PARALLEL RECURSIVE COMPUTATIONS WHERE BOTH RECOMBINATION AND PARTITION OVERHEADS ARE PROBLEM–DEPENDENT

Dr. Arindam Saha
Dept. of Electrical & Computer Engineering
and NSF Engineering Research Center for CFS
Mississippi State University, P.O. Box 6176
Mississippi State, MS 39762

Dr. Meghanad D. Wagh
Electrical Engineering and Computer Science
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015

Abstract: ***Parallel recursive computations incorporating the unavoidable and significant parallel computing overheads, encompassing a wide variety of applications, can be modeled as***

$$T(n) = \begin{cases} t_0, & \text{for } n < n_0, \\ \min\limits_{0<r<n} \{\max\{T(n-r),\ T(r)+k(r)\} + \lambda(n,r)\} & \text{otherwise.} \end{cases}$$

where k(r) and λ(n,r) represent the partition and recombination overheads respectively. The optimal partition size (solution to r of the above minmax recurrence relation) is nontrivial and is very different from the n/2 value conventionally used. Using the optimal partitions at every stage of the recursion enhances the performance greatly. In this paper we solve a challenging case of our parallel recursive model where the overhead functions are problem–dependent.

I. INTRODUCTION

Many parallel processing algorithms are based on the recursive paradigm [1]. This technique allows a problem to be partitioned into smaller instances of itself, and combining the partial results to obtain the final solution. Since these smaller instances can themselves be recursively partitioned into even smaller sizes, the procedure reduces to solving minimal sized problems and recombining their results. The recursive technique is ideal for parallel processing because the smaller sized problems are mutually independent and hence can be executed concurrently in different processors.

The performance of any parallel recursive algorithm is greatly affected by the architectural and algorithmic overheads inevitable in any realistic parallel computing environment. These overheads, among others, include the cost of interprocessor communication to distribute data and collect results, the cost 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. Conventionally these overheads are neglected in the design of algorithms [2] – [4]. This leads to ad hoc equal partitioning and nonoptimal performance of the recursive technique on real world parallel machines.

It has recently been [5] shown that by proper choice of partitions, one may extract optimal performance from recursive algorithms in the presence of overheads. The complexity of such optimal algorithms shows a considerable improvement over algorithms using equal partitions [5]. But unfortunately, the choice of optimal partition sizes is greatly influenced by the nature of the overheads. It is therefore imperative to correctly model the architectural and algorithmic overheads and then efficiently determine the optimal partitions given any problem size. This paper develops procedures to obtain such optimal partitions for certain special cases.

Section II of this paper describes the model for parallel recursive algorithm incorporating both the symmetric and asymmetric overheads inherent in any parallel computing environment. A specific case of the generalized model is solved in Section III. Section IV concludes the paper with some discussions.

II. PARALLEL RECURSIVE MODEL

The complexity of a parallel recursive algorithm may be described by the following equation.

$$T(n) = \begin{cases} t_0, & \text{for } n < n_0, \\ \min\limits_{0<r<n} \{\max\{T(n-r),\ T(r)+k(r)\} + \lambda(n,r)\} & \text{otherwise.} \end{cases} \quad (1)$$

where, T(n) is the complexity of a size n problem, n_0 is some small problem size below which the recursion is not applied and all problems have the same complexity t_0, k(r) is the partition overhead dependent on the partition size, and λ(n,r) is recombination overhead as a function of the problem size and/or the partition size.

While the symmetric recombination overhead, λ(n,r) characterizes the costs associated with the recombination of the partial results, the asymmetric partition overhead k(r) is related to the costs incurred during interprocessor communication as well as any extra computations one of the partitions may require. Note that in some cases both processors participate in the communication phase, but such a symmetric cost can be easily incorporated in the λ(n,r) term.

The aim of this paper is two–fold. 1) We want to determine optimal r values in the solution of (1) for all problem sizes of interest. Such r values will be referred to as the optimal partitions. It will be shown in this paper that that the optimal partitions are very different from the ad hoc equal partitions n/2. Note that our model *does not* suggest binary partitioning, but with the help of our

partitioning algorithms one can easily generate *many partitions simultaneously.* The complexity of such algorithms is of little consequence because they are going to be used by parallelizing compilers infrequently (analogous to time–consuming place–and–route in VLSI). Similar partitioning algorithms have been previously published by the authors [8]. 2) We want to derive either a closed–form or an asymptotic solution of the complexity function T(n). Again, we will show that the complexity T(n) is far better considering the optimal partitions as opposed to the ad hoc equal partitions.

