1996

Toward efficient scheduling of evolving computations on rings of processors

LX Gao
AL Rosenberg

An Empirical Study of Dynamic Scheduling on Rings of Processors

Extended Abstract

Dawn E. Gregory Lixin Gao Arnold L. Rosenberg Paul R. Cohen

Department of Computer Science, University of Massachusetts
Amherst, MA 01003, USA

Abstract

We empirically analyze and compare two distributed, low-overhead policies for scheduling dynamic tree-structured computations on rings of identical PEs. Our experiments show that both policies give significant parallel speedup on large classes of computations, and that one yields almost optimal speedup on moderate size rings. We believe that our methodology of experiment design and analysis will prove useful in other such studies.

1. Introduction

The promise of parallel computers to accelerate computation relies on the algorithm designer's ability to keep all (or most) of a computer's processors fruitfully occupied all (or most) of the time. In this paper, we study the problem of efficiently scheduling evolving binary-tree-structured computations on a ring-structured parallel computer. We thus face the challenge of efficiently scheduling a computation whose ultimate "shape" is unknown (precluding "offline" planning as in [3, 4, 7]), on an architecture that has a large diameter (precluding efficient random placements as in [6]) and small bisection bandwidth (precluding efficient massive data transmission as in [5]).

We empirically analyze and compare two distributed, low-overhead dynamic-scheduling policies. Our simpler policy—called KOSO, for "keep-one-send-one"—has each PE keep one child of a spawning task and pass the other to its clockwise neighbor in the ring; our more sophisticated policy—called KOSO*—operates similarly, but allows child-passing only from a more heavily loaded PE to a more lightly loaded one. Based on partial (mathematical) analyses of the policies, we formulated two conjectures about the efficiency of the schedules they produce. (a) Both policies yield close to (optimal) p-fold speedup on p-PE rings, for large, significant classes of evolving tree-structured computations; (b) policy KOSO* generally outperforms policy KOSO on these computations. Our experiments largely substantiate both conjectures, at least on moderate-size rings.

Our concern here is with the design and analysis of simulation experiments as much as with comparing scheduling policies.

2. The Formal Setting

The architecture. A p-PE ring of processing elements (PEs) has p identical PEs, denoted $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{p-1}$, with each PE $\mathcal{P}_i$ connected to its *clockwise neighbor* $\mathcal{P}_{i+1 \bmod p}$ and its *counterclockwise neighbor* $\mathcal{P}_{i-1 \bmod p}$.

The computational load. In order to simplify the design of our experiments, we formulate a purely combinatorial environment which abstracts the behavior of multigrid algorithms. Our abstract computational load comprises dynamically growing *binary tree-dags* (*BT*s, for short).

The (dynamic) computation that generates a BT $\mathcal{T}$ proceeds as follows, until no active leaves remain. Initially, $\mathcal{T}$ has a single node, named 1, which is simultaneously its *root* and its (current) *active leaf*. At each step, some set (depending on the schedule) of then-current active leaf-tasks get *executed*. An executed task x may:

- *halt*, so x becomes a *permanent* leaf;
- *spawn* two *children*, the new active leaves $2x$ and $2x+1$, so x becomes a nonleaf.

Policies KOSO **and** KOSO*. Our two scheduling policies differ in their load-balancing regimens but share the same scheduling policy. Both have PEs retain tasks awaiting execution in a local priority queue, ordered by the tasks' heights in the BT $\mathcal{T}$ being scheduled. Each computation begins with the root of $\mathcal{T}$ as the sole occupant of PE $\mathcal{P}_0$'s task-queue and all other PEs' task-queues empty. Subsequently, the task-queue of each PE contains some subset of the then-active leaves of $\mathcal{T}$. At each step, each $\mathcal{P}_i$ having a nonempty task-queue:

1. executes the active leaf x in its task-queue which is first in the mandated order;

2. if task x spawns two children, then $\mathcal{P}_i$ adds the new leaf $2x$ to its task-queue. Under KOSO, $\mathcal{P}_i$ simultaneously sends the new leaf $2x+1$ to the task-queue of PE $\mathcal{P}_{i+1 \bmod p}$. Under KOSO*, $\mathcal{P}_i$ sends task $2x+1$ to $\mathcal{P}_{i+1 \bmod p}$ only if the latter PE has a lighter load than $\mathcal{P}_i$; otherwise, $\mathcal{P}_i$ adds this task also to its own task-queue.

We assess one time unit for the entire process of executing a task and performing the balancing actions just described. Thus, we ignore the fact that a step of KOSO* consumes a bit more real time than a step of KOSO, because of the required comparisons of loads at each step. Our timing assessment is congruous with the fine-grain nature of the motivating multigrid algorithms.

