Throughput Optimization in Disk-Based Real-Time Application Specific Systems

Stephen Docy, Inki Hong, and Miodrag Potkonjak
Computer Science Dept., University of California, Los Angeles, CA

ABSTRACT

Traditionally, application specific computations have been focusing on numerically intensive data manipulation. Modern communications and DSP applications, such as WWW, interactive high resolution TV, video-on-demand, and wireless communications, however, are increasingly more data management oriented. At the same time, the technological trends indicate that magnetic disk system performance is rapidly becoming a principal component in overall system behavior.

In this paper, we initiate behavioral and system level research in synthesis of disk-based application specific system by introducing cost and throughput optimization problems for synthesis of disk-based application specific systems (DASS). We formulate, establish computational complexity, and develop efficient optimization algorithms for data assignment on a single and multiple disks so that the seek time overhead is minimized. Extensive experimental results clearly indicate the importance of the synthesis problem and effectiveness of the proposed optimization algorithms.

1.0 Introduction

1.1 Motivation: Why Synthesis of Disk-based Systems?

The rapidly increasing importance of massive storage systems in application specific systems is the consequence of convergence of application and technological trends. Both recent trends and future market prediction indicate that emphasis in application specific systems has been shifting from the data manipulation paradigm to the data management paradigm. Internet-based applications (e.g. world wide web), video-on-demand, interactive television, and wireless communications are only a subset of exciting and economically important modern data-management based application specific systems.

At the same time, all technology trends also indicate that key design metrics of modern and future application specific designs, such as throughput, latency, power, weight, and cost, will be dominated by massive storage elements. Both modern microprocessor and ASIC technology is advancing at exponential rates and speedups of more than 50% compounded annually have become the standard. Although, disk capacity increase and disk cost decrease follow even faster trends (between 60 and

80%), performance of disks has been improving at rate smaller than 10% compounded annually. And disk is already a bottleneck in current computing and communication systems. While the trends do not indicate that data manipulation is not important anymore, it is clear that dominating bottlenecks of a system are in data management of massive storage systems.

Traditional behavioral synthesis has been data-path centered, where even storage elements in data-path received only secondary attention in comparison with execution units and interconnect. Although, recently, there has been several efforts which addressed design of background memory in application specific systems, it is the first effort, to the best of our knowledge which addresses synthesis and optimization of disk-based application specific systems.

1.2 Motivation Examples

In disk-based application-specific systems, tasks may require data from disks for its execution. Given a static periodic schedule of tasks, the disk head movement distance for the given task schedule will be significantly different and so will be the throughput of the system, depending on how the data blocks are stored on a disk. We make some assumptions here to clarify the explanations to be made in this subsection, though the assumptions will be furnished in more details in the later sections. We assume that the data block for a task is stored in contiguous integer multiples of cylinders. The disk head movement distance between two blocks is also assumed to be the number of disk cylinders between the center points of the two blocks. We consider a small example of 10 tasks where a single disk is used for the storage of all data blocks and each task requires one disk cylinder for storage of its data. Given a task schedule <7, 6, 1, 9, 7, 2, 5, 8, 4, 10, 6, 7, 1, 6, 4, 6, 3, 1, 4, 8>, a positioning solution <6, 8, 2, 1, 3, 10, 5, 9, 4, 7> results in 100 cylinders of disk head movement per period while a different positioning solution <2, 5, 8, 10, 4, 6, 7, 1, 3, 9> incurs only 36 cylinders of disk head movement per period. Even on this small example, a properly selected positioning solution can improve the resulting throughput almost by a factor of 3.

It is common that the multiple disks are used for the storage of data blocks to exploit economies of scale. Some video-on-demand storage server contains more than 100 disk drives. In the limit case, if a disk can hold only a single cylinder of data and the same number of disks as that of tasks are used, the disk head movement distance becomes zero. However, the minimum size of available disks is usually much larger than the data requirement for a single task. Thus, we only deal with the cases when a few disks are available, where they can just fit in data for all the tasks and each disk must store data for many tasks. We consider the same example of 10 tasks. The two disks with 5 cylinders each are available. The problems that we need to address are how to partition the tasks and where in each disk the partitioned tasks are positioned. A partitioning and positioning solution <Disk 1: 9, 1, 6, 5, 10 Disk 2: 3, 4, 8, 7, 2> makes the head to move 24 disk cylinders per period for the given schedule. The use of 2 smaller disks reduced the disk seek time by a factor of 1.5 from the larger single disk solution.

