On Balancing Computational Load on Rings of Processors

Lixin Gao   *Arnold L. Rosenberg*

Department of Computer Science
University of Massachusetts
Amherst, Mass. 01003, USA

Abstract. We consider a very simple, deterministic policy for scheduling certain genres of dynamically evolving computations — specifically, computations in which tasks that spawn produce precisely two offspring — on rings of processors. Such computations include, for instance, tree-structured branching computations. We believe that our policy yields good parallel speedup on most computations of the genre, but we have not yet been able to verify this. In the current paper, we show that when the evolving computations end up having the structure of complete binary trees or of two-dimensional pyramidal grids, our strategy yields almost optimal parallel speedup. Specifically, a ring having p processors can execute a computation that evolves into the height-n complete binary tree (which has $2^n - 1$ nodes) in time

$$T_{tree}(n;p) \leq \frac{1}{p}(2^n - 1) + \alpha_p^n + p,$$

for some constant $\alpha_p < 2$ that depends only on p. Similarly, the ring can execute a computation that evolves into the side-n pyramidal grid (which has $\binom{n+1}{2}$ nodes) in time

$$T_{grid}(n;p) \leq \frac{1}{p}\binom{n+1}{2} + \frac{3}{2}n + 2.$$

1 Introduction

1.1 Motivation

The promise of parallel computers to accelerate computation relies on the algorithm designer's ability to keep all (or most) of the computers' processors fruitfully[1] occupied all (or most) of the time. The problem of balancing computational loads so as to approach this goal has received considerable attention since the advent of (even the promise of)

[1]The importance of this qualifier should become obvious from our discussion.

parallel computers (cf. [2]). In this paper, we describe a simple strategy that we believe balances and schedules loads well for a variety of dynamically evolving computations, on parallel architectures whose underlying structure is a ring of identical processors. The challenge of balancing loads on such architectures is their large diameters, which precludes certain randomizing strategies that depend on low diameters. While we have not yet been able to delimit the class of computations that our strategy works well on, we report here on a first step, which establishes that our strategy yields near-optimal speedup on dynamically evolving computations that end up having have the structure of complete binary trees or of two-dimensional pyramidal grids. Specifically, our strategy allows a ring having p processors to execute a computation that ends up with the structure of the height-n complete binary tree (which has $2^n - 1$ nodes) in time

$$T_{tree}(n;p) \leq \frac{1}{p}(2^n - 1) + \alpha_p^n + p,$$

for some constant $\alpha_p < 2$ that depends only on p. Similarly, it allows the ring to execute a computation that ends up with the structure of the side-n pyramidal grid (which has $\binom{n+1}{2}$ nodes) in time

$$T_{grid}(n;p) \leq \frac{1}{p}\binom{n+1}{2} + \frac{3}{2}n + 2.$$

1.2 Background

A. What Is "Load Balancing?"

Somewhat surprisingly, there is a lack of universal agreement on what "load balancing" means. Most naively, we want all of the parallel computers' processors to perform (roughly) the same amount of work during the course of the computation of interest; cf. [7, 10]. The weakness of this definition is that, in principle, even though a computer's processors all perform the same amount of work, they could perform their work seriatim (for some unpredictable, undesirable reason), so that one achieves no speedup over a sequential computer. This pathological possibility motivates demanding also that (almost) all processors be occupied doing fruitful work (almost) all of the time. This goal (which we share with, say, [9]), if achieved, guarantees that the computation of interest is sped up by a factor of (roughly) p when executed on a p-processor machine — which is, of course, the most one could hope for. Two quite different approaches to balancing computational loads have evolved.

General heuristics. Systems-oriented practicioners have developed numerous heuristics — often fine-tuned for targeted architectures — for balancing large classes of computational loads. Such "general" schemes often enjoy the desirable characteristic of *unobtrusiveness*, in the sense that they do not materially complicate the programming of an application, and they do not consume many cycles of the machine (cf. footnote 1); but they usually elude rigorous analysis, due to the generality of the loads that they balance. Significant attempts to understand classes of "general" schemes can be found in [4, 7, 8, 10].

Application-specific techniques. Both algorithm designers and applications-oriented practicioners have devised a number of load-balancing strategies that exploit the specific characteristics of targeted classes of parallel computations, often focussing on targeted classes of architectures. Such "specialized" schemes usually impact algorithmic strategy significantly, often achieving only asymptotic speedups (sometimes with constant factors that cannot be ignored). However, these strategies often admit rigorous analyses that establish their benefits (at least within a probabilistic framework); cf. [5, 6, 9].

B. The Present Study

Where we fit in. The present study falls primarily into the "application-specific" class, in that we focus on load-balancing and scheduling a specific class of computations on a targeted architecture. However, we strive for the unobtrusiveness of the better "general" strategies in addition to the rigorous analyzability of the better "application-specific" strategies. Our study diverges from most comparable studies in three respects.

1. We study a more general class of computations. Whereas studies like [6, 9] focus on dynamically evolving tree-structured computations, we allow (in principle) any dynamically evolving computation in which a task that spawns produces precisely two new tasks.

2. We focus on a simple, unobtrusive, deterministic load-balancing and scheduling strategy, in an attempt to gauge how well a simple scheme can balance loads on a ring. The schemes in [6, 9] employ randomization in an essential way.

3. We focus on parallel computation on rings of identical processors, in an attempt to understand how to balance loads in high-diameter networks. The architectures in [6] are essentially PRAMs, while the ones in [9] are essentially butterfly networks (of identical processors).

Seeking unobtrusiveness. Parallel computing is more than just the sharing of work by multiple agents, because of the complication of orchestrating interprocessor communication. It is well known that the overhead for communication can offset the gains from concurrent computation. This problem is especially acute in the context of procedures like load balancing, wherein overloaded processors must somehow transfer work to underloaded ones. In the present study, we "finesse" the problem of communication costs, by always having communication consist of one processor's passing a small packet of work to an immediate neighbor.

