On Scheduling Mesh-Structured Computations for Internet-Based Computing

Arnold L. Rosenberg, *Fellow*, *IEEE*

Abstract—Advances in technology have rendered the Internet a viable medium for employing multiple independent computers collaboratively in the solution of a single computational problem. A variety of mechanisms—e.g., Web-based computing, peer-to-peer computing, and grid computing—have been developed for such *Internet-based computing* (*IC*). Scheduling a computation for IC presents challenges that were not encountered with earlier modalities of parallel or distributed computing, especially when the computation's constituent tasks have interdependencies that constrain their order of execution. The process of scheduling such computations for IC is studied via a "pebble game" that abstracts the process of orchestrating the allocation of a computation's interdependent tasks to participating computers. A quality measure for plays of this game is developed that addresses the danger of "gridlock" in IC when a computation stalls because (due to dependencies) no tasks are eligible for execution. This measure rewards schedules that maximize the number of tasks that are eligible for execution at every step of the computation, one avenue for minimizing the likelihood of "gridlock." The resulting formal setting is illustrated via the problem of scheduling computations whose intertask dependencies have the structure of "evolving" meshes of finite dimensionalities. Within an idealized setting, a simple scheduling strategy is shown to be optimal when the dependencies have the structure of a two-dimensional mesh and within a constant factor of optimal for meshes of higher dimensionalities. The strategy remains optimal for a generalization of two-dimensional meshes whose structures are determined by abelian monoids (a monoid-based version of Cayley graphs). The optimality results for the idealized setting provide scheduling guidelines for real settings.

Index Terms—Internet-based computing, grid computing, P2P computing, Web computing, scheduling, mesh-structured computations, monoid dags, Cayley graphs.

1 Introduction

Advances in technology have rendered the Internet a viable medium for employing multiple independent computers collaboratively in the solution of a single computational problem. A variety of mechanisms have been developed for such *Internet-based computing* (*IC*, for short), including Web-based computing (*WC*, for short), Peer-to-Peer computing (*P2P*, for short), and Grid computing (*GC*, for short). Most forms of IC—including the three just cited—lend themselves naturally to the master-slave computing metaphor in which a "master" computer enlists the aid of remote "slave" (or "client") computers to collaborate in the computation of a massive collection of compute-intensive tasks of (roughly) equal complexities. In this paper, we seek (algorithmic) remedies for a problem that arises in IC when scheduling a computation whose constituent tasks have dependencies that constrain their order of execution. The problem, which arises from the difficulty of predicting the timing of interactions among remote clients in most IC environments, is that such computations can encounter a form of "gridlock" in which a computation stalls because (due to intertask dependencies) no task can be executed pending the execution of already allocated tasks.

• *The author is with the Department of Computer Science, University of Massachusetts Amherst, Amherst, MA 01003. E-mail: rsnbrg@cs.umass.edu.*

Manuscript received 6 Mar. 2003; revised 7 Nov. 2003; accepted 23 Dec. 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 118400.

1.1 The Growing Importance of IC

The success of IC in enabling a large variety of computations that could not be handled efficiently by any fixed-size assemblage of dedicated computing agents (e.g., multiprocessors or clusters of workstations) is attested to by the large—and growing—number of IC projects and project infrastructures. An early example of P2P computing is described in [34], whose computationally intensive biological computations would tax the resources of any single computing facility. The IC mechanism we are calling Web-based computing proceeds essentially as follows (think of SETI@home [20]). Volunteers (clients) register with a WC Web site. Thereafter, each registered volunteer visits the Web site from time to time to receive a task to compute. Some time after completing the task, the volunteer returns the results from that task and receives a new task. And the cycle continues. A small sampler of recent WC projects includes [20], [33], wherein astronomical calculations benefit from IC because of the volume of their workloads; [16], [21], wherein medical test analyses benefit for similar reasons; and [30], wherein security-motivated number-theoretic calculations benefit because of both the number and computational complexity of their individual tasks. An interesting "general-purpose" WC project is the World-Wide Grid (http://www.cs.mu.oz.au/~raj/grids/wwg), which aims to enlist a massive number of computers worldwide to create a supercomputer from idle cycles. As described in [10], [11], the IC mechanism we are calling Grid computing occurs within a Computational Grid: a consortium of (usually geographically dispersed) computing sites that contract to share resources. A small sampler of

GC projects and/or infrastructures includes commercial infrastructures available from, e.g., IBM (http://www.ibm.com) and Entropia (http://www.entropia.com), in addition to research-oriented grids and infrastructures such as: the LHC Computing Grid (http://lcg.web.cern.ch/LCG), which is dedicated to subatomic particle related computations; TeraGrid (http://www.teragrid.org) and the GUSTO Grid (http://www.globus.org), which are both world-wide "virtual supercomputer" projects; the UK e-Science Grid (http://www.escience-grid.org.uk), which is a UK-centered analogue of GUSTO and TeraGrid.

Thus far, there is little in the way of an algorithmic theory for Internet-based computing in its various guises. One finds in [6] a variety of application-motivated algorithmic challenges for GC. The problem of scheduling general (classes of) computations for IC faces a number of challenges that are specific to particular IC mechanisms.

- Some forms of IC (such as WC) employ "pull"-based task allocation; others (such as GC) employ "push"-based allocation.
- GC clients tend to be known and stable, hence trusted; WC clients tend to be anonymous and volatile, heightening the need for security measures (cf. [26]) and (if warranted by one's workload) protection against clients who never return assigned tasks.
- Grid members trade resources, which raises fairness-related concerns (cf. [7]), an issue that does not arise in WC.

Yet other scheduling problems arise from challenges that are common to all forms of IC.

- It is hard to predict the timing of communications over the Internet.
- The remote clients in all modes of IC display a level of heterogeneity in capabilities and resources that does not occur in other modalities of parallel or distributed computing.
- Remote clients in IC projects typically do not guarantee *when*—or, in the case of WC, *if*—they will perform an assigned task.

This paper focuses on a problem that arises because of the two types of temporal unpredictability just mentioned.

1.2 The Danger of Gridlock in IC

The absence of task completion-and-return timing guarantees from remote clients in IC projects is largely an annoyance when the tasks comprising the shared workload are mutually independent: The (usually massive) size of the workload ensures that some task is always eligible for execution— hence, for allocation to a client. However, when the workload's tasks have interdependencies that constrain their order of execution, the lack of timing guarantees creates a nontrivial scheduling challenge, which is discussed from a systems perspective in [19]. Specifically, such dependencies can potentially engender gridlock when no new tasks can be allocated for an indeterminate period, pending the execution of already allocated tasks. Although a variety of "safety devices" have been developed to address this danger—e.g., allocation of tasks to multiple clients (a technique used, e.g., in SETI@home [20]) or deadline-triggered reallocation of tasks (a technique described, e.g., in [5], [19])—each such device has significant drawbacks. To wit:

- Multiple allocation of tasks significantly thins out the remote workforce.
- Well-planned reallocation of tasks requires a reliable model of clients' computing behaviors.

Moreover—and most significantly—no such device eliminates the danger of gridlock since the "backup" remote client(s) assigned a given task may be as dilatory as the primary one.

1.3 The Current Study

In this paper, we begin to study the problem of scheduling, for IC platforms, computations whose intertask dependencies are structured as a *dag* (directed acyclic graph);[1] see [12], [13]. We focus on strategies for scheduling the allocation of the tasks of a computation-dag in a way that minimizes the danger of gridlock by always maximizing the number of tasks that are eligible for execution. Within an idealized setting, we identify a scheduling strategy that provably maximizes the number of such eligible tasks at every step of the computation for certain mesh-structured computation-dags and that comes within a constant factor of that goal for other mesh-structured dags. Of course, even within our idealized setting, such a scheduling strategy neither eliminates the danger of gridlock nor obviates "safety devices" such as those just mentioned. Rather, such a strategy provides guidelines for provably enhancing the effectiveness of the "safety devices." And, importantly, these guidelines prescribe actions that are under the control of the IC master and are independent of the behavior of the remote clients! (We see later that the proposed scheduling strategy accommodates the heterogeneity of clients in an implicit way.)

The contributions of our study are of three types.

1. We develop an IC-oriented variant of the *pebble games* that have proven, over decades, to be useful in the formal study of a large variety of scheduling problems (Section 2.3). A major facet of developing such a model is devising a formal criterion for comparing the qualities of competing schedules. We formulate such a criterion that is tailored to a broad class of computation-dags, including those whose dependencies have the structure of an "evolving" mesh of any finite dimensionality (a *mesh-dag*, for short). Mesh-dags model a variety of numerical linear algebra computations that arise in a broad range of scientific and engineering applications. (In a sequel to this paper [29], we perform an analogous study of some common computation-dags of other structures.) The quality-measuring component of our model formalizes and quantifies the intuitive goal that a schedule "maximizes the number of tasks that are eligible for execution at each step of the computation."

