MINMAX RECURRENCES IN ANALYSIS OF ALGORITHMS

by

Dr. Arindam Saha
Dept. of Electrical & Computer Engg.
Mississippi State University
P.O. Drawer EE
Mississippi State, MS 39762.

Dr. Meghanad D. Wagh
CSEE Department
Lehigh University
Packard Lab #19
Bethlehem, PA 18015.

Abstract: **Recurrence relations with minimization and maximization, called minmax recurrence relations are commonly encountered in the analysis of algorithms. In this paper we present the solution of one such challenging recurrence relation. We characterize the optimal partition sizes as well as derive the order of complexity of the overall recurrence relation. It is proved that *ad hoc* equal partitioning is never the optimal choice. We also provide a survey of three other interesting minmax recurrence relations found in the literature.**

1. Introduction

The complexity analysis of several recursive algorithms requires the solution of a variety of recurrence relations. A special type of recurrence relations is the *minmax* recurrence relation, which involves recursive minimization and maximization. The standard techniques used for simple recurrences such as the generating functions, the characteristic equations and the summing factors fail to deliver adequate results with minmax recurrences. Thus, often the solution of minmax recurrences becomes a challenging and interesting task. Unfortunately, the current knowledge of researchers in this field is not mature enough to enable establishing general solution techniques. Hence minmax recurrences are mostly solved in a case-by-case basis with not much commanality between different solution methods. In this paper, in Section 2, we first show three examples of minmax recurrences found in the literature, how they are derived from the analysis of algorithms and how they are solved. The remaining and major part of this paper, Section 3, deals with our solution of minmax recurrence derived from the analysis of a parallel divide-and-conquer computational model. In Section 4 we draw some conclusions.

2. A Survey of Minmax Recurrences

In this section we present three interesting minmax recurrences found in the literature. This paper deals with functions that are convex in nature. A real-valued function *f(n)* over the nonnegative integers is defined convex if its second difference is nonnegative, i.e., if

$$g(n+2) - g(n+1) \geq g(n+1) - g(n) \text{ for all } n \geq 0.$$

The following result [1] shows that the *convolution minimization* of convex functions is particularly simple.

Claim 1: Let a(n) and b(n) be two convex functions and define the new function

$$c(n) = \min \{a(r)+b(n-r)\}, \text{ for all } n \geq 0$$

Then c(n) is convex. Moreover, if $c(n) = a(k) + b(n-k)$, then

$$c(n+1) = \min\{a(k)+b(n+1-k), a(k+1)+b(n-k)\}$$

Proof: Let $\delta f(n) = f(n+1) - f(n)$. The two sequences

$$\begin{matrix} \delta a(0), \delta a(1), \delta a(2), \ldots, \\ \delta b(0), \delta b(1), \delta b(2), \ldots, \end{matrix} \quad (1)$$

are nondecreasing by hypothesis. Suppose that the smallest *n* elements of (1) are $\delta a(0), \ldots, \delta a(i-1), \delta b(0), \ldots, \delta b(j-1)$, where $i+j = n$. If $r<i$, then $n-r-1 \geq j$; hence $\delta b(n-r-1) \geq \delta a(r)$, i.e,

$$a(r) + b(n-r) \geq a(r+1) + b(n-r-1).$$

On the other hand, if $r \geq i$, then $n-r-1<j$; hence

$$a(r) + b(n-r) \leq a(r+1) + b(n-r-1).$$

Thus it follows that $c(n) = a(i) + b(j)$. As we increase *n* to *n+1*, we increase *i* or *j* by one, depending on which of $\{\delta a(i), \delta b(j)\}$ is larger. In other words, the sequence

$$\delta c(0), \delta c(1), \delta c(2), \ldots,$$

is precisely the result obtained when merging the two sequences (1) into nondecreasing order. The claim then follows immediately. □

2.1. An Insertion Problem

We are interested in knowing the average number of comparisons, *h(n)*, required to insert an element A_n into a sorted list $A_1 \leq A_2 \leq \ldots \leq A_{n-1}$ [2].