1.3 Our Load-Balancing Problem

A. The Architecture

We focus here on rings of identical processors (PEs, for short). The p-PE version $\mathbf{R}_p$ of this architecture has p identical PEs, denoted $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{p-1}$, with each PE $\mathcal{P}_i$ connected directly to the two PEs $\mathcal{P}_{i \pm 1 \bmod p}$. We call PE $\mathcal{P}_{i+1 \bmod p}$ (resp., $\mathcal{P}_{i-1 \bmod p}$) the *clockwise* (resp., the *counterclockwise*) neighbor of PE $\mathcal{P}_i$.

B. The Computational Load

The computations we wish to balance have the structure of dynamically growing leveled dags (i.e., directed acyclic graphs) coming from two families, binary tree-dags and two-dimensional grid-dags. The nodes of these dags represent computational tasks, and their arcs denote "parenthood" (in a sense that will become clear).

Binary tree-dags. An *N-node binary tree-dag* (*tree*, for short) $\mathcal{T}$ is a dag whose nodes comprise a *full, prefix-closed* set of N binary strings. By *full*, we mean that the string $x0$ is a node of $\mathcal{T}$ precisely when the string $x1$ is; by *prefix-closed*, we mean that the string x is a node of $\mathcal{T}$ whenever $x0$ and $x1$ are. The arcs of $\mathcal{T}$ lead from each *parent* node x to its *left child* $x0$ and its *right child* $x1$. The null string λ is the *root* of $\mathcal{T}$; each node that has no child in the node-set is a *leaf* of $\mathcal{T}$. The length $|x|$ of a string x is the corresponding node's *level* in $\mathcal{T}$ (so the root is the unique node at level 0). The *weight* of a node x, denoted $Wgt(x)$, is the number of 1s in the binary string x. Of particular interest is the class of *complete* trees. For each n, the *height-n complete binary tree-dag* $\mathcal{T}_n$ is the tree whose nodes comprise all $2^n - 1$ binary strings of length $< n$. The *leaves* of $\mathcal{T}_n$ are the 2^{n-1} nodes at level n.

The (dynamic) computation that "generates" a tree $\mathcal{T}$ proceeds as follows.

Initially: $\mathcal{T}$ has precisely one node, which is simultaneously its root and its (current) *active* leaf.

Inductively: (The tasks corresponding to) some subset of the then-current active leaves (the particular subset depending on our computation schedule) get *executed.* An executed task/leaf may:

- *halt*, thereby becoming a *permanent* leaf,
- *spawn* two new active leaves, thereby becoming an interior node.

The computation ends when no active leaves remain.

To aid the reader's intuition, we show how one "real" computational problem abstracts to a tree. Consider the problem of numerically integrating a function f on a real interval

$[a, b]$ using the trapezoid rule. Each task in this computation corresponds to a subinterval of $[a, b]$. The task associated with subinterval $[c, d]$ proceeds as follows.

1. Evaluate the area of the trapezoid T having corners (in clockwise order) $(c, 0)$, $(c, f(c))$, $(d, f(d))$, $(d, 0)$.
2. If the quantity $\frac{1}{2}(d - c)$ is less than some prespecified (resolution) threshold, then return the area of T as the integral of f on $[c, d]$, and halt; otherwise, proceed.
3. Evaluate the area of the trapezoid T' having corners $(c, 0)$, $(c, f(c))$, $(\frac{1}{2}(c+d), f(\frac{1}{2}(c+d)))$, $(\frac{1}{2}(c+d), 0)$; evaluate the area of the trapezoid T'' having corners $(\frac{1}{2}(c+d), 0)$, $(\frac{1}{2}(c+d), f(\frac{1}{2}(c+d)))$, $(d, f(d)), (d, 0)$.
4. If the sum of the areas of T' and T'' differs from the area of T by less than some prespecified (accuracy) threshold, then return the area of T as the integral of f on $[c, d]$, and halt; otherwise proceed.
5. Solve the two new tasks corresponding to the intervals $[c, \frac{1}{2}(c+d)]$ and $[\frac{1}{2}(c+d), d]$.

Step 5 corresponds to a current leaf's spawning two new leaves; steps 2 and 4 correspond to a current leaf's halting.

Two-dimensional grid-dags. An *N-node two-dimensional grid-dag* (*grid*, for short) $\mathcal{G}$ is a dag whose nodes comprise a *full, prefix-closed* set of N pairs of nonnegative integers. By *full*, we mean that the pair $\langle k+1, \ell \rangle$ is a node of $\mathcal{G}$ precisely when the pair $\langle k, \ell+1 \rangle$ is; by *prefix-closed*, we mean that, if the pair $\langle k, \ell \rangle$ is a node of $\mathcal{G}$, then:

- If $k = 0$, then[2] $\langle k, \ell \ominus 1 \rangle$ is also a node of $\mathcal{G}$;
- if $\ell = 0$, then $\langle k \ominus 1, \ell \rangle$ is also a node of $\mathcal{G}$;
- else, at least one of $\langle k-1, \ell \rangle$ and $\langle k, \ell - 1 \rangle$ is also a node of $\mathcal{G}$.

The arcs of $\mathcal{G}$ lead from each *parent* node $\langle k, \ell \rangle$ to its *left child* $\langle k, \ell+1 \rangle$ and its *right child* $\langle k+1, \ell \rangle$. The pair $\langle 0, 0 \rangle$ is the *origin-node* of $\mathcal{G}$; each node that has no child is a *sink-node* of $\mathcal{G}$. The sum $k + \ell$ is the *level* of node $\langle k, \ell \rangle$ in $\mathcal{G}$ (so the origin-node is the unique node at level 0). Of particular interest is the class of *pyramidal* grids. For each n, the *side-n pyramidal grid-dag* $\mathcal{G}_n$ is the grid whose nodes comprise all $\binom{n+1}{2}$ pairs of nonnegative integers $\langle k, \ell \rangle$ such that $k + \ell < n$. The *sink-nodes* of $\mathcal{G}_n$ are the n pairs at level $n - 1$.