Table 1 illustrates how KOSO and KOSO* distribute the nodes of the 6-level complete binary tree $\mathcal{T}_6$ in the ring $\mathcal{R}_4$.

PE	level	KOSO-resident nodes	KOSO*-resident nodes
$\mathcal{P}_0$	0	1	1
	1	2	2
	2	4	4, 5
	3	8	8, 10, 11
	4	16, 31	16, 17, 20, 21, 22
	5	32, 47, 55, 59, 61, 62	32, 33, 34, 35, 40, 41, 42, 44, 45
$\mathcal{P}_1$	1	3	3
	2	5, 6	6
	3	9, 10, 12	9, 12, 13
	4	17, 18, 20, 24	18, 23, 24, 25, 26
	5	33, 34, 36, 40, 48, 63	36, 37, 43, 46, 47, 48, 50, 52
$\mathcal{P}_2$	2	7	7
	3	11, 13, 14	14
	4	19, 21, 22, 25, 26, 28	19, 27, 28, 29
	5	35, 37, 38, 41, 42, 44, 49, 50, 52, 56	38, 49, 51, 53, 54, 55, 56, 57, 58
$\mathcal{P}_3$	3	15	15
	4	23, 27, 29, 30	30, 31
	5	39, 43, 45, 46, 51, 53, 54, 57, 58, 60	39, 59, 60, 61, 62, 63

Table 1. The node-assignments when $\mathcal{R}_4$ executes $\mathcal{T}_6$ under policies KOSO and KOSO*.

3. Motivating our conjectures

Our conjectures about the individual and comparative behaviors of policies KOSO and KOSO* are founded on two analytical results which may lend the reader some intuition about how the policies work. The first motivating result asserts that policy KOSO schedules BTs that ultimately grow into *complete* binary trees asymptotically optimally.

Theorem 1 *Under policy* KOSO, $\mathcal{R}_p$ *executes any BT that grows into the height-n complete binary tree $\mathcal{T}_n$ within time* $(2^n - 1)/p + (\textit{low-order terms})$. *[2]*

The second motivating result focuses on how KOSO and KOSO* distribute work to the PEs of $\mathcal{R}_p$ while the evolving BT keeps growing. In both parts of the following theorem, we start growing a BT with a single task in PE $\mathcal{P}_0$ and all other PEs idle, and we observe the pattern of work distribution throughout $\mathcal{R}_p$.

Theorem 2 *Focus on an evolving BT in which every executed task spawns two new tasks.*
(a) *After $N \geq p-1$ steps of policy* KOSO, *the numbers of unexecuted tasks residing in the heaviest and lightest loaded PEs of $\mathcal{R}_p$ differ by $p-2$. [2]*
(b) *After $N \geq (p-1)^2$ steps of policy* KOSO*, *the numbers of unexecuted tasks residing in the heaviest and lightest loaded PEs of $\mathcal{R}_p$ differ by* 1.

4. The Experimental Setup

Because multigrid computations rarely generate complete binary trees in practice, we wish to explore the behaviors of KOSO and KOSO* under more realistic conditions. To this end, we designed a suite of experiments based on simulation of $\mathcal{R}_p$ under each policy.

The inputs to the experiments. A key issue in simulation experiments is the generation of random test instances that model realistic computation. We created an abstract analogue of a real, important class of tree-structured computations, namely, the computations that arise from numerically integrating real-valued functions on real intervals using refinement techniques such as the Trapezoid Rule or Simpson's Rule. (Our abstraction extends easily to higher-dimensional multigrid-type algorithms.) BTs arise naturally from such algorithms: tasks correspond to real intervals; spawning corresponds to bisecting the current interval to get more accuracy; halting corresponds to achieving adequate accuracy or encountering inadequate (computer) resolution.

We seek a probabilistic growth model that leads to BTs similar in structure to those one might expect to be generated by random applications of the Trapezoid Algorithm. To this end, we desire a growth model that produces BTs which:

- are "bushy" near the root (since most nonlinear functions will require some interval refinement initially);
- rather quickly get "scrawny" (since most "actual" functions will be rather smooth throughout most of the domain of interest);
- stay rather "shallow" (since [numerical] resolution will be exceeded before a BT gets very deep).

Our formal model employs a single real-valued parameter $\delta < 1$ to generate a suite of random trees for each of our experiments. Each experiment works on a family $\mathcal{F}(\delta)$ of random trees, which is defined by the rule:

Each level-ℓ node of an evolving BT in $\mathcal{F}(\delta)$ spawns (two children) with probability δ^{ℓ} and halts with probability $1-\delta^{\ell}$.

Clearly our BTs will be shallow and will quickly get scrawny. To foster "bushiness" near our BTs' roots, we use values of δ that are close to unity—specifically, no smaller than 0.96.