1. Precise definitions of all required notions appear in Section 2.

2. We identify a strategy for scheduling mesh-dags, that is optimal (according to our formal criterion) for two-dimensional mesh-dags (Section 3.1) and within a constant factor of optimal for mesh-dags of higher dimensionalities (Section 3.2).
3. We augment this second contribution by showing that the optimal strategy for two-dimensional mesh-dags remains optimal for a generalization of such dags whose structures are determined by abelian monoids (Section 4).

2 A Formal Model for Scheduling Dags for IC

In this section, we develop the formal entities that underlie our study: computation-dags and the pebble games that model the process of scheduling the dags' computations.

2.1 Computation-Dags

A *directed graph* (*digraph*, for short) $\mathcal{G}$ is given by a set of *nodes* $N_{\mathcal{G}}$ and a set of *arcs* (or, *directed edges*) $A_{\mathcal{G}}$, each having the form $(u \to v)$, where $u, v \in N_{\mathcal{G}}$. A *path* in $\mathcal{G}$ is a sequence of arcs that share adjacent endpoints, as in the following path from node u_1 to node u_n:

$$(u_1 \to u_2),\ (u_2 \to u_3),\ \ldots,\ (u_{n-2} \to u_{n-1}),\ (u_{n-1} \to u_n) \tag{1}$$

A *dag* (short for *directed acyclic graph*) $\mathcal{G}$ is a digraph that has no cycles; i.e., in a dag, there does not exist a path of the form (1) wherein $u_1 = u_n$. When a dag $\mathcal{G}$ is used to model a computation, i.e., is a *computation-dag*:

- Each node $v \in N_{\mathcal{G}}$ represents a task of the computation.
- An arc $(u \to v) \in A_{\mathcal{G}}$ represents the dependence of task v on task u: v cannot be executed until u is.

For each arc $(u \to v) \in A_{\mathcal{G}}$, we call u a *parent* of v and v a *child* of u in $\mathcal{G}$. The transitive extensions of the parent and child relations are, respectively, the *ancestor* and *descendant* relations. Every dag $\mathcal{G}$ (except for the degenerate one that has no nodes) has at least one parentless node; such a node is called a *source* of $\mathcal{G}$.

We have purposely not posited the *finiteness* of computation-dags. While the intertask dependencies in nontrivial computations usually have cycles—typically caused by loops—it is useful to "unroll" these loops when scheduling the computation's individual tasks. This converts the computation's (possibly modest-size) computation-digraph into a sequence of expanding "prefixes" of what "evolves" into an enormous—often infinite—computation-dag. One typically has better algorithmic control over the "steady-state" scheduling of such computations if one expands these computation-dags to their infinite limits and concentrates on scheduling tasks in a way that leads to a computationally expedient sequence of evolving prefixes. The case study that dominates the current paper focuses on an important class of such computations, namely, those whose underlying computation-dags have the structure of finite-dimensional meshes. (For completeness, we note that there are important classes of computation-dags that are inherently finite; one notable example are *reduction-dags*, all of whose arcs point away from the sources, toward the childless nodes; see, e.g., [18], [29].)

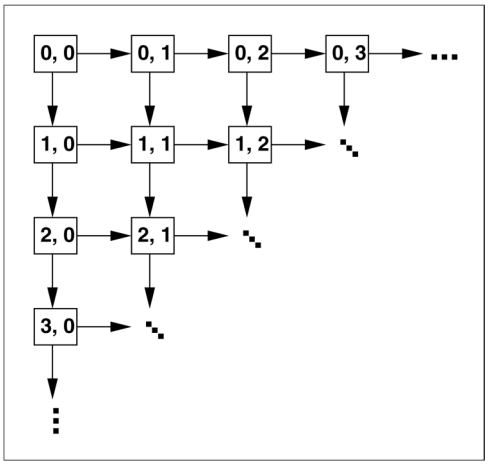


Fig. 1. The first four diagonal levels of the mesh-dag $\mathcal{M}_2$.

2.2 Mesh-Structured Computation-Dags

Let N denote the set of nonnegative integers. For each positive integer d, the *d-dimensional mesh-dag* $\mathcal{M}_d$ has node-set N^d, the set of d-tuples of nonnegative integers. The arcs of $\mathcal{M}_d$ connect each node $\langle v_1, v_2, \ldots, v_d\rangle \in \mathrm{N}^d$ to its d children $\langle v_1, v_2, \ldots, v_j + 1, \ldots, v_d\rangle$, for all $1 \le j \le d$. Node $\langle 0, 0, \ldots, 0\rangle$ is $\mathcal{M}_d$'s unique source node, often called its *origin*.

The *diagonal levels* of the dags $\mathcal{M}_d$ play an essential role in our study. Each such level is the subset of $\mathcal{M}_d$'s nodes that share the sum of their coordinates: For each $\ell = 0, 1, \ldots$, level ℓ of $\mathcal{M}_d$ is the set

$$L_\ell^{(d)} \stackrel{\text{def}}{=} \{\langle v_1, v_2, \ldots, v_d\rangle \mid v_1 + v_2 + \cdots + v_d = \ell\}.$$

Fig. 1 depicts the first four levels, $L_0^{(2)}, L_1^{(2)}, L_2^{(2)}, L_3^{(2)}$, of the two-dimensional mesh-dag $\mathcal{M}_2$.

Mesh-dags naturally model a large variety of computational situations. As just two examples, such situations include the numerical linear-algebraic computations that arise in myriad scientific and engineering applications, as well as database computations in which data organizes naturally into (multidimensional) tables. The hallmark of these situations is that each datum can be viewed naturally as residing at a "lattice point" $\langle x, \ldots, y, \ldots, z\rangle$ of N^d and that the value at each such point depends computationally on some or all of its immediate "predecessors" $\langle x, \ldots, y-1, \ldots, z\rangle$. When one includes the close relatives of mesh-dags we study in Section 4, the modeled class of computational situations expands even further.

2.3 The Internet-Computing Pebble Game

A variety of so-called *pebble games* on dags have been shown, over the course of several decades, to yield elegant formal analogues of a variety of problems related to scheduling the tasks/nodes of a computation-dag. The basic idea underlying such games is to use tokens (called pebbles) to model the progress of a computation on a dag: The placement or removal of pebbles of various types —which is constrained by the dependencies modeled by the dag's arcs—represents the changing (computational) status

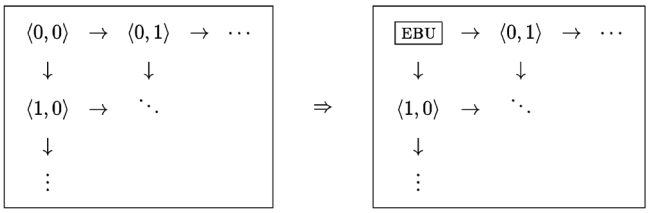


Fig. 2. Illustrating Rule 1.

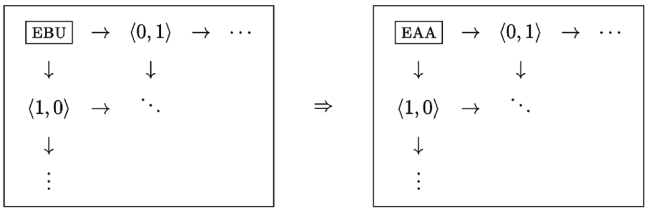


Fig. 3. Illustrating Rule 2.1.

of the tasks represented by the dag's nodes. Pebble games have been used to study problems as diverse as register allocation [22], [8], interprocessor communication in parallel computers [15], "out-of-core" memory accesses [14], and the bandwidth-minimization problem for sparse matrices [28]. Additionally, pebble games have been shown to model many complexity-theoretic problems perspicaciously [23]. The pebble game that we study here shares its basic structure with the "no recomputation allowed" pebble game of [28], but it differs from that game in the resource one strives to optimize. For brevity, we describe the Internet-Computing (I-C) Pebble Game within the "pull"-based context of WC; the reader can easily adapt our description to a "push"-based GC version of the Game.

2.3.1 The Rules of the I-C Pebble Game

The *Internet-Computing* (*I-C*, for short) *Pebble Game* on a dag $\mathcal{G}$ involves one player S, called the *Server*, and an indeterminate number of players $C_1, C_2, \ldots$, called the *Clients*. The Server has access to unlimited supplies of three types of pebbles: ELIGIBLE-BUT-UNALLOCATED (EBU, for short) pebbles, ELIGIBLE-AND-ALLOCATED (EAA, for short) pebbles, and EXECUTED (XEQ, for short) pebbles. The Game's moves are designed to reflect the successive stages in the "life-cycle" of a task/node in a computation-dag, from the time the task becomes eligible for execution (hence, receives an EBU pebble) through its actual execution (when it receives an XEQ pebble). We now present the rules of the Game, interspersed with illustrations of their application. The reader should note how the moves of the Game expose the danger of a play's being stalled indefinitely by dilatory Clients (the "gridlock" referred to in the Introduction).

