Time-Resource Trade-offs for Optimal and Near-Optimal Execution of a Class of Parallel Algorithms

Harish Sethu (Contact Author)
Department of Electrical and Computer Engineering
Drexel University
3141 Chestnut Street, Philadelphia, PA 19104-2875.
Phone: (215) 895-5876, Fax: (215) 895-1695
E-mail: sethu@ece.drexel.edu

and

Meghanad D. Wagh
Department of Electrical Engineering and Computer Science
Lehigh University
Packard Laboratory #19, Bethlehem, PA 18015.
Phone: (215) 758-4142, Fax: (610) 758-6279
E-mail: mdw0@lehigh.edu

Abstract

In this paper, we consider the optimal and near-optimal implementation of a class of parallel algorithms based on the principle of divide-and-conquer. We use a computational model incorporating both communication and the associated computational overheads inherent to a divide-and-conquer methodology. We consider the time-optimal implementation of this class of parallel algorithms on a generic network topology, and we show that the optimal topology of the interconnection network required for such an implementation is readily mappable on a hypercube. We obtain a recursive analytical expression for the minimum degree of the hypercube, $D(n)$, required to solve a problem of size n in a *time-optimal* fashion, and describe some important characteristics of $D(n)$ as a function of n. Finally, we prove an interesting result that shows how a small constant deviation from time-optimality may be used to achieve a large O($\log n / \log\log n$) reduction in the degree of the hypercube required to solve a problem of size n. This result quantifies the statement that solving problems in a time-optimal manner can be very expensive in terms of resource requirements, and that a small compromise in execution time can sometimes yield very favorable trade-offs. We present an intuitive explanation of this phenomenon and analyze its dependence on the relationship between the overheads associated with the problem. Besides parallel divide-and-conquer on hypercube topologies, the principal results presented here have wider applications such as in the construction of key-based worst-case optimal search trees while minimizing the required key length.

1 Introduction

This work focuses on a class of parallel algorithms based on the principle of divide-and-conquer, used in numerous applications in science, engineering and finance. The divide-and-conquer methodology to parallelization is used in a wide array of algorithms including sorting and search algorithms, a variety signal and image processing algorithms, and associative operations on large sets of data such as those in commercially important applications such as data mining. This technique of divide-and-conquer is among the few general techniques available for parallel algorithm design, and is among the most widely used and studied [1–7].

Divide-and-conquer achieves parallelism by recursively partitioning the original problem into smaller independent instances of the same problem. Examples for which sub-problems are of fixed size in relation to the original problem include the computation of an N-point DFT and the multiplication of square matrices. Many problems, however, can be partitioned into arbitrary sizes such as database searching, associative operation on large data sets, dot products of long vectors, sorting and searching operations. This flexibility can be exploited to allocate smaller subtasks to the processors subjected to higher overheads. This strategy as applied to the divide-and-conquer algorithm has been modeled and analyzed in [8, 9]. At each stage of the divide-and-conquer recursion, the problem is partitioned into two subproblems. The various algorithmic and architectural overheads are lumped together into an asymmetric partition overhead, k, attached to only one partition and a symmetric merging overhead, λ, added to both the partitions. These overheads include the cost of interprocessor communication to collect results, the algorithmic costs of combining the partial results and the costs associated with converting one or both subproblems to conform to the exact form of the original problem. In this paper, we use a recursive computational model that captures these overheads, and therefore, allows an analysis of resource requirements.

The computational model can be summarized by a typical recursion step in which a problem of size n is partitioned into problems of sizes r and $n - r$ as follows,

$$T(n) = \begin{cases} \min\limits_{1 \le r \le n-1} [\max\{T(r),\ T(n-r)+k\}] + \lambda, & n \ge n_0 \\ T_0, & n < n_0, \end{cases} \tag{1}$$

where $T(n)$ is the minimal execution time or the *time-complexity* of P_n, a size n instance of the problem being solved. Problems smaller than a certain size n_0 are solved in a single processor in constant time T_0. The quantity n_0 can be thought of as the smallest size for which the partitioning is advantageous or as a quantity dictated by the problem granularity. The divide-and-conquer algorithm for these problems is completely defined by specifying the partition sizes at each stage of the recursion. It may be noted that the optimal partition sizes for any given n is not necessarily unique. The above recursion has also been analyzed in [10], in the context of search procedures using lop-sided binary trees.

Consider a simple example of a problem which is modeled by (1). Consider the problem of adding n numbers on a parallel system. Two processors X_1 and X_2 may split this problem and assume the task of adding r and $n - r$ numbers, respectively. Now, if we choose to combine the results by adding the two sums in processor X_1, we would have X_2 communicating its result, the sum of $n - r$ numbers, to X_1. This "keep some, send some" technique of allocating divide-and-conquer tasks to processors is described in detail in [1]. Let the time taken to communicate the result computed in X_2 to X_1 be k, and let the time taken to merge this result with that computed in

X_1 be equal to λ. One may now see that the computational model described in (1) uses the value of r that best overlaps computation and communication, and achieves the best possible execution time given the constraints. This process of splitting the problem into smaller problems in an optimal fashion can be carried out recursively.

Given an unlimited number of processors with the desired connectivity, the partitioning as given by (1) can be carried on to completion yielding the optimal execution time. However, on real systems or in multi-user environments, it is vital to minimize the resource requirements. Denote by $D(n)$ the degree of the smallest hypercube that can solve a problem of size n in *optimal* time, as given by (1). An implementation of an algorithm for a problem of size n on a hypercube of degree less than $D(n)$ will lead to a sub-optimal execution time. On the other hand, using a hypercube of degree greater than $D(n)$ will not contribute to improving the execution time. Our work concentrates on minimizing $D(n)$, the degree of the hypercube required to solve a problem of size n while preserving an optimal execution time in the presence of the parallel computing overheads. Note that a degree-d hypercube has 2^d processors, and therefore minimizing the degree of the hypercube greatly reduces the number of processing nodes required to solve the problem. In a more theoretical sense, it will be apparent from Section 3 that the quantity $D(n)$ is actually the minimum possible height of a time-optimal binary tree task graph of the parallel divide-and-conquer algorithm.