<u>*Claim 2:*</u> h(1) = 0,

$$h(n) = 1 + \min_{0<r<n} \{(r/n)h(r) + ((n-r)/n)h(n-r)\}.$$

Proof: We begin the sorting process by comparing A_n with some element A_r in the sorted list. Since all permutations are assumed equally likely,

$$\Pr(A_n < A_r) = r/n$$
$$\Pr(A_n > A_r) = (n-r)/n.$$

If the result of the comparison is $A_n > A_r$, we are left with the problem of inserting A_n into the ordered list $A_r, \ldots, A_{n-1}$. Similarly if $A_n < A_r$ then we have to insert A_n into the ordered list $A_1, \ldots, A_{r-1}$. When we multiply the required number of comparisons in each case by the probability of occurrence of the case, add 1 for the first comparison, and choose the value of *r* that minimizes the expression, we obtain the result given in the claim. □

In order to solve *h(n)* it is worthwhile to study the function *f(n) = nh(n)*. Hence we have

f(1) = 0,

$$f(n) = n + \min_{0<r<n} \{f(r) + f(n-r)\}. \qquad (2)$$

The minmax recurrence relation (2) has been studied extensively as the generalized subadditive inequality [3,4]. This generalized sequence is as follows

f(1) = 0,

$$f(n) = g(n,r) + \min_{0<r<n} \{f(r) + f(n-r)\}. \qquad (3)$$

Batty and Rogers [4] discuss maximal solutions of (3) for various types of *g(n,r)* function. Some of their results are listed here without proof.

1. Let *g(n,r) = g(n)* be independent of *r* and monotonic in *n*.

a) If *g* is decreasing, then $1 \in R_n$, $n \geq 2$, where R_n is the set of *r* values that satisfy the minmax recurrence for any given *n*.

b) If *g* is increasing and convex, then $\lfloor n/2 \rfloor \in R_n$, $n \geq 2$, and *f* is also convex.

c) If *g* is increasing and concave, then $\varrho(n) \in R_n$, $n \geq 2$, where for $p \geq 0$,

$$\varrho(2^p + q) = 2^{p-1}, \qquad \text{for } 0 \leq q \leq 2^{p-1},$$
$$\varrho(2^p + q) = q, \qquad \text{for } 2^{p-1} \leq q \leq 2^{p-1}.$$

d) If *g* is increasing and nonnegative, then *f* is increasing.

1. Let *g(n,r) = g(r)* be independent of *n* and monotonic in *r*.

a) If *g* is increasing, then $1 \in R_n$, $n \geq 2$.

b) If *g* is decreasing and concave, then $\lfloor n/2 \rfloor \in R_n$, $n \geq 2$, and *f* is also convex.

c) If *g* is decreasing and convex, then $\sigma(n)$ and $\varrho(n) \in R_n$, $n \geq 2$, where for $p \geq 0$,

$$\sigma(2^p + q) = q \qquad \text{for } 0 \leq q \leq 2^p.$$

Moreover, if the monotonicity of *g* is strict then the only *r* in R_n are those given above.

2.2. A Graph Matching Problem

The problem of drawing a graph on a mechanical plotter with prespecified vertex locations [5], where the distances satisfy the triangle inequality, arises in numerous applications. To draw the graph efficiently we must minimize wasted plotter-pen movement, i.e., movement of the pen off the paper. If the graph contains an Eulerian cycle or path (a cycle or a path that traverses every edge of the graph exactly once), then it can be drawn with no wasted pen movement; otherwise, the graph contains an even number $n > 2$ of vertices of odd degree. In the latter case, as a simple consequence of the triangle inequality, the minimum wasted pen movement can be achieved by finding a minimum weighted matching (a matching is the set of edges no two of which have a vertex in common) of the *n* vertices of odd degree. The graph can then be drawn by traversing the Eulerian cycle that exists when the edges of the minimum matching are added to the original graph with *these* edges traversed with the pen off the paper.

The currently known best algorithm for finding the minimum weighted complete matching is $O(n^3)$. In order to reduce this time complexity many heuristics have been used. One obvious heuristic is the *greedy* algorithm : repeatedly match the two closest unmatched remaining vertices, resolving any ties arbitrarily. For *n* vertices this can be done in the worst-case time $O(n^2 \log n(n))$ by sorting the *n* distances. But one must consider how far from the minimum matching the resultant greedy matching will be.

We now analyze the ratio of the cost of the matching found by the greedy algorithm to the cost of the minimum matching in the worst case. Observe that the union of any two matchings is a collection of disjoint cycles, the edges of which alternate between the two matchings. Consider the collection of such cycles that results from taking the union of the greedy and the minimum matchings. Without loss of any generality we can consider the ratio of the two costs when the union of the two matchings is a single cycle.