1.3 Paper Organization

The rest of the paper is organized in the following way. We first present the background material which for the sake of completeness includes a short disk technology description and several disk

timing models. After presenting related works in Section 3, Section 4 contains synthesis and optimization core of the paper, where the problem we address is formally defined and technical details of the approach for reducing seek time in application specific systems are presented. After presenting extensive experimental results in Section 5, we conclude the paper by outlining main contributions.

2.0 Background Material

In this section we first provide an overview of disk technology. After that, we briefly discuss the most popular timing models of a magnetic disk. We conclude the section, by explaining the selected hardware and computational models.

2.1 Disk Technology

Due to the superior trade-off with respect to common design metrics such as cost, memory capacity, latency, data input-output bandwidth and reliability in comparison with all other massive storage alternatives, magnetic disks have become the de-facto standard for providing non-volatile high volume memory in modern computer systems. Unfortunately, disk performance has become a major bottleneck in the performance of the entire system because of the following factors. Since disk performance is dependent on the speed at which its sizable mechanical components can physically move, magnetic disks are inherently much slower than other system-components, such as the processor, which are electronic in nature. Also, while the performance of magnetic disks has increased steadily over the years, the performance of other components has increased much more rapidly. Since any improvement in the time required to satisfy requests sent to the disk is generally mirrored by a proportional increase in overall system performance, developing efficient means of optimizing disk access time is crucially important. Note that this is not necessarily true for other system components which may spend a significant amount of time “waiting” on slower devices, such as the disk.

We now describe the basic concept of disks. The major components of a magnetic-disk are the platters and the read/write heads. The platters are circular surfaces with a metallic coating which allows them to retain a magnetic charge. Multiple platters are placed directly over each other, forming a cylindrical object. A gap is left between each of the platters to accommodate a read/write head. The read/write head floats just above the surface of the platter and, as the platter rotates beneath it, the head may detect (read) or alter (write) the charge of specific areas on the platter. All read/write heads are attached to an arm assembly which moves the heads either toward the center or toward the edge of the platters.

Each platter consists of a series of concentric circles called tracks. A track can be defined as the total area of a rotating platter that passes beneath a stationary read/write head. Thus, a head may read or write to any position within the current track without needing to move. Equivalent tracks on all platters are typically referred to as a cylinder. Each track, in turn, is made up of multiple, fixed-length sections called sectors. While the size of a sector may be different for various disk models, on any particular disk, the sector size is constant for all tracks on all platters. Because track size

decreases as you move toward the center of the platter, the number of sectors per track also decreases.

We now provide the typical values of parameters for most modern magnetic disks. Disks usually store 1-2 gigabytes of data. They have 14-18 platters which are sized between 1.3 and 8 inches (2.5 and 3.5 inches are most popular) and rotate 3600-7200 times per minute. The number of cylinders is 2000-2600. With the data transfer rate of 50-70 megabits per second and the average seek time of 9-15 milliseconds, it takes about 20-35 milliseconds for the access of 8K data block.

2.2 Disk - Timing Model

A seek moves the disk head (arm) from track to track. Recently, several techniques have been proposed for analytic and empirical modeling of seek time [Rue94, Wor95]. All the models share the common property that longer distance which arm has to travel corresponds to larger time overhead. Typical seek times for an IBM disk are given in Table 1 [Pat90].

seek distance range [tracks]	seek time [ms]
1 - 50	$1.9 + \sqrt{d} - d / 50$
51 - 100	$8.1 + 0.044 * (d - 50)$
101 - 500	$10.3 + 0.025 * (d - 100)$
501 - 884	$20.4 + 0.017 * (d - 500)$

Table 1: The typical seek times for an IBM disk.

Typical seeks times for two Hewlett-Packard disks are given in Table 2.

Disk Type	HP C2200A	HP 97560
seek time[ms] short	$3.45 + 0.597 \sqrt{d}$	$3.24 + 0.400 \sqrt{d}$
long	$10.8 + 0.012d$	$8.00 + 0.008 d$
boundary	616	383

Table 2: The typical seek times for two Hewlett-Packard disks.