Besides parallel divide-and-conquer, the fundamental results presented in this paper have a variety of other applications. One such example is a dictionary machine supporting key-based search operations using a partial mapping from keys to data. Keys are typically strings of symbols over an alphabet of a certain size and are stored in a structure commonly known as the doubly-chained tree. Solving a problem of size n using parallel divide-and-conquer using the above model described by (1) is analogous to creating a worst-case time-optimal search tree for a collection of n records [11]. Minimizing the hypercube degree for a time-optimal solution becomes analogous to the problem of constructing a worst-case time-optimal tree while minimizing the length of the key required for the search.

Section 2 describes some definitions and results on the time-optimality of execution as given by (1). Section 3 obtains the analytical definition of $D(n)$ and presents some fundamental properties of $D(n)$ as a function of n. Section 4 further investigates some characteristics of $D(n)$ and proceeds to analyze time-resource trade-offs that can be achieved by deviating from the goal of time-optimal execution. In this section, we prove that in solving a problem of size n, a small constant deviation from time-optimal execution may allow a large drop of order O($\log n / \log \log n$) in the degree of the hypercube required to solve the problem. Section 4 also gives an intuitive explanation of this result, and makes some observations on the dependence of this large trade-off on the relationship between the overheads k and λ. Section 5 concludes the paper, and suggests several enhancements to the computational model to accurately capture the behavior of a wider variety of algorithms and their resource requirements.

2 Properties of the Computational Model

Let the size of a problem, n, be a positive integer. In all of the following, we also assume that the time complexity, $T(n)$, and the overheads k and λ are non-negative integers. This assumption is without loss of generalization, since the unit of time-complexity can always be defined in such a way that k, λ and $T(n)$ for all n are always integer quantities.

Definition 1. *Denote by* S_m*, the set of sizes of problems that have the same execution time* m *as given by (1). Thus, we have,* $\mathrm{S}_m = \{n \in \mathbf{Z} \,|\, T(n) = m\}$*. Note that it is possible that* $\mathrm{S}_m = \phi$*.*

Definition 2. *Define* η_m *recursively as the largest element of* S_m *if* $\mathrm{S}_m \neq \phi$ *and* η_{m-1} *if* $\mathrm{S}_m = \phi$*. The quantity* η_m*, therefore, is the size of the largest problem that can be solved* optimally *within time* m*.*

It has been shown in [10] that the η_m's are related by,

$$\eta_m = \eta_{m-\lambda} + \eta_{m-\lambda-k}, \quad m \geq T(n_0). \tag{2}$$

One may understand Equation (2) intuitively by noting that the size of the largest problem that can be solved in time m is the sum of the sizes of the largest problem that can be solved in time $m - \lambda$ (since it will take a recombination overhead of λ to combine the results) and the largest problem that can be solved in time $m - \lambda - k$ (since it will take an additional communication overhead of k before the two subproblems can be combined).

Since the size of the complexity class, $|\mathbf{S}_m|$, is nothing but $\eta_m - \eta_{m-1}$, we can easily derive from (2) that the sizes of the complexity classes are also similarly related. The following equation expresses this recursive relationship.

$$|\mathbf{S}_m| = |\mathbf{S}_{m-\lambda}| + |\mathbf{S}_{m-\lambda-k}|, \quad m \geq T(n_0). \tag{3}$$

Definition 3. *Denote by* $\mathbf{R}_n$*, the set of all optimal partition sizes* r *as given by (1), for a problem of size* n*. Thus,* $\mathbf{R}_n = \{1 \leq r \leq n-1 \,|\, T(n,r) = T(n)\}$*, where* $T(n,r) = \max\{T(r), T(n-r)+k\} + \lambda$*.*

By the above definition, if $p \in \mathbf{R}_n$, we can say that, $T(n-p) \leq m - \lambda - k$, and $T(p) \leq m - \lambda$. These two conditions imply that $n - p \leq \eta_{m-\lambda-k}$, and $p \leq \eta_{m-\lambda}$. We can now derive the limits on p as $n - \eta_{m-\lambda-k} \leq p \leq \eta_{m-\lambda}$. This result can be formally stated as follows.

$$\mathbf{R}_n = \{r \in \mathbf{Z} \,|\, n - \eta_{m-\lambda-k} \leq r \leq \eta_{m-\lambda}\}, \quad n \geq n_0, \tag{4}$$

A close inspection of the recursive equation (1) reveals that if $T(n) = m$ for some problem size n, m is such that it can be expressed as $m = T(n_0) + i(k+\lambda) + j\lambda$, for some $i, j \geq 0$. For very large m, it is possible to say that m is of the form $T(n_0) + ig$ where $g = \gcd(k, \lambda)$, and i is some positive integer. This result can be more formally expressed as follows.

$$\mathbf{S}_m \neq \phi \;\text{ iff }\; g|(m - T(n_0)), \;\text{ for } m \gg T(n_0) \tag{5}$$

It is shown in [10] that η_m is of the order of a^m, where $1/a$ is the largest real root of $x^{\lambda+k} + x^{\lambda} = 1$. Now, if $\mathbf{S}_m \neq \phi$, we know that $|\mathbf{S}_m| = \eta_m - \eta_{m-1} > 0$, and the order of $|\mathbf{S}_m|$ can be given by,

$$|\mathbf{S}_m| = \Theta(a^m), \tag{6}$$

It is also shown in [10] that the time-complexity $T(n)$ of a size n problem as given by (1) is $\mathrm{O}(\log n)$, that is,

$$T(n) = \mathrm{O}(\log n). \tag{7}$$

Note that the results on time-optimality in this section may be applied to determining a partitioning algorithm not just on hypercube topologies but on all systems where the communication cost between any two processors is not dependent upon or only slightly dependent upon exactly which processors are communicating. Examples of such systems include many real systems based on both direct and indirect networks where the topology-dependent hardware latency through the network is a small fraction of the overall latency between any two processors.