The I-C Pebble Game:

Rule 1. At any step of the Game, S may place an EBU pebble on any unpebbled source node of $\mathcal{G}$.
/*Unexecuted source nodes are always eligible for execution, having no parents whose prior execution they depend on. The Game must begin with a move of this type*/ (see Fig. 2).

Rule 2. Say that Client C_i approaches S requesting a task. If C_i has previously been allocated a task that it has not completed, then C_i's request is ignored; otherwise, the following occurs.

1. If at least one node of $\mathcal{G}$ contains an EBU pebble, then S gives C_i the task corresponding to one such node and replaces that node's pebble by an EAA pebble (see Fig. 3).
2. If no node of $\mathcal{G}$ contains an EBU pebble, then C_i is told to withdraw its request, and this move is a no-op.

Rule 3. When a Client returns (the results from) a task/node, S replaces that task/node's EAA pebble by an XEQ pebble. S then places an EBU pebble on each unpebbled node of $\mathcal{G}$ all of whose parents contain XEQ pebbles (see Fig. 4).

Rule 4. S's goal is to allocate nodes in such a way that every node v of $\mathcal{G}$ *eventually* contains an XEQ pebble.
/*This modest goal is necessitated by the possibility that $\mathcal{G}$ is infinite.*/

2.3.2 The Quality of a Play of the Game

We strive to determine how to play the I-C Pebble Game in a way that maximizes the number of nodes that hold EBU pebbles at every moment—so that we maximize the chance that the Server always has a task to allocate when approached by a Client. With the aid of some simplifying assumptions (which, we argue below, are benign ones), we derive a formal version of this goal that is appropriate for large classes of computation-dags, including the mesh-like dags that we study here and the reduction-dags of various structures that we study in [29]. We hope that the analyses that uncover optimal scheduling strategies for these classes of computation-dags will form the basis for discovering and analyzing analogous scheduling strategies for other important classes. A primary consideration as we contemplated the simplifying assumptions that we present now is that *these assumptions allow a Server to focus solely on issues that are within its control, rather than depending on the Clients' (unpredictable) behavior.*

1. **Restrict the unpredictability of Clients.** Our first simplifying assumption builds on the fact that, by monitoring the status of Clients, an IC master can often restrict the Clients' unpredictability to the timing of, rather than the order of task completion.
   Simplifying assumption. *Tasks are executed in the same order as they are allocated.*

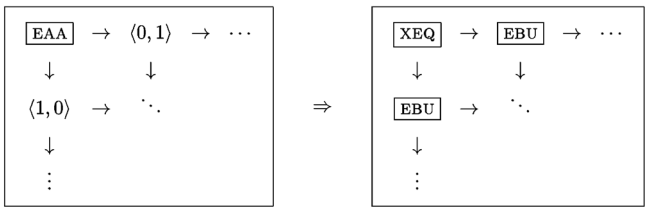


Fig. 4. Illustrating Rule 3.

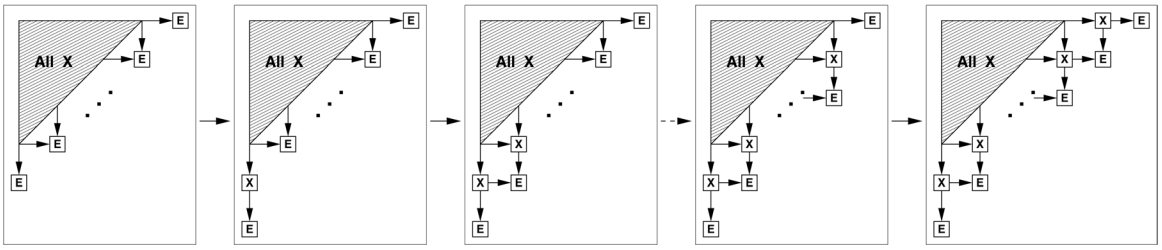


Fig. 5. Computing a typical diagonal level of $\mathcal{M}_2$. "X" denotes an EXECUTED node; "E" denotes an ELIGIBLE node.

Rationale. Without some restriction on the behavior of Clients, a malicious adversary (read: unfortunate behavior by Clients) could confute any attempt to produce EBU pebbles rapidly by executing allocated tasks in a pessimal order. As we see in the next sections, the proposed simplification suffices to afford the Server an avenue for maximizing the production rate of EBU pebbles.

Let us put this simplifying assumption in perspective.

a. While the assumption idealizes the world in which the Server allocates tasks, it is not fanciful: The desired order can be at least approximated in practice, using known techniques. To wit, IC master sites often monitor the status of Clients before matching a Client with an eligible task; see, e.g., [5], [19], [32], as well as the IC-enabling software developed by Entropia, Inc. (http://www.entropia.com). Such monitoring allows the master/Server to enhance the likelihood of, even if not to guarantee, the desired execution order.
b. While the assumption simplifies the chore of scheduling a computation-dag well (within the context of our eligible-task-producing goal), it decidedly does not trivialize the chore.

The preceding points should be viewed in the light of results such as those in Sections 3.1.1 and 3.1.2, which exhibit two equally intuitive scheduling strategies for mesh-dags, one of which is optimal and one pessimal within the context of our goal. On the one hand, these results indicate the importance of the monitoring mentioned in point 1. On the other hand, these results indicate that intuition will not suffice to develop good schedules.

2. **Simplify the repertoire of pebbles.** Our second simplification, which is enabled by our first one, allows us to focus on only two types of pebbles.

Simplification. *Ignore the distinction between EAA and XEQ pebbles, lumping both together henceforth as XEQ pebbles.*

Rationale. The number n of XEQ pebbles at a step of the I-C Pebble Game is determined by the structure of the target computation-dag and by the strategy the Server uses when playing the Game. In contrast, the breakdown of n into the relative numbers of EAA and XEQ pebbles reflects the (actual) computation rates and frequencies of approaches by Clients during that play—which is beyond the Server's control!

In the light of our simplifications, we enhance legibility by henceforth renaming EAA and XEQ pebbles collectively as EXECUTED pebbles and renaming EBU pebbles as ELIGIBLE pebbles. For simplicity, we also, henceforth, refer to a node that holds a pebble of type $T \in$ {ELIGIBLE, EXECUTED} as a "T node."

The preceding simplifications afford us the following conceptually simple, mathematically tractable formalization of our informal scheduling goal. For each step t of a play of the I-C Pebble Game on a dag $\mathcal{G}$, let $X^{(t)}$ denote the number of EXECUTED pebbles on $\mathcal{G}$'s nodes at step t and let $E^{(t)}$ denote the number of ELIGIBLE pebbles on $\mathcal{G}$'s nodes at step t.

We measure the quality of a play of the I-C Pebble Game on a dag $\mathcal{G}$ by the relative sizes of $E^{(t)}$ and $X^{(t)}$ at each step t of the play—the bigger $E^{(t)}$ is, for fixed $X^{(t)}$, the better. Our goal is an **I-C optimal** schedule in which, for all steps t, $E^{(t)}$ is as big as possible, given $X^{(t)}$.

3 (Near-)Optimal Schedules for Mesh-Dags

In this section, we exhibit schedules for mesh-dags of all finite dimensionalities that are either I-C optimal or within a constant factor thereof. In Section 3.1, we present an I-C optimal schedule for the two-dimensional mesh-dag $\mathcal{M}_2$. In Section 3.2, we present a schedule for each mesh-dag $\mathcal{M}_d$ of fixed dimensionality $d > 2$, that is within a constant factor of I-C optimal; the constant factor depends only on d. In all cases, the exhibited schedules allocate a mesh-dag $\mathcal{M}$'s tasks along $\mathcal{M}$'s diagonal levels. Interestingly, the diagonal-level allocation/execution[2] strategy for meshes is optimal under several optimization criteria, including, e.g., parallel execution time [17], [31].

3.1 I-C Optimal Schedules for Two-Dimensional Mesh-Dags

3.1.1 I-C Optimal Schedules

Theorem 1 (The Two-dimensional mesh-dag $\mathcal{M}_2$). ***1)*** *For any schedule that allocates nodes along successive diagonal levels of $\mathcal{M}_2$, $E^{(t)} = n$ whenever $X^{(t)}$ lies in the range*

$$\binom{n}{2} \leq X^{(t)} < \binom{n+1}{2}. \tag{2}$$

2) *For any schedule for $\mathcal{M}_2$, if $X^{(t)}$ lies in the range (2), then $E^{(t)}$ can be as large as n, but no larger.*

It follows that any diagonal-threading schedule for $\mathcal{M}_2$ is I-C optimal.

2. Since nodes are executed in order of allocation, we can use "execute" and "allocate" interchangeably.

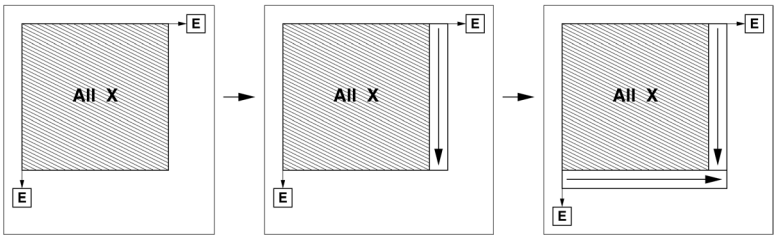


Fig. 6. A schedule for $\mathcal{M}_2$ that traverses *square* levels. "X" denotes an EXECUTED node; "E" denotes an ELIGIBLE node. The long arrows indicate sequences of node-executions.