Table 1. contains a list of all the relevant symbols used in this paper and their corresponding definitions.

Symbol	Definition
g	gcd(k, λ).
n	Problem Size
η_m	Largest n such that complexity T(n)=m.
S_m	Set of problem sizes of same complexity m.
$\mathbf{R}_n$	Set of optimal partition sizes for n.
r_m	Unique optimal partition size for η_m.
T(n,r)	Complexity of problem size n. and partition size r

Table 1. Symbols and their definitions

III. SOLUTION OF THE MODEL

The logarithmic merging or recombination overheads occur in many real world algorithms that include many parallel sorting and searching problems. This scenario is modeled by the following recursion.

$$T(n) = \begin{cases} t_0, & \text{for } n < n_0, \\ \min\limits_{0<r<n} \{\max\{T(n-r),\ T(r)+kr\} + \lambda\lfloor \log n \rfloor\}, & \text{otherwise.} \end{cases} \quad (2)$$

The determination of the optimal partition size in recurrence (2) for any n is of utmost importance for design and implementation of optimal algorithms. In this section, we present relevant results to characterize the optimal partition sizes. Clearly, $T(n)$ is monotonically increasing with n.

Lemma 1: $T(n)$ is a monotonically increasing function of n.

Proof: (By induction over n) Trivial. ❑

Note that although according to (2), any integer between *1* to *n–1 can* be an optimal partition size, the search domain for the optimal partition size, is in actuality, highly restricted as shown by the following result.

Lemma 2: If $p \in R_n$, then $T(n+1) = \min\{T(n+1,p),\ T(n+1,\ p+1)\} + \lambda\lfloor \log_2(n+1) \rfloor$.

Proof: This result is similar to that of the convolution minimization stated by Fredman and Knuth in [7]. Please refer to it for the proof. ❑

Due to the logarithmic factor in the recombination overhead function, one can realize that any problem size of the form of a power of 2 is special in the sense that the contribution of the recombination part necessarily increases there. This intuitive notion is formalized in the following lemma.

Lemma 3: $T(2^i) - T(2^i–1) \geqq \lambda$, for any positive i.

Proof: Let $p \in \mathbf{R}_{2^i-1}$

$\therefore$ $T(n) = T(2^i - 1) = \max\{\ T(2^i - 1 - p) + \lambda\lfloor \log(2^i - 1) \rfloor,\ T(p) + kp + \lambda\lfloor \log(2^i - 1) \rfloor\ \}$. $T(2^i - 1) = \max\{T(2^i - 1 - p) + \lambda(i - 1),\ T(p) + kp + \lambda(i - 1)\ \}$. Since $T(2^i–1) \geqq T(2^i–1,p+1) = \max\{T(2^i–2–p), T(p+1) + kp+k\} + \lambda i–\lambda$, the second term in the max function must get chosen in $T(2^i–1,p+1)$.Thus

$T(p+1) + kp + k \geqq \max\{T((2^i–1–p), T(p)+kp\}$ (3)

Also, $T(2^i–1) \leqq T(2^i–1,p–1) = \max\{T(2^i–2–p), T(p–1)+kp–k\} + \lambda i–\lambda$. Here the first term in the max wins to give

$T(2^i–p) \geqq \max\{T((2^i–1–p), T(p)+kp\}$ (4)

Combining (3) and (4), one has

$\min\{T(2^i–p),\ T(p+1) + kp + k\} \geqq \max\{T(2^i–1–p),\ T(p)+kp\}$ (5)

Considering the problem size 2^i, according to Lemma 2, $T(2^i) = \min\{T(2^i,p),\ T(2^i,p+1)\} + \lambda i$ After expansion, we have $T(2^i) = \min\{\max\{T(2^i–2–p),\ T(p) + kp\}, \max\{T(2^i–1–p), T(p+1)+ kp+k\} + \lambda i$. Using (3) and (4), we can write $T(2^i) = \min\{T(2^i–1–p)\ T(p+1)+kp+k\} + \lambda i$. Then, $T(2^i) + T(2^i–1) = \min\{T(2^i–1–p), T(p+1)+kp+k\} - \max\{T(2^i–1–p), T(p) + kp\} + \lambda$. Using (5) one thus gets, $T(2^i) + T(2^i–1) \geq \lambda$. ❑

There may exist more than one optimal partition size for any n. But as shown in Theorem 1, it is uniquely defined at every η_m, the largest problem size with complexity m.