2.3 Computational Model

We assume that each of tasks follows homogeneous synchronous data flow semantics and syntax individually [Lee95]. Therefore, each task is periodic, and is fully specified by its period and its running time of one iteration of an available processor. In the schedule period, a task can appear zero or a few times in the schedule. Since context switching overhead is usually high for modern systems, we assume no task preemption. Note that this preemption restriction, actually does not impact any of the proposed methods, since in all discussed design cases non-preemptive policies yield superior results in comparison with preemptive policies.

Furthermore, each task has a need to read or read and write data on the disk. We assume that for each task a set of continuous cylinders is allocated, where the required data for the task is stored. Seek time is proportional to distance which disk's head has to travel. The goal is to properly position task data blocks so that given a periodic static task schedule, the disk seek time is minimized.

3.0 Related Work

In this section we first survey information about magnetic disks, mainly about their architecture, technical details, and massive storage alternatives. Next, we describe the disk modeling approaches. Lastly, we provide the pointers to the references of the partitioning problem which is related to our optimization problem.

Although there have been constant stream of alternative massive storage technologies, such as charge-coupled devices (CCD), bubble memories, optical and magneto-optical disks, semiconductor disks, rhodopsin-based biological memories, and holographic storage, magnetic disks have been dominated secondary storage since the mid sixties. They provide superior trade-off of cost, memory capacity, bandwidth, and reliability in comparison with all other alternatives. Detailed description of magnetic disks can be found in many books [Cam88, Hoa91, Jor96, Mee90]. An introductory exposition of basic disk principles is also given in modern architecture [Pat90] and operating systems [Pet85] textbooks. Relative comparison of magnetic disks with other massive storage alternatives is given on [Dou94]. Wood and Hodhes [Woo93] survey state-of-the-art and technology trends in direct access storage devices, mainly magnetic disks. In particular, they stress that all data and technology developments indicate that the relative cost of magnetic disks will decrease at least as fast as the cost of semiconductor memories.

The early disk-related research in operating systems has been focused on development of scheduling algorithms for efficient use of high-volume storage in time-shared mainframes [Den67, Teo72, Hof80]. Later, operating systems researchers developed new disk scheduling algorithms for new general-purpose computing platforms assuming increasingly more realistic and complex disk models [Gei87, Sel90].

Disk modeling recently attracted a great deal of interests [Pat90, Rue94, Wor95]. While initially very simple linear models have been used [Won83], the most recent models include variety of technical effects, including statistically validated models [Wor95]. While there is an abundance of macroscopic models for disks, there are surprisingly few detailed quantitative descriptions of mechanical, electrical, and magnetic components of a disk [Wag92].

One of our key optimization involves partitioning. Partitioning has been popular topic. Numerous partitioning techniques at physical design level of abstraction have been surveyed by Alpert and Kahng [Alp95]. Behavioral synthesis partitioning techniques were initially studied by McFarland [McF83], Rajan and Thomas [Raj85], and Camposano and Brayton [Cam87]. Lagnese and Thomas [Lag89] generalized their work by considering multi-stage clustering. Gupta and De Micheli [Gup90] proposed both simulated annealing and heuristic approaches for partitioning of programmable and hardwired designs [Gup90]. Other notable efforts in partitioning include [Kuc91, Pra91, Vah92, Rao92]

4.0 Problems Formulation, Graph-Theoretic Abstraction, Complexity, and Optimization Strategy

In this section, we formulate the problem of throughput optimization. Next, we show how to reformulate the problem into a graph-theoretic problem. After that, we establish the computational complexity of the problem by showing that the problem is NP-complete. Lastly, we provide our optimization strategies for the problem.

4.1 Problems Formulation

The throughput optimization using positioning problem is defined using the standard Garey-Johnson [Gar79] format.

Problem: Disk Head Movement Minimization using Positioning

Instance: Given a schedule of tasks, the data requirements of the tasks and a disk which just fits in all data blocks.

Question: Is there a positioning of data blocks for the tasks in a disk such that the disk head movement distance for the given schedule is at most *L*?

The throughput optimization using partitioning and positioning problem is similarly defined in the standard Garey-Johnson format.

Problem: Disk Head Movement Minimization using Positioning and Partitioning

Instance: Given a schedule of tasks, the data requirements of the tasks and *K* disks which just fit in all data blocks.

Question: Are there a partitioning of data blocks for the tasks into the *K* disks and a positioning of assigned data blocks on each disk such that the disk head movement distance for the given schedule is at most *L*?