Proof. 1) Let $\mathcal{S}$ be any schedule for $\mathcal{M}_2$ that executes ELIGIBLE nodes along successive diagonal levels; Fig. 5 depicts such a schedule, which proceeds *up* successive diagonals of $\mathcal{M}_2$. We claim that $\mathcal{S}$ achieves the advertised upper bound. To see this, note that precisely n nodes of $\mathcal{M}_2$ lie on the nth diagonal level $L^{(2)}_{n-1}$. These nodes are all ELIGIBLE as soon as all of their $\binom{n}{2}$ ancestors (which comprise all the nodes at lower diagonal levels of $\mathcal{M}_2$) have been EXECUTED. As we proceed from the position wherein the ELIGIBLE nodes comprise the entire level $L^{(2)}_{n-1}$ to the analogous position for level $L^{(2)}_n$, executing ELIGIBLE nodes as we go, we create one new ELIGIBLE node with each node execution, save the last, which creates two new ELIGIBLE nodes. It follows that, during the period when there are precisely n ELIGIBLE nodes under schedule $\mathcal{S}$, the number of EXECUTED nodes always has the form $\binom{n}{2} + k$ for some $k < n$.

2) A first lower bound on $X^{(t)}$ when $E^{(t)} = n$. Although we could argue a bit more simply if we considered only the case $d = 2$ here, with just a bit more effort, we establish the basis for the proofs of our lower bounds for mesh-dags of all finite dimensionalities. Our proofs build on the following fact, which is a direct consequence of the I-C Pebble Game's rules (hence, it is stated without proof).

Lemma 1. *If the node v of $\mathcal{M}_d$ is ELIGIBLE, then:*

- *every ancestor of v must be EXECUTED;*
- *no descendant of v can be either ELIGIBLE or EXECUTED.*

We restate the lemma in a bit more detail, focusing on an arbitrary dimensionality $d \geq 2$. For each $(d-1)$-tuple of nonnegative integers $\langle u_1, u_2, \ldots, u_{a-1}, u_{a+1}, \ldots, u_d \rangle$:

1. There is at most one nonnegative integer u_a such that the node $v \stackrel{\text{def}}{=} \langle u_1, u_2, \ldots, u_{a-1}, u_a, u_{a+1}, \ldots, u_d \rangle$ of $\mathcal{M}_d$ is ELIGIBLE.
2. If node v is ELIGIBLE, then:
   a. For all $u < u_a$, node
   $$\langle u_1, u_2, \ldots, u_{a-1}, u, u_{a+1}, \ldots, u_d \rangle$$
   of $\mathcal{M}_d$ is already EXECUTED.
   b. For all $u > u_a$, node
   $$\langle u_1, u_2, \ldots, u_{a-1}, u, u_{a+1}, \ldots, u_d \rangle$$
   of $\mathcal{M}_d$ is neither EXECUTED nor ELIGIBLE.

Restricting attention to the case $d = 2$, we find the following important consequences of Lemma 1.

- No two ELIGIBLE nodes of $\mathcal{M}_2$ reside in the same row or the same column.[3]
- Every row- and column-ancestor of each ELIGIBLE node of $\mathcal{M}_2$ has already been EXECUTED.

These conditions indicate that, at any instant when there are n ELIGIBLE nodes on $\mathcal{M}_2$, we must have n distinct rows of $\mathcal{M}_2$, each containing a distinct (perforce, nonnegative) number of EXECUTED nodes. An easy argument shows that the number of EXECUTED nodes is minimized in this case when the n ELIGIBLE nodes comprise level $L^{(2)}_{n-1}$ of $\mathcal{M}_2$. (Any other arrangement could be "compactified" without increasing the number of EXECUTED nodes.) When the ELIGIBLE nodes comprise level $L^{(2)}_{n-1}$, we have $\sum_{k=0}^{n-1} k = \binom{n}{2}$ EXECUTED nodes.

Completing the proof. The preceding lower bound on $X^{(t)}$ when $E^{(t)} = n$ tells us that, in order to have $n+1$ ELIGIBLE nodes on $\mathcal{M}_2$, we need at least $\binom{n+1}{2}$ EXECUTED nodes. The proof of part 1 of the theorem tells us that, as soon as we have $\binom{n}{2}$ EXECUTED nodes on $\mathcal{M}_2$, we can have n ELIGIBLE nodes. Combining these facts establishes part 2 of the theorem, hence verifies the I-C optimality of schedules that allocate $\mathcal{M}_2$'s nodes along successive diagonal levels. □

3.1.2 Some Perspective: I-C Pessimal Schedules

Theorem 1 singles out a specific strategy for executing (the nodes of) $\mathcal{M}_2$ as being the best in terms of maximizing the production rate of ELIGIBLE nodes. In fact, the prescribed diagonal-level schedules are *unboundedly better* in this respect than many other natural schedules for $\mathcal{M}_2$. We illustrate this point by contrasting the production rate of ELIGIBLE nodes exposed in the proof of Theorem 1 part 1 with the production rate of the "square-shell" schedule depicted schematically in Fig. 6. If the reader fleshes out this schematic depiction to a level of detail commensurate with that of Fig. 5, they will find that, under the "square-shell" schedule, no more than *three* nodes of $\mathcal{M}_2$ are ever simultaneously ELIGIBLE. (The number 3 here is, in fact, pessimal.) Thus, Theorem 1 points out a distinction in I-C quality that is far from just of academic interest.

3.2 Near-Optimal Schedules for Higher-Dimensional Mesh-Dags

We now extend the development of Section 3.1 to mesh-dags of dimensionality $d > 2$. In this case, we are able to

3. A *row* (respectively, *column*) of $\mathcal{M}$ is the induced subdag on a set of nodes of the form $\{i\} \times \mathrm{N}$ (respectively, $\mathrm{N} \times \{j\}$).

establish the I-C optimality of diagonal-following schedules only to within constant factors (that depend on d).

Theorem 2 (Mesh-dags of higher dimensionality). *For any fixed dimensionality $d > 2$, there exist positive constants $\kappa_1^{(d)}$ and $\kappa_2^{(d)}$, depending only on d, such that:* ***1)*** *For any schedule for $\mathcal{M}_d$ that schedules nodes along successive diagonal levels and lexicographically within levels, there are $\geq n$ ELIGIBLE nodes on $\mathcal{M}_d$ at time t of the schedule—i.e., $E^{(t)} \geq n$—whenever $X^{(t)} \geq (1/d)n^{d/(d-1)} + \kappa_1^{(d)} n$.* ***2)*** *When $X^{(t)} \leq \kappa_2^{(d)} n^{d/(d-1)}$, then $E^{(t)} \leq n$.*

It follows that any diagonal-threading schedule for $\mathcal{M}_d$ is within a constant factor of I-C optimal.

Proof. 1) Let $\mathcal{S}$ be any schedule for $\mathcal{M}_d$ that executes ELIGIBLE nodes along successive diagonal levels of $\mathcal{M}_d$, using a *lexicographic regimen*; i.e., $\mathcal{S}$ executes the ELIGIBLE nodes along each diagonal level of $\mathcal{M}_d$ in lexicographic order of their coordinates. We claim that $\mathcal{S}$ achieves the advertised upper bound. To see this, note that precisely $\binom{m+d-1}{d-1}$ nodes of $\mathcal{M}_d$ lie on the mth diagonal level $L_{m-1}^{(d)}$. These nodes are all ELIGIBLE whenever all of their $\sum_{j=0}^{m-1} \binom{j+d-1}{d-1} = \binom{m+d-1}{d}$ ancestors (which comprise all the nodes on lower diagonal levels of $\mathcal{M}_d$) are EXECUTED.

Focus on a moment t for which exactly n nodes of $\mathcal{M}_d$ are ELIGIBLE—i.e., $E^{(t)} = n$. Consider how many additional ELIGIBLE nodes must become EXECUTED under $\mathcal{S}$ before $n+1$ nodes of $\mathcal{M}_d$ are ELIGIBLE. Due to $\mathcal{S}$'s order of executing nodes, this delay is maximum when the n ELIGIBLE nodes at time t form a complete diagonal level of $\mathcal{M}_d$, say the mth diagonal level, $L_{m-1}^{(d)}$—in which case, $n = \binom{m+d-1}{d-1}$. As $\mathcal{S}$ proceeds from the position wherein the ELIGIBLE nodes comprise level $L_{m-1}^{(d)}$ to the first moment when the number of ELIGIBLE nodes on $\mathcal{M}_d$ exceeds n, $\mathcal{S}$ must execute the $\binom{m+d-2}{d-2}$ nodes from level $L_{m-1}^{(d)}$ that have 0 as their first coordinate. It follows that during the period when $\mathcal{M}_d$ has precisely n ELIGIBLE nodes, the number of EXECUTED nodes always has the form $\binom{m+d-1}{d} + k$, for some $k \leq \binom{m+d-2}{d-2}$. This reckoning shows that, in general, $\mathcal{S}$ will have produced $\binom{m+d-1}{d-1}$ ELIGIBLE nodes on $\mathcal{M}_d$ by the time that it has produced $\binom{m+d-1}{d} + \binom{m+d-2}{d-2}$ EXECUTED nodes. Rewriting this fact from the perspective of n, rather than m, yields the bound of part 1.