Define *f(n)* to be the *smallest* fraction of the total length that can consist of edges of the matching due to the minimum matching of the *n* vertices. Then it can be shown [5] that

f(1) = 0,

$$f(n) = \min_{1<r<n-1} \{\alpha f(r) + \beta f(n-r)\}. \qquad (4)$$

where $\alpha \geq 1-\alpha-\beta > 0$ and $\beta \geq 1-\alpha-\beta > 0$.

Since the extremum of a linear function on a polyhedron must occur at a vertex of the polyhedron [6], the only possible values α and β can be (1,0), (0,1) and (1/3,1/3) of which (1/3,1/3) is the only nontrivial value. Thus in this case the value of r is always n/2. One can then easily show by induction that $f(n) = O(n^{\log(2/3)})$. The recurrence (5) has been solved in a generalized form in [7].

2.3. Optimum Lopsided Binary Trees

Lopsided binary search trees with costs α and β, respectively, on the left and right edges are encountered in various applications [8]. Such trees with weights $\alpha = \beta = 1$ are the normal binary trees, whereas with $\alpha = 1$, $\beta = 2$ correspond to the familiar Fibonacci trees. An important application of optimum lopsided trees is the construction of optimum prefix-free codes. When the alphabet consists of two letters (say 1 and 0) with costs c_1 and c_2, the problem is modeled by lopsided binary trees with $\alpha = c_1$ and $\beta = c_2$. The optimum lopsided tree with n leaves gives the minimum-cost prefix-free code for n symbols occurring with uniform probability.

Let T be a lopsided binary search tree with n leaves and let d(E), the weighted depth of a node E, bet the sum of the weights on the path from the root to the node E. The cost of the tree, C(T), is defined to be the max(d(E)) over all the leaves in T. Let $\tau(n)$ be the worst-case optimum lopsided binary search tree, i.e., $C(\tau(n)) = \min(C(T))$ over all possible trees T with n leaves. Clearly, the value of $C(\tau(n))$ is given by the recurrence

$$f(n) = \min_{0<r<n} \{\max\{\alpha + f(r), \beta + f(n-r)\}\}. \qquad (5)$$

where $\beta \geq \alpha \geq 0$ and α/β is rational.
We can view $\tau(n)$ as being the result of a sequence of replacements of a leaf by an internal node and two leaves as children, starting with an initially empty tree because the following is true.

Claim 3: $\tau(n)$ is obtained from $\tau(n-1)$ by replacing a leaf with minimum cost in $\tau(n-1)$ by an internal node with two leaves as children.
Proof: Follows easily from Claim 1. □

When $\alpha=0$, $\tau(n-1)$ has a leaf of depth 0 and $\tau(n)$ is obtained by replacing this node. Hence when $\beta \geq \alpha = 0$, $f(n) = \beta$, the minimum possible cost. When $\alpha>0$, it follows from Claim 3 that every leaf will be placed eventually and $f(n)$ will be if the form $i\alpha + j\beta$ for some $i \geq 0$, $j \geq 1$. The difficult recurrence (5) is solved indirectly by solving a simpler recurrence in a quantity that specifies the maximum number of leaves with a certain depth. Let $\alpha/\beta = p/q$ where p and q are mutually prime integers. Then every integer $k \geq pq$ can be represented as $ip + jq$, $i \geq 0$, $j \geq 1$. Thus for all $k \geq pq$, $\alpha k/p$ will be the depth of a leaf in $\tau(n)$ for some n. If P_k is the maximum number of leaves with depth $\alpha k/p$, then P_k satisfies the recurrence

$$P_k = P_{k-p} + P_{k-q} \qquad (6)$$

since leaves of depth $\alpha k/p$ are obtained by replacing leaves of depths $\alpha(k-p)/p$ and $\alpha(k-q)/p$. In order to solve (6) we use the following result from [8,9] without proof.

Claim 4: Let r be the root of largest magnitude of $z^q - z^{q-p} = 1$, where p and q are mutually prime integers. Then r is real and simple, $1<r\leq 2$, and there is no other root of absolute value r.