The (dynamic) computation that "generates" a grid $\mathcal{G}$ proceeds as follows.

[2] $\ominus$ denotes *positive subtraction*: $m \ominus 1 \overset{def}{=} \max(0, m-1)$.

Initially: $\mathcal{G}$ has precisely one node, which is simultaneously its origin-node and its (current) *active* sink-node.

Inductively: (The tasks corresponding to) some subset of the then-current active sink-nodes (the particular subset depending on our computation schedule) get *executed.* An executed task/sink-node may:

- *halt*, thereby becoming a *permanent* sink-node,
- *spawn* two new *task-arcs*, thereby becoming an interior node. Each newly spawned task-arc is associated with either a *unary task* or a *binary task.*
  - A unary task-arc leads from the executed task to a newly created active sink-node.
  - A binary task-arc leads from the executed task-node either to a newly created *inactive* sink-node, or to a pre-existing inactive sink-node that was created by another executed node,[3] which thereby becomes active.

The computation ends when all remaining sink-nodes are permanent ones. Intuitively, a sink-node switches from inactive to active status when it gets the correct number of parents, meaning that its associated task has received all needed inputs.

"Solving" the load-balancing problem. Our goal is to discover and analyze: (a) a simple, deterministic regimen for balancing the computational load generated by dynamically growing trees and grids on rings of PEs; (b) a policy for scheduling the resulting load, that is provably efficient in the following sense. Let the phrase "unit time" denote the aggregate time it takes one PE of a ring to

- execute one node of a tree or grid
- transmit to an immediate neighbor a description of the task denoted by a single tree- or grid-node.

Then, we seek balancing-plus-scheduling policies which, for each integer p, provably allow a p-PE ring $\mathbf{R}_p$ to execute any dynamically growing tree or grid that ends up having N nodes, in time $\leq N/p + p + o(N)$. The term N/p represents the ideal p-fold parallel speedup; the term p represents the overhead of "loading" $\mathbf{R}_p$ with work; the remaining term represents the extent to which our policies deviate from the ideal.

What we achieve. The problem just described seems to be very challenging technically: its solution has eluded us thus far. We have discovered a simple balancing-plus-scheduling

[3]The last clause precludes having "parallel" arcs from one node to another.

policy, which we believe works well on randomly generated trees and grids and on large explicit classes of such dags; however, we have not yet been able to verify or refute our belief. However, we have succeeded in proving that this policy "solves" the load-balancing problem in the (analytically) much simpler situations in which the dynamically evolving tree grows into a *complete* tree and the dynamically evolving grid grows into a *pyramidal* grid. (In the numerical integration scenario, for instance, such growth corresponds to the situation where all leaves of the tree halt because of the resolution threshold.) In these cases, our policy achieves essentially optimal speedup, in the sense made explicit at the end of Section 1.1.

The KS-BF balancing-plus-scheduling policy. We call our balancing-plus-scheduling policy KS-BF, for mnemonic reasons. The balancing component of the policy has each PE observe the regimen *K*eep-left-*S*end-right in response to a spawning task, meaning that a PE keeps the left child of the spawning task and sends the right child to its neighbor. The scheduling component of the policy mandates that each PE execute the tasks assigned to it in a locally *B*readth-*F*irst manner. A PE satisfies the latter requirement by keeping its as-yet unexecuted tasks in a priority-queue. The tasks are kept in the queue in order of their levels (in the tree or grid being executed) and, within a level, in breadth-first, or, "left-to-right" order. In the case of a tree, "left-to-right" order means lexicographic order of the string-names of the nodes; in the case of a grid, "left-to-right" order means in order of the first entries of the integer-pair-names of the nodes.

The details of the KS-BF policy for trees are as follows. Each computation begins with the root (and initial leaf) λ of the dynamically growing tree $\mathcal{T}$ as the sole occupant of PE $\mathcal{P}_0$'s task-queue. At each step of the computation, the task-queue of each PE $\mathcal{P}_i$ contains some subset of the then-current active leaves of $\mathcal{T}$. Each $\mathcal{P}_i$ having a nonempty task-queue performs the following actions.

1. $\mathcal{P}_i$ executes that active leaf x in its task-queue which is first in breadth-first order.

2. If leaf x spawns two children, then $\mathcal{P}_i$ adds the new leaf $x0$ (the left child) to its task-queue, and it transmits the new leaf $x1$ (the right child) to the task-queue of its clockwise neighbor $\mathcal{P}_{i+1 \bmod p}$.

We assess one time unit for the entire process of executing a task and performing the balancing actions just described. Adapting these details to grids is straightforward.

In order to lend some intuition for the KS regimen, we illustrate in Table 1 how the regimen distributes the nodes of $\mathcal{T}_5$ in $\mathbf{R}_4$.

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

Table 1. *The node-assignments when $\boldsymbol{R}_4$ executes $\mathcal{T}_5$ under the* KS *regimen.*

In fact, we can go even further and specify exactly which PE of $\mathbf{R}_p$ will execute each node of an given input tree or grid.

Lemma 1.1 *Under the* KS *regimen:*

(a) *each node x of an input tree is executed at PE $\mathcal{P}_{Wgt(x) \bmod p}$ of $\boldsymbol{R}_p$.*

(b) *each node $\langle k, \ell \rangle$ of an input grid $\mathcal{G}$ is executed at PE $\mathcal{P}_{k \bmod p}$ of $\boldsymbol{R}_p$.*

Proof: Straightforward inductions establish the result. We present a proof just for a tree $\mathcal{T}$, leaving the similar proof for grids to the reader.

Note first that the root λ of $\mathcal{T}$, which has weight $Wgt(\lambda) = 0$, is executed at PE $\mathcal{P}_0$.

Assume inductively that some given (but arbitrary) nonleaf node x of $\mathcal{T}$ is executed at PE $\mathcal{P}_{Wgt(x) \bmod p}$. By definition of the KS regimen:

- If the left child $x0$ of x is also in $\mathcal{T}$, then it is executed at PE $\mathcal{P}_{Wgt(x) \bmod p}$ (the "keep left" part of the regimen).