2) We proceed by induction on d, using the case $d = 2$ from Theorem 1 as our base case and operating with the following inductive hypothesis.

Inductive Hypothesis. For each dimensionality $\delta < d$, there exists a constant $c_\delta > 0$ such that, at any instant when $\mathcal{M}_\delta$ contains n ELIGIBLE nodes, it also contains at least $c_\delta n^{\delta/(\delta-1)}$ EXECUTED nodes.

To extend the induction, focus on any instant t when $\mathcal{M}_d$ contains n ELIGIBLE nodes—i.e., $E^{(t)} = n$. Let m be the smallest integer such that the diagonal node $\langle m, m, \ldots, m\rangle$ of $\mathcal{M}_d$ is *neither* ELIGIBLE nor EXECUTED at time t. Let $X^{(t)}(n)$ denote the number of EXECUTED nodes on $\mathcal{M}_d$ at time t, expressed as a function of n. We distinguish two cases.

Case 1: $\underline{m \geq n^{1/(d-1)}}$. This case can be viewed as saying that we are *really* exploiting the full d-dimensionality of $\mathcal{M}_d$, by quickly filling up many diagonal levels. We achieve our goal (of extending the induction) quickly in this case.

$$When\ m \geq n^{1/(d-1)},\ we\ have\ X^{(t)}(n) \geq n^{d/(d-1)} - 1.$$

This is verified as follows: Because node $\langle m, m, \ldots, m\rangle$ is the first diagonal node that is neither ELIGIBLE nor EXECUTED at time t, we know, by Lemma 1, that every ancestor of node $\langle m-1, m-1, \ldots, m-1\rangle$ is EXECUTED by time t. Since there are $m^d - 1$ such ancestors, we have

$$X^{(t)}(n) \ \geq\ m^d - 1 \ \geq\ n^{d/(d-1)} - 1.$$

Case 2: $\underline{m < n^{1/(d-1)}}$. In this case, since we are not quickly filling up many diagonal levels of $\mathcal{M}_d$, we are not exploiting the dag's full d-dimensionality. This means, intuitively, that we are essentially scheduling within a lower-dimensional mesh-dag—which means that we are producing EXECUTED nodes at a "fast" rate. Formalizing this intuition requires a bit more effort than in Case 1.

By Lemma 1, no descendant of node $\langle m, m, \ldots, m\rangle$ can be either ELIGIBLE or EXECUTED at time t. These descendants comprise the "positive orthant of $\mathcal{M}_d$ with node $\langle m, m, \ldots, m\rangle$ at its origin," i.e., the set of nodes $\{\langle m_1, m_2, \ldots, m_d\rangle \mid \text{each } m_i \geq m\}$. This means that all n ELIGIBLE nodes on $\mathcal{M}_d$ at time t must reside in one (or more) of the dm copies of $\mathcal{M}_{d-1}$ that are obtained by the various ways of specifying $d-1$ of $\mathcal{M}_d$'s d coordinate axes plus a position $p \in \{0, 1, \ldots, m-1\}$ along the unspecified axis. Let us denote the m copies along the kth axis, where $k \in \{1, 2, \ldots, d\}$, in some (arbitrary) way, as

$$\mathcal{M}_{d-1}^{(k,0)}, \mathcal{M}_{d-1}^{(k,1)}, \ldots, \mathcal{M}_{d-1}^{(k,m-1)}.$$

(Note that the union of these copies comprises $\mathcal{M}_d$ minus all descendants of node $\langle m, m, \ldots, m\rangle$.) Now, many of $\mathcal{M}_d$'s n ELIGIBLE nodes at time t actually reside in more than one of the mesh-dags $\mathcal{M}_{d-1}^{(k,j)}$. If we assign each ELIGIBLE node in some (arbitrary) way to one of the dags $\mathcal{M}_{d-1}^{(k,j)}$ that it resides in, then we effect a partition of the integer n into dm nonnegative parts: $n = \sum_{k=1}^{d} \sum_{j=0}^{m-1} n_{k,j}$. This double sum denotes the assignment of $n_{k,j}$ of the n ELIGIBLE nodes to copy $\mathcal{M}_{d-1}^{(k,j)}$ of $\mathcal{M}_{d-1}$.

Now, let us aggregate the dm copies of $\mathcal{M}_{d-1}$ into d groups, according to their "unspecified" axis: The kth group comprises mesh-dags $\mathcal{M}_{d-1}^{(k,0)}, \mathcal{M}_{d-1}^{(k,1)}, \ldots, \mathcal{M}_{d-1}^{(k,m-1)}$. Note that the mesh-dags in each group are pairwise disjoint: No two share any node. It follows that at least one of these groups—call it group $k^\star$—contains at least n/d of the n ELIGIBLE nodes at time t. Let us restrict attention now to the copies of $\mathcal{M}_{d-1}$ in group $k^\star$. Because of the internode dependencies within each mesh-dag $\mathcal{M}_{d-1}^{(k^\star,i)}$, our inductive hypothesis implies that the $n_{k^\star,i}$ ELIGIBLE nodes residing in $\mathcal{M}_{d-1}^{(k^\star,i)}$ betoken the existence of at least $c_{d-1} n_{k^\star,i}^{(d-1)/(d-2)}$ EXECUTED nodes in $\mathcal{M}_{d-1}^{(k^\star,i)}$—hence, also in $\mathcal{M}_d$—at time t. It follows that the total number of EXECUTED nodes on $\mathcal{M}_d$ at time t is no smaller than

$$\xi_t \ =\ c_{d-1} \sum_{i=0}^{m-1} n_{k^\star,i}^{(d-1)/(d-2)}.$$

A standard convexity argument demonstrates that the sum ξ_t is bounded below by the sum obtained by setting all of the $n_{k^\star,i}$ to their average value, which is no smaller than $n/(dm)$. (Recall that the m numbers $n_{k^\star,i}$ sum to *at least* n/d.) This gives us the following lower bound on ξ_t:

$$\begin{aligned}\xi_t &\geq c_{d-1} m \Big(\frac{n}{dm}\Big)^{(d-1)/(d-2)} \\ &= [c_{d-1} \cdot d^{-(d-1)/(d-2)}] \cdot n^{(d-1)/(d-2)} \cdot m^{-1/(d-2)} \\ &= c_d \cdot n^{(d-1)/(d-2)} \cdot m^{-1/(d-2)}.\end{aligned} \tag{3}$$

When $m < n^{1/(d-1)}$, we achieve the claimed lower bound on $X^{(t)}(n)$ via substitution in (3):

$$\begin{aligned}X^{(t)}(n) \geq\ \xi_t\ &\geq\ c_d \cdot n^{(d-1)/(d-2)-1/((d-1)(d-2))} \\ &\geq\ c_d \cdot n^{d/(d-1)}.\end{aligned}$$

This completes the proof. □

4 Optimal Schedules for Mesh-Like Monoid Dags

Although Theorem 1 refers narrowly to the mesh-dag $\mathcal{M}_2$, its scheduling prescription extends to a large class of close relatives of $\mathcal{M}_2$. This section is devoted to identifying one family of such close relatives whose structures are induced by certain families of abelian monoids. From an algebraic perspective, these dags are monoid-theoretic analogues of the familiar group-induced Cayley graphs. From a graph-theoretic perspective, these dags can be viewed as somewhat distorted copies of $\mathcal{M}_2$ that are enhanced with regularly placed "shortcut" arcs. We show that the mesh-like dags in the identified family share those aspects of the structure of $\mathcal{M}_2$ that underlie Theorem 1.

4.1 Mesh-Like Cayley Dags of Monoids

A large literature that goes back decades amply illustrates the benefits of exploiting the algebraic, as well as the combinatorial, structure of graphs associated with a wide range of computational problems. This algebraic structure is manifest in the *Cayley digraphs* of monoids. A *finitely generated monoid* is an algebraic system $M = M(\Gamma, \times, \mathbf{1})$, where:

- $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ is a finite set of *generators* for M;
- $\times$ is a binary associative *multiplication* for M;
- $\mathbf{1}$ is a (two-sided) multiplicative identity: For all $\xi \in M$, we have $\xi \times \mathbf{1} = \mathbf{1} \times \xi = \xi$.

