TOWARDS AN ARCHITECTURE-INDEPENDENT ANALYSIS OF PARALLEL ALGORITHMS

(Extended Abstract)

Christos H. Papadimitriou[1] and Mihalis Yannakakis[2]

1. INTRODUCTION

Harnessing the massively parallel architectures soon to become available into efficient algorithmic cooperation is one of the most important intellectual challenges facing Computer Science today. To the theoretician, the task seems similar to that of understanding the issues involved in the performance of sequential algorithms (which motivated Knuth's books, among other important works), only infinitely more complex. In sequential computation, the design process involves (a) choosing an algorithm and (b) analyzing it (mostly, counting its steps). In the parallel context, however, we have at least four stages: (1) Choose the algorithm (say, a directed acyclic graph (dag) indicating the elementary computations and their interdependence, a model in which evaluation of sequential performance is trivial). (2) Choose a particular multiprocessor architecture. (3) Find a *schedule* whereby the algorithm is executed on the processors (so that all necessary data are available at the appropriate processor at the time of each computation). (4) Only now can we talk about the performance of the algorithm, measured as the makespan of the schedule (elapsed time for computing the last result). In our opinion, it is this multi-layered nature of the problem that lies at the heart of the difficulties encountered in the development of the necessary ideas, principles, and tools for the design of parallel algorithms.

Is there a way to shortcut the process, thus improving our chances of finally gaining some insight into parallel algorithms? It would be very interesting if we could combine stages (3) and (4) into a single step whereby the performance of the algorithm chosen in (1) can be evaluated in a simple, direct manner (at least in principle), pretty much like it is done in sequential computation (stage (b)). At first, the task seems impossible, since it is well-known that the performance of a parallel algorithm depends critically on the architecture adopted in stage (2), and the space of architectures is too rich to use as a parameter. One first attempt at this problem was made in [PU]. In that paper three parameters of an algorithm were isolated, that are relevant in any architecture, and thus can be used as first measures of the performance of the algorithm (these parameters were: elapsed time ignoring communication, total communication traffic, and total (back-to-back) communication delay). In [PU] nontrivial trade-offs between these parameters were shown for the pyramid (called "diamond dag" in [PU]). Those results were not completely satisfactory for at least two reasons: First, [PU] bypassed the problem of parametrizing architectures by considering many performance measures. Second, no general principle or technique, applicable to all dags, was in sight (this was mentioned as a challenging open problem in [PU]; the results presented here apply to all dags, and, when restricted to the pyramid, imply Theorem 2 in [PU], thus resolving that question).

In this paper we present a technique which, we feel, may lead to an important new understanding of parallel computation. We are concerned with stages (2), (3), and (4) alone. In other words, we start with the algorithm-dag given and fixed; usually, the dag will in fact be a family thereof, indexed by its number of inputs, such as the pyramid, the full binary tree, the FFT, etc. We assume that we are interested in the case in which the available number of processors is adequate for dealing with the whole width of the dag (thus the number of processors involved is no longer a parameter). It would seem then that the

[1] Department of Computer Science and Engineering, University of California at San Diego.
[2] AT&T Bell Laboratories.

crucial measure of the complexity of the dag would be its *depth*, since it is the depth that dominates its parallel complexity in idealized models of computation, such as a PRAM. The point is that in real parallel architecures there is a significant *communication delay* between the time some information is produced at a processor and the time it can be used by another, and the depth fails to capture this. Such communication delay, denoted τ, is measured in elementary steps of the processors (equivalently, nodes of the dag).

The status of τ as the single most important parameter of parallel computers was pointed out by C. Gordon Bell [Be]. As a function of n, the number of processors, τ can range, depending on the architecture, from n (on a ring) to $\sqrt{n}$ (a grid) to $\log n$ (a hypercube). For actual computers, one hears of values of τ ranging from small constants (smaller than one in the case of GF11 [Be]) to several millions for slow clusters. We propose that τ is a parameter of the architecture which is important enough to be the basis of our scheme.