3 Hypercube Degree Requirement

Consider a computational binary tree task graph as shown in Figure 1. A typical task graph of recursive partitioning has a tree topology as shown in this figure. Note that this topology is not necessarily a complete binary tree. Each leaf of this tree represents a subtask computation while the intermediate nodes represent the computations corresponding to the merging of the subtask results. By our mapping strategy, when two processors work on two different parts of the same problem, the results of these subproblems are combined in one of these two processors rather than in a third processor, as discussed in [1]. It is obvious that this serves the objective of minimizing communication and maximizing processor utilization. Without loss of generality, we choose to evaluate a parent node and its left child in the same processor. Thus, computation tasks represented by N_1, N_2, N_4 and N_8 in Figure 1 are all assigned to the same processor.

The resulting connections between processors that is required to implement such a computational tree with dilation 1, form a *binomial tree* topology, discussed in [1]. By the task-numbering scheme used in this work as shown in the computational tree of Figure 1, level of a task in the tree is the position of the leftmost 1 in the binary representation of its label. Thus, N_1 is on level 0, while N_{12} is on level 3. One can see that a 3-level computational tree maps onto a hypercube of degree 3 perfectly with dilation 1, as shown in Figure 1. In general, if a processor i of the hypercube is assigned to a task in level l of the tree, then the child tasks of this task are assigned to processors i and j, where i and j differ in the l-th bit. Thus, a binomial tree resulting from a level-L computational tree graph maps on to a degree-L hypercube perfectly. Note that in the mapping of an algorithm to a topology, a processor may perform computations corresponding to many tasks at different levels of the tree. It is also obvious from this discussion that $D(n)$, the minimum degree of a hypercube required to solve a problem P_n in optimal time is the same as the minimum possible number of levels in a time-optimal computational tree of problem P_n.

Let a problem of size n, P_n, be divided into two problems P_r and P_{n-r}. The number of levels in a tree corresponding to P_n is exactly one more than the larger of the number of levels corresponding to the sub-problems P_r and P_{n-r}. Our objective is to preserve time-optimality while minimizing $D(n)$. Therefore we search within $\mathbf{R}_n$ for that partition size which would yield the least number of levels in the tree. Recall that $\mathbf{R}_n$ is the set of all partition sizes r that yield the minimal execution time of P_n as given by (1). The minimum degree of the hypercube required to solve a problem of size n optimally, is therefore given by,

$$D(n) = 1 + \min_{r \in \mathbf{R}_n} [\max\{D(r), D(n-r)\}]. \tag{8}$$

The degree of the hypercube required to solve problems of size smaller than n_0 is obviously equal to 0, since they are optimally solved in a single processing node. This gives the initial condition for the recursion (8).

We now discuss some properties of $D(n)$ as a function of n. It is proved in [4] that $D(n)$ as expressed by (8) can be easily evaluated at $n = \eta_m$, by the following expression.

$$D(\eta_m) = \lceil (m - T(n_0) + 1)/\lambda \rceil, \quad m \geq T(n_0). \tag{9}$$

If two problems have the same optimal time complexity, the larger problem would require a hypercube of a larger degree than that of the smaller problem. In other words, $D(n)$ is a monotonically increasing function of n within each complexity class. This intuitively obvious result is formally expressed as follows.

$$D(n_1) \leq D(n_2) \text{ if } n_1, n_2 \in \mathbf{S}_m \text{ and } n_1 \leq n_2. \tag{10}$$

Definition 4. *Denote by $\mu(m, d)$, the size of the largest problem that can be* optimally *solved within time-complexity m on a hypercube of degree d or less.*

Note that the problem of size $\mu(m, d)$ does not necessarily have a time complexity m. It is shown in [4] that,

$$\mu(m, l) = \mu(m - \lambda, l - 1) + \mu(m - \lambda - k, l - 1), \;\; m \geq T(n_0). \tag{11}$$

This recursion (11) is used in the next section to derive results on some interesting trade-offs between system size and time-optimality.

4 Time-Resource Trade-offs

A typical plot of $D(n)$ as a function of n, shown in Figure 2, reveals an interesting and somewhat counter-intuitive feature which has important practical implications. One can see that the value of $D(n)$ frequently drops as n increases beyond one of the η points on the graph. It is therefore possible to time-optimally solve problems of size larger than size η_m but with a smaller hypercube than that required for an optimal solution of a problem of size η_m. In this section, we attempt to study this property of $D(n)$ as a function of n.

Since $D(\eta_m + 1)$ is typically less than $D(\eta_m)$, a time-optimal execution of a size $\eta_m + 1$ problem would typically require a smaller hypercube than a problem of size η_m. Thus, even while executing a size η_m problem, it might be worthwhile to add a dummy element and solve it as a size $\eta_m + 1$ problem. In fact, all problems in the complexity class $\mathbf{S}_m$ that require a hypercube of degree greater than $D(\eta_m + 1)$ can be solved on a smaller hypercube of degree $D(\eta_m + 1)$ as problems of size $\eta_m + 1$ by adding an appropriate number of dummy elements. In this section, we show that the reduction in the degree of the hypercube, and therefore the number of processors required is substantial in comparison to the sacrifice in time-complexity that such a procedure incurs.