Theorem 1: The optimum partition set $R\eta_m$ includes the element $r_m = \{\eta_m - \eta_{(m\ -\ \lambda\lfloor \log\eta_m \rfloor)}\}$.

Proof: For any $p \in \mathbf{R}\eta_m$,

$m = T(\eta_m, p) = \max\{\ T(\eta_m - p) + \lambda\lfloor \log\eta_m \rfloor, T(p) + kp + \lambda\lfloor \log\eta_m \rfloor\ \}$ (6)

Thus, considering the first term in the max function in (6), we have $m \geq T(\eta_m–p) + \lambda\lfloor \log_2(\eta_m) \rfloor$., or, $T(\eta_m–p) \leqq m - \lambda\lfloor \log_2(\eta_m) \rfloor$. From the definition of an eta point, we can write $\eta_m–p \leqq \eta_{m-\lambda}\lfloor \log_2(\eta_m) \rfloor$. Thus,

$p \geq \eta_m - \eta_{(m\ -\ \lambda\lfloor \log\eta_m \rfloor)}$. (7)

We prove this theorem by considering two cases.

Case I: Let η_m be of the form $2^i–1$ for some i. In other words, $\lfloor \log_2(\eta_m) \rfloor = \lfloor \log_2(\eta_{m+1}) \rfloor$. We will now show that p cannot be strictly bigger than $\eta_m - \eta_{m-\lambda}\lfloor \log_2(\eta_m) \rfloor$. If it were true, then $p \geq 1+\eta_m–\eta_{m-\lambda}\lfloor \log_2(\eta_m) \rfloor$. In other words, $(1+\eta_m–p) \leqq \eta_{m-\lambda}\lfloor \log_2(\eta_m) \rfloor$. This implies that $T(1+\eta_m–p) \leqq m–\lambda\lfloor \log_2(\eta_m) \rfloor$, or

$T(1+\eta_m–p) + \lambda\lfloor \log_2(\eta_m) \rfloor \leqq m$. (8)

Also, considering the second term in the max function of (6), we get $T(p) + kp + \lambda\lfloor \log\eta_m \rfloor \leqq m$. (9)

We know that $T(\eta_m+1) \leqq T(\eta_m+1,p) = \max\{T(1+\eta_m–p),\ T(p) + kp\} + \lambda\lfloor \log_2(\eta_m+1) \rfloor$. But by

assumption of this case $\lfloor log_2(\eta_m) \rfloor = \lfloor log_2(\eta_m+1) \rfloor$. Thus, $T(\eta_m+1) \leq \max\{T(1+\eta_m-p), T(p)+kp\} + \lambda\lfloor log_2(\eta_m) \rfloor$. (10)

Comparing (8), (9) and (10), we find that $T(\eta_m+1) \leq m$. But this is impossible. Therefore,

$$p \leq \eta_m - \eta_{(m - \lambda\lfloor \log\eta_m \rfloor)} \quad . \tag{11}$$

From (7) and (11), we conclude that $p = \eta_m - \eta_{m-\lambda\lfloor log_2(\eta_m) \rfloor}$

Case II: Let η_m be of the form 2^i-1 for some i, i.e., $\lfloor log_2(\eta_m+1) \rfloor = 1 + \lfloor log_2(\eta_m) \rfloor$. Similar to the reasoning for Case I, we can show that for this case $T(\eta_m+1) \leq m + \lambda$. But according to Lemma 3, $T(\eta_m+1) \geq m + \lambda$. Thus, $T(\eta_m+1) = m + \lambda$ (12)

The equation corresponding (18) in this case is

$T(\eta_m+1) \leq \max\{T(1+\eta_m-p), T(p) + kp\} + \lambda + \lambda\lfloor log_2(\eta_m) \rfloor$ (13)

To satisfy (12) and (13), we can easily see that $p=\eta_m-\eta_{m-\lambda}\lfloor \log_2(\eta_m) \rfloor$. ❑

It can be shown that for every η_m which is not of the form 2^i-1, r_m is the only partition size. For an η_m of the form 2^i-1, r_m is the smallest possible optimal partition size. When the problem size is not an r_m, there may exist numerous partition sizes. But, as the following result shows, r_m can still be used as an optimal partition size.