- If the right child $x1$ of x is also in $\mathcal{T}$, then it is executed at the clockwise neighbor $\mathcal{P}_{Wgt(x)+1 \bmod p}$ of PE $\mathcal{P}_{Wgt(x) \bmod p}$ (the "send right" part of the regimen).

These cases extend the induction, because $Wgt(x0) = Wgt(x)$, and $Wgt(x1) = Wgt(x)+1$. ∎

We can infer directly from Lemma 1.1 that the KS-BF policy does not perform well on all trees or all grids.

Trees. The *h-level w-complete binary tree* is the binary tree whose node-set comprises all binary strings of length $< h$ that have weight $\leq w$. Consider the family of $(\ell_p \overset{def}{=} \log p)$-complete binary trees. When h is significantly larger than ℓ_p, the number of nodes in the h-level member of this family is easily shown to be $N(h;p) \overset{def}{=} \Theta(h^{\ell_p})$. Note first that, in this case, h is not the dominant term in the ideal p-PE running time $O(N(h;p)/p + h)$. Note next that no policy that employs the KS balancing regimen can achieve even close to (asymptotic) optimal parallel speedup (i.e., speedup proportional to p), since the regimen assigns work to only $\log p$ PEs.

Grids. Let us focus on the family of grids whose node-sets have the form $\{i,j\} \times \{0,1,\ldots,m-1\} \cup \{0,1,\ldots,m-1\} \times \{0\}$, where $i \equiv 0 \bmod p$, and $j \equiv 1 \bmod p$. Note that, for sufficiently large m, the KS balancing regimen assigns almost all the work to PEs $\mathcal{P}_0$ and $\mathcal{P}_1$.

Despite its bad worst-case beavior, we believe that the KS-BF balancing-plus-scheduling policy does perform well on large classes of trees and grids, as well as on most "randomly growing" trees and grids. The best we can prove at this point is that the policy is almost optimal when the input tree grows into a complete tree and when the input grid grows into a pyramidal grid. Formally,

Theorem 1.1 *When the ring $\boldsymbol{R}_p$ uses the* KS-BF *balancing-plus-scheduling policy:*

*(**a**) It executes each input tree that grows into the height-n complete binary tree $\mathcal{T}_n$ in time*

$$T_{tree}(n;p) \leq \frac{1}{p}(2^n - 1) + \alpha_p^n + p,$$

where $\alpha_p < 2$ is a constant that depends only on p.

*(**b**) It executes each input grid that grows into the side-n pyramidal grid $\mathcal{G}_n$ in time*

$$T_{grid}(n;p) \leq \frac{1}{p}\binom{n+1}{2} + \frac{3}{2}n + 2.$$

We prove part (a) of Theorem 1.1 in Section 2 and part (b) in Section 3.

2 Analyzing the KS-BF Policy on Trees

In this section, we prove part (a) of Theorem 1.1, in two steps. First, in Section 2.1, we prove that the KS regimen approximately balances the amount of work that the PEs of $\mathbf{R}_p$ perform while executing any complete binary tree $\mathcal{T}_n$. Then, in Section 2.2, we prove that breadth-first scheduling of a KS-balanced workload ensures that, once a PE of $\mathbf{R}_p$ first receives a task of $\mathcal{T}_n$ to execute, it will always have work to do until it has completed all of its work. This suffices to establish part (a) of Theorem 1.1. Note that, whereas p is a fixed but arbitrary constant throughout, n ranges over all (positive) integers for each value of p.

2.1 Work Distribution under the KS Regimen

Our analysis of the amount of work done by each PE of $\mathbf{R}_p$ while executing a tree that grows into $\mathcal{T}_n$ builds on Lemma 1.1(a), which allows us to profile the distribution of work among the PEs. For $0 \leq i \leq p-1$, let $W_i(n)$ denote the total work done by PE $\mathcal{P}_i$ during this execution.

Theorem 2.1 *The exact value of $W_i(n)$ is given by:*

$$W_i(n) = \sum_{k \equiv i+1 \bmod p} \binom{n}{k}. \tag{2.1}$$

This yields the implicit bound,

$$\left| W_i(n) - \frac{2^n - 1}{p} \right| \leq \alpha_p^n, \tag{2.2}$$

where $\alpha_p < 2$ is a constant depending only on p.

Proof: Note first that, for any k, the number of length-k binary strings of weight w is precisely

$$\binom{k}{w}.$$

It is immediate, therefore, by Lemma 1.1(a) and the definition of complete binary tree, that precisely

$$W_i(n;\ell) = \sum_{j \equiv i \bmod p} \binom{\ell}{j}$$

nodes from level ℓ of $\mathcal{T}_n$ (where $0 \le \ell < n$) are executed by PE $\mathcal{P}_i$ of $\mathbf{R}_p$. Of course, the workshare $W_i(n)$ is just the summation of $W_i(n;\ell)$ over all levels of $\mathcal{T}_n$. In other words,

$$W_i(n) = \sum_{\ell=0}^{n-1} W_i(n;\ell) = \sum_{\ell=0}^{n-1} \sum_{j \equiv i \bmod p} \binom{\ell}{j}.$$

Elementary manipulations simplify this double summation.

$$\begin{aligned} W_i(n) &= \sum_{\ell=0}^{n-1} \sum_{j \equiv i \bmod p} \binom{\ell}{i} \\ &= \sum_{j \equiv i \bmod p} \sum_{\ell=0}^{n-1} \binom{\ell}{j} \\ &= \sum_{k \equiv i+1 \bmod p} \binom{n}{k}, \end{aligned}$$

whence equation (2.1).

To obtain the more perspicuous bound (2.2) on $W_i(n)$, which gauges the actual workshares' deviations from the ideal workshare, we use the Discrete Fourier Transform (DFT) (see Chapter 7 of [1]).[4] In what follows, ω is a primitive pth root of unity. Denote by $F_p(\omega)$ the order-p DFT matrix

$$F_p(\omega) = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{p-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(p-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(p-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{p-1} & \omega^{2(p-1)} & \omega^{3(p-1)} & \cdots & \omega^{(p-1)^2} \end{pmatrix}.$$

The following facts allow us to obtain information about the workshares $W_i(n)$ via calculations involving the DFT matrix $F_p(\omega)$.