Definition 5. *Denote by $U(m)$ the quantity $D(\eta_m) - D(\eta_m + 1)$. Thus, $U(m)$ represents the possible reduction in the degree of the hypercube when executing a problem of size η_m as a problem of size $\eta_m + 1$.*

Definition 6. *Define ζ_m as the smallest problem that needs at least time m for a time-optimal solution. When $\mathbf{S}_m \neq \phi$, $\zeta_m = \min \mathbf{S}_m$.*

Definition 7. *Let $\vartheta_m = D(\eta_m) - D(\zeta_m)$. Thus, ϑ_m is the difference between the degrees of the hypercubes required for the largest and smallest problems belonging to the complexity class $\mathbf{S}_m$. For mathematical convenience, let $\vartheta_m = 0$ when $\mathbf{S}_m = \phi$.*

Note from (9) that $D(\eta_m)$ as a function of m increases only in step sizes of 1 and therefore, $U(m-1) \le \vartheta_m \le U(m-1)+1$. Clearly, $\mathrm{O}(\vartheta_m) = \mathrm{O}(U(m))$ and therefore, an investigation into the order of $U(m)$ leads us to a study of ϑ_m. The following lemma proves that $\vartheta_{i\lambda}$ is a monotonically increasing function of i.

Lemma 1. *$\vartheta_m \ge \vartheta_{m-\lambda}$, for all m.*

Proof. The proof is by induction. It is easily seen from (9) that $D(\eta_{T(n_0)}) = 1$ and by the definition of $D(n)$ and n_0, $D(\zeta_{T(n_0)}) = D(n_0) = 1$. Thus, $\vartheta_m = 0$ for all $m \le T(n_0)$.

Assume that the statement of the lemma is true for $m \le t$. Consider $m = t+1$. If $\mathbf{S}_{t+1}$ is an empty set, then $\mathbf{S}_{t+1-\lambda}$ is also an empty set according to (3), satisfying the lemma with $\vartheta_m = \vartheta_{m-\lambda} = 0$. If $\mathbf{S}_{t+1}$ is non-empty, let us assume that $\vartheta_{t+1-\lambda} = p$, i.e.,

$$D(\eta_{t+1-\lambda}) - D(\zeta_{t+1-\lambda}) = p. \tag{12}$$

It is required to prove that $\vartheta_{t+1} \ge p$. Let t' be the smallest m such that $t+1-m = i\lambda$ for some non-negative integer i and $\vartheta_m = p$. By definition of t' and the induction assumption (12), $\vartheta_{t'-\lambda} < p$, that is, $D(\eta_{t'-\lambda}) - D(\zeta_{t'-\lambda}) \le p-1$. However, from (9), $D(\eta_{t'}) = 1 + D(\eta_{t'-\lambda})$ and therefore,

$$D(\zeta_{t'-\lambda}) \ge D(\eta_{t'}) - p = D(\zeta_{t'}). \tag{13}$$

Consider a problem of size $n = \zeta_{t'}$, with partition sizes r and $n-r$ corresponding to a time-optimal solution on a system of degree $D(n)$. Since $T(n) = t'$, by the expression for $\mathbf{R}_n$ given in (4), time-optimality requires,

$$r \le \eta_{t'-\lambda}. \tag{14}$$

Simultaneously, using only the optimal size topology requires that, $D(r) \le D(n)-1 = D(\zeta_{t'})-1$. From (13), this translates to

$$D(r) < D(\zeta_{t'-\lambda}) \ \text{ and } \ D(n-r) < D(\zeta_{t'-\lambda}). \tag{15}$$

From (14) and (15), and since $D(n)$ is a monotonically increasing function of n within a complexity class, $r < \zeta_{t'-\lambda}$. This implies that $n - r > \zeta_{t'} - \zeta_{t'-\lambda}$. From the definition of ζ's, we have $n-r > \eta_{t'} - |\mathbf{S}_{t'}| + 1 - \eta_{t'-\lambda} + |\mathbf{S}_{t'-\lambda}| - 1$. From (2) and (3), one gets $n - r > \eta_{t'-\lambda-k} - |\mathbf{S}_{t'-\lambda-k}|$, i.e., $n - r \ge \zeta_{t'-\lambda-k}$. However, from (4), $n-r \le \eta_{t'-\lambda-k}$ and therefore,

$$T(n-r) = t' - \lambda - k. \tag{16}$$

Since $\zeta_{t'-\lambda-k}$ is the smallest element in the complexity class $\mathbf{S}_{t'-\lambda-k}$, $D(\zeta_{t'-\lambda-k}) \le D(n-r)$. Therefore, using (15),

$$D(\zeta_{t'-\lambda-k}) \le D(\zeta_{t'}) - 1. \tag{17}$$

Now, from (9),

$$D(\eta_{t'-\lambda-k}) = D(\eta_{t'}) - \lceil(\lambda + k - z)/\lambda\rceil. \tag{18}$$

where $z = (t' - T(n_0)) \bmod \lambda$. Using (17) and (18), one gets,

$$\vartheta_{t'-\lambda-k} \geq D(\eta_{t'}) - \lceil(\lambda + k - z)/\lambda\rceil - D(\zeta_{t'}) + 1$$

This implies,

$$\vartheta_{t'-\lambda-k} \geq p - \lceil(k - z)/\lambda\rceil. \tag{19}$$

By definition, $t' \leq t + 1$. Since λ and k are non-zero, $t + 1 - \lambda - k < t$ and therefore, applying the induction assumption and using (19), we get,

$$\vartheta_{t+1-\lambda-k} \geq \vartheta_{t'-\lambda-k} \geq p - \lceil(k - z)/\lambda\rceil$$

Using (9), we get,

$$D(\zeta_{t+1-\lambda-k}) \leq D(\eta_{t+1-\lambda-k}) - p + \lceil(k - z)/\lambda\rceil = D(\eta_{t-\lambda-k}) - p$$