As a consequence of claim 4, we have $P_k = cr^k + o(r^k)$ where c is some constant. Thus for $\beta \geq \alpha > 0$, it can be shown (after some algebraic manipulation) that

$$f(n) = \alpha(\lceil \log_r(n+1) \rceil - 1 + q - \log_r(c/(r-1)) + o(1).$$

3. Our Minmax Recurrence Relation

In the remaining part of this paper we derive and solve a challenging minmax recurrence that arises from a parallel divide-and-conquer computational model. Divide-and-conquer allows partitioning of a problem in (typically) two smaller instances of itself, and combining the partial results to get the final solution. Since these smaller instances can be recursively partitioned into even smaller sizes, the procedure reduces to solving minimal sized problems and recombining their results. The divide-and-conquer paradigm is ideal for parallel processing because the subproblems are mutually independent and can be executed concurrently in different processors. Unfortunately, the performance of any parallel divide-and-conquer algorithm is significantly affected by both architectural and algorithmic overheads that are inevitable in any parallel processing environment.

The overheads can be broadly classified into two groups - the partition overhead and the recombination overhead. The partition overhead typically includes interprocessor communication cost necessary for data distribution and result collection. On the other hand, the recombination overhead results from the cost of combining the partial results. Conventionally, these overheads are neglected during the design of algorithms. This leads to *ad hoc* equal partitioning and nonoptimal performance of the divide-and-conquer technique on real-world parallel machines.

The complexity of a parallel divide-and-conquer algorithm may be described by the following minmax recurrence

$$T(n) = \min_{0<r<n} \{\max\{T(n-r), T(r)+kr\}+\lambda\}. \qquad (7)$$

where T(n) is the complexity of a size n problem, kr is the partition overhead that depends upon the partition size, and λ is the constant recombination overhead. Note that (7) is true for all problem sizes $n \geq n_0$, where n_0 is some small problem size below which recursive partitioning is not employed and all problem sizes up to n_0 have same complexity equal to some predefined constant say t_0. The recurrence (7) is derived by noting that the execution of size r problem in one processor together with the interprocessor communication are carried out concurrently with the execution of problem size $n-r$. The *max* function in (7) is due to the fact that the slower of these two tasks dominate. In order to ensure that the best possible partitions are chosen for proper optimization we use the *min* function. The minmax recurrence relation (7) can *realistically* model a wide range of common applications.

Note that the minmax recurrence (7) is different from the previously solved minmax recurrences (3), (4), and (5). Both equations (3) and (4) did not involve any recursive maximization. Although equation (5) is a true minmax recurrence, it is simpler than (7) because of the constant nature of α and β. Thus (7) is considerably more challenging and interesting than the previously solved recurrences.

There are two main aspects related with minmax recurrences in general and (7) in particular that need to be addressed. We are interested not only in the behavior of $T(n)$ and the order of complexity, but also in the proper characterization of the optimal partition values for every problem size.

3.1. Main Results

Let S_m denote the set of all the problem sizes *n* that have the same complexity $T(n) = m$. Denote the maximum of this set by η_m. In other words η_m's are those problem sizes such that the complexity value necessarily changes at $\eta_m + 1$. For mathematical convenience assume that $\eta_m = \eta_{m-1}$ when $|S_m| = 0$. Recall that R_n denotes the set of all optimum partition sizes for a problem size *n*. Now we list some of our results.

We begin by substantiating the intuitive belief that solving larger problems takes more time.

Lemma 1: $T(n)$ is a monotonically increasing function of n.
Proof: (By induction over n). Recall that for $n<n_0$, $T(n)=t_0$. For $n=n_0$, $r=1$ is optimal and thus $T(n_0) = t_0 + k + \lambda > T(n_0 - 1)$. Assuming that the result is true up to certain $n \geq n_0$, its truth for n+1 can be established as follows. Let r denote the optimum partition size for problem size n+1, i.e.,

$$T(n+1) = \max\{T(n+1-r)+\lambda,\ T(r)+kr+\lambda\} \quad (8)$$

If r=n, then $T(n+1) = T(n) = kn + \lambda > T(n)$. If r<n, note that

$$T(n) \leq \max\{T(n-r)+\lambda,\ T(r)+kr+\lambda\} \quad (9)$$

Since the monotonicity is assumed for problem sizes up to n, $T(n-r) \leq T(n+1-r)$. Combining this with (8) and (9) gives $T(n) \leq T(n+1)$ for r<n also. □

Now we characterize the set R_n. This seemingly formidable task is simplified by the following result that shows that the optimal partition size is uniquely defined at special problem sizes η_m.