Fact 2.1 *The cumulative amount of work performed by all PEs of* $\boldsymbol{R}_p$ *when executing* $\mathcal{T}_n$ *is*

$$\sum_{i=0}^{p-1} W_i(n) = 2^n - 1.$$

[4]The authors discovered after completing this paper that essentially the same analysis appears, with different motivation, in the unpublished Master's thesis [3].

Fact 2.1 is true because every node of $\mathcal{T}_n$ is executed exactly once. ∎

Fact 2.2 *For each $k \in \{1, 2, \ldots, p-1\}$,*

$$\begin{aligned}\sum_{i=0}^{p-1} W_i(n)\omega^{k(i+1 \bmod p)} &= \sum_{i=0}^{p-1} \omega^{k(i+1 \bmod p)} \sum_{\ell \equiv i+1 \bmod p} \binom{n}{\ell} \\ &= (1+\omega^k)^n - 1.\end{aligned}$$

Fact 2.2 is true (by regrouping terms) because ω is a *primitive* pth root of unity. ∎

Facts 2.1 and 2.2 combine to validate the following matrix-vector product.

$$F_p(\omega) \begin{pmatrix} W_{p-1}(n) \\ W_0(n) \\ W_1(n) \\ \vdots \\ W_{p-2}(n) \end{pmatrix} = \begin{pmatrix} 2^n - 1 \\ (1+\omega)^n - 1 \\ (1+\omega^2)^n - 1 \\ \vdots \\ (1+\omega^{p-1})^n - 1 \end{pmatrix} \tag{2.3}$$

It is well known that the matrix $F_p(\omega)$ is nonsingular and has inverse

$$F_p^{-1}(\omega) = \frac{1}{p} F_p(\omega^{-1}).$$

We can, therefore, premultiply both sides of equation (2.3) by $F_p^{-1}(\omega)$ to obtain the following aggregated expression for the workshares $W_i(n)$.

$$\begin{pmatrix} W_{p-1}(n) \\ W_0(n) \\ W_1(n) \\ \vdots \\ W_{p-2}(n) \end{pmatrix} = F^{-1}(\omega) \begin{pmatrix} 2^n - 1 \\ (1+\omega)^n - 1 \\ (1+\omega^2)^n - 1 \\ \vdots \\ (1+\omega^{p-1})^n - 1 \end{pmatrix} \tag{2.4}$$

By expanding equation (2.4), we obtain the following explicit expression for $W_i(n)$ in terms of ω.

$$W_i(n) = \frac{1}{p}\left[(2^n - 1) + \sum_{j=1}^{p-1} \omega^{-j(i+1)} \left((1+\omega^j)^n - 1\right)\right]. \tag{2.5}$$

By noting that $\sum_{j=0}^{p-1} \gamma^j = 0$ for any pth root of unity γ (since the sum is invariant under multiplication by $\gamma \neq 0$), one easily simplifies expression (2.5):

$$W_i(n) = \frac{1}{p}\left[2^n + \sum_{j=1}^{p-1} \omega^{-j(i+1)} (1+\omega^j)^n\right]. \tag{2.6}$$

Expression (2.6) affords us a direct path to bound (2.2). First, by direct manipulation of expression (2.6), we find that

$$\left|W_i(n) - \frac{2^n - 1}{p}\right| = \left|\frac{1}{p}\sum_{j=1}^{p-1} \omega^{-j(i+1)}(1+\omega^j)^n + \frac{1}{p}\right|. \tag{2.7}$$

Next, we invoke the triangle inequality to convert equation (2.7) to the inequality

$$\left|W_i(n) - \frac{2^n - 1}{p}\right| \leq \frac{1}{p}\sum_{j=1}^{p-1} \left|\omega^{-j(i+1)}(1+\omega^j)^n\right| + \left|\frac{1}{p}\right|. \tag{2.8}$$

Since every power of ω is a pth root of unity, inequality (2.8) simplifies to

$$\left|W_i(n) - \frac{2^n - 1}{p}\right| \leq \frac{1}{p}\sum_{j=1}^{p-1} \left|(1+\omega^j)^n\right| + \left|\frac{1}{p}\right|. \tag{2.9}$$

Let $\alpha_p = \max_{0<j<p} |1+\omega^j|$. Inequality (2.9) immediately yields a bound of the form of inequality (2.2):

$$\left|W_i(n) - \frac{2^n - 1}{p}\right| \leq \frac{p-1}{p}\alpha_p^n + \frac{1}{p} \leq \alpha_p^n. \tag{2.10}$$

By noting that $|1+\omega^j| < 2$ for all $j \neq 0$, one verifies that $\alpha_p < 2$, so inequality (2.10) is, in fact, the sought bound (2.2). ∎

2.2 Running Time under the KS-BF Policy

In this section, we analyze the time required by $\mathbf{R}_p$ to execute the tree $\mathcal{T}_n$ under the KS-BF policy. To this end, we establish the following notation.

- $\pi(x) \overset{def}{=} Wgt(x) \bmod p$ denotes the index of the PE at which node x of $\mathcal{T}_n$ is executed.
- $N(x)$ denotes the set of nodes of $\mathcal{T}_n$ that are assigned to PE $\mathcal{P}_{\pi(x)}$ by the KS regimen.
- $N^<(x)$ denotes the subset of $N(x)$ comprising tree-nodes that precede x in breadth-first order.
- $\nu(x) \overset{def}{=} |N^<(x)|$ denotes the cardinality of the set $N^<(x)$.