4.2 Graph-Theoretic Abstraction

The Disk Head Movement Minimization Using Positioning problem can be reformulated using the graph-theoretic abstraction. The given schedule of tasks can be described by a undirected weighted graph *G(V,E)* of tasks. The weight *W(i,j)* of an edge between task *i* and *j* is the number of times that the two tasks *i* and *j* are adjacent in the schedule. Let *D(i,j)* denote the disk head movement distance between the blocks *i* and *j* in a disk. The disk head movement distance for a given schedule is the sum of *W(i,j)*D(i,j)* over all *(i,j)* in *E*. When a graph *G* is not weighted and each task requires only one track for its data, the resulting problem is reduced to the Optimal Linear Arrangement which is NP-complete [Gar79]. For a special case when the graph of *n* nodes is a tree, there exists an optimal algorithm whose running time is O(nlogn) [Ado73].

The Disk Head Movement Minimization Using Positioning and Partitioning problem can be also reformulated using the similar graph-theoretic abstraction. Once a partition is performed, for each partition of tasks, a separate schedule can be constructed from the original schedule by considering only the tasks in the partition. For each schedule, a undirected weighted graph *G(V,E)* of tasks is generated. The disk head movement distance for the original schedule is the sum of the disk head

movement distance for each partition which is computed in the same way as described above.

4.3 Computational Complexity

We only need to prove that the Disk Head Movement Minimization Using Positioning problem is NP-complete because the Disk Head Movement Minimization Using Positioning and Partitioning problem is reduced to it when only a single disk is available. We proved, using the standard Karp's polynomial reduction technique that the Disk Head Movement Minimization Using Positioning problem is NP-complete, with the transformation from a known NP-complete problem, Optimal Linear Arrangement [Gar79] into an instance of our problem.

4.4 Optimization Strategy

Since the computational complexity of the throughput optimization problem forbids an exact or optimal solution, an efficient and effective heuristic method has been developed for the problem. We first describe our heuristic for the positioning with the assumptions of uniform task data requirement and uniform cylinder capacity, and then show how to extend it to handle the positioning problem without the restrictions. After that, we discuss our heuristic for the partitioning problem.

4.4.1 Positioning with Uniform Task Data Requirement and Uniform Cylinder Capacity

Given the order of disk block accesses, intelligent decisions can be made when deciding how to place the data blocks. Proper arrangement of the data on the disk improves the efficiency of disk access. The goal of such an efficient arrangement would be to minimize the seek time required when moving the read/write head to the location of the data for the next task to be run. In order to simplify the construction of such an arrangement, it is initially assumed that the data for each task will occupy exactly one cylinder on the disk. While this may be unrealistic, especially given the varying capacities of different cylinders, it enables clearer explanation of the essence of the problem and heuristic techniques. Later in this section we examine the situation in which the number of cylinders required varies from task to task.

Given a task schedule, the cost of any specific arrangement can be determined. The cost of an arrangement is defined as the number of disk cylinders that must be traversed during the entire schedule period. Whenever one task finishes and another begins, the read/write head must be moved to the location of the data for the new task. Obviously, this distance depends on the arrangement of the task data on the disk. If the data is located on an adjacent cylinder, the cost of moving is 1. If the head must move over k intervening cylinders to reach the appropriate data, the movement cost is k+1. Each time the head moves between data locations the total cost is increased by the required amount.

We now present our heuristic approach for addressing this computationally intractable problem. The following is the pseudo code for our heuristic.

```
Sort tasks in descending order of scheduled turns
Task data will be placed according to this order
Place first two tasks
While there remain tasks to be placed
```

```
Choose next task
Compute cost of all possible placements for this task
Choose the placement with the lowest cost
Update current task placement to reflect this choice
```

Our heuristic arranges task data one task at a time. A task may be assigned to any position, even between tasks that are already placed, but once the task has been added, the group of tasks may not be rearranged. Subsequent tasks may be inserted anywhere within the group, but the order of the previous tasks is not altered. By doing this, the number of combinations which is examined is reduced from $n! / 2$ to less than $(n^2 + n) / 2$.

Since many possible arrangements are eliminated from consideration, great care must be taken to insure that the remaining arrangements contain a solution which is as close to optimal as possible. And the quality of the final arrangement chosen by our heuristic is very dependent on the order in which the tasks are placed. In an attempt to arrive at a simple, yet meaningful, criterion for ordering the task placement, the number of times each task is scheduled during the period is totaled. Since tasks need to access their data every time they run, the total number of schedule times appears to be a useful measure when ranking the tasks for placement. This is much simpler, though surely less accurate, than trying to determine the relationship between each of the tasks.