The elements of M comprise all finite products of instances of elements $\Gamma \cup \{\mathbf{1}\}$.[4] The Cayley digraph $\mathcal{G}(M)$ associated with $M = M(\Gamma, \times, \mathbf{1})$ has node-set $N_{\mathcal{G}(M)} = M$ and arc-set $A_{\mathcal{G}(M)} = \{(\xi \to \xi \times \gamma_i) \mid \xi \in M,\ \gamma_i \in \Gamma\}$.

The computation-modeling literature focuses mainly on Cayley digraphs of *groups*; see, e.g., [2], [9]. However, major insights can emerge from studying Cayley digraphs of weaker algebraic structures; see, e.g., the "group-action graphs" of [1], [3] and the monoid-graphs of [24], [25]. Our study of Cayley computation-dags is inspired by the last two sources.

The Cayley digraphs of many monoids are *cyclic*. This is obvious for groups because of inverses; it is, less obviously, true for all *finite* monoids because each arc leads to some node and there are only $|M|$ nodes. The *acyclicity* of computation-dags is, thus, possible only for Cayley digraphs of *infinite* monoids; it manifests itself as a natural restriction on the monoids underlying the digraphs. The reader can easily verify the following:

Lemma 2. *The Cayley digraph $\mathcal{G}(M)$ is acyclic if and only if the monoid M does not contain elements ξ, η, with $\eta \neq \mathbf{1}$, such that $\xi = \xi \times \eta$.*

We say that a monoid M with generator-set $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ is *k-dimensional*, for some $k \leq n$, if there exists a k-element subset $\Gamma' \subseteq \Gamma$ such that each $\xi \in M$ is a product of (instances of) generators from Γ'. In this situation, the $n - k$ generators from $\Gamma - \Gamma'$ supply "shortcuts" in the Cayley digraph $\mathcal{G}(M)$. We call Γ' a *k-basis* for M.

Mesh-Like Monoid Dags. We have yet to discuss what makes a computation-dag "mesh-like." After close inspection of the analyses in Section 3, we have decided to insist on the following three properties:

A *Mesh-Like Monoid dag* (*MLM-dag*, for short) $\mathcal{M}$ is the Cayley digraph of a *finitely generated, infinite* monoid M, such that:

- The monoid M is *abelian*, meaning that the following *commutative* identities hold:

$$\gamma_1 \times \gamma_2\ =\ \gamma_2 \times \gamma_1\ \text{ for all generators } \gamma_1, \gamma_2 \text{ of } M. \tag{4}$$

- $\mathcal{M}$ is *acylic*; i.e., its underlying monoid M satisfies Lemma 2.
- $\mathcal{M}$ has *infinite width*, meaning that there exist successful plays of the I-C Pebble Game on $\mathcal{M}$ in which, for every integer $n > 0$, there is a step at which at least n nodes of $\mathcal{M}$ contain ELIGIBLE pebbles.[5]

An MLM-dag $\mathcal{M}$ is *two-dimensional* (is a *2MLM-dag*) if its underlying monoid is two-dimensional.

Mesh-dags are the most restrictive MLM-dags in the sense that their underlying monoids are (algebraically) *free:* The identities (4) are the *only* nontrivial relations among their elements. In detail, the mesh-dag $\mathcal{M}_d$ is the MLM-dag based on the monoid M_d that has:

- d generators $g_1, \ldots, g_d$, each g_i being a d-tuple of the form $g_i = \langle 0, \ldots, 0, 1, 0, \ldots, 0\rangle$ where the sole 1 is in position i;
- coordinatewise addition of d-tuples of integers as its "multiplication;"
- the identity $\mathbf{1} = \langle 0, 0, \ldots, 0\rangle$.

4.2 Delimiting the Structure of 2MLM-Dags

We now show that every 2MLM-dag is actually a copy of $\mathcal{M}_2$ augmented with a structurally constrained set of "shortcut" arcs. The inheritance of $\mathcal{M}_2$'s structure means that 2MLM-dags cannot be scheduled with greater

4. As is common, "M" ambiguously denotes both the monoid $M(\Gamma, \times, \mathbf{1})$ and the set of its elements; context will clarify our intention.

5. We see from Section 3.1.2 that not every successful play of the Game need satisfy this condition.

I-C quality than can $\mathcal{M}_2$; the constraints on the "shortcut" arcs means that they can be scheduled with the same I-C quality. Let us focus, henceforth, on an arbitrary 2MLM-dag $\mathcal{M}$ whose underlying monoid M has a 2-basis $\{\lambda, \rho\}$.

Theorem 3. *The basis generators λ, ρ of the monoid M underlying $\mathcal{M}$ do not obey any equation of the form*[6]

$$\lambda^a \times \rho^b \;=\; \lambda^c \times \rho^d \tag{5}$$

for nonnegative integers a, b, c, d, except for the trivial equation wherein $a = c$ and $b = d$.

Proof. For the sake of contradiction, assume that an equation of the form (5) holds for some nonnegative integers a, b, c, d, where either $a \neq c$ or $b \neq d$ (or both). We expose the absurdity of this assumption by repeatedly invoking the following property of monoids.

Fact 1. If $\xi = \eta$ for $\xi, \eta \in M$, then, for all $\zeta \in M$, we have $\xi \times \zeta = \eta \times \zeta$.

We isolate, in turn, three cases that exhaust the ways in which $a \neq c$ or $b \neq d$ (or both).

Case 1: $[a < c \text{ and } b \leq d]$ or $[a \leq c \text{ and } b < d]$. In this case, (5) can be rewritten as:

$$\begin{aligned} \lambda^a \times \rho^b &= \lambda^{a+(c-a)} \times \rho^{b+(d-b)} \\ &= (\lambda^a \times \rho^b) \times (\lambda^{c-a} \times \rho^{d-b}). \end{aligned} \tag{6}$$

Since $(\lambda^{c-a} \times \rho^{d-b}) \in M - \{\mathbf{1}\}$, Lemma 3 indicates that (6) contradicts $\mathcal{M}$'s alleged acyclicity. We conclude that (5) cannot hold with parameters that satisfy this case.

Case 2: $a + b = c + d$. This condition says that nodes $\lambda^a \times \rho^b$ and $\lambda^c \times \rho^d$ lie on the same "diagonal level" $\ell \stackrel{\text{def}}{=} a + b = c + d$ of $\mathcal{M}$. Therefore, we can rewrite (5) as:

$$\lambda^a \times \rho^{l-a} \;=\; \lambda^c \times \rho^{l-c}. \tag{7}$$

Say, with no loss of generality, that $a < c$ and let $e = c - a$.

Let us use the relations that hold within the monoid M —specifically, (4) and (7)—to determine the population of level l of $\mathcal{M}$.

- There are a or fewer nodes "to the left of" node $\lambda^a \times \rho^{l-a}$; all have the form $\lambda^x \times \rho^{l-x}$, where $0 \leq x < a$.
- There are $l - c$ or fewer nodes "to the right" of node $\lambda^c \times \rho^{l-c}$; all have the form $\lambda^x \times \rho^{l-x}$, where $c < x \leq l$.
- There is node $\lambda^a \times \rho^{l-a} = \lambda^c \times \rho^{l-c}$.
- There are $e - 1$ or fewer nodes "between" node $\lambda^a \times \rho^{l-a}$ and node $\lambda^c \times \rho^{l-c}$; all have the form $\lambda^x \times \rho^{l-x}$, where $a < x < c$.

In toto, level l of $\mathcal{M}$ thus contains no more than l nodes. In the same way, let us determine the population of level $l + e$ of $\mathcal{M}$.

- There are a or fewer nodes "to the left of" node $\lambda^a \times \rho^{l+e-a}$; all have the form $\lambda^x \times \rho^{l+e-x}$, where $0 \leq x < a$.

6. As usual, for any $\xi \in M$, we have $\xi^0 = 1$ and $\xi^{i+1} = \xi \times \xi^i$.

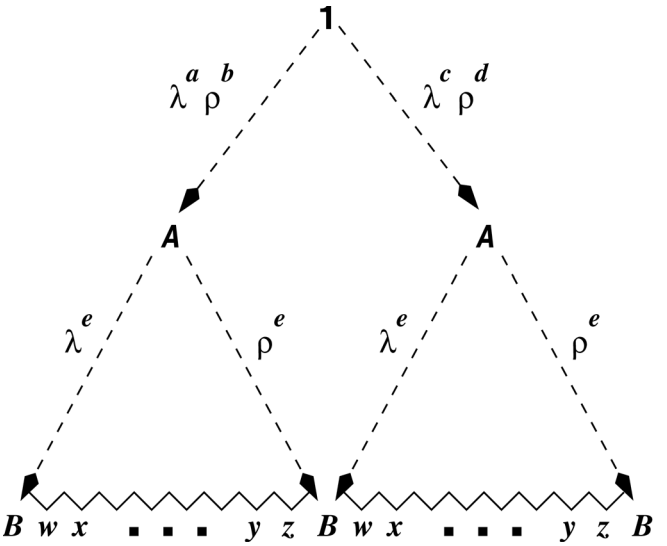


Fig. 7. Lemma 3, pictorially. Node A is $\lambda^a\rho^b = \lambda^c\rho^d$; Node B is $\lambda^a\rho^{b+e} = (\lambda^c \times \rho^{d+e} = \lambda^{a+e}\rho^b) = \lambda^{c+e}\rho^d$.

