Proceedings of the IASTED International Conference on
Parallel and Distributed Computing and Systems
November 3–6, 1999 in Cambridge, Massachusetts, USA

Node-Degree Requirements for Time-Optimal Execution of a Class of Parallel Algorithms

Harish Sethu
sethu@ece.drexel.edu
Department of ECE
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104

Meghanad D. Wagh
mdw0@lehigh.edu
Department of EECS
Lehigh University
Packard Laboratory #19
Bethlehem, PA 18015

Abstract. *In this paper, we consider the time-optimal execution of a class of parallel algorithms based on the principle of divide-and-conquer. We use a computational model of this class of algorithms which incorporates both communication and the associated computation overheads that are unavoidable in a divide-and-conquer methodology. We consider the time-optimal implementation of this class of parallel algorithms on a generic network topology and derive properties of the node-degree requirements of the processors in the topology. We say that a processor is assigned a problem of size n if it is assigned the task of the root node in the computational tree graph for a problem of size n. We obtain a recursive analytical expression for $C(n)$ as a function of n, defined as the lower bound on the node-degree of a processor assigned to solve, in optimal time, a problem of size n. We prove several interesting properties of $C(n)$ as a function of n, including its well-behaved but non-monotonic nature. In this paper, we lay the groundwork based on which one may derive algorithms for complete analytical characterization of $C(n)$, and therefore, optimal partitioning strategies for time-optimal execution with minimal connectivity requirements.*

Keywords. node-degree, optimal partitioning, divide-and-conquer, overheads.

1 Introduction

Among the few general techniques available for parallel algorithm design, the technique of divide-and-conquer is one of the most widely used and studied [1–6]. The most common applications that use the divide-and-conquer approach include many signal and image processing tasks, evaluation of arithmetic expressions, finding the extremum, simple database searches or complex data mining operations, and many other applications which involve associative operations on large sets of data. 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 [7, 8]. 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 communication overhead, k, attached to only one partition and a symmetric merging overhead, λ, added to both the partitions. The communication overhead, k, primarily includes the cost of interprocessor communication to collect results, while the merging overhead, λ, primarily includes the algorithmic costs of combining the partial results. Depending on the specific problem, either k or λ, or both, may include the costs associated with converting one or both subproblems to conform to the exact form of the original problem. This model can be summarized by a typical recursion step in which a problem of size $n \geq n_0$, is partitioned into problems of sizes r and $n - r$ as follows.

$$T(n) = \min_{1 \leq r \leq n-1} [\max\{T(r),\, T(n-r) + k\}] + \lambda \quad (1)$$

where $T(n)$ is the minimal execution time or the *time-complexity* of P_n, a size n instance of the problem being solved. In solving the above recursion, we assume that 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 size for any given n is not necessarily unique. The above recursion has also been analyzed in [9].

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 P_1 and P_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 P_1, we would have P_2

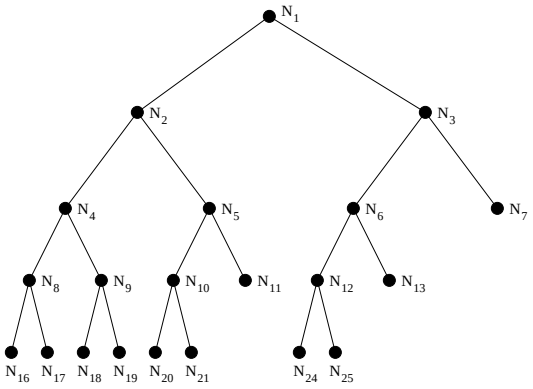

Figure 1: A computational tree graph.

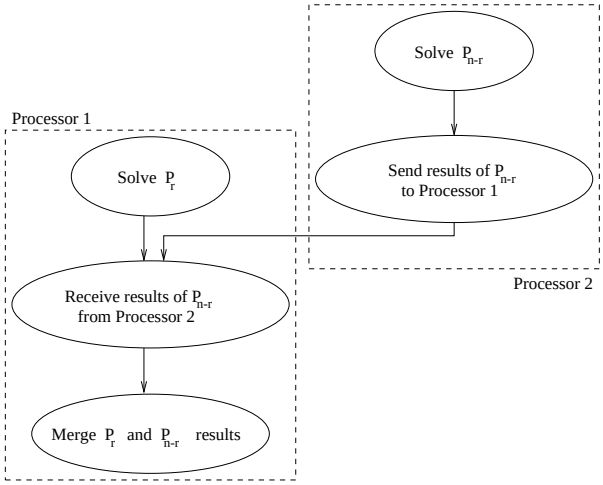

Figure 2: Task assignment on two processors.

communicating its result, the sum of $n - r$ numbers, to P_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 P_2 to P_1 be k, and let the time taken to merge this result with that computed in P_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.