Theorem 2.2 *Node x of $\mathcal{T}_n$ is executed by PE $\mathcal{P}_{\pi(x)}$ of $\boldsymbol{R}_p$ at step $\tau(x) \overset{def}{=} \nu(x) + \pi(x)$. Hence, $\boldsymbol{R}_p$ executes an input tree that grows into $\mathcal{T}_n$ in time*

$$T_{tree}(n;p) \leq p + \max_{0 \leq i \leq p-1} W_i(n).$$

Proof: We argue: (a) that the KS balancing regimen ensures that each PE $\mathcal{P}_i$ of $\mathbf{R}_p$ starts working at time i; (b) that the BF scheduling policy guarantees that each PE $\mathcal{P}_i$ of $\mathbf{R}_p$ performs all of its work in an uninterrupted block of $W_i(n)$ steps. It will follow that $\mathbf{R}_p$ finishes executing $\mathcal{T}_n$ by time $\max_{0 \leq j < p} W_j(n) + p$.

Assertion (a) is immediate by induction, since only $\mathcal{P}_0$ has work at step 0 of the execution, and the KS regimen passes work along only one PE per step.

We establish assertion (b) by verifying the schedule in the statement of the theorem, namely, that each node x of $\mathcal{T}_n$ gets executed by PE $\mathcal{P}_{\pi(x)}$ at step $\tau(x)$. Note first that node x could not be executed any earlier than step $\tau(x)$, because PE $\mathcal{P}_{\pi(x)}$ does not start working until step $\pi(x)$, and there are $\nu(x)$ tree-nodes that $\mathcal{P}_{\pi(x)}$ must execute (because of the BF policy) before it gets to node x. We complete the proof by verifying that node x is available for execution at time $\tau(x)$. We proceed by induction on the breadth-first order of the nodes of $\mathcal{T}_n$.

As the base of the induction, we remark that the root λ of $\mathcal{T}_n$ gets executed at PE $\mathcal{P}_0$ at step $\tau(\lambda) = 0$, as predicted by the fact that $\pi(\lambda) = \nu(\lambda) = 0$.

Next, focus on a nonroot node $x\delta$ of $\mathcal{T}_n$, where $\delta \in \{0,1\}$, and assume that every node y of $\mathcal{T}_n$ which precedes $x\delta$ in breadth-first order is executed at step $\tau(y)$.

Consider first the case when $\delta = 0$, so that node $x\delta = x0$ is a *left* child of its parent x. Now, since x precedes $x0$ in breadth-first order, our inductive hypothesis ensures that node x is executed at step $\tau(x)$. It follows that node $x0$ resides in the task-queue of PE $\mathcal{P}_{\pi(x)}$ (since $\pi(x0) = \pi(x)$ under the KS regimen) beginning at step $\nu(x) + \pi(x0) + 1$. Since, obviously, $\nu(x) < \nu(x0)$, this means that node $x0$ is available to be executed by step $\tau(x0)$.

Consider next the case when $\delta = 1$, so that node $x\delta = x1$ is a *right* child of its parent x. As before, since x precedes $x1$ in breadth-first order, our inductive hypothesis assures us that node x is executed at step $\tau(x)$. It follows that node $x1$ resides in the task-queue of PE $\mathcal{P}_{\pi(x1)}$ beginning at step $\nu(x) + \pi(x) + 1$. Note the following:

1. $\pi(x1) = \pi(x) + 1 \bmod p$.
   This equation is an immediate consequence of the KS regimen.

2. $\nu(x1) \geq \nu(x)$.
   This inequality holds because, for each tree-node $y \in N^{<}(x)$, the tree-node $y0^{|x|-|y|}1 \in$

$N^{<}(x1)$. We know that $y0^{|x|-|y|}1$ is a node of $\mathcal{T}_n$ because it precedes $x1$ in breadth-first order, and by hypothesis, $x1$ is a node of $\mathcal{T}_n$.

To continue with the analysis, we distinguish two subcases. Say first that $\pi(x) < p-1$, so $\pi(x1) = \pi(x)+1$. In this subcase,

$$\begin{aligned} \nu(x)+\pi(x)+1 &= \nu(x)+\pi(x1) \\ &\leq \nu(x1)+\pi(x1). \end{aligned}$$

In the other subcase, $\pi(x) = p-1$, so $\pi(x1) = 0 = \pi(x)-p+1$. In this subcase, we know that node $x1$ has length at least p, since it contains a 1 and has (by Lemma 1.1(a)) $Wgt(x1) \equiv 0 \bmod p$. It follows that, *in addition to* the $\nu(x)$ elements in $N^{<}(x1)$ guaranteed by Assertion (2) above, $N^{<}(x1)$ must also contain at least the $p+1$ additional nodes $\{0^i \mid 0 \leq i \leq p\}$. We thus have:

$$\begin{aligned} \nu(x)+\pi(x)+1 &= \nu(x)+\pi(x1)+p \\ &< \nu(x1)+\pi(x1). \end{aligned}$$

In either subcase, node $x1$ is available to be executed no later than step $\tau(x1)$.

In summary: we have shown that, in all cases, node $x\delta$ is executed precisely at step $\tau(x\delta)$. This extends our induction and completes the proof. ∎

3 Analyzing the KS-BF Policy on Grids

This section is devoted to proving part (b) of Theorem 1.1. Since our proof will follow both the organization and underlying reasoning of Section 2, we shall be somewhat sketchy in this section.

3.1 Work Distribution under the KS Regimen

Our analysis of the amount of work done by each PE of $\mathbf{R}_p$ while executing a grid that grows into $\mathcal{G}_n$ builds on Lemma 1.1(b). For $0 \leq i \leq p-1$, let $W_i(n)$ denote the total work done by PE $\mathcal{P}_i$ of $\mathbf{R}_p$ during this execution.