The simulation environment. The input generator and network simulation were written in GNU C and run on a DEC Alpha under OSF1.[1]

Our probabilistic input model employs a linear congruential random number generator. Because spawning decisions are made locally, the order in which tasks are executed modifies the shape of the evolving tree. Thus, both the number of PEs and the policy may result in different trees, even if the number generator is initialized with the same "seed" value.

The simulation is configured by three parameters: the number of PEs, $p \geq 3$, the load-balancing regimen, $Alg \in \{\text{KOSO},\text{KOSO}^{\star}\}$, and the input family, $\delta \geq .96$. After running to completion, the simulator reports N, the total number of tasks consumed, h, the maximum height of encountered tasks, and T, the total running time.

5. The Experiments and Their Results

Our first analyses are based on data from a *pilot experiment*, in which we exercised both policies under a wide range of conditions. We executed 20 random samples from the input families $\delta = \{.96, .965, .97\}$ on the networks $\mathcal{R}_3$, $\mathcal{R}_6$, $\mathcal{R}_{10}$, and $\mathcal{R}_{20}$ under each policy, yielding 480 data points (2 policies $\times$ 20 instances $\times$ 3 input families $\times$ 4 networks).

Because our conjectures are concerned with network efficiency, we formulate this analysis in terms of the *overhead*, $\epsilon \geq 0$, incurred over a trial. ϵ represents the difference between actual running time and optimal running time, computed as $\epsilon = T - \lceil N/p \rceil$.

To verify our first conjecture—that both of our policies are "good"—we consider the relationship between ϵ and N: if the policies are performing optimally, then ϵ and N will be unrelated. Figure 1 displays this relationship as captured in the pilot experiment; clearly, ϵ increases as N increases. We derive an equation for this relationship using linear regression: $\epsilon = .013N + 26.9$. Even though the slope .013 is fairly small, it can be statistically distinguished from zero, and we must conclude that N and ϵ are related. On the other hand, the line accounts for only 16% of the variance in ϵ, suggesting that other factors are at work.

Our second conjecture—that KOSO* outperforms KOSO—suggests that ϵ should be significantly smaller for KOSO* than

[1] Source code for the simulation can be obtained via the WWW at `http://eksl-www.cs.umass.edu/~gregory/SPDP.html`.

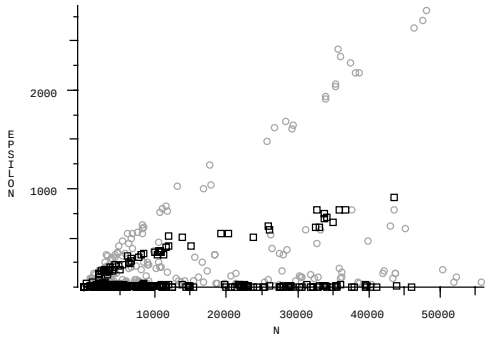

Figure 1. The relationship between N and ϵ. Light circles denote KOSO trials, dark squares represent KOSO*.

for KOSO. As a first test of this, we compare the overhead incurred by each policy, using a *t-test* to determine if the difference we observe could have arisen by random chance. Comparing the average overhead for KOSO (305.9) with that of KOSO* (91.8) yields a test statistic of 5.75, hence a probability $< 10^{-5}$ that the policies incur equal overhead.

Extending the analysis. Our first experiment has shown that KOSO* is apparently a better load-balancing policy, but it has not resolved the issue of (approximate) optimality. Because we know that large rings may incur additional overhead due to the cost of initial loading, we extend the analysis to account for ring size p. Two-factor *analysis of variance* (ANOVA) separates the influences of p and Alg on ϵ by generating *F-statistics* for three effects: one for p ($F = 100.9$), one for Alg ($F = 57.5$), and one for the non-linear *interaction* between them ($F = 18.9$). All three of these values are highly significant (indicating a probability of no effect $< 10^{-5}$), which means that both p and Alg affect the value of ϵ, and that p affects the way that Alg affects the value of ϵ. The average ϵ under each condition is reported in table 2.

Policy	3	6	10	20
KOSO	29.1	65.8	229.6	899.1
KOSO*	2.8	7.3	19.8	337.2

Table 2. Mean ϵ for Alg and p.

Under both policies, ϵ increases significantly between $p = 10$ and $p = 20$. This leads us to suspect the results are heavily influenced by the *startup costs* associated with loading the ring. To isolate startup costs from other overhead, we ran another experiment that measures two new variables: *Startup*, the number of steps before all PEs become busy, and *Steady*, the number of steps in which all PEs are occupied. We divide each by total running time T, to arrive at the fraction of running time spent in each condition.

We repeated the two-factor ANOVA of p and Alg on each of these variables, in hopes of eliminating one or more of the effects. The analysis of *Startup* shows that p has a significant impact, but the effect of Alg and the nonlinear interaction have gone away. This is precisely what we predicted: ring size affects the overhead due to startup costs.