From (12), we get,

$$D(\zeta_{t+1-\lambda-k}) \leq D(\zeta_{t+1-\lambda}). \tag{20}$$

Now, consider a problem of size $n' = \zeta_{t+1-\lambda} + \zeta_{t+1-\lambda-k}$. Clearly, $n' = \eta_{t+1-\lambda} - |\mathbf{S}_{t+1-\lambda}| + 1 - \eta_{t+1-\lambda-k} + |\mathbf{S}_{t+1-\lambda-k}| - 1$. Therefore, from (2) and (3),

$$n' = \eta_{t+1} - |\mathbf{S}_{t+1}| + 2. \tag{21}$$

We know from (12) that $\mathbf{S}_{t+1-\lambda}$ is non-empty. From (16), we know that $\mathbf{S}_{t'-\lambda-k}$ is non-empty and by definition of t', $t + 1 - t' = i\lambda$ for some integer i. Therefore, using (3), $\mathbf{S}_{t+1-\lambda-k}$ is also non-empty. Clearly now, again from (3), $|\mathbf{S}_{t+1-\lambda}| + |\mathbf{S}_{t+1-\lambda-k}| = |\mathbf{S}_{t+1}| \geq 2$ and therefore, from (21), $n' \in \mathbf{S}_{t+1}$. For the problem of size n', let us now choose partition sizes r' and $n' - r'$ as $\zeta_{t+1-\lambda}$ and $\zeta_{t+1-\lambda-k}$, respectively. It is easily seen that these partition sizes satisfy the conditions of time-optimality as required by (1). Therefore, from (8),

$$D(n') \leq 1 + \max\{D(\zeta_{t+1-\lambda}), D(\zeta_{t+1-\lambda-k})\}. \tag{22}$$

From (20) and (22),

$$D(n') \leq 1 + D(\zeta_{t+1-\lambda})$$

Thus, there exists $n' \in \mathbf{S}_{t+1}$, such that $D(\eta_{t+1}) - D(n') = D(\eta_{t+1-\lambda}) + 1 - D(n') \geq D(\eta_{t+1-\lambda}) - D(\zeta_{t+1-\lambda})$ which equals p as defined in (12). From (10), $D(\zeta_{t+1}) \leq D(n')$ and therefore, $\vartheta_{t+1} \geq p$. □

We now proceed with the analysis of the trade-offs using a scheme of classification that eases our study. An obvious convenience is achieved when we group together all problem sizes that have the same optimal time-complexity and also require the same degree hypercube for an optimal solution.

Definition 8. *Denote by $\mathbf{Q}(m, i)$, the set of all values of n for which $T(n) = m$ and $D(n) = D(\eta_m) - i$.*

Clearly, for $\mathbf{S}_m = \phi$, $\mathbf{Q}(m, i) = \phi$ for all i. Also, since $D(n)$ is a monotonically increasing function of n within each complexity class, $\mathbf{Q}(m, i) = \phi$ for $i > D(\eta_m) - D(\zeta_m)$. Of course, $\mathbf{Q}(m, i) = \phi$ for $i < 0$ as well. The following lemma expresses a recursive relationship between the sizes of these classes and subsequently leads to determining the order of ϑ_m.

Lemma 2. *For $i < \vartheta_{m-\lambda}$ and $z = (m - T(n_0)) \bmod \lambda$,*

$$|\mathbf{Q}(m, i)| = |\mathbf{Q}(m - \lambda, i)| + |\mathbf{Q}(m - \lambda - k, i - \lceil (k - z)/\lambda \rceil)|. \tag{23}$$

Proof. Since $D(n)$ is a monotonically increasing function of n within complexity classes, we have for all m and $i < \vartheta_m$,

$$|\mathbf{Q}(m, i)| = \mu(m, D(\eta_m) - i) - \mu(m, D(\eta_m) - i - 1). \tag{24}$$

Let $\vartheta_{m-\lambda} = p$. Let t' be the smallest t such that $m - t = j\lambda$ for some non-negative integer j and $\vartheta_t = p$. As in the proof of Lemma 1, we would have $\mathbf{S}_{t'-\lambda-k}$ non-empty and $\vartheta_{t'-\lambda-k} \geq p - \lceil (k - z)/\lambda \rceil$, where $z = (t' - T(n_0)) \bmod \lambda$. Therefore, from Lemma 1, we get,

$$\vartheta_{m-\lambda-k} \geq p - \lceil (k - z)/\lambda \rceil$$

Thus, for $i < p - \lceil (k - z)/\lambda \rceil$, using (24), one gets,

$$\begin{aligned} |\mathbf{Q}(m - \lambda - k, i)| = \; & \mu(m - \lambda - k, D(\eta_{m-\lambda-k}) - i) \\ & -\mu(m, D(\eta_{m-\lambda-k}) - i - 1). \end{aligned} \tag{25}$$

Recalling that $p = \vartheta_{m-\lambda}$, replacing i by $i - \lceil (k - z)/\lambda \rceil$, and using (25) along with (9), one gets, for $i < \vartheta_{m-\lambda}$,

$$\begin{aligned} |\mathbf{Q}(m - \lambda - k, i - \lceil (k - z)/\lambda \rceil)| = \; & \mu(m - \lambda - k, D(\eta_{m-\lambda}) - i) \\ & -\mu(m - \lambda - k, D(\eta_{m-\lambda}) - i - 1). \end{aligned} \tag{26}$$

From Lemma 1, $\vartheta_{m-\lambda} \leq \vartheta_m$ and therefore, applying (24) for all $i < \vartheta_{m-\lambda}$, one gets,

$$|\mathbf{Q}(m, i)| = \mu(m, D(\eta_m) - i) - \mu(m, D(\eta_m) - i - 1). \tag{27}$$

$$|\mathbf{Q}(m - \lambda, i)| = \mu(m - \lambda, D(\eta_{m-\lambda}) - i) - \mu(m, D(\eta_{m-\lambda}) - i - 1). \tag{28}$$

Using the above two equations along with (26) and (11), the lemma is proved. □

Theorem 1. *$U(m)$ is of order $\Omega(m/\log m)$.*

Proof. In proving this theorem, recall from (9) that $D(\eta_m)$ increases in steps of size no more than 1. Therefore, $U(m-1) \leq \vartheta_m \leq U(m-1) + 1$ and thus, it suffices to prove that ϑ_m is of the said order.