Fig. 1 illustrates the computational tree graph of this task distribution strategy. The tasks in the figure are labeled from N_1 through N_{25}. Note that the leaves of the tree in Fig. 1 represent the computation of subproblems that are not further divided and its other nodes represent merging of the results of subproblems. Each node of the tree thus represents a computation and *not* a processing unit. In our task distribution strategy, tasks N_1, N_2, N_4, N_8 and N_{16}, for example, are all executed in the same processing unit. Similarly, tasks N_3, N_6, N_{12} and N_{24} are all executed in the same processing unit.

The recursive task distribution technique described above to minimize communication and maximize processor utilization can be illustrated using the simple case of two processors. Consider a problem P_n divided into subproblems P_r and P_{n-r}. A processor is said to have been assigned P_n (or is the root processor of P_n) if it combines the results of its subproblems P_r and P_{n-r}. In our model, P_r is assigned the same processor as P_n. P_{n-r} is assigned a separate processor and the results of the subproblems are combined in the processor that was assigned the parent problem. Fig. 2 illustrates this task distribution for two processors.

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 while attempting to design reconfigurable or dedicated systems for this class of algorithms, it is vital to minimize the resource requirements. One of the more important resource constraints tends to be the connectivity or the node-degree of processing units, i.e., the number of other processing units any given processing unit may be directly connected to. In this paper, we study the connectivity requirements of time-optimal divide-and-conquer algorithms in the presence of parallel computing overheads. In our analysis, lower bounds on the node degrees of each of the processors and also the specific interconnections required are determined as functions of the problem size. This can directly lead to the design of optimal network topologies for reconfigurable or dedicated systems, thus improving the processor and link utilization while retaining time-optimality.

It will be shown in Section 2, that a multitude of optimal partition sizes, r, exist for any given problem of size n, i.e., a number of different values of r satisfy the equation, $T(n) = \max\{T(r), T(n-r)+k\} + \lambda$. Each of these optimal solutions may impose different requirements on the node-degrees of individual processors. The required network connectivity, therefore, differs from one solution to another. The analysis presented here concentrates on obtaining those solutions which use minimal connectivity while preserving time-optimality.

In designing an optimal topology for a dedicated or a reconfigurable system, we need quantitative estimates of the node-degree requirements of each of the processors participating in the algorithm execution. Denote by $C(n)$, the least number of links required of the processor assigned to P_n in solving the problem optimally as in (1). In accordance with the task distribution strategy explained earlier, P_n and the subproblem P_r are assigned the same processor. This processor needs to communicate with the processor assigned to P_{n-r} besides those it already communicates with to solve P_r. Therefore, to solve a larger problem P_n, a processor requires exactly one more link than to solve P_r. Optimizing the communication network complexity while

preserving time complexity then implies that $C(n)$ be obtained at every stage of recursion through,

$$C(n) = 1 + \min_{r \in \mathbf{R}_n} C(r), \quad n \geq n_0. \tag{2}$$

where, $\mathbf{R}_n$ is the set of all optimal partition sizes r as given by (1), for a problem of size n. Problems of size less than n_0 are not partitioned and are solved in one processor, and therefore, $C(n) = 0$ when $n < n_0$. It should be noted that $C(n)$ denotes the smallest number of times that a problem of size n requires to be divided while solving it optimally. At every stage of the recursion, use of a partition size r with the smallest $C(r)$ ensures this minimization. Using the lower bounds on connectivity as dictated by (2), therefore, leads to a topology which is optimal in terms of communication network complexity.

$C(n)$ plays a central role in defining the connectivity requirements of the class of algorithms under consideration. In the following section, we present some of the earlier results obtained on properties of the recursion (1). In Section 3, we investigate and prove some of the properties of $C(n)$ as a function of n, a few of which were stated without proof in [10]. These include the somewhat counter-intuitive non-monotonic nature of $C(n)$ as a function of n. Section 4 describes a few more properties of $C(n)$ and lays down the basic principles using which it is algorithmically possible to readily obtain a complete characterization of $C(n)$.