Theorem 3.1 *The exact value of $W_i(n)$ is given by:*

$$W_i(n) = n - i + \left(n - i - \frac{p}{2}\right)\left\lfloor \frac{n-i}{p} \right\rfloor - \frac{p}{2}\left\lfloor \frac{n-i}{p} \right\rfloor^2. \tag{3.1}$$

This yields the implicit bound,

$$\left| W_i(n) - \frac{1}{p}\binom{n+1}{2} \right| \leq \frac{3}{2}n + 2. \tag{3.2}$$

Proof: Lemma 1.1(b) tells us that each node $\langle k, \ell \rangle$ of $\mathcal{G}_n$ in executed by PE $\mathcal{P}_{k \bmod p}$. It follows, therefore, that, for each $i \in \{0, 1, \ldots, p-1\}$,

$$W_i(n) = \sum_{k \equiv i \bmod p} (n-k) = \sum_{j=0}^{\lfloor \frac{n-i}{p} \rfloor} (n - (i + jp)).$$

The latter expression evaluates, by standard techniques, to

$$W_i(n) = \left(n - i - \frac{p}{2}\left\lfloor \frac{n-i}{p} \right\rfloor\right)\left(\left\lfloor \frac{n-i}{p} \right\rfloor + 1\right). \tag{3.3}$$

Elementary manipulations convert expression (3.3) to expression 3.1.

In order to verify bound (3.2), we quantify the deviation of the actual workshares from the ideal. We begin by noting the amount of work each PE of $\mathbf{R}_p$ would do when executing $\mathcal{G}_n$ in a perfectly balanced world.

Fact 3.1 *In a perfectly balanced computation of $\mathcal{G}_n$, each PE of $\boldsymbol{R}_p$ would do work*

$$W \stackrel{def}{=} \frac{1}{p}\sum_{i=0}^{p-1} W_i(n) = \frac{1}{p}\binom{n+1}{2}.$$

Obviously, some workshares $W_i(n)$ exceed W, while others are smaller than W. In fact, the progression from the smallest workshare to the largest is quite regular.

Fact 3.2 *For all $i \in \{1, 2, \ldots, p-1\}$,*

$$W_{i-1}(n) + i - 1 \geq W_i(n) + i.$$

In particular, for all n,

$$W_0(n) > W_1(n) > \cdots > W_{p-1}(n).$$

Fact 3.2 can be verified as follows. For each $k \in \{0, 1, \ldots, n-1\}$, we call the set of grid-nodes $\{\langle k, \ell \rangle \mid 0 \leq \ell \leq n-k-1\}$ the kth *row* of $\mathcal{G}_n$. Lemma 1.1(b) assures us that all nodes in a given row of $\mathcal{G}_n$ are executed at the same PE of $\mathbf{R}_p$. We partition the set of rows of $\mathcal{G}_n$ into *bands*: the ith band, where $0 \le i \le \lfloor (n-1)/p \rfloor$, comprises those rows of $\mathcal{G}_n$ whose indices fall in the set $\{ip, ip+1, ip+2, \ldots, ip+p-1\}$; the somewhat awkward wording here is because the last band may contain fewer than p rows. Now, note that, within each band, the sum of the row-sizes and the row-indices stays constant (as long as the row exists). This verifies Fact 3.2. ∎

Facts 3.1 and 3.2 combine to ensure that $W_{p-1}(n) < W < W_0(n)$. We can bound the deviation of the actual workshares from the ideal, therefore, by bounding the differences $W_0(n) - W$ and $W - W_{p-1}(n)$. For the former difference, we have

$$W_0(n) - W = n + \left(n - \frac{p}{2}\right)\left\lfloor \frac{n}{p} \right\rfloor - \frac{p}{2}\left\lfloor \frac{n}{p} \right\rfloor^2 - \frac{1}{p}\binom{n+1}{2}.$$

Conservative estimates show this quantity to be no greater than

$$W_0(n) - W \le \left(\frac{3}{2} - \frac{1}{2p}\right) n < \frac{3}{2} n. \tag{3.4}$$

For the latter difference, we have

$$W - W_{p-1}(n) = \frac{1}{p}\binom{n+1}{2} - n + p - 1 - \left(n - \frac{3}{2}p + 1\right)\left\lfloor \frac{n-(p-1)}{p} \right\rfloor + \frac{p}{2}\left\lfloor \frac{n-(p-1)}{p} \right\rfloor^2.$$

Conservative estimates show this quantity to be no greater than

$$W - W_{p-1}(n) \le \left(\frac{3}{2} - \frac{1}{p}\right) n + \left(\frac{1}{2} + \frac{3}{2p}\right) \le \frac{3}{2} n + 2. \tag{3.5}$$

The bound (3.2) follows. ∎

3.2 Running Time under the KS-BF Policy

In this section, we analyze the time required by $\mathbf{R}_p$ to execute the grid $\mathcal{G}_n$ under the KS-BF policy. To this end, we adapt the following notation from Section 2.2.

- $\pi(k,\ell) \overset{def}{=} k \bmod p$ denotes the index of the PE at which node $\langle k, \ell \rangle$ of $\mathcal{G}_n$ is executed.
- $N(k,\ell)$ denotes the set of nodes of $\mathcal{G}_n$ that are assigned to PE $\mathcal{P}_{\pi(k,\ell)}$ by the KS regimen.

- $N^{<}(k,\ell)$ denotes the subset of $N(k,\ell)$ comprising tree-nodes that precede $\langle k,\ell\rangle$ in breadth-first order.
- $\nu(k,\ell) \overset{def}{=} |N^{<}(k,\ell)|$ denotes the cardinality of the set $N^{<}(k,\ell)$.

Theorem 3.2 *Node $\langle k,\ell\rangle$ of $\mathcal{G}_n$ is executed by PE $\mathcal{P}_{\pi(k,\ell)}$ of $\boldsymbol{R}_p$ at step $\tau(k,\ell) \overset{def}{=} \nu(k,\ell)+\pi(k,\ell)$. It follows that $\boldsymbol{R}_p$ executes an input grid that grows into $\mathcal{G}_n$ in time*

$$\begin{aligned} T_{grid}(n;p) &\leq \max_{0\leq i\leq p-1}(W_i(n)+i) \\ &\leq W_0(n). \end{aligned}$$

Proof: As in Theorem 2.2, we argue, first, that each PE $\mathcal{P}_i$ of $\mathbf{R}_p$ starts working at time i and, second, that each PE $\mathcal{P}_i$ of $\mathbf{R}_p$ performs all of its work in an uninterrupted block of $W_i(n)$ steps. The first of these assertions is proved just as in Theorem 2.2, so we focus on the second assertion.