The proof is by determining the rate at which the quantities $|\mathbf{S}_m|$ and $|\mathbf{Q}(m, i)|$ change with increasing m, since ϑ_m is nothing but the number of non-empty subsets $\mathbf{Q}(m, i)$ of the set $\mathbf{S}_m$.

From (5), we know that for large m, $\mathbf{S}_m$ is non-empty if and only if $(m - T(n_0))$ is an integer multiple of $g = \gcd(k, \lambda)$. Since we are interested in the asymptotic order of ϑ_m, we shall assume that m is large. Consequently, if $\mathbf{S}_m$ is non-empty, it may be assumed that for finite positive integers i, $\mathbf{S}_{m-ig}$ is also non-empty. Now, depending upon the value of j for which $(m - T(n_0)) \bmod \lambda = (j\lambda + jk) \bmod \lambda$, all possible non-empty complexity classes can be grouped under λ/g different classes.

Let $i_s = \lceil (k+1)/\lambda \rceil - 1$, the smallest possible value of i below which $|\mathbf{Q}(m, i)|$ is necessarily equal to $|\mathbf{Q}(m - \lambda, i)|$ as determined by Lemma 2. Now, $|\mathbf{Q}(m, i)| = \phi$ for $i < 0$ and therefore, for $i < i_s$, Lemma 2 gives,

$$|\mathbf{Q}(m, i)| = |\mathbf{Q}(m - \lambda, i)|. \tag{29}$$

Consider $i = i_s$ and m corresponding to $j = 0$. Note that when $j = 0$, $(m - T(n_0)) \bmod \lambda = 0$ since $(j\lambda + jk) \bmod \lambda = 0$. For such m, the term $|\mathbf{Q}(m - \lambda - k, i_s - \lceil (k - z)/\lambda \rceil)|$ is 0 using (29) since $z = 0$ and $i_s - \lceil k/\lambda \rceil < i_s$. Thus, using Lemma 2, for m corresponding to $j = 0$ and $i = i_s$,

$$|\mathbf{Q}(m, i)| = |\mathbf{Q}(m - \lambda, i)|. \tag{30}$$

Consider now $j = 1$ and again, $i = i_s$. Now, in the term $|\mathbf{Q}(m - \lambda - k, i_s - \lceil (k - z)/\lambda \rceil)|$, $i_s - \lceil (k - z)/\lambda \rceil \leq i_s$ and $m - \lambda - k$ corresponds to $j = 0$. From (30), this is a constant for all m corresponding to $j = 0$ and therefore, from Lemma 2, $|\mathbf{Q}(m, i_s)|$ is a linearly increasing function of m for m corresponding to $j = 1$. By similar reasoning, $|\mathbf{Q}(m, i_s)|$ increases as m^2 for m corresponding to $j = 2$. The maximum possible value of j is $\lambda/g - 1$ and therefore, $|\mathbf{Q}(m, i_s)|$ for any m can at most increase as $m^{(\lambda/g)-1}$.

Consider now, $i = i_s + 1$ and $j = 0$. As before, since $i_s + 1 - \lceil (k - z)/\lambda \rceil \leq i_s$, the term $|\mathbf{Q}(m - \lambda - k, i_s - \lceil (k - z)/\lambda \rceil)|$ increases at most as $m^{(\lambda/g)-1}$ and therefore, from Lemma 2, $|\mathbf{Q}(m, i_s)|$ increases at most as $m^{\lambda/g}$. By similar arguments as before, $|\mathbf{Q}(m, i_s + 1)|$ for any m increases at most as $m^{(2\lambda/g)-1}$. In general, $|\mathbf{Q}(m, i)|$ for any m and $i \geq i_s$ is of order O($m^{(i - i_s + 1)(\lambda/g) - 1}$). Of course, as given by (29), $|\mathbf{Q}(m, i)|$ is O(1) for $i < i_s$.

Now, by definition,

$$|\mathbf{S}_m| = \sum_{i=0}^{\vartheta_m} |\mathbf{Q}(m, i)|. \tag{31}$$

Therefore, using (6), with $1/a$ being the largest real root of $x^{\lambda+k} + x^{\lambda} = 1$, asymptotically,

$$\begin{aligned} a^m &= \sum_{i=0}^{\vartheta_m} |\mathbf{Q}(m, i)| \leq \sum_{i=i_s}^{\vartheta_m} (m^{(i - i_s + 1)(\lambda/g) - 1}) &(32)\\ &= \sum_{i=1}^{\vartheta_m - i_s + 1} (m^{i(\lambda/g) - 1}) &(33)\\ &= m^{(\lambda/g)(\vartheta_m - i_s + 1) - 1}. &(34) \end{aligned}$$

Simplifying $a^m \leq m^{(\lambda/g)(\vartheta_m - i_s + 1) - 1}$ asymptotically, one gets $\vartheta_m \geq m/\log m$. That is, $\vartheta_m = \Omega(m/\log m)$. □

From (7), we know that $T(n)$ is of order $\mathrm{O}(\log n)$. Thus, the order of the reduction in the degree of the hypercube required when such a trade-off can be used for a problem of size n is O($\log n/\log \log n$). This large decrease in the degree of the hypercube required forces an increase in the execution time equal to the difference in the complexities of the adjacent complexity classes. From (5), this difference is exactly equal to g for problems larger than a certain size. Recall that g is a quantity equal to or smaller than even λ and k, and that λ is just the computational cost of a single merging step and k is just the cost of a single communication step. Thus, with a small constant increase in the execution time, one can achieve a large decrease in the size of the hypercube required.