2 Definitions and Preliminary Results

Let the size of a problem, n, be a positive integer. Let $\mathbf{S}_m$ denote the set of sizes of problems that have the same execution time m as given by (1). Thus, we have,

$$\mathbf{S}_m = \{n \in \mathbf{Z} \,|\, T(n) = m\}. \tag{3}$$

Note that it is possible that $\mathbf{S}_m = \phi$. Define η_m recursively as the largest element of $\mathbf{S}_m$ if $\mathbf{S}_m$ is non-empty and η_{m-1} if $\mathbf{S}_m$ is empty. 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 [9] that the η_m's are related by,

$$\eta_m = \eta_{m-\lambda} + \eta_{m-\lambda-k}, \quad m \geq T(n_0). \tag{4}$$

One may understand (4) 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 (4) that the sizes of the complexity classes are also similarly related, given by the recursion,

$$|\mathbf{S}_m| = |\mathbf{S}_{m-\lambda}| + |\mathbf{S}_{m-\lambda-k}|, \quad m \geq T(n_0). \tag{5}$$

Recall that $\mathbf{R}_n$ is 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 $T(n,r) = \max\{T(r), T(n-r)+k\} + \lambda$. Therefore, 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 can be interpreted to mean 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.

$$\mathbf{R}_n = [n - \eta_{m-\lambda-k}, \eta_{m-\lambda}], \quad n \geq n_0, \tag{6}$$

From (4) and (6), it is easily derived that the optimal partition size, r, for a problem of size η_m, is given by,

$$\mathbf{R}_{\eta_m} = \{\eta_{m-\lambda}\} \tag{7}$$

A close inspection of (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, k and λ are positive integers. To be more general and include non-integer values of k and λ in our results, we redefine g as,

$$g = \gcd(\lambda\alpha, k\alpha)/\alpha, \tag{8}$$

where α is such that both $\lambda\alpha$ and $k\alpha$ are integer quantities. This result can be more formally expressed as follows.

$$\mathbf{S}_m \neq \phi \text{ iff } g|(m - T(n_0)), \text{ for } m \gg T(n_0) \tag{9}$$

3 Elementary Properties of C(n)

The function $C(n)$ of n is a well behaved function, exhibiting piecewise monotonicity and having an asymptotic order of the time-complexity. Theorem 1 states that within a complexity class, $C(n)$ increases monotonically with problem size n. Theorem 2 gives the value of $C(n)$ when n is on the upper boundary, η_m, of the complexity class $\mathbf{S}_m$.

Theorem 1. For $n \in \mathbf{S}_m$, $C(n)$ is a monotonically increasing function of n.

Proof. When $T(n) = m < T(n_0)$, $n < n_0$ and therefore, $C(n) = 0$ for all $n \in \mathbf{S}_m$. Thus the theorem is true for all such m.

Consider problem sizes $n_1, n_2 \in \mathbf{S}_m$, $m \geq T(n_0)$. It is required to prove that if $n_1 \leq n_2$, then $C(n_1) \leq C(n_2)$. The optimal partition sizes of these problems given by (6) are,

$$\mathbf{R}_{n_1} = [n_1 - \eta_{m-\lambda-k}, \eta_{m-\lambda}], \tag{10}$$

$$\text{and} \qquad \mathbf{R}_{n_2} = [n_2 - \eta_{m-\lambda-k}, \eta_{m-\lambda}]. \tag{11}$$

If $n_1 \leq n_2$, then $\mathbf{R}_{n_2} \subseteq \mathbf{R}_{n_1}$, implying,

$$\min_{r \in \mathbf{R}_{n_1}} C(r) \leq \min_{r \in \mathbf{R}_{n_2}} C(r). \tag{12}$$

Now, by the definition of $C(n)$ in (2), $C(n_1) \leq C(n_2)$. □

The statement of Theorem 1 is consistent with intuition. It is easy to comprehend that if two problems require the same time, the minimum network complexity required by the larger problem is at least as large as that of the smaller one.

The monotonicity of $C(n)$ within complexity classes enables us to obtain an easy upper bound on $C(n)$ by merely determining the value of $C(n)$ at upper boundaries of complexity classes. Theorem 2 achieves this by providing the value of $C(n)$ for $n = \eta_m$.

Theorem 2. For all $m \geq T(n_0)$, $C(\eta_m) = \lceil \frac{(m-T(n_0)+1)}{\lambda} \rceil$.

Proof. From (7), the optimal partition size for a problem of size η_m is unique and equal to $\eta_{m-\lambda}$. Therefore,

$$C(\eta_m) = 1 + C(\eta_{m-\lambda}) = i + C(\eta_{m-i\lambda}). \qquad (13)$$

The above recursion can be carried out until $\eta_{m-i\lambda} < n_0$, or, $m - i\lambda < T(n_0)$. At this point $C(\eta_{m-i\lambda}) = 0$ and thus $C(\eta_m)$ is merely equal to the smallest such i. □

The following Corollary which readily follows from Theorems 1 and 2, has important design implications.

Corollary 1. For all $n \leq \eta_m$, $C(n) \leq C(\eta_m)$.