We argue that each node $\langle k,\ell\rangle$ of $\mathcal{G}_n$ gets executed by PE $\mathcal{P}_{\pi(k,\ell)}$ at step $\tau(k,\ell)$. By the same argument as in Theorem 2.2, node $\langle k,\ell\rangle$ could not be executed any earlier than step $\tau(k,\ell)$. Hence, we need only verify that node $\langle k,\ell\rangle$ is available for execution at time $\tau(k,\ell)$. We proceed by induction on the breadth-first order of the nodes of $\mathcal{G}_n$.

As the base of the induction, we remark that the origin $\langle 0,0\rangle$ of $\mathcal{G}_n$ gets executed at PE $\mathcal{P}_0$ at step $\tau(0,0)=0$, as predicted by the fact that $\pi(0,0)=\nu(0,0)=0$.

Next, focus on a non-origin node $x' \overset{def}{=} \langle k+\rho,\ell+\sigma\rangle$ of $\mathcal{G}_n$, where $\{\rho,\sigma\}=\{0,1\}$, and assume that every node $\langle\phi,\psi\rangle$ of $\mathcal{G}_n$ which precedes node x' in breadth-first order is executed at step $\tau(\phi,\psi)$.

Consider first the case $\sigma=1$ (so $\rho=0$), in which node x' is a *left* child of its parent $x \overset{def}{=} \langle k,\ell\rangle$. Now, since node x precedes node x' in breadth-first order, our inductive hypothesis ensures that node x is executed at step $\tau(x)$. It follows that node x' resides in the task-queue of PE $\mathcal{P}_{\pi(x)}$ (since $\pi(x')=\pi(x)$ under the KS regimen) beginning at step $\nu(x)+\pi(x')+1$. Since, obviously, $\nu(x)<\nu(x')$, this means that node x' is available to be executed by step $\tau(x')$.

Consider next the case $\rho=1$ (so $\sigma=0$), in which node x' is a *right* child of its parent x. As before, since x precedes x' in breadth-first order, our inductive hypothesis ensures that node x is executed at step $\tau(x)$. It follows that node x' resides in the task-queue of PE $\mathcal{P}_{\pi(x')}$ beginning at step $\nu(x)+\pi(x)+1$. Note the following:

1. $\pi(x')=\pi(x)+1 \bmod p$.
   This is a consequence of the KS regimen.

2. $\nu(x') \geq \nu(x)$.
   This is because, for each grid-node $\langle a, b\rangle \in N^{<}(x)$, the grid-node $\langle a+1, b\rangle \in N^{<}(x')$. We know that $\langle a+1, b\rangle$ is a node of $\mathcal{G}_n$ because it precedes x' in breadth-first order, and by hypothesis, x' is a node of $\mathcal{G}_n$.

To continue with the analysis, we distinguish two subcases. Say first that $\pi(x) < p - 1$, so $\pi(x') = \pi(x) + 1$. In this subcase,

$$\begin{aligned} \nu(x) + \pi(x) + 1 &= \nu(x) + \pi(x') \\ &\leq \nu(x') + \pi(x'). \end{aligned}$$

In the other subcase, $\pi(x) = p-1$, so $\pi(x') = 0 = \pi(x) - p + 1$. In this subcase, we know that the level of node x' in $\mathcal{G}_n$ is at least p, since, being a right child, x' is at least in row 1 of $\mathcal{G}_n$, and has (by Lemma 1.1) $k + 1 \equiv 0 \bmod p$. It follows that, *in addition to* the $\nu(x)$ elements in $N^{<}(x')$ guaranteed by Assertion (2) above, $N^{<}(x')$ must also contain at least the p additional nodes $\{\langle k+i, \ell-i\rangle \mid 1 \leq i \leq \ell\} \cup \{\langle j, k+\ell+1-j\rangle \mid 0 \leq j \leq k\}$. We thus have:

$$\begin{aligned} \nu(x) + \pi(x) + 1 &= \nu(x) + \pi(x') + p \\ &\leq \nu(x') + \pi(x'). \end{aligned}$$

In either subcase, node x' is available to be executed no later than step $\tau(x')$.

Summarizing our argument: we have shown that, in all cases, node $\langle k+\rho, \ell+\sigma\rangle$ is executed precisely at step $\tau(k+\rho, \ell+\sigma)$. This extends our induction. The proof is finally completed by appealing to Fact 3.2. ∎

Acknowledgments. It is a pleasure to acknowledge helpful and stimulating conversations with Vittorio Scarano and Zhi-Li Zhang.

This research was supported in part by NSF Grant CCR-92-21785. A portion of this research was done while the second author was visiting the Department of Computer Science, The Technion, Haifa, Israel.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman (1974): *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass.

[2] R.P. Brent (1974): The parallel evaluation of general arithmetic expressions. *J. ACM 21*, 201-206.

[3] G. Even (1991): *Construction of Small Probability Spaces for Deterministic Simulation* (in Hebrew). M.Sc. Thesis, The Technion.

[4] A. Gerasoulis and T. Yang (1992): A comparison of clustering heuristics for scheduling dags on multiPEs. *J. Parallel and Distr. Comput.*

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

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

[7] R. Lüling and B. Monien (1993): A dynamic, distributed load-balancing algorithm with provable good performance. *5th ACM Symp. on Parallel Algorithms and Architectures*, 164-172.

[8] R. Lüling, B. Monien, F. Ramme (1990): Load-balancing in large networks: a comparative study. Typescript, Univ. Paderborn.

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

[10] L. Rudolph, M. Slivkin, E. Upfal (1991): A simple load balancing scheme for task allocation in parallel machines. *3rd ACM Symp. on Parallel Algorithms and Architectures*, 237-244.