Theorem 2: For any $n \in S_m$, R_n includes r_m.

Proof: As a consequence of Lemma 4 we know that for any $n \in S_m$, one has $\lfloor \log n \rfloor = \lfloor \log\eta_m \rfloor$ (14)

With the help of Theorem 1, from the expression of $T(\eta_m) = m = T(\eta_m,r_m) + \lambda\lfloor \log_2(\eta_m) \rfloor$, we can write

$$T(r_m) + kr_m \leq m - \lfloor \log_2(\eta_m) \rfloor. \tag{15}$$

Since $n \in S_m$ and using (14), we have $T(n) = m \leq T(n, r_m) + \lambda\lfloor log_2(\eta_m) \rfloor$. Upon expansion, it gives

$$m \leq \max\{T(n-r_m), T(r_m)+kr_m\} + \lambda\lfloor \log_2(\eta_m) \rfloor. \tag{16}$$

From (16), one gets $T(n-r_m) \leq m-\lambda\lfloor \log_2(\eta_m) \rfloor$ (17)

Combining (15) and (17), we have

$$m \geq \max\{T(n-r_m), T(r_m)+kr_m\} + \lambda\lfloor \log_2(\eta_m) \rfloor. \tag{18}$$

Combining (16) and (18), we have $m = \max\{T(n-r_m), T(r_m) + kr_m\} + \lambda\lfloor log_2(\eta_m) \rfloor = T(n,r_m) = T(n)$. ❑

We now investigate the behavior of the complexity $T(n)$. First, we look at the allowable complexity values.

Lemma 4: For any complexity m for which $|S_m| \neq 0$, $g \mid (m-t_0)$, where $g = gcd(k,\lambda)$.

Proof: This result is obvious from the fact that for $n<n_0$, complexity of size n problem is t_0, and at every stage of computation, it differs from an earlier complexity by $(kr+\lambda\lfloor log_2(\eta_m) \rfloor$ or $\lambda\lfloor log_2(\eta_m) \rfloor$, both of which are multiples of g. ❑

Lemma 4 shows that the complexity $T(n)$ is not a continuous function of the problem size n and the gaps in the complexity are at least equal to g. In general, the converse of Lemma 4 is not true. The necessary and sufficient condition for the a complexity class to be nonempty is much more complex and is specified by the following theorem.

Theorem 3: $|S_m| \neq 0$ *iff* $|S_{m-\lambda\lfloor log_2(\eta_m) \rfloor}| \neq 0$ or for some r, $m=T(r)+kr+\lambda\lfloor log_2(\eta_m) \rfloor$.

Proof: Let $|S_m| \neq 0$. This means that there is a problem of size η_m whose complexity is m. Let $r \in R\eta_m$. Clearly then, $m = T(\eta_m, r) = \max\{ T(\eta_m - r) + \lambda\lfloor \log\eta_m \rfloor, T(r) + kr + \lambda\lfloor \log\eta_m \rfloor \}$.

If m equals the first term of the max function, then the size $(\eta_m - r)$ problem has a complexity $(m - \lambda\lfloor \log\eta_m \rfloor$ implying $|S_{(m - \lambda\lfloor \log\eta_m \rfloor)}| \neq 0$. On the other hand if the second term was chosen, then $T(r) + kr + \lambda\lfloor \log\eta_m \rfloor = m$, as specified in the statement of the theorem. This proves necessity of given conditions for $|S_m|$ to be $\neq 0$.

We prove sufficiency by showing that either of the two conditions implies $|S_m| \neq 0$. Let's consider first $|S_{(m - \lambda\lfloor \log\eta_m \rfloor)}| \neq 0$. We will prove that $|S_m|=0$ is a contradiction to this as follows. If $|S_m|=0$, then there exists a problem size n such that $T(n)<m$ and $T(n+1) > m$. Obviously n is the maximum of its complexity class and from Theorem 1, if r is its optimum partition size, then, $T(n-r) = T(n) - \lambda\lfloor \log(n) \rfloor < m - \lambda\lfloor \log(n) \rfloor$. (19)

Also, $m < T(n+1) \leq T(n+1, r) = \max\{T(n-r+1), T(r) + kr \} + \lambda\lfloor \log\eta_m \rfloor$. (20)