This Corollary states that if a processor satisfies the connectivity requirements of a problem of size η_m assigned to it, then it will satisfy the requirements of any problem of size less than or equal to η_m.

As seen in Theorem 1, the connectivity requirements of the root processor increase monotonically within a given complexity class as the problem size n increases. An important design issue, however, is the smallest size of a problem that necessarily requires a certain network complexity for a time-optimal execution. Theorem 3 addresses this question. The proof of Theorem 3 requires the following two lemmas.

Lemma 1. If $\lambda|(m - T(n_0))$ and $j \leq |\mathbf{S}_{T(n_0)}|$, then

$$C(\eta_m - j) = C(\eta_m) \Leftrightarrow C(\eta_{m-\lambda} - j) = C(\eta_{m-\lambda}).$$

Proof. We assume the statement on the left hand side of the equivalence and derive the right hand side. The converse can be proved by backtracking the steps.

Recall that the sizes of the complexity classes are related as $|\mathbf{S}_m| = |\mathbf{S}_{m-\lambda}| + |\mathbf{S}_{m-\lambda-k}|$ given by (5). Clearly, $|\mathbf{S}_m| \geq |\mathbf{S}_{m-\lambda}|$. Note that $|\mathbf{S}_{T(n_0)}|$ is non-zero by definition of n_0 and therefore, $\lambda|(m-T(n_0))$ implies that $|\mathbf{S}_m|$ is non-zero and greater than or equal to $|\mathbf{S}_{T(n_0)}|$. Thus, when $j \leq |\mathbf{S}_{T(n_0)}|$, either $(\eta_m - j) \in \mathbf{S}_m$ or $(\eta_m - j) = \eta_p$ for some $p < m$. However, the latter case is impossible because $\lambda|(m - T(n_0))$ implies $C(\eta_m - j) < C(\eta_m)$ from Theorem 2, contradicting the left hand side of the equivalence. Hence, if $j \leq |\mathbf{S}_{T(n_0)}|$, and $\lambda|(m - T(n_0))$, then $C(\eta_m - j) = C(\eta_m)$, implying $(\eta_m - j) \in \mathbf{S}_m$.

From equation (6) and the definition of $C(n)$, and upon further simplification using (4), we get,

$$\begin{aligned} C(\eta_m - j) &= 1 + \min_{r \in [\eta_m - j - \eta_{m-\lambda-k},\ \eta_{m-\lambda}]} C(r) \\ &= 1 + \min_{r \in [\eta_{m-\lambda} - j,\ \eta_{m-\lambda}]} C(r). \end{aligned}$$

Combining the above equation with (7), and the assumption of the left hand side, $C(\eta_m - j) = C(\eta_m)$, one gets,

$$C(\eta_{m-\lambda}) = \min_{r \in [\eta_{m-\lambda} - j,\ \eta_{m-\lambda}]} C(r). \qquad (14)$$

However, using Theorem 2 and the fact that $\lambda|(m - \lambda - T(n_0))$, one can see that if $\eta_{m-\lambda} - j \notin \mathbf{S}_{m-\lambda}$, then, $C(\eta_{m-\lambda} - j) < C(\eta_{m-\lambda})$. This is in contradiction with (14). Therefore, $\eta_{m-\lambda} - j \in \mathbf{S}_{m-\lambda}$ and therefore, in the light of Theorem 1, Equation (14) yields $C(\eta_{m-\lambda} - j) = C(\eta_{m-\lambda})$. □

Lemma 2. If $\lambda|(m - T(n_0))$ then the largest integer j such that $C(\eta_m - i) = C(\eta_m)$ for $0 \leq i \leq j$, is $|\mathbf{S}_{T(n_0)}| - 1$.

Proof. From the monotonic nature of $C(n)$ within a complexity class and because $|\mathbf{S}_m| \geq |\mathbf{S}_{T(n_0)}|$, the theorem could be proved by showing that,

$$C(n) < C(\eta_m) \text{ for } n \leq \eta_m - |\mathbf{S}_{T(n_0)}|, \qquad (15)$$

$$\text{and } C(\eta_m - |\mathbf{S}_{T(n_0)}| + 1) = C(\eta_m) \qquad (16)$$

If $(\eta_m - |\mathbf{S}_{T(n_0)}|) \notin \mathbf{S}_m$, then $(\eta_m - |\mathbf{S}_{T(n_0)}|) = \eta_p$ for some $p < m$ since $|\mathbf{S}_m| \geq |\mathbf{S}_{T(n_0)}|$. Now, $\lambda|(m - T(n_0))$ and therefore, from Theorem 2, $\eta_p < \eta_m$. Using Corollary 1 now, Equation (15) is readily proved.