Theorem 1: The optimum partition set $R_{\eta_m} = \{\eta_m - \eta_{m-\lambda}\}$.
Proof: For any $p \varepsilon R_{\eta_m}$, $m = \max\{T(\eta_m - p)+\lambda,\ T(p)+kp+\lambda\}$. The first term of the max yields $T(\eta_m - p) \leq T(\eta_{m-\lambda})$, implying from Lemma 1,

$$p \geq \eta_m - \eta_{m-\lambda} \quad (10)$$

Similarly, the second term of the max gives
Now if $p>\eta_m - \eta_{m-\lambda}$, then $\eta_m - p+1 \leq \eta_{m-\lambda}$ or, $T(\eta_m - p+1) \leq m-\lambda$.

$$T(p)+kp+\lambda \leq m$$

Thus, in this case,

$$T(\eta_m+1) \leq \max\{T(\eta_m+1-p)+\lambda, T(p)+kp+\lambda\} \leq m,$$

a result contradicting the definition of η_m. Therefore, $p \leq \eta_m - \eta_{m-\lambda}$. Combining this with (10) gives the desired result. □

Let $r_m = \eta_m - \eta_{m-\lambda}$. This partition size r_m is important for two reasons. Firstly, r_m is optimal for a large number of problems whose complexities satisfy some inequality. Secondly, it turns out to be the optimal partition size for all problem sizes in S_m. These to facts are illustrated in the following two results.

Lemma 2: The partition size r_m is optimal for problems whose complexity m satisfies

$$T(r_m)+kr_m+\lambda \leq m < T(r_m+1)+k(r_m+1)+\lambda\}$$

Proof: Follows from the definition of η_m and Theorem 1.

Theorem 2: For any $n \varepsilon S_m$, the set of optimal partition sizes is given by $R_n = \{p \mid n - \eta_{m-\lambda} \leq p \leq r_m\}$.
Proof: we first prove that any $p \varepsilon R_n$ is an optimal partition size for problem size $n \varepsilon S_m$. The lower bound on p and the monotonicity of $T(n)$ yields $T(n-p) \leq m-\lambda$, and the upper bound gives $T(p)+kp+\lambda \leq T(r_m)+kr_m+\lambda \leq m$. Consequently, $\max\{T(n-p)+\lambda, T(p)+kp+\lambda \leq m$. However $m = T(n)$. In other words $\max\{T(n-p)+\lambda, T(p)+kp+\lambda \geq m$. Therefore any $p \varepsilon R_n$ is an optimal partition.

Conversely, to prove that any $p \notin R_n$ is not an optimal partition we proceed as follows. For any $p>r_m$, one has $p \geq r_m+1$. From Lemma 1 and Lemma 2, one gets $T(n-p)+kp+\lambda>m$. This implies that with such a p as partition size $T(n)>m$, clearly ruling out such a possibility. Similarly, if $p<n-\eta_{m-\lambda}$, then $n-p \geq \eta_{m-\lambda}+1$. Thus, in this case with such a p as a partition size $T(n-p)>m-\lambda$, implying that $T(n)$ will be greater than m.Thus such a p is also ruled out. □

Theorem 2 has been used by the authors [10] to design some optimal partition algorithms. The reader is referred to [10] for accompanying results. Such algorithms to compute the optimal partition size given any problem size, although vital, are omitted in this paper for brevity purposes.

Conventionally, researchers ignore the effect of overheads in parallel divide–and–conquer and end up using the nonoptimal *ad hoc* n/2 partitioning. Now we prove that n/2 is almost *never* an optimal partition size when the overhead k is nonzero (as is true in all practical cases because k represents the inevitable interprocessor communication overhead in parallel processing).