Once the tasks have been ranked according to their respective number of scheduled turns, they are sorted into descending order and their data is placed with respect to this order. Tasks that will be scheduled frequently during the period are placed first and those that will be scheduled the least (or not at all) are placed last. The order of placement for tasks with equal rank is not specified by the algorithm, but the current implementation places them in order of task number. This is one area in which a slightly more complex scheme could decide more intelligently on the order of task placement. This placement ordering was arrived at during the initial development of our heuristic. Initially, tasks were added only to the beginning or at the end of the group of previously arranged tasks. They were not inserted between tasks which were already in place. This arrangement scheme was based on the premise that frequently accessed data should be placed on cylinders near the middle of the disk, while seldomly used data should be kept out of the way on cylinders near the edge or near the center of the disk. In this way, the maximum seek distance for regularly accessed data would be kept to around half the disk. This algorithm reduced the number of combinations that were examined to 2n. While this resulted in good performance, the solutions offered were often poor. By expanding the arrangements considered to include all possible placements for new tasks, rather than only two, much better results were obtained.

4.4.2 Positioning with Nonuniform Task Data Requirement and Nonuniform Cylinder Capacity

Restricting task data to a single disk cylinder is often unrealistic assumption. Expanding our heuristic to consider different cylinder needs for each task would allow the heuristic to model much more reasonable situations. While introducing this limitation did much to simplify the design of the heuristic, the resulting algorithm is simple enough that it should be possible to extend its capabilities. There are two factors which could result in a task needing more than a single cylinder

for its data. The first is the size of the data that needs to be stored for the task and the second is the varying capacity of disk's cylinders. Each of these has different implications and will be looked at individually.

As the amount of data for a task increases, so may the required number of cylinders. Since we are not yet concerned with the varying capacity of different cylinders, the number of cylinders required by a particular task would remain constant regardless of where on the disk the data is placed. If this quantity is known, changing the cost calculation function to take the data size into consideration would be quite simple. Instead of adding one for each task crossed while seeking, the number of cylinders required by each task crossed would be added. A much more complex problem may be deciding in what order to arrange the data within the assigned cylinders. Since the read/write head may be moving in either direction when traveling to the data, the layout of the data will affect the distance required to reach the beginning of the data. Thus, it is reasonable that the distance is the number of cylinders between the center points of two disk blocks. Also of importance is the effect the varying cylinder requirements should have on the order in which the tasks are assigned for placement. Using an approach similar to the one originally used to decide upon placement order, it would seem reasonable to place tasks which require the fewest cylinders in the middle of the disk, and to place those which require the largest number of cylinders near the edge or near the center of the disk. This would attempt to prevent frequent seeks over tasks which occupy a significant number of cylinders. Of course, the number of cylinders and the number of scheduled turns may result in conflicting desired placements. Where do you place a task which is scheduled frequently but requires many cylinders? Different approaches to placement ordering based on the frequency and the size of the tasks should be experimented with to find a reasonable solution.

Note that the disk itself also complicates matters. Because of the structure of the disk, cylinders decrease in size (and capacity) the closer they are to the center of the disk. So, while the size of the data required by each task may not change, the number of cylinders required to store the data will change depending on which cylinders are allocated. Once again, it should be relatively simple to allow the heuristic to calculate the actual number of cylinders required given the size of the task's data and its position within the arrangement. However, another side-effect of varying cylinder capacities is that reversals of arrangements, which were previously considered equal, may now have vastly different costs. The placement 0,2,3,1,4 and the placement 4,1,3,2,0 will now have to be considered independently. This can easily be done within the cost calculation function, but will double the already considerable amount of work performed in order to calculate the cost of each arrangement.

4.4.3 Partitioning

Another situation which merits attention is the case in which the task data is stored on multiple small disks rather than on a single large disk. This is an attractive implementation option and may be used to significantly improve the efficiency of disk access. Smaller disks have a smaller number of cylinders and, thus, the maximum seek distance required to access a particular cylinder is proportionally less. For example, with 100 tasks stored on a single, 100 cylinder disk, the maximum seek distance would be 99 cylinders. By storing the tasks on two, 50 cylinder disks, the maximum

seek distance is reduced to 49 cylinders. Also, with a greater number of active disks, the likelihood that the desired data is beneath (or within a few cylinders of) the current position of one of the read/write heads is increased. Of course, the increased cost of buying and maintaining multiple disks must be weighed against the benefit of increased performance. A balance between desired cost and desired efficiency must be found.