If $(\eta_m - |\mathbf{S}_{T(n_0)}|) \in \mathbf{S}_m$, we prove (15) by contradiction. Assume $C(\eta_m - |\mathbf{S}_{T(n_0)}|) = C(\eta_m)$. Lemma 1 can then be applied recursively to get $C(\eta_{T(n_0)} - |\mathbf{S}_{T(n_0)}|) = C(\eta_{T(n_0)})$. Now, $C(\eta_{T(n_0)} - |\mathbf{S}_{T(n_0)}|) = C(n_0 - 1) = 0$, while Theorem 2 tells us that $C(\eta_{T(n_0)}) = 1$, a contradiction. Therefore,

$$C(\eta_m - |\mathbf{S}_{T(n_0)}|) < C(\eta_m). \qquad (17)$$

As before, since $\lambda|(m-T(n_0))$, $\eta_m > \eta_p$ for $p < m$, from Theorem 2. Now, (15) follows using Corollary 1, Theorem 1 and (17).

Consider the following equation, both sides of which are initially equal to 1.

$$C(\eta_{T(n_0)} - |\mathbf{S}_{T(n_0)}| + 1) = C(\eta_{T(n_0)}). \qquad (18)$$

Equation (16) is proved by repeated application of Lemma 1 to Equation (18). □

We now introduce a function β_d to denote the smallest n such that $C(n) = d$. This essentially means that all problems of size less than β_d can be solved optimally with less than d links from the root processor. β_d is an important characterizing parameter since it marks the first failure from optimality in a given interconnection network topology with limited communication links.

Note that from definition (2), given any $\beta_d = n$, there exists an r such that $C(r) = d - 1$. Since $r < n$, $\beta_{d-1} < \beta_d$, and therefore, β_d is a strictly increasing function of d. The following theorem expresses β_d in terms of the complexity class boundaries.

A plot of $C(n)$ against n, is shown in Figure 3, illustrating the notations employed in this article.

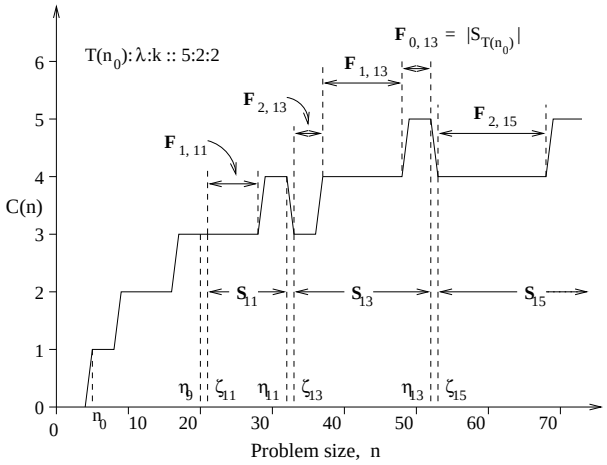

Figure 3: An illustrative plot of $C(n)$ against n.

Theorem 3. For all $d \geq 1$, $\beta_d = \eta_{(d-1)\lambda+T(n_0)} - |\mathbf{S}_{T(n_0)}| + 1$.

Proof. For all $d \geq 1$, by Lemma 2, $\eta_{(d-1)\lambda+T(n_0)} - |\mathbf{S}_{T(n_0)}| + 1$ is the smallest n such that $C(n) = C(\eta_{(d-1)\lambda+T(n_0)}) = d$. □

4 Complete Characterization of C(n)

In this section we lay the groundwork based upon which one may completely characterize $C(n)$ as a function of n, and thus obtain partitioning strategies for time-optimal execution with minimal connectivity requirements. The need for a complete characterization may arise when the network topology is reconfigurable or when the overheads are known and a dedicated system is to be designed. While such reconfigurability enables one to tailor the topology according to the problem at hand, a time-optimal execution requires a thorough knowledge of the connectivity requirements of the problem.

In several applications in science and engineering, the overheads as well as the expected problem sizes are known prior to execution time. In such situations, if problems of only a known kind are expected, the design of a dedicated system with an optimal network topology is a cost-effective alternative. An example is an image processing engine on satellites or robotic vehicles, implemented using multiple processing units computing in parallel. Determining this topology for a proposed dedicated network also requires a complete characterization of the connectivity function, which is the objective in the following sections of this article.

The value of $C(n)$ for n at the upper boundaries of complexity classes is known from Theorem 2. Also since $C(n)$ is monotonically increasing in each complexity class as stated by Theorem 1, it is a piecewise staircase function as shown in Fig. 3. It is easily seen that $C(n)$ is uniquely determined for every n if the step sizes are known for each of the steps in the different complexity classes. The following gives a formal definition of these step sizes.