But $m > (T(r) + kr + \lambda\lfloor \log\eta_m \rfloor)$ because $m > T(n) = \max\{ T(n-r) + \lambda\lfloor \log\eta_m \rfloor, T(r) + kr + \lambda\lfloor \log\eta_m \rfloor \}$. Therefore (20) implies $m < T(n-r+1) + \lambda\lfloor \log\eta_m \rfloor$, or

$$T(n-r+1) > m - \lambda\lfloor \log\eta_m \rfloor. \tag{21}$$

Equations (19) and (21) together imply that $|S_{(m - \lambda\lfloor \log\eta_m \rfloor)}| = 0$ which is a contradiction. Therefore we have in this case, $|S_m| \neq 0$.

To prove the sufficiency of the second condition, consider now that $|S_{(m - \lambda\lfloor \log\eta_m \rfloor)}| = 0$ but for some r, $m = T(r) + kr + \lambda\lfloor \log\eta_m \rfloor$. Relation $|S_{(m - \lambda\lfloor \log\eta_m \rfloor)}| = 0$ implies that there exists a problem size n such that $T(n) < m - \lambda\lfloor \log\eta_m \rfloor$ and $T(n+1) > m - \lambda\lfloor \log\eta_m \rfloor$. We will now show that problem size $n_1 = n + r$ has complexity m, thus proving that $|S_m| \neq 0$. We can write

$$T(n_1) = \min_{0<p<n1} T(n_1, p) = \min_{0<p<n1} \{\max\{T(n_1 - p) + \lambda\lfloor \log\eta_m \rfloor, T(p) + kr + \lambda\lfloor \log\eta_m \rfloor\}\}.$$

For $p > r$, $T(p) + kp + \lambda\lfloor \log\eta_m \rfloor > m$ implying that $T(n_1, p) > m$. Similarly, for $p < r$, $T(n_1 - p) \geq T(n_1 - r + 1) > (m - \lambda\lfloor \log\eta_m \rfloor)$ and $T(n_1, p) > m$ once again. Finally, for $p = r$, $T(n_1 - p) < (m - \lambda\lfloor \log\eta_m \rfloor)$ and $T(p) + kp + \lambda\lfloor \log\eta_m \rfloor = m$. Thus the minimum value of $T(n_1, p)$ is obtained at $p = r$ and this value is $T(n_1) = m$. ❑

We now present an analysis to determine the order of complexity. Interestingly enough, this scenario is similar to the model solved by us in [8]. By using the results derived so far in this section and the results of complexity in [8] we now derive the order of T(n).

Theorem 4: T(n) is of order $O([n\log(n)]^{0.5})$.

Proof: Rewriting recurrence relation in (2) we have,

$$T(n)-\lambda\lfloor \log(n) \rfloor = \min\{\max\{T(n-r), T(r)+kr\}. \tag{22}$$

Now add a sufficiently large constant $P = \lambda\log(N)$ to the right hand side of equation (22), where N is the largest problem size of interest, and subtract appropriate terms from each element of the *max* function to get $T(n) - \lambda\lfloor \log(n) \rfloor \leq \min\{ \max\{T(n-r) - \lambda\lfloor \log(n-r) \rfloor, T(r) - \lambda\lfloor \log(r) \rfloor + kr \} + P \}$. (23)

B : making a simple substitution, $f(n) = T(n) - \lambda\lfloor\log(n)\rfloor$. we can transform (23) into

$f(n) \leq \min\{ \max\{ f(n-r), f(r) + kr \} + P \}$. (24)

Please refer to [10] to find that (24) is similar to the recursion solved in that paper for the special case of a constant s=1. A similar result has also been derived in [6]. So from [6] and [8] we get the order of N, a complexity m problem, to be $O((m^2)\log(N))$. (25)

Note here that N denotes the problem size n and m represents the value of the function f(n). Thus (25) is interpreted to imply that the order of f(n) is $O([n\log(n)]^{0.5})$. Thus T(n) is of the order $O(([n\log(n)]^{0.5}) + \log(n)) = O([n\log(n)]^{0.5})$. ❑

If n/2 is chosen as the partition size at every stage of the recursion (2), then one gets $T(n) = T(n/2) + kn/2 + \lambda\lfloor\log(n)\rfloor$. Clearly, this forces the complexity T(n) to be $O(n + \text{loglog}(n))$. Thus one of the most significant impacts of optimal partitioning as opposed to the commonly used equal partitioning is the drastic reduction of the overall computational complexity. We have succeeded in reducing the complexity from O(n) to $O([n\log(n)]^{0.5})$.