This surprising trade-off can also be understood by noting that, for a given problem, the rate of decrease in the execution time reduces as the size of the topology used to solve the problem increases beyond a certain point. Increasing the degree of the hypercube to something larger than $D(n)$, by definition, does not help improve the execution time. Thus, the rate of decrease in the execution time really becomes small as it reaches the point where there is no more of a reduction in execution time with an increase in the size of the hypercube. Thus, at this point, a small sacrifice in the execution time easily leads to a large reduction in the degree of the hypercube required.

Figure 3 shows the behavior of the quantity ϑ_m plotted only for m such that $(m - T(n_0))$ is an integer multiple of λ. A simple constant complexity behavior is assumed for $n < n_0$ and the two cases considered differ only in the value of λ. Figure 3 confirms the statement of equation (34) that the rate of increase of ϑ_m with respect to m is also dependent on the quantity λ/g. The smaller this quantity, the higher is the rate of increase of ϑ_m. Figure 3 and Equation (34) highlight an apparently unexpected behavior of $U(m)$—a very small increase or decrease in the overheads λ or k can significantly change the value of g, and therefore the possible time-resource trade-offs. However, recall that at asymptotic values, i.e., for large m, adjacent complexity classes differ in their complexities by g and therefore, when g is large, a larger sacrifice in the time-complexity is incurred. This larger increase in the time leads to a larger decrease in the degree of the hypercube required. The dependence of ϑ_m on the overheads λ and k as shown by Figure 3 is therefore predictably consistent with intuition.

5 Conclusion and Future Work

In this work, we have considered the problem of minimizing the degree of the hypercube required to solve the problem in optimal time using the parallel divide-and-conquer technique. Based on a mathematical computational model, we have obtained a recursive analytical expression for the minimum degree $D(n)$ of a hypercube topology needed to solve a problem of size n in *optimal* time. We have obtained several properties of $D(n)$ as a function of n. Recall that $D(n)$ is actually a property (height or the number of levels) of the optimal computational tree and therefore, the results presented here need not be seen as specific to the hypercube topology. For example, the quantity $D(n)$ can also be interpreted as the minimum length of keys required to create a worst-case time-optimal key-based search tree where the search at level-i of the tree uses the i-th symbol in the key.

Finally we show that it may be possible to achieve some very attractive trade-offs between the size of the hypercube required and the execution time. In particular, we have shown that, with a small constant sacrifice in time-optimality, it may be possible to reduce the degree of the hypercube required for a problem of size n by an amount of the order of O($\log n/\log\log n$). This constant sacrifice in time is equal to λ which is just the cost in execution time of a single merging step in the divide-and-conquer recursion.

The discussion in this paper considers only the size of the hypercube required to solve a problem optimally, but not on how one might execute a problem optimally given a hypercube of a certain size. Investigating this latter harder problem requires knowledge of the sequential time-complexity of the problem, i.e., a complete characterization of the execution time of the problem on a single processor, as a function of the problem size. Incorporating this into the computational model of (1), and solving the recursive equations for partition sizes is a matter for future research.

Some divide-and-conquer algorithms have non-constant overheads associated with communication or merging steps. For example, in merge sorting, the merging overhead λ is not a constant but a logarithmic function of n, the problem size. A study of such algorithms will require modifications to the computational model used in this paper.

References

[1] V. Lo, S. Rajopadhye, and J. A. Telle, "Parallel divide and conquer on meshes," *IEEE Trans. Par. Dist. Sys.*, vol. 7, pp. 1049–1058, Oct. 1996.

[2] E. W. Mayr and R. Werchner, "Divide-and-conquer algorithms on the hypercube," *Theoretical Computer Science*, no. 162, pp. 283–296, 1996.

[3] Z. Li and E. M. Reingold, "Solution of a divide-and-conquer maximin recurrence," *SIAM J. Comput.*, vol. 18, pp. 1188–1200, Dec. 1989.

[4] H. Sethu and M. D. Wagh, "Design of time-optimal hardware-efficient divide-and-conquer algorithms," in *Proc. Int't Conf. Par. Proc.*, (St. Charles, IL), Aug. 1992.

[5] Q. F. Stout, "Supporting divide-and-conquer algorithms for image processing," *J. Par. Dist. Comput.*, vol. 4, pp. 95–115, Feb. 1987.

[6] B. S. Veroy, "Average complexity of divide-and-conquer algorithms," *Info. Proc. Let.*, vol. 29, no. 6, pp. 319–326, 1988.

[7] M. J. Atallah, R. Cole, and M. T. Goodrich, "Cascading divide-and-conquer: A technique for designing parallel algorithms," *SIAM J. Comput.*, vol. 18, pp. 499–532, June 1989.

[8] A. Saha and M. D. Wagh, "On parallel recursive computations with variable partition overheads and constant recombination overheads," *Int'l J. Appl. Soft. Tech.*, vol. 2, no. 1, pp. 1–19, 1996.

[9] A. Saha and M. D. Wagh, "Solutions of two minmax recurrnces in parallel processing with variable recombination overhead," *Applied Mathematics and Computation*, vol. 76, pp. 173–211, May 1996.

[10] S. Kapoor and E. M. Reingold, "Optimum lop-sided binary trees," *J. ACM*, vol. 36, pp. 573–590, July 1989.

[11] D. M. Choy and C. K. Wong, "Construction of optimal α-β leaf trees with applications to prefix code and information retrieval," *SIAM J. Comput.*, vol. 12, pp. 426–446, Aug. 1983.