For $\mathbf{S}_m \neq \phi$, define ζ_m as the size of the smallest problem that necessarily incurs a time-complexity of at least m for a time-optimal solution, i.e., for $\mathbf{S}_m \neq \phi$, $\zeta_m = \min \mathbf{S}_m$. For mathematical convenience, we define $\zeta_m = \eta_m$ when $\mathbf{S}_m = \phi$. However, in all of the following, $\mathbf{S}_m \neq \phi$ is an implicit assumption associated with any reference to ζ_m, unless otherwise stated.

Let $\mathbf{S}_m \neq \phi$. Define $\mathbf{F}_{x,m}(k,\lambda)$, $0 \leq x < C(\eta_m) - C(\zeta_m)$, as the set of integers n such that $T(n) = m$ and $C(n) = C(\eta_m) - x$. Clearly, since $C(n)$ is monotonically increasing within complexity classes, $\mathbf{F}_{x,m}(k,\lambda)$ is a set of contiguous integers for all x, m, k and λ. For $\mathbf{S}_m = \phi$, i.e., in case of empty complexity classes, $\mathbf{F}_{x,m}(k,\lambda) = \phi$. Note that from Lemma 2, if $\lambda|(m - T(n_0))$, then $\mathbf{F}_{0,m}(k,\lambda) = |\mathbf{S}_{T(n_0)}|$. In the following, $\mathbf{F}_{x,m}(k,\lambda)$ is merely referred to as $\mathbf{F}_{x,m}$ whenever the overheads are not central to the discussion or when the values of the overheads are contextually obvious.

In all of the following analysis, the time complexity of problems smaller that n_0 is assumed to be a constant equal to t_0. This is a valid assumption for a divide-and-conquer approach with fine-grain parallelism such as in image processing.

Theorem 4. For $\mathbf{S}_m \neq \phi$, and $0 \leq x < C(\eta_m) - C(\zeta_m)$,

$$|\mathbf{F}_{x,m+\lambda}(k,\lambda)| = |\mathbf{F}_{x,m}(k,\lambda)|. \tag{19}$$

Proof. We set out to prove that the function $C(\eta_m - i)$ has the same shape as that of $C(\eta_{m-\lambda} - i)$ for $i < |\mathbf{S}_{m-\lambda}|$. This will prove that the cardinalities of $\mathbf{F}_{x,m}(k,\lambda)$ are the same for values of m that differ by λ.

Consider two problems of sizes $\eta_{m+\lambda} - i$ and $\eta_{m+\lambda} - i - 1$ where $i < |\mathbf{S}_m| - 1$. Using (5), $i + 1 < |\mathbf{S}_{m+\lambda}|$ and therefore, both these problem sizes belong to the complexity class $\mathbf{S}_{m+\lambda}$. From Theorem 1, i.e., from the monotonic nature of $C(n)$ within a class, one gets $C(\eta_{m+\lambda} - i - 1) \leq C(\eta_{m+\lambda} - i)$. Let

$$C(\eta_{m+\lambda} - i - 1) + l = C(\eta_{m+\lambda} - i), \quad l \geq 0. \tag{20}$$

Now, by the definition of $C(n)$ as in (2), one gets,

$$l + 1 + \min_{r \in \mathbf{R}_{\eta_{m+\lambda}-i-1}} C(r) = 1 + \min_{r \in \mathbf{R}_{\eta_{m+\lambda}-i}} C(r). \tag{21}$$

Recall from (6) that, $\mathbf{R}_{\eta_{m+\lambda}-i}$ and $\mathbf{R}_{\eta_{m+\lambda}-i-1}$ are given by,

$$\mathbf{R}_{\eta_{m+\lambda}-i} = [\eta_{m+\lambda} - i - \eta_{m-k}, \eta_m] \tag{22}$$

$$\mathbf{R}_{\eta_{m+\lambda}-i-1} = [\eta_{m+\lambda} - i - 1 - \eta_{m-k}, \eta_m] \tag{23}$$

Using (4), Equations (22) and (23) become,

$$\mathbf{R}_{\eta_{m+\lambda}-i} = [\eta_m - i, \eta_m] \tag{24}$$

$$\mathbf{R}_{\eta_{m+\lambda}-i-1} = \eta_m - i - 1, \eta_m] \tag{25}$$

Now, since $i + 1 < |\mathbf{S}_m|$, the ranges of optimal partition sizes given by equations (24) and (25) are entirely within

the set $\mathbf{S}_m$. Note once again that the connectivity function is monotonically increasing within a complexity class and therefore, the minimum value of $C(n)$ within such a range corresponds to that of the smallest n. Thus, Equation (20) becomes,

$$C(\eta_m - i - 1) + l = C(\eta_m - i). \tag{26}$$

The derivation of (26) from (20) implies that whenever there is a drop in $C(n)$ as we trace its value down from $\eta_{m+\lambda}$, there is a corresponding drop at exactly the same intervals as we move down from η_m. Clearly, the steps (20–26) imply that for $1 \leq i \leq |\mathbf{S}_m| - 1$,

$$C(\eta_{m+\lambda}) - C(\eta_{m+\lambda} - i) = C(\eta_m) - C(\eta_m - i). \tag{27}$$