Once we have parametrized away the choice of the architecture, we must concentrate on stage (3), scheduling the dag on the processors. But given τ, and under our assumption of enough processors, this is a concrete scheduling problem: Schedule a dag of unit time tasks on an unbounded number of processors so that, if task i is executed at time t on processor p and j is a parent of i, then either (a) j is executed on p by time $t-1$, or (b) j is executed on some other processor p' *by time* $t-1-\tau$. The optimum makespan of this scheduling problem, always a function of τ, is therefore a fair measure of the parallel complexity of the dag. Unfortunately, this is a nontrivial function: The scheduling problem is NP-complete...

Fortunately, we propose in this paper a simple way to calculate the optimum makespan *within a factor of two*. We compute a simple function e on the nodes of the dag, visiting them in topological order, and show that this quantity is *between half the optimum makespan and the optimum makespan*, and thus can serve as an adequate estimate.

There may be a healthy suspicion that perhaps our method, by disregarding the number of processors, will yield unrealistic, processor-wasteful algorithms. We present strong evidence that this is not so: Using our method, we derive asymptotic upper and lower bounds for the parallel complexity of common algorithms (the FFT, the pyramid, the full binary tree), as a function of n (the number of inputs) and τ. Interestingly, these bounds are achieved by time-optimal algorithms that are also *processor optimal* (for fixed time), and, furthermore, they are *the most obvious and intuitive parallel algorithms* for the corresponding problem on a parallel machine with ratio τ. They are all simple variants of the algorithm directly suggested by our method. Furthermore, starting from our lower bound for the pyramid, we can derive the execution time vs. communication delay tradeoff in [PU].

2. THE APPROXIMATION ALGORITHM

We are given a directed acyclic graph $D=(V,A)$, whose nodes are computational tasks of equal (unit) execution time requirements, and whose arcs, as is customary in parallel algorithms, represent both time precedence and functional dependence. We are also given a positive integer τ. A *schedule* S of D is a finite set of triples $S \subset V \times \omega \times \omega$ (the second component is the process on which the node is scheduled, and the third is the time), such that: (1) For each $v \in V$ there is a triple $(v,p,t) \in S$. (2) There are no two triples $(v,p,t),(v',p,t) \in S, v \neq v'$. (3) If $(u,v) \in A$ and $(v,p,t) \in S$ then either there is another triple $(u,p,t') \in S$ with $t' \leq t-1$, or there is another triple $(u,p',t') \in S$ with $t' \leq t-1-\tau$.

In other words, we schedule the dag with possible repetitions of the nodes on an unlimited number of processors, so that there is a delay τ for communication between the processors. Our goal is to minimize $T_{\max}$, the largest time appearing in S.

Theorem 1. It is an NP-complete problem to decide, given a directed acyclic graph (V,A), integer τ, and deadline $T_{\max}$, whether there exists a schedule S such that no time greater than $T_{\max}$ is used. □

Given a dag (V,A) and an integer τ, we can compute the following function $e: V \to \omega$, inductively on the depth of the nodes of the dag:

(1) If v is a source, then $e(v)=0$.

(2) Otherwise, consider the set of all predecessors of v, ordered in decreasing $e(u)$ (recall that e has already been evaluated on the predecessors) $e(u_1) \geq e(u_2) \geq \ldots \geq e(u_p)$. Let k be the smallest among $\tau+1$ and p. Then we define $e(v) = \max\{e(u_1)+1, \ldots, e(u_k)+k\}$.