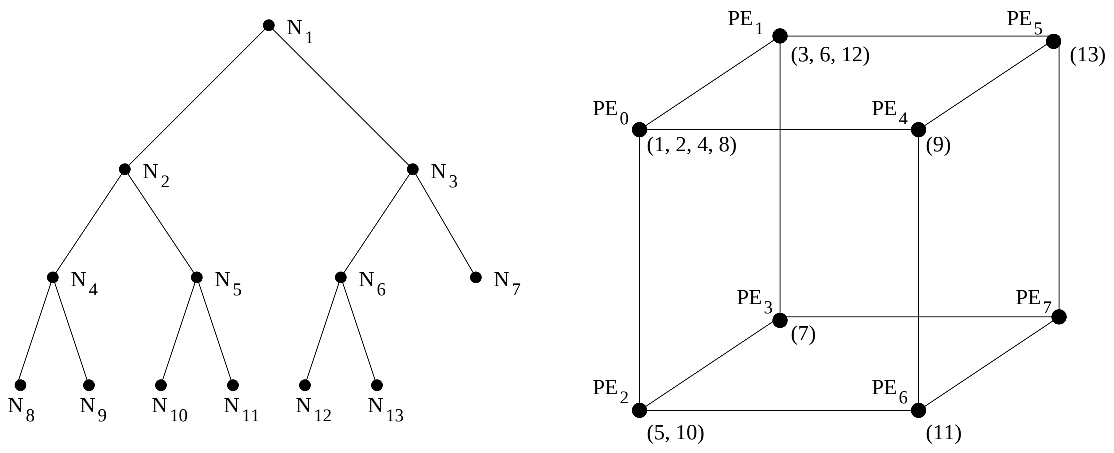

Figure 1: A 3-level computational tree and its mapping on a hypercube.

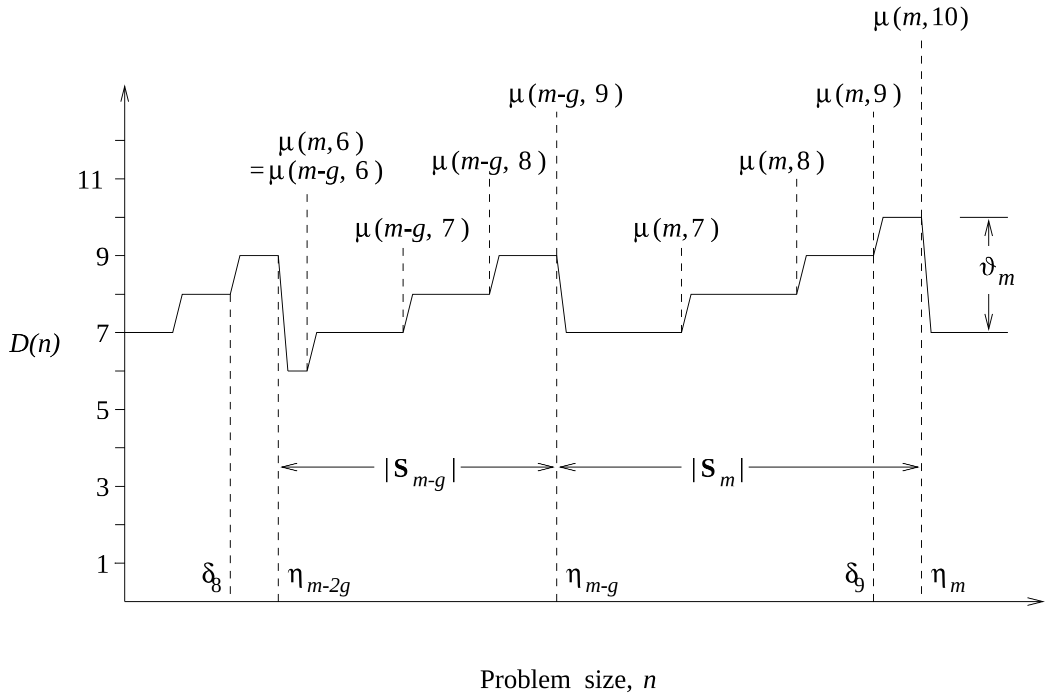

Figure 2: A typical plot of $D(n)$ as a function of n.

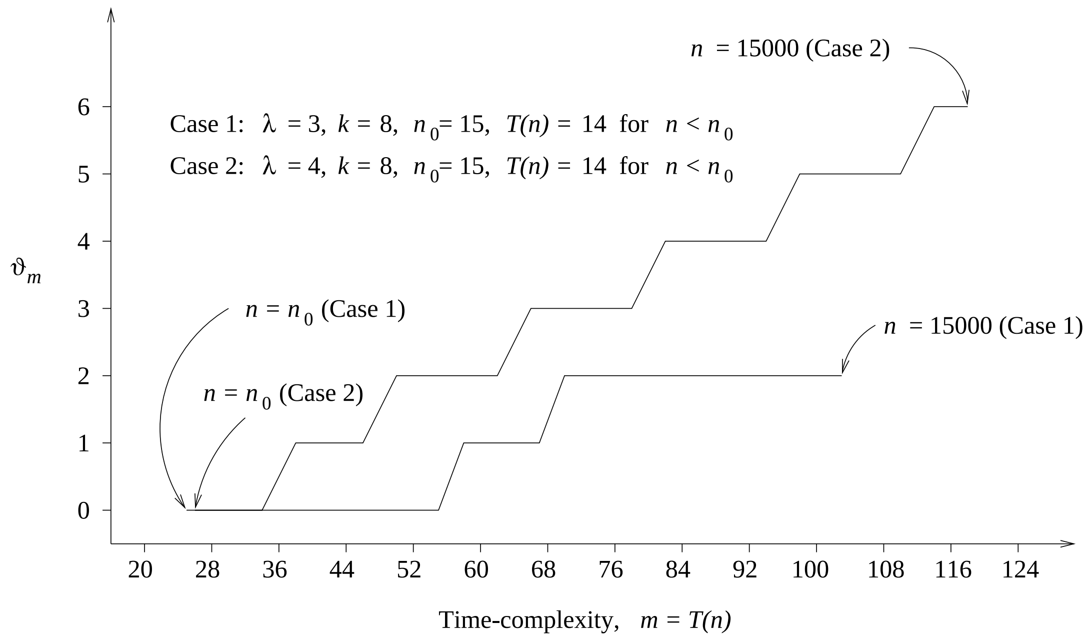

Figure 3: Relationship between the overheads and the trade-offs.