Translating the above Equation (27) in terms of $\mathbf{F}_{x,m}(k,\lambda)$ and $\mathbf{F}_{x,m+\lambda}(k,\lambda)$, one readily obtains the statement of the theorem. □

Note that the difference in the levels of the consecutive steps in Fig. 3 are never more than one within each complexity class. This observation that the value of $C(n)$ with each complexity class increases at most by 1 between steps, is stated in the following theorem.

Theorem 5. For $n, n+1 \in \mathbf{S}_m$, $C(n+1) \leq C(n) + 1$.

Proof. This proof is by induction. The basis step is readily verified for all $m \leq T(n_0)$.

Using the definition of $C(n)$ given by (2), it is required to be proved that,

$$\min_{r \in \mathbf{R}_{n+1}} C(r) \leq 1 + \min_{r \in \mathbf{R}_n} C(r). \tag{28}$$

for $n, n+1 \in \mathbf{S}_t$, while the statement of the theorem is assumed true for $m < t$. Now, from (6), the set $\mathbf{R}_{n+1}$ contains problem sizes from $n + 1 - \eta_{t-\lambda-k}$ to $\eta_{t-\lambda}$ while the set $\mathbf{R}_n$ contains all of these in addition to that corresponding to problem size $n - \eta_{t-\lambda-k}$. Now, letting $\min \mathbf{R}_{n+1} = r'$, Equation (28) translates to,

$$C(r') \leq 1 + \min\{C(n - \eta_{t-\lambda-k}), C(r')\}. \tag{29}$$

Now, problem sizes $n - \eta_{t-\lambda-k}$ and r' may or may not belong to the same complexity class. If they do belong to the same class, then since $n - \eta_{t-\lambda-k} < r'$, by definition of r', $C(r') = C(n + 1 - \eta_{t-\lambda-k})$. Now, since both $T(r')$ and $T(n - \eta_{t-\lambda-k})$ are less than m, by the induction assumption,

$$C(r') \leq 1 + C(n - \eta_{t-\lambda-k}). \tag{30}$$

which proves Equation (29).

Now, if $n - \eta_{t-\lambda-k}$ and r' do not belong to the same complexity class, one gets,

$$C(r') \leq C(n + 1 - \eta_{t-\lambda-k}). \tag{31}$$

If $n+1-\eta_{t-\lambda-k}$ belongs to the same class as $n-\eta_{t-\lambda-k}$, then Equation (29) is proved along the same lines as before. However, if $n+1-\eta_{t-\lambda-k}$ belongs to a class corresponding to a complexity higher than that of $n-\eta_{t-\lambda-k}$, then, clearly $n-\eta_{t-\lambda-k}$ must be at the boundary of its complexity class. In other words,

$$n - \eta_{t-\lambda-k} = \eta_j, \tag{32}$$

for some j. Therefore,

$$T(n + 1 - \eta_{t-\lambda-k}) = j + \epsilon, \tag{33}$$

where $j + \epsilon$ is the smallest complexity larger than j corresponding to a non-empty complexity class. From the recursion of (5), $\epsilon \leq \lambda$. Now, applying Theorem 2, one gets,

$$C(\eta_{j+\epsilon}) \leq 1 + C(\eta_j). \tag{34}$$

From (33) and Corollary 1, $C(n+1-\eta_{t-\lambda-k}) \leq C(\eta_{j+\epsilon})$, and therefore, using (32) and (34),

$$C(n + 1 - \eta_{t-\lambda-k}) \leq 1 + C(n - \eta_{t-\lambda-k}). \tag{35}$$

From (31) and (35),

$$C(r') \leq 1 + C(n - \eta_{t-\lambda-k}). \tag{36}$$

which proves Equation (29). □

Theorem 5 is an extremely useful characterization of the function $C(n)$. In the light of this theorem, it is worthwhile to note the important implication of Theorem 4 in the study of the behavior of $C(n)$. Using Theorems 4 and 5, the knowledge of $|\mathbf{F}_{x,m}|$ for a certain set of values of m over a range spanning λ consecutive complexity values, is sufficient to characterize $C(n)$ for problems of smaller complexity. The goal of complete connectivity characterization, thus, simplifies to merely finding $|\mathbf{F}_{x,m}|$ for a small set of values of m and x.