Theorem 2. There is no schedule in which node v is scheduled before time $e(v)$. Furthermore, there is

a schedule in which each node v node v is scheduled at or before time $2e(v)$. □

A Generalization: By a rather complicated generalization of the algorithm, we can obtain schedules with the same worst-case approximation ratio even when each node $v \in V$ has an execution time $x(v)$ and a delay $\tau(v)$; that is, if $(v,u) \in A$, v cannot start until $x(v) + \tau(v)$ time units have passed after v started at another processor. This models a situation in which tasks have unequal processing times, and their results differ in size, and therefore in transmission delay. In our original problem, all $x(v)$'s are one, and all $\tau(v)$ are equal (and equal to τ).

The following generalization of our algorithm computes the estimate $e(v)$ for this case:

For any source node v, $e(v)$ is zero, as before.

For any node v other than a source consider the set of its predecessors, and for each such predecessor u compute the function $f(u) = e(u) + x(u) + \tau(u)$. Now sort the predecessors in decreasing f : $f(u_1) \geq f(u_2) \geq \ldots$ (notice that f may not be monotonic).

Consider an integer j (intuitively, this is a candidate value for $e(v)$), and suppose that $f(u_k) > j \geq f(u_{k+1})$. Let $N_v(j)$ be the subdag of D consisting of all nodes $u_i, i \leq k$ for which there is a path to v using only nodes $u_i, i \leq k$. Consider then the following scheduling problem S_j for the nodes of $N_v(j)$ (excluding v): The deadline is j, the release time for job v_i is $e(v_i)$, where we have assumed that $e(v_1) \geq e(v_2) \geq \ldots \geq e(v_\ell)$, and $N_v(j)$ contains ℓ nodes (plus v). Obviously, the optimum schedule has length $L_j = \max_{i=1}^{k}[e_i + \sum_{q=1}^{i} x(q)]$. *Now choose the least j such that $j \geq L_j$. This is the value of $e(v)$.*

Theorem 3. In the generalized problem and algorithm, there is no schedule in which node v is scheduled before time $e(v)$. Also, for each node v there is a schedule in which node v is scheduled at time $2e(v)$. Furthermore, the algorithm can be implemented in $O(|V|^2 \log |V| + |V||A|)$ time. □

If there are communication delays on the arcs (not nodes) of the dag, the same technique yields a ratio of two. The only difference is that, instead of the criterion based on the function f, $N_j(v)$ is now constructed arc-by-arc, where an arc (u,w) is included if $e(u)+x(u)+\tau(u,w) > j$. The rest remains the same[1].

[1] Many thanks to Gene Lawler for pointing out to us another approximation algorithm for this problem (with a larger worst-case ratio) thus motivating this subsection.

3. APPLICATIONS

In this Section we apply our technique to three well-known families of dags: The full binary tree, the fast Fourier transform, and the pyramid.

Full Binary Tree

Suppose that the dag is a full binary tree with n nodes. Recall that our algorithm entails scheduling on the same processor with v its τ highest (in e value) predecessors. In the binary tree, these predecessors are at the $\log \tau$ next leveles of the tree. Therefore, in this case our algorithm degenerates to the following: Divide the $\log n$ leveles of the tree into $\frac{\log n}{\log \tau}$ layers of height $\log \tau$. Compute each of the $\frac{n}{\tau}$ resulting subtrees, with all subtrees at the same level computed in parallel. The time is $O(\frac{\tau \log n}{\log \tau})$; since it is the resultof our method, it is asymptotically optimal. $O(n/\tau)$ processors are used. The number of processors can be made optimal (number of nodes divided by time) by simply letting the last layer of the tree have height $\log(\frac{\tau \log n}{\log \tau})$, instead of $\log \tau$. The time at most doubles.

The Fast Fourier Transform

From the point of view of any output, the n-input FFT is a full binary tree with n leaves, and thus the optimal time is, by the results of the previous subsection, $O(\frac{n \log \tau}{\tau})$. The proof of Theorem 2 suggests an algorithm which computes each output separately.