- There are $l - c$ or fewer nodes "to the right" of node $\lambda^{c+e} \times \rho^{l-c}$; all have the form $\lambda^{x+e} \times \rho^{l-x}$, where $c < x \leq l$.
- There is node

$$\begin{aligned} \lambda^a \times \rho^{l+e-a} &= (\lambda^{a+e} \times \rho^{l-a} = \lambda^c \times \rho^{l+e-c}) \\ &= \lambda^{c+e} \times \rho^{l-c}. \end{aligned}$$

  The equations $\lambda^a \times \rho^{l+e-a} = \lambda^c \times \rho^{l+e-c}$ and $\lambda^{a+e} \times \rho^{l-a} = \lambda^{c+e} \times \rho^{l-c}$ both follow from Fact 1. The parenthesized equation $\lambda^{a+e} \times \rho^{l-a} = \lambda^c \times \rho^{l+e-c}$ follows by direct calculation.
- There are $e - 1$ or fewer nodes "between" node $\lambda^a \times \rho^{l+e-a}$ and node $\lambda^{a+e} \times \rho^{l-a}$; all have the form $\lambda^x \times \rho^{l-x}$, where $a < x < a + e$.

  There are $e - 1$ or fewer nodes "between" node $\lambda^c \times \rho^{l+e-c}$ and node $\lambda^{c+e} \times \rho^{l-c}$; all have the form $\lambda^x \times \rho^{l-x}$, where $c < x < c + e$.

  However, the preceding two sets of nodes are identical, as we show now.

Lemma 3. *On level $l + e$ of $\mathcal{M}$, the set of nodes that lie "between" nodes $\lambda^a \times \rho^{l+e-a}$ and $\lambda^{a+e} \times \rho^{l-a}$ is identical to the set of nodes that lie "between" nodes $\lambda^c \times \rho^{l+e-c}$ and $\lambda^{c+e} \times \rho^{l-c}$.*

Verification. For each $j \in \{1, 2, \ldots, e-1\}$:

$$\begin{aligned} \lambda^{a+j} \times \rho^{l+e-(a+j)} &= (\lambda^a \times \rho^{l-a}) \times (\lambda^j \times \rho^{l-j}) \\ &= (\lambda^c \times \rho^{l-c}) \times (\lambda^j \times \rho^{l-j}) \\ &= \lambda^{c+j} \times \rho^{l+e-(c+j)} \end{aligned} \tag{8}$$

The $e - 1$ instances of (8) prove the lemma. Fig. 7 illustrates the argument pictorially.

Thus, in toto, level $l + e$ of $\mathcal{M}$ contains no more than l nodes.

Aside. Reasoning mirroring that in system (8) shows that the set of nodes of $\mathcal{M}$ that populate the entire "triangle"

subtended by nodes $\lambda^a \times \rho^{l-a}$, $\lambda^a \times \rho^{l+e-a}$, and $\lambda^{a+e} \times \rho^{l-a}$ is identical to the set of nodes of $\mathcal{M}$ that populate the entire "triangle" subtended by nodes $\lambda^c \times \rho^{l-c}$, $\lambda^c \times \rho^{l+e-c}$, and $\lambda^{c+e} \times \rho^{l-c}$.

One easily extends the reasoning that leads to system (8) to show that every level $l + me$ of $\mathcal{M}$, where $m = 0, 1, 2, \ldots$, contains no more than l nodes. Since every node of $\mathcal{M}$ has the form $\lambda^x \rho^y$, it follows that $\mathcal{M}$ has finite width (in the sense of Section 4.1). To wit:

Lemma 4. *Say that there exist fixed constants s and $k(s)$ such that, for all but finitely many lines of slope s in the λ-ρ plane, a line segment of length $k(s)$ contains all of the nodes of the dag $\mathcal{M}$ along that line. Then, $\mathcal{M}$ has finite width.*

Verification. Invoking the reasoning that proves the lower bound in part 2 of Theorem 1, we note that two nodes of $\mathcal{M}$ can simultaneously be ELIGIBLE only if they reside in distinct rows and columns of the λ-ρ plane. Under the assumption of the lemma, no more than $k(s)$ such independent nodes can coexist.

Thus, the assumption of Case 2 contradicts the fact that $\mathcal{M}$, being a 2MLM-dag, has infinite width. We conclude that (5) cannot hold with parameters that satisfy this case.

Case 3: $a + b \neq c + d$. Say, with no loss of generality, that $a + b < c + d$ and that $a < c$ (so, by Lemma 2, $b > d$). Let $e = c - a$ and $f = b - d$. Since this case can be viewed as a "skewed" version of Case 2, we shall be a bit sketchier than in that case.

In addition to the motivating (5), we now have the triple equation

$$\lambda^a \rho^{b+f} = (\lambda^{a+e} \rho^b = \lambda^c \rho^{d+f}) = \lambda^{c+e} \rho^d.$$

The equations $\lambda^a \rho^{b+f} = \lambda^c \rho^{d+f}$ and $\lambda^{a+e} \rho^b = \lambda^{c+e} \rho^d$ both follow from Fact 1. The parenthesized equation $\lambda^{a+e} \rho^b = \lambda^c \rho^{d+f}$ follows by direct calculation.

Using an argument that mirrors (8), one shows the following by direct calculation.

Lemma 5. *The set of nodes of $\mathcal{M}$ that lie along the line between nodes $\lambda^{a+e} \rho^b$ and $\lambda^a \rho^{b+f}$ is identical to the set of nodes of $\mathcal{M}$ that lie along the line between nodes $\lambda^{c+e} \rho^d$ and $\lambda^c \rho^{d+f}$.*

Aside. One can strengthen the assertion of Lemma 5 to the two "triangles" subtended, respectively, by the nodes $\{\lambda^a \rho^b, \lambda^{a+e} \rho^b, \lambda^a \rho^{b+f}\}$ and the nodes $\{\lambda^c \rho^d, \lambda^{c+e} \rho^d, \lambda^c \rho^{d+f}\}$.

A similar direct calculation reveals the following:

Lemma 6. *The set of nodes of $\mathcal{M}$ that lie along the line between nodes $\lambda^a \rho^b$ and $\lambda^c \rho^d$ is equinumerous to the set that lie along the line between nodes $\lambda^{a+e} \rho^b$ and $\lambda^a \rho^{b+f}$.*

Lemmas 5 and 6 easily imply that

Lemma 7. *For all integers $m \geq 0$, the sets of nodes of $\mathcal{M}$ that lie along the line between nodes $\lambda^{a+me} \rho^b$ and $\lambda^a \rho^{b+mf}$ have equal, finite cardinality.*

In the presence of Lemma 4, Lemma 7 implies that, in Case 3, too, the dag $\mathcal{M}$ has finite width, contradicting its being a 2MLM-dag. We conclude that (5) cannot hold with parameters that satisfy this case.

As Cases 1, 2, and 3 exhaust the situations that support (5), the theorem is proven. □

4.3 Optimal Schedules for 2MLM-Dags

Theorem 4. *Let $\mathcal{M}$ be a 2MLM-dag. Theorem 1 holds with every instance of "$\mathcal{M}_2$" replaced by "$\mathcal{M}$."*

Proof. Theorem 3 tells us that the nodes of $\mathcal{M}$ comprise all lattice points of the nonnegative λ-ρ quadrant; i.e., $N_{\mathcal{M}} = \{\lambda^a \times \rho^b \mid a, b = 0, 1, \ldots\}$. Therefore, we can repeat the lower-bound proof of Theorem 1 verbatim for $\mathcal{M}$, with only the changes in notation necessitated by the shift from "$\mathcal{M}_2$" to "$\mathcal{M}$."

