given.

Question: Is there a static periodic schedule of the tasks such that the total distance of the disk head movement for the schedule is at most D?

We proved, using the standard Karp's polynomial reduction technique from TSP that the disk scheduling problem is NP-complete.

5.2 Optimization Algorithm

Using the transformation method shown in the earlier section, the disk scheduling problem is transformed into an instance of the TSP. Because the original problem is directly transformed into TSP, only one TSP needs to be solved. LSMC heuristic with 100 iterations of 3-Opt local optimization algorithms has been used for all the examples.

5.3 Experimental results

Tables 3 and 4 illustrate the effectiveness of the algorithm for the minimization of disk head

Number Of Tasks	Number Of Blocks	Worst Random Schedule	Average Random Schedule	Best Random Schedule	Average Optimized Schedule	Best Optimized Schedule	CPU Time [sec]
100	10	382	351.6	322	18	18	4.89
100	50	1786	1634	1452	94	94	1.64
100	100	3914	3542.2	3228	198	198	1.73
200	50	3498	3218.6	3088	98	98	4.39
200	100	7082	6656.4	6042	194	194	3.75
200	200	13932	13213.8	12218	396	396	2.81
300	100	10524	9969.4	9270	196	196	6.08
300	200	20880	19031.4	17550	398	398	5.00
300	300	31602	28728.2	26844	592	592	4.52
500	100	17068	16823.2	16422	198	198	12.90
500	300	53210	50584	46964	596	596	7.49
500	500	88354	82548.4	78418	996	996	7.70

Table 3: Type 1 Examples of the Disk Scheduling Problem

Number Of Tasks	Number Of Blocks	Worst Random Schedule	Average Random Schedule	Best Random Schedule	Average Optimized Schedule	Best Optimized Schedule	CPU Time [sec]
100	10	504	469	436	156	156	6.86
100	50	2708	2567.4	2394	1000	1000	2.90

Table 4: Type 2 Examples of the Disk Scheduling Problem

Number Of Tasks	Number Of Blocks	Worst Random Schedule	Average Random Schedule	Best Random Schedule	Average Optimized Schedule	Best Optimized Schedule	CPU Time [sec]
100	100	5084	4972.4	4848	2016	2016	2.67
200	50	5036	4721.6	4468	1802	1802	8.39
200	100	11498	10958.6	10548	4560	4560	9.97
200	200	20648	19850.8	18784	7750	7750	8.91
300	100	15986	15395	14908	5618	5618	14.55
300	200	33070	31818.6	30160	12980.6	12976	17.21
300	300	49204	47708.4	46452	19843	19842	26.33
500	100	25408	24669.6	23956	8452.8	8436	25.80
500	300	74814	73460.8	72460	24878.6	24872	28.91
500	500	137228	133228.2	130540	54457	54432	53.97

Table 4: Type 2 Examples of the Disk Scheduling Problem

movement. The heuristic has been applied to two types of examples. In the examples of the first type, all the tasks access only one disk block. For the second type, each task accesses one or two blocks. Examples are randomly generated for the various combinations of the number of tasks and blocks. First, the task scheduling is done randomly 100 times, and the worst, the average, and the best distances are shown for each example. For the type 1(2) examples, the average improvement of disk head movement distance by a factor of 43.3(2.5) compared to the best random solution has been achieved. For the type 2 examples, the distance that the disk head moves within each task is responsible for the most of the total head movement distance of the optimized solutions.

6.0 Conclusion

We introduced a new approach for addressing a large family of modern digital signal processing scheduling problems. The approach is based on transforming a given scheduling problem into traveling salesman problem and employing available exceptionally efficient heuristics for solving the transformed problem. In particular, we addressed two new problems: block processing scheduling with simultaneous architecture and algorithm selection and disk scheduling. The effectiveness of the algorithms is demonstrated on numerous examples.

7.0 References

[Bha95] 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, Osaka, Japan, October 1995

[Cha95] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, R. Brodersen, “Optimizing Power Using Transformations”, *IEEE Transactions on CAD*, Vol. 14, No. 1, pp. 13-32, January 1995.

[Dai94] S.J. Daigle, J.K. Strosnider, “Disk Scheduling for Multimedia Data Streams”, Proc. SPIE High-

Speed Networking and Multimedia Computing, Vol. 2188, pp.212-223, 1994.

[Feu83] 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.

[Gar79] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. H. Freeman and Co., 1979.

[Gon77] M.J. Gonzales, "Deterministic Processor Scheduling", ACM Computing Surveys, Vol. 9, No. 3, pp. 173-204, 1977.

[Hen90] J. Hennessy, D. Patterson, Computer Architecture: A quantitative approach, Morgan Kaufmann, 1990.

[Joh90] 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, July 1990.

[Jon83] R. Jonker and T. Volgenant, "Transforming Asymmetric into Symmetric Traveling Salesman Problems", Operations Research Letters, Vol. 2, No. 4, 1983.

[Jor96] F. Jorgensen, "The Complete Handbook of Magnetic Recording", 4th ed, TAB Books, New York, NY, 1996.

[Kan80] P.-C. Kanellakis and C. H. Papadimitriou, "Local Search for the Asymmetric Traveling Salesman Problem", Operations Research, Vol. 28, pp. 1086-1099, 1980.

[Kun88] S.Y. Kung, VLSI Array Processors, Prentice Hall, Englewood Cliffs, NJ, 1988.

[Law85] 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.

[Lee95] E.A. Lee, T.M. Parks, "Dataflow Process Networks", Proc. of the IEEE, Vol. 83, No. 5, pp. 773-799, 1995.

[Lin73] S. Lin and B. W. Kernighan, "An effective Heuristics Algorithm for the Traveling-Salesman Problem", Operations Research, Vol. 31, pp. 498-516, 1973.

[Liu73] 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.

[Mar92] 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.

[McF90] 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, February 1990.

[Mil91] D. L. Miller and J. K. Pekny, "Exact Solution of Large Asymmetric Traveling Salesman Problems", Science, Vol. 251, pp. 754-761, 1991.

[Pot94] M. Potkonjak, J. Rabaey, "Algorithm Selection: A Quantitative Computation-Intensive Optimization Approach", *ICCAD94 International Conference on Computer-Aided Design*, pp. 90-95, November 1994.

[Rei92] G. Reinelt, "Fast Heuristics for Large Geometric Traveling Salesman Problems", ORSA Journal on Computing, Vol. 4, No. 2, pp. 206-217, 1992.

[Rue94] C. Ruemmler, J. Wilkes, "An introduction to disk drive modeling", IEEE Computer Magazine, vol.27, (no.3):17-28, 1994.

[Sri95] M. B. Srivastava, M. Potkonjak, "Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput", IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 3, No. 1, pp. 2-19, 1995.

[Teo72] T.J.Teorey, T.B. Tinkerton, A comparative Analysis of Disk Scheduling Policies, Communications of the ACM, Vol. 15, No. 3, pp. 177-184, 1972.

[Wor95] B.L. Worthington, G.R. Ganger, Y.N. Patt, J. Wilkes, "On-line extraction of SCSI disk drive parameters", Performance Evaluation Review, vol.23, (no.1):146-56, 1995.

[Ziv94] V. Zivojnovic, S. Ritz, H. Meyr, “Retiming of DSP programs for Optimum Vectorization”, ICASSP-94 International Conference on Acoustic, Speech, and Signal Processing, Vol. 2, pp. 465-468, 1994.