Once the number of disks has been chosen, it must then be decided how to arrange the task data on the disks. Once again, the simplicity of our heuristic makes this transition quite natural. Assuming that the tasks will be placed in decreasing order of scheduled turns (as with the current, single disk implementation), a logical way to begin would be to place one task on each of the disks. By placing the most frequently accessed data on separate disks, the greatest benefits of using multiple disks can be seen. Once these initial tasks have been placed, the heuristic can operate in the same basic fashion that it did with a single disk. For each subsequent task, consider all possible placements and choose the arrangement with the least cost. While the separation of data onto multiple disks will increase the total number of considered arrangements, the increase will not be very significant.

5.0 Experimental Results

5.1 Design Examples

We have generated random examples by varying the number of tasks and the schedule period. The numbers of the tasks considered are 50, 100, 150, and 200 to see the effects of problem size on the performance. For all the cases, we consider several different periods to see how the schedule length affects the heuristic performance. We have generated 10 different schedules for each case. For all the test examples, we assume the uniform task data requirement such that each task requires a single cylinder for its data, and the uniform cylinder capacity.

5.2 Experimental Results

Tables 3 and 4 present the experimental results for the test examples. The random solution statistics

Task	Period	Random Solution		Heuristic Solution		Improvements	
		Average	Best	Average	Best	Average	Best
50	50	613.2	502	182.8	157	3.35	3.20
50	100	1359.2	1300	587.6	538	2.31	2.42
50	150	2131.2	2028	1088.3	1026	1.96	1.98
100	100	2656.5	2560	734.1	662	3.62	3.87
100	200	5758.3	5560	2499.5	2309	2.30	2.41
100	300	8810.5	8632	4558.8	4350	1.93	1.98
150	150	6251.2	6036	1744.3	1647	3.58	3.66
150	300	13290.0	12458	5706.2	5135	2.33	2.43
150	450	20476.8	20280	10717.0	10311	1.91	1.97
200	200	11182.8	10866	2973.5	2914	3.76	3.73
200	400	23384.4	22876	9649.7	9333	2.42	2.45
200	600	36382.8	35688	18835.3	17859	1.93	2.00

Table 3: The results for the positioning problem

are generated from the best of 100 random solutions. Because we have generated 10 different schedules for each test case, we report the average and the best of the heuristic results.

Number of Disks	Random Solution		Heuristic Solution		Improvements	
	Average	Best	Average	Best	Average	Best
1	36382.8	35688	18835.3	17859	1.93	2.00
2	18477.3	18316	9585.2	9207	1.93	1.99
5	7322.0	7134	3672.7	3531	1.99	2.02

Table 4: The results of the example of 200 tasks with schedule period 600 for the partitioning problem.

5.3 Data Analysis

Experimental results illustrate the effectiveness of the throughput optimization using positioning and partitioning for the test examples. All the comparisons of the performances are based on the best results. Using positioning, our heuristic reduces the disk head movement by a factor of 2.68 for the examples from a random solution. We note that the heuristic solution quality does not depend on the number of tasks, but on the schedule period. The longer the schedule period, the less the improvement has been achieved. Using partitioning and positioning, our heuristic has produced a factor of 1.94 (5.06) reduction in disk head movement with 2(5) disks for the examples of 200 tasks compared to a heuristic solution for a single disk without using more disk space. Experimental results show the effectiveness of the heuristics used. They show that the careful positioning of data blocks can significantly improve the throughput and the use of multiple disks significantly improves the throughput without the need to increase the storage capacity.

6.0 Conclusion

We studied a novel problem of throughput optimization in disk-based hard real-time application specific systems. We formulated the problem, established its computational complexity, and developed a simple, but effective heuristics for data assignment and disk partitioning so that overall performances of a disk-based application specific system are optimized. The approach yielded significant improvements on a large set of simulated designs.

7.0 References

[Ado73] D. Adolphson, T. C. Hu, "Optimal Linear Ordering", *SIAM J. Appl. Math.*, Vol. 25, No. 3, pp. 403-423, 1973.

[Alp95] C.J. Alpert, A.B. Kahng, Recent directions in netlist partitioning: a survey", *Integration the VLSI journal,* Vol. 19, No. 1&2, pp. 1-81, 1995.