We now present a generalized algorithm to compute the optimal partition size r for any problem size n as a solution to (2). The algorithm is self-explanatory and is based on the results derived in this section.

Algorithm to compute optimal partition size

Step 1. *(Initialization)*

```
m[0] = 0; eta[0] = 1; rm[0] = 0;
m[1] = t0 + k + λ; eta[1] = 2; rm[1] = 1;
r = 1; log = 1; loglimit = 3;
i = 2;          /* row about to be created */
j = 1;          /* row that creates a new row */
newt = μ(2);    /* μ(r) = T(r) + kr + λlog */
```

Step 2. *(Computation)*

```
while r < N do {
  eta[i] = eta[j] + r;
  if (eta[i] > loglimit) and
      (eta[i-1] < loglimit) then {
      eta[i] = loglimit; m[i] = m[j] + λlog;
      if (m[i] ≥ newt) then {
        m[i] = newt; r++; newt = μ(r+1); }
      rm[i] = r; j—; goto final; }
  if (eta[i] > loglimit) then {
      log++; loglimit = 2*loglimit + 1;
      newt = μ(r+1); } m[i] = m[j] + λlog;
      rm[i] = r;
  if (m[i] > newt) then {
      r++; j—; eta[i] = eta[j] + r; m[i] = newt;
      rm[i] = r; newt = μ(r+1); }
  elseif (m[i] = newt) then {
      r++; newt = μ(r+1);
      if ( eta[i] ≠ loglimit) then eta[i] = eta[j] + r;
      else j—; rm[i] = r; }
final: i++; j++ }
```

IV. CONCLUSIONS

This paper presents the design and analysis of parallel algorithms based on the principle of recursive partitioning. Crucial to the analysis of optimality is the development of a well defined analytical model in the form of a minmax recurrence relation that incorporates the algorithmic and architectural overheads encountered in real world parallel computing environments. Conventionally researchers have ignored these unavoidable overheads in their design and analysis of parallel recursive computations and most have chosen the equal partition size on an ad hoc basis. We have demonstrated a novel technique of including these overheads at the outset such that optimal partitioning decisions can be made to mitigate the adverse effect of these overheads. Our model *does not* suggest binary partitioning, but with the help of our partitioning algorithms one can easily generate *many partitions simultaneously.*

The case solved in this paper, deals with the recombination overhead as logarithmic function of the problem size. We have shown that the order of complexity can be significantly reduced from O(n) to $O([n\log(n)]^{0.5})$ by using optimal partitions instead of the ad hoc equal partitions at every stage of the recursion. We have also designed a generalized algorithm to compute these optimal partition sizes.

REFERENCES

[1] E. Horowitz and S. Sahni, *Fundamentals of computer Algorithms*, Computer Science Press Potomac, Maryland, 1978.

[2] M. J. Atallah, R. Cole and M. T. Goodrich, "Cascading Divide–and–Conquer: A technique for designing parallel algorithms," *SIAM J. Comput.*, 18 (1989), pp. 499–532.

[3] Q. F. Stout, "Supporting Divide–and–conquer algorithms for image processing," *J. Parallel and Distributed Comput.*, 4 (1987), pp. 95–115.

[4] B. Abramson, "Divide and Conquer under global constraints: A solution to the N–Queens problem," *J. Parallel and Distributed Comput.*, 6 (1989), pp. 649–662.

[5] A. Saha, *Design and Analysis of Parallel Recursive Computations in the Presence of Overheads*, Ph.D Dissertation, Lehigh Univ., 1991.

[6] S. Kapoor and E. M. Reingold, "Optimum lopsided binary trees," *J. ACM*, 36 (1989), pp. 573–590.

[7] M. L. Fredman and D. Knuth, "Recurrence relations based on minimization," *J. Math. Analysis and Appl.*, 48 (1974), pp. 534–559.

[8] A. Saha and M. D. Wagh, "Algorithms for determining optimal partitions in parallel divide–and–conquer computations," *Proceedings of the ICPP*, St. Charles, Vol. III, (1991), pp. 75–82.