Unfortunately, our analysis shows that *Steady* is subject to the same effects as the overhead ϵ. Thus, while the new experiment yielded one interesting result, we have reached another dead end.

Looking inside the computation. Still lacking precise quantification of the effects of either p or Alg, we ran another experiment to look even deeper inside the computation. In this experiment, we collected *time-series* of the number of busy PEs and the net queue-size for each PE on each step of the trial. From a sample of such trials, we selected four representative instances of approximately equal length, one for each algorithm on two different size rings. These data are displayed in figure 2.

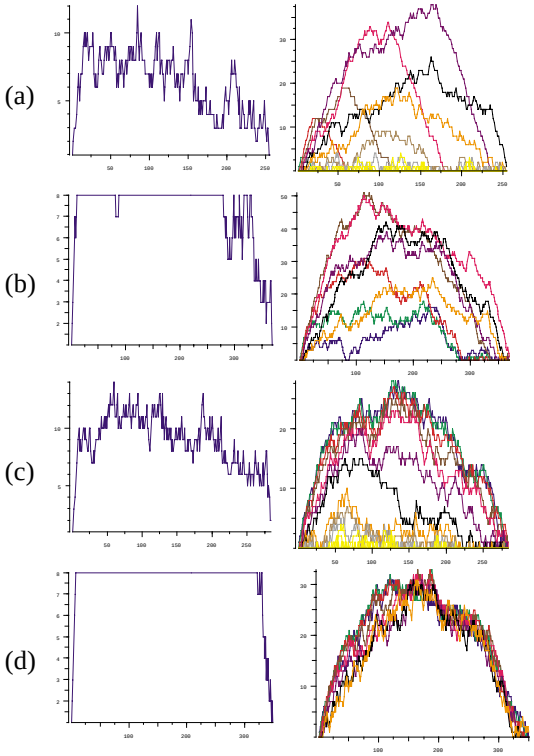

Figure 2. Busy PEs (left) and queue-sizes (right) for: KOSO **on (a)** $\mathcal{R}_{16}$ **and (b)** $\mathcal{R}_8$**;** KOSO* **on (c)** $\mathcal{R}_{16}$ **and (d)** $\mathcal{R}_8$**.**

The plots in figure 2 show a variety of interesting features. First, note that the number of busy PEs (left) never stabilizes on $\mathcal{R}_{16}$ (plots (a) and (c)), while on $\mathcal{R}_8$ (plots (b) and (d)) this number quickly reaches a plateau and remains there for most of the trial. This plateau is indicative of the ring reaching its steady-state, where all PEs are busy. Thus, we see further evidence of ring size affecting the startup time: larger rings never reach the steady state.

The graphs of queue-size (right) tell an even more interesting story. When the ring never reaches its steady state (plots (a) and (c)), we see a large variance among the queue-sizes. When the network spends most of its time in steady state (plots (b) and (d)), this variance should be reduced; however, *this occurs only under policy* KOSO*. In plot (d) we see that the queues for KOSO* are nearly the same size over the entire trial. This is an important result, because it shows that KOSO* results in *more even queue sizes* than KOSO; thus, KOSO* literally does a better job of *balancing the load* among the PEs, by minimizing variance in the length of PE queues.

6. Ongoing Work

Having (largely) substantiated our conjectures on an abstract computational load, we are currently devising an experimental study that will compare the results reported here with those obtained from using KOSO and KOSO* to schedule "real" tree-structured computations, specifically those generated by multigrid algorithms.

Acknowledgements. The research of L.-X. Gao and A. L. Rosenberg was supported in part by NSF Grant CCR-92-21785. The research of D. E. Gregory was supported by an NSF Graduate Research Fellowship.

References

[1] P.R. Cohen (1995): *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA.

[2] L.-X. Gao and A.L. Rosenberg (1996): Toward efficient scheduling of evolving computations on rings of processors. *J. Parallel Distr. Comput.*, to appear.

[3] L.-X. Gao, A.L. Rosenberg, R.K. Sitaraman (1995): Optimal architecture-independent scheduling of fine-grain tree-sweep computations. *7th IEEE Symp. on Parallel and Distr. Processing*, 620-629.

[4] S.L. Johnsson (1987): Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distr. Comput. 4*, 133-172.

[5] R.M. Karp and Y. Zhang (1988): Randomized parallel algorithms for backtrack search and branch-and-bound computation. *J. ACM 40*, 765-789.

[6] A.G. Ranade (1994): Optimal speedup for backtrack search on a butterfly network. *Math. Syst. Theory 27*, 85-101.

[7] T. Yang and A. Gerasoulis (1991): A fast static scheduling algorithm for dags on an unbounded number of processors. *Supercomputing '91*, 633-642.