[Cam88] Marvin Camras, "*Magnetic recording handbook*", Van Nostrand Reinhold Co., New York, NY, 1988.

[Cam87] R. Camposano, R.K. Brayton, "Partitioning Before Logic Synthesis", *ICCAD'87*, pp. 324-326, 1987.

[Den67] P.J. Denning, "Effects of Scheduling on File Memory Operations", *Proc. of the AFIPS Spring Joint Computer Conference,* pp. 9-21, 1967.

[Dou94] F. Douglis, F. Kaashoek, B. Marsh, R. Caceres, K. Li, J.A. Tauber, "Storage alternatives for mobile computers", Fi*rst USENIX Symposium on Operating Systems Design and Implementation (OSDI),* pp. 25-37. Berkeley, CA, 1994.

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

[Gei87] R. Geist, S. Daniel, "A Continuum of Disk Scheduling Algorithms, *ACM Transactions on Computer Systems,* Vol. 5,

No. 1, pp. 59-68, 1987.

[Gup90] R. Gupta, G. De Micheli, "Partitioning of Functional Models of Synchronous Digital Systems, *ICCAD90 International Conference on Computer-Aided Design*, pp. 216-219, 1990.

[Hoa91] A. S. Hoagland, James E. Monson, "*Digital magnetic recording"* 2nd ed, John Wiley, New York, NY, 1991.

[Hof80] M. Hofri, "Disk Scheduling: FCFS vs. SSTF Revisited", *Communications of the ACM,* Vol. 23, No. 11, pp. 545-653, 1972.

[Jor96] F. Jorgensen. "*The complete handbook of magnetic recording"*, 4th ed, TAB Books, New York, NY, 1996.

[Kuc91]K. Kucukcakar, A.C. Parker, "CHOP: A Constraint-Driven System-Level Partitioner", *28th DAC,* pp. 514-519, 1991.

[Lag89] E.D. Lagnese, D.E. Thomas, "Architectural Partitioning for System Level Design", *26th DAC,* pp. 62-67, 1989.

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

[McF83] M.C. McFarland, "Computer-Aided Partitioning of Behavioral Hardware Descriptions", *20th DAC*, pp. 472-480, 1983.

[Mee90] D. Mee, E. D. Daniel, editors, "*Magnetic recording handbook: technology and applications*" McGraw-Hill, New York, NY, 1990.

[Pat90] D.A. Patterson, J.L. Hennessy, "*Computer Architecture: A Quantitative Approach"*, Morgan Kaufmann, San Mateo, CA, 1990.

[Pet85] J.L. Peterson, A. Silberschatz, "*Operating Systems Concepts*", 2nd ed, Addison Wesley, Reading, MA, 1985.

[Pra91] S. Prakash, A.C. Parker, "Synthesis of Application-Specific Multiprocessor Architectures", *28th DAC,* pp. 8-13, 1991.

[Raj85] J.V. Rajan, D.E. Thomas, "Synthesis by Delayed Binding Decisions", *22nd DAC,* pp. 367-373, 1985.

[Rao92] D.S. Rao, F.J. Kurdahi, "Partitioning by Regularity Extraction", *29th DAC,* pp. 235-238, 1992.

[Rue94] C. Ruemmler, J. Wilkes, "An introduction to disk drive modeling", *IEEE Computer Magazine,* Vol. 27, No. 3, pp. 17-28, 1994.

[Sel90] M. Seltzer, P. Chen, J. Ousterhout, "Disk Scheduling Revisited", *Proc. of USENIX,* pp. 313-323, 1990.

[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.

[Vah92] F. Vahid, D.D. Gajski, "Specification Partitioning for System Design", *29th DAC,* pp. 219-224, 1992.

[Wag92] J.A. Wagner, "Access time minimization of a moving coil actuator with volumetric, magnetic and current constraints", *IEEE Industry Applications Society Annual Meeting,* pp. 148-155, 1992.

[Won83] C.K. Wong, "*Algorithmic Studies in Mass Storage Systems*", Springer-Verlag, Berlin, Germany, 1983.

[Woo93] C. Woods, P. Hodges, "DASD Trends: Cost, Performance, and Form Factor", *Proc. of the IEEE,* Vol. 81, No. 4, pp. 573-585, 1993.

[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, pp.:146-56, 1995.