However, the following algorithm achieves the same bound: For $k \approx \frac{\tau}{\log \tau}$, partition the FFT dag into stripes of width $\log k$. Each stripe contains $\frac{n}{k}$ FFT's on k points. Each of these FFT's is executed in sequential time $O(k \log k) \approx \tau$ by a processor, with all FFT's in a stripe executed in parallel. Once a stripe is completed, the results are exchanged and the next stripe starts. The total time is $O(\frac{\tau \log n}{\log \tau})$ (optimal), and there are $O(\frac{n \log \tau}{\tau})$ processors (also optimal!).

The Pyramid

Arguing in the pyramid with n nodes along the same lines as with the full binary tree, the optimal time bound is obtained by computing, in the same processor as node v, the subpyramid with τ nodes with v as apex. The time required is $2\sqrt{n\tau}$. This method uses $\sqrt{n} - \sqrt{\tau}$ processors. However, the *stripes* method [PU], in which the each processor computes a diagonal stripe of width $\sqrt{\tau}$, and processors are synchro-

nized in a pipelined fashion, gives the same asymptotic time, and optimum number of processors: $\sqrt{\frac{n}{\tau}}$.

We can now reconstruct the idealized time-communication delay tradeoff, shown in [PU], as follows: Recall that the idealized time T of a schedule is the time required if all communication delays are zero, and the communication delay d is the time required if all execution times are zero and $\tau = 1$. It was shown in [PU] that (in our notation), for the n-node pyramid, $Td = \Omega(n)$. We need the following Lemma, relating d and T with our $T_{\max}$.

Lemma 3. If there is a schedule S with idealized time T and communication delay d, then there is a schedule S' with $T_{\max} \leq T + d\tau$.

Sketch: We can transform any legal schedule S under [PU] to a legal schedule S' in our framework by adding to the execution time of any node as many multiples of τ as there are arcs with endpoints executed by different processors in any path into the node. □

Thus, $T_{\max} \leq T + d\tau$. Continuing our argument for the pyramid with n nodes, we know that $T_{\max} \geq \sqrt{n\tau}$. Combining the two inequalities, $T + d\tau \geq \sqrt{n\tau}$. Recall now that this inequality is valid for all positive τ. Picking a τ such that $T = \frac{1}{2}\sqrt{n\tau}$, we have that $d\tau \geq \frac{1}{2}\sqrt{n\tau}$, from which $Td \geq \frac{1}{4}n$ follows (note that n here is the number of nodes in the pyramid, and not its height, as in [PU]).

4. OPEN PROBLEMS

Although our approximation algorithm may produce schedules with many processors and a lot of recomputation, we observed that, in all examples considered, the same bounds are achieved without recomputation. Is this a general pattern, namely that any parallel algorithm using recomputation can be simulated by an algorithm that does not use recomputation, and has the same asymptotic time requirements? This is an interesting open question, akin to open problems proposed in [PU]. It is easy to see that it fails is we allow dags with arbitrarily high out-degrees, but there seems to be no easy counterexample with bounded degrees (all three of our examples have in-degree and out-degree two) [1]. A stronger, but less likely, conjecture would be that there is a time-optimal and at the same time processor-optimal (asymptotically) algorithm for any dag with bounded degrees.

Finally, since e, the central part of our method, is a rather simple parameter of the dag, there is hope that we can develop the necessary algebraic tools to show tight bounds for the parallel complexity of *problems* (*not* algorithms) —e.g., for the discrete Fourier transform, as opposed the FFT, matrix multiplication, as opposed to the full binary tree, etc.

[1] It was recently shown by Lefteris Kirousis and Paul Spirakis that for the binary *out-tree* significant recomputation is required in order to achieve optimal time.

References

[Be] C. Gordon Bell "Gordon Bell on the Future of Computers", interview in *SIAM NEWS*, February 1987.

[PU] C. H. Papadimitriou and J. D. Ullman "A Communication-Time Tradeoff", *Proc. 1984 STOC*. Also, to appear in *SIAM J. on Computing*.