Let γ be any generator of M. By hypothesis, γ must be a nonidentity[7] element of the monoid $M(\{\lambda, \rho\}, \times, \mathbf{1})$; i.e., $\gamma = \lambda^a \times \rho^b$, for some a, b such that $a + b > 0$. Now, if node $\lambda^x \times \rho^y$ of $\mathcal{M}$ is ELIGIBLE at some step t of the I-C Pebble Game, then all of its ancestors $\{\lambda^u \times \rho^v \mid [u < x] \text{ and } [v < y]\}$ must be EXECUTED at step t. Thus, the ELIGIBLE status of node $\lambda^x \times \rho^y$ is unaffected by the presence or absence of an arc $(\lambda^{x-a} \times \rho^{y-b} \rightarrow \lambda^x \times \rho^y)$. (This arc would betoken the fact that $\lambda^x \times \rho^y = \lambda^{x-a} \times \rho^{y-b} \times \gamma$ in M.) It follows that node $\lambda^x \times \rho^y$ is ELIGIBLE at step t of the I-C Pebble Game on $\mathcal{M}$ if and only if node $\langle x, y \rangle$ is ELIGIBLE at step t of the I-C Pebble Game on $\mathcal{M}_2$. The upper bound of the theorem thus follows from the upper-bound proof of Theorem 1, with only the changes in notation necessitated by the shift from mesh-dags to monoid-dags. □

5 Conclusions

We have formulated a variant of the classical pebble game on dags that models the process of executing a computation-dag within an Internet-based computing environment (Section 2.3.1). Within this model, we have proposed a formalism, I-C quality, for assessing the relative qualities of individual executions of a broad range of computation-dags, including mesh-like ones (Section 2.3.2). We have then identified strategies that are exactly I-C optimal for both two-dimensional mesh-dags (Section 3.1) and their monoid-theoretic kin, 2MLM-dags (Section 4), and are within constant factors of I-C optimal for all mesh-dags (Section 3). We have also noted that I-C optimal schedules for mesh-dags are unboundedly better than I-C pessimal ones (Section 3.1.2).

Many inviting challenges remain in this research area. The arguments that we have used here to establish the (near) I-C optimality of schedules rely extensively on the specific structures of the dags being scheduled. This specificity is true also for the I-C optimal schedules identified in this paper's sequel [29]. Yet, perusing the analyses of the schedules, one has the feeling that there are "big," as-yet unidentified, principles that are ensuring the

7. When $\gamma = \mathbf{1}$, the associated digraph $\mathcal{M}$ has self-loops, hence is not acyclic.

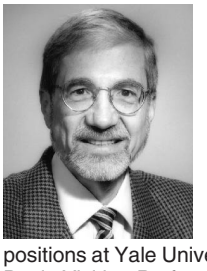
I-C quality of the schedules. Seeking such principles is an enticing and potentially significant challenge.

ACKNOWLEDGMENTS

This research was supported in part by US National Science Foundation (NSF) Grant CCR-00-73401. The author is grateful to Fran Berman for suggesting the scheduling problem studied herein and to Micah Adler, Henri Casanova, Matt Yurkewych, and the anonymous referees for many helpful comments and suggestions. A portion of this paper appeared in the *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '03)* as [27].

REFERENCES

[1] F.S. Annexstein, M. Baumslag, and A.L. Rosenberg, "Group Action Graphs and Parallel Architectures," *SIAM J. Computing,* vol. 19, pp. 544-569, 1990.
[2] B.W. Arden and K.W. Tang, "Representations and Routing of Cayley Graphs," *IEEE Trans. Comm.,* vol. 39, pp. 1533-1537, 1991.
[3] M. Baumslag and A.L. Rosenberg, "Processor-Time Tradeoffs for Cayley Graph Interconnection Networks," *Proc. Sixth Distributed Memory Computing Conf.,* pp. 630-636, 1991.
[4] A.R. Butt, S. Adabala, N.H. Kapadia, R. Figueiredo, and J.A.B. Fortes, "Fine-Grain Access Control for Securing Shared Resources in Computational Grids," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS '02),* 2002.
[5] R. Buyya, D. Abramson, and J. Giddy, "A Case for Economy Grid Architecture for Service Oriented Grid Computing," *Proc. 10th Heterogeneous Computing Workshop,* 2001.
[6] H. Casanova, *Distributed Computing Research Issues in Grid Computing,* typescript, Univ. of California, San Diego, 2002.
[7] W. Cirne and K. Marzullo, "The Computational Co-Op: Gathering Clusters into a Metacomputer," *Proc. 13th Int'l Parallel Processing Symp.,* pp. 160-166, 1999.
[8] S.A. Cook, "An Observation on Time-Storage Tradeoff," *J. Computer and System Sciences,* vol. 9, pp. 308-316, 1974.
[9] V.V. Dimakopoulos and N.J. Dimopoulos, "Optimal Total Exchange in Cayley Graphs," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, pp. 1162-1168, 2001.
[10] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure.* San Francisco: Morgan-Kaufmann, 1999.
[11] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. Supercomputer Applications,* 2001.
[12] A. Gerasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling Dags on Multiprocessors," *J. Parallel and Distributed Computing,* vol. 16, pp. 276-291, 1992.
[13] L. He, Z. Han, H. Jin, L. Pan, and S. Li, "DAG-Based Parallel Real Time Task Scheduling Algorithm on a Cluster," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '2000),* pp. 437-443, 2000.
[14] J.-W. Hong and H.T. Kung, "I/O Complexity: The Red-Blue Pebble Game," *Proc. 13th ACM Symp. Theory of Computing,* pp. 326-333, 1981.
[15] J.E. Hopcroft, W. Paul, and L.G. Valiant, "On Time versus Space," *J. ACM,* vol. 24, pp. 332-337, 1977.
[16] "The Intel Philanthropic Peer-to-Peer program,"www.intel.com/cure, 2003.
[17] R.M. Karp, R.E. Miller, and S. Winograd, "The Organization of Computations for Uniform Recurrence Equations," *J. ACM,* vol. 4, pp. 563-590, 1967.
[18] R.M. Karp, A. Sahay, E. Santos, and K.E. Schauser, "Optimal Broadcast and Summation in the logP Model," *Proc. Fifth ACM Symp. Parallel Algorithms and Architectures,* pp. 142-153, 1993.
[19] D. Kondo, H. Casanova, E. Wing, and F. Berman, "Models and Scheduling Guidelines for Global Computing Applications," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS '02),* 2002.
[20] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, "SETI@home: Massively Distributed Computing for SETI," *Computing in Science and Eng.,* 2000.
[21] *The Olson Laboratory Fight AIDS@Home Project,* www.fightaidsathome.org, 2003.
[22] M.S. Paterson and C.E. Hewitt, "Comparative Schematology," *Proc. Project MAC Conf. Concurrent Systems and Parallel Computation,* pp. 119-127, 1970.
[23] N.J. Pippenger, "Pebbling," *Proc. Fifth IBM Symp. Math. Foundations of Computer Science,* 1980.
[24] A.L. Rosenberg, "Data Graphs and Addressing Schemes," *J. Computing Systems Sciences,* vol. 5, pp. 193-238, 1971.
[25] A.L. Rosenberg, "Addressable Data Graphs," *J. ACM,* vol. 19, pp. 309-340, 1972.
[26] A.L. Rosenberg, "Accountable Web-Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, pp. 97-106, 2003.
[27] A.L. Rosenberg, "On Scheduling Collaborative Computations on the Internet: Mesh-Dags and Their Close Relatives," *Int'l Parallel and Distributed Processing Symp. (IPDPS '03),* 2003.
[28] A.L. Rosenberg and I.H. Sudborough, "Bandwidth and Pebbling," *Computing,* vol. 31, pp. 115-139, 1983.
[29] A.L. Rosenberg and M. Yurkewych, "Guidelines for Scheduling Some Common Computation-Dags for Internet-Based Computing,," submitted for publication, 2004.
[30] *The RSA Factoring by Web Project,* http://www.npac.syr.edu/factoring (with foreword by A. Lenstra). Northeast Parallel Architecture Center, 2003.
[31] W. Shang and J. Fortes, "Time-Optimal Linear Schedules for Algorithms with Uniform Dependencies," *IEEE Trans. Computers,* vol. 40, pp. 723-742, 1991.
[32] X.-H. Sun and M. Wu, "GHS: A Performance Prediction and Task Scheduling System for Grid Computing," *Proc. IEEE Int'l Parallel and Distributed Processing Symp.,* 2003.
[33] C. Weth, U. Kraus, J. Freuer, M. Ruder, R. Dannecker, P. Schneider, M. Konold, and H. Ruder, *XPulsar@home-Schools Help Scientists,* typescript, Univ. of Tübingen, 2000.
[34] S.W. White and D.C. Torney, "Use of a Workstation Cluster for the Physical Mapping of Chromosomes," *SIAM NEWS,* pp. 14-17, Mar. 1993.



Arnold L. Rosenberg is a Distinguished University Professor of Computer Science at the University of Massachusetts (UMass) Amherst, where he codirects the Theoretical Aspects of Parallel and Distributed Systems (TAPADS) Research Laboratory. Prior to joining UMass, he was a professor of computer science at Duke University from 1981 to 1986, and a research staff member at the IBM T.J. Watson Research Center from 1965 to 1981. He has held visiting positions at Yale University and the University of Toronto; he was a Lady Davis Visiting Professor at the Technion (Israel Institute of Technology) in 1994, and a Fulbright Research Scholar at the University of Paris-South in 2000. Dr. Rosenberg's research focuses on developing algorithmic models and techniques to deal with the new modalities of "collaborative computing" that result from emerging technologies. He is the author or coauthor of more than 145 technical papers on these and other topics in theoretical computer science and discrete mathematics and is the coauthor of the book *Graph Separators, with Applications*. He is a fellow of the ACM, a fellow of the IEEE, and a Golden Core member of the IEEE Computer Society.