Theorem 4 implies that for any given i there can be no more than λ distinct non-zero values of $|\mathbf{F}_{i,m}(k,\lambda)|$, assuming $C(\eta_m) - C(\zeta_m) \geq i$. However, the quantity $|\mathbf{F}_{i,m}(k,\lambda)|$ is non-zero only when $\mathbf{S}_m \neq \phi$. From (9), we know that $\mathbf{S}_m = \phi$ for all m such that g does not divide $m - T(n_0)$, where $g = \gcd(k,\lambda)$. Thus, $|\mathbf{S}_m|$ may be non-zero only for every g-th value of the complexity m. Thus, in effect, there is at most λ/g distinct non-zero values of $|\mathbf{F}_{i,m}(k,\lambda)|$. Therefore, the notation for the quantities $|\mathbf{F}_{i,m}(k,\lambda)|$ can be simplified by the following definition.

Denote by $A_v(k,\lambda)$, $0 \leq v < \lambda/g$, the infinite series defined by the elements $\{a_{v,0}(k,\lambda), a_{v,1}(k,\lambda), \dots\}$ where,

$$a_{v,i}(k,\lambda) = |\mathbf{F}_{i,m}(k,\lambda)|, \tag{37}$$

for some m such that, $v = [(m - T(n_0)) \bmod \lambda]/g$ and, $C(\eta_m) - C(\zeta_m) > i$.

By the above definition, we have λ/g different series, in which the i-th element is $|\mathbf{F}_{i,m}(k,\lambda)|$, the different series being generated by different sets of values of m. For the sake of brevity, in the following, we shall drop the overheads as qualifiers to the notations of the series and their elements. Thus, A_v denotes the series corresponding to

values of m for which $v = [(m - T(n_0)) \bmod \lambda]/g$, while $a_{v,0}$, $a_{v,1}$, etc., denote its elements.

When the communication overhead is dominant, i.e., $k \gg \lambda$, one may assume with only a minor approximation, that k is an integer multiple of λ. In a subsequent paper, we will prove that, when $k \gg \lambda$, there is only one series $A = \{a_0, a_1, \dots\}$, given by,

$$a_i = |\mathbf{S}_{T(n_0)+(\lambda+k)(i+1)-\lambda}|. \tag{38}$$

The above equation helps us to completely characterize the connectivity requirements of a topology as a function of the problem size, and also leads to some interesting insights into the trade-offs between node-degree requirements and time-optimality.

5 Conclusion

It is well-known that parallel computing overheads severely limit the performance of parallel algorithms on real systems. In this paper, we have used a computational model of a parallel algorithm based on the principle of divide-and-conquer, and which incorporates both algorithmic and communication overheads in the model. The effect of these unavoidable overheads may be alleviated by choosing appropriate partition sizes for optimal processor and link utilization. In this paper, we have derived the basic concepts necessary to compute the partition sizes, not just for time-optimality but also to minimize the node-degree requirements of the processors in the system.

We defined $C(n)$ as the lower bound on the node-degree of the processor assigned the task corresponding to the root of the time-optimal computational tree graph for a problem of size n. Obtaining values of $C(n)$ recursively at each stage of the divide-and-conquer recursion, allows us to derive an optimal topology with minimal node-degree requirements while preserving time-optimality. We study a number of properties of $C(n)$ and describe the basis on which we seek to completely characterize $C(n)$ as a function of n. As will be shown in subsequent papers by the authors, these results help us design fast algorithms for partitioning problems into subproblems in such a way that the execution time is optimal and the connectivity requirements are minimal necessary for time-optimality.

Even though the recursion in (1) has been previously studied by other researchers, we are not aware of any other analytical study of the relationships between node-degree requirements of processors and time-optimality. Speedup of parallel algorithms has traditionally been defined in terms of the number of processors, even though other resource constraints (such as node-degree constraints) are frequently encountered in real system design. In extensions to this work, we expect to analytically obtain further theoretical insights into the relationships and trade-offs that exist between node-degrees of processors and the execution time.

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

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

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

[5] 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.

[6] E. Horowitz and A. Zorat, "Divide-and-conquer for parallel processing," *IEEE Trans. Comput.*, vol. 32, pp. 582–585, June 1983.

[7] 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.

[8] 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.

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

[10] H. Sethu and M. D. Wagh, "Node-degree requirements of parallel divide-and-conquer," in *Proc. 4th ISMM/IASTED Int'l Conf. Par. & Dist. Comput. & Sys.*, (Washington, D. C.), Oct. 1991.