Theorem 3: If $k \neq 0$ then n/2 is never an optimal partition size for any problem size $n>2(\lambda/k)$.
Proof: Suppose n is not an η. If $n>2(\lambda/k)$ and $n/2 \varepsilon R_n$, then clearly n/2 is the largest element of R_n. Further, since n is not an η, from Theorem 2, $(n/2)-1 \varepsilon R_n$ as well. However equating the two expressions of $T(n)$ with these two as partition sizes one gets

$$T(n/2)+k(n/2)+\lambda = T(1+n/2)+\lambda\} \qquad (11)$$

But it can be easily shown that $T(n+1)-T(n) \leq \lambda$ [10]. Use of this fact in (11) contradicts the fact that $n>2(\lambda/k)$. Therefore $n/2 \notin R_n$. Now suppose $n=\eta_m$. Assume if possible $\eta_m/2$ is the optimal partition size for n. From Theorem 1 one then gets $\eta_m - \eta_{m-\lambda} = \eta_m/2$. Simple algebraic manipulation yields $r_m = \eta_{m-\lambda}$. and $T(\eta_m) = m = \max\{m, m+k\eta_{m-\lambda}\}$. However since $k \neq 0$ this cannot be correct. therefore $\eta_m/2$ is not an optimal partition size for problem size η_m. □

The conclusion of Theorem 3 is that n/2 should never be arbitrarily used to partition any moderately sized problem. A simple brute-force solution of (7) using r=n/2 at every step of the recursion yields T(n) = O(n). The following result shows that we can improve the order of complexity of T(n).

Theorem 4: T(n) is of the order $O(\sqrt{n})$.
Proof: The proof of this result is long and involved. It is omitted in this paper for brevity purposes. The reader is referred to Chapter 3 of the first author's dissertation[11] for the proof.

The result of Theorem 4 shows that a proper partitioning strategy, as opposed to the arbitrary equal partitioning scheme, can reduce the complexity of a parallel divide-and-conquer computation considerably.

4. Conclusions

The major contribution of this paper is the solution of a challenging minmax recurrence relation. This minmax relation is derived from a model of parallel divide-and-conquer computations that incorporates the unavoidable and significant parallel processing overheads.

The minmax recurrence is solved by characterizing the properties of the optimal partition sizes. It is shown that the optimal partition size, given a problem size *n*, is nontrivial and very different from the *ad hoc* n/2 value taken conventionally. It is also shown that the complexity of the algorithm reduces from O(n) to $O(\sqrt{n})$ by choosing the optimal partition size instead of the equal partition size size at every stage of the recursion.

The other contribution of this paper is a survey of some of the existing theory of minmax recurrence relations. We mention three interesting recurrences, how they are derived in the analysis of algorithms and how they are solved by various authors.

We note that the current knowledge of researchers in this field is not mature enough to enable establishing general solution techniques. Hence minmax recurrences are mostly solved in a case-by-case basis with not much commanality between different solution methods. Our solution of a minmax recurrence relation in this paper will hopefully enrich this field and help others to solve such problems in future.

References

[1] M.L. Fredman and D.E. Knuth, "Recurrence relations based on minimization," J. of Math. Analysis an Appl., 48, 1974, pp 435-559.

[2] R. Morris, "Some theorems on sorting," SIAM J. Applied Math., Vol. 17, No. 1, January 1969, pp 1-6.

[3] J.M. Hammersley and G.R. Grimmett, "Maximal solutions of the generalized subadditive inequality," in Stochastic Geometry, E.F. Harding and D.G. Kendall eds., John Wiley, London, 1974, pp 270-284.

[4] C.J.K. Batty and D.G. Rogers, *Some maximal solutions of the generalized subadditive inequality*, SIAM J. Alg. Discrete Methods., Vol. 13, No. 3, 1982, pp 369-378.

[5] E.M. Reingold and R.E. Tarjan, *On a greedy heuristic for complete matching*, SIAM J. Computing Vol. 10, No. 4, November 1981, pp 676-681.

[6] G.B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, NJ, 1963, Chapter 3, page 154.

[7] S. Kapoor and E.M. Reingold, "Recurrence relations based on minimization and maximization," J. Math. Analysis and Appl., 109, 1985, pp 591-604.

[8] S. Kapoor and E.M. Reingold, "Optimum lopsided binary trees," J. of the ACM, Vol. 36, No. 3, July 1989, pp 573-590.

[9] M.D. Wagh and G. Bakdash, "Optimality in parallel divide-and-conquer algorithms in the presence of overheads," in review.

[10] A. Saha and M.D. Wagh, "Optimal partition algorithms for parallel divide-and-conquer computations," 1991 Proc. of the Int'l Conf. on Parallel Processing, St. Charles IL, pp 75-82.

[11] A. Saha, "Design and analysis of parallel recursive computations in the presence of overheads," Ph.D. Dissertation, Chapter 3, pp 39-45.