Applying IC-Scheduling Theory to Familiar Classes of Computations

Gennaro Cordasco*, Grzegorz Malewicz†, Arnold L. Rosenberg#

*Univ. di Salerno
Dipt. di Informatica e Applicazioni
Fisciano 84084, ITALY
cordasco@dia.unisa.it

†Google, Inc.
Dept. of Engineering
Mountain View, CA 94043, USA
malewicz@google.com

#Univ. of Massachusetts
Dept. of Computer Science
Amherst, MA 01003, USA
rsnbrg@cs.umass.edu

Abstract

Earlier work has developed the underpinnings of IC-Scheduling theory*, an algorithmic framework for scheduling computations having intertask dependencies for Internet-based computing (IC). The Theory aims to produce schedules that render tasks eligible for execution at the maximum possible rate, so as to: (a) utilize remote clients' computational resources well, by always having work available for allocation; (b) lessen the likelihood that a computation will stall for lack of tasks that are eligible for execution. The current paper reconnects the Theory, which models computations abstractly, with a variety of significant* real *computations and computational paradigms, by illustrating how to schedule these computations optimally.*

1. Introduction

Earlier work [7, 18, 19, 20] develops *IC-Scheduling theory*, an algorithmic framework for scheduling computations having intertask dependencies for Internet-based computing (IC, for short). The Theory's goal is to produce schedules that maximize the rate at which tasks are rendered eligible for allocation to remote clients (hence for execution), with the dual aim of: (a) enhancing the utilization of remote clients, by always having work to allocate to an available client; (b) lessening the likelihood of a computation's stalling pending execution of already-allocated tasks.

IC-Scheduling theory, as developed in [7, 8, 18], treats computations as abstract *directed acyclic graphs* (*dags*). While such treatment is convenient for algorithmicists, it may obscure the range of applicability of the theory to real computations for those who do not deal daily with such abstractions. This paper reconnects the Theory with "real" IC by describing a variety of significant specific computations and generic computational paradigms that IC-Scheduling theory provides optimal schedules for, providing for each:

1. an analysis of its intertask dependency structure;
2. a sketched formal analysis of how this structure is scheduled optimally by the theory;
3. a discussion of how this structure can be rendered *multi-granular*, to adjust task granularities, at least over a wide range, while maintaining a desirable intertask dependency structure. Multi-granularity is quite important in IC: it allows one to tailor task granularity to remote clients' computing resources and to diminish the volume of (expensive) inter-client communication.

We illustrate each paradigm with one or more significant applicative computations. Our overriding concern here is to illustrate how the structure of each computation's intertask dependencies is accommodated by the Theory. Other issues—even critical ones such as communication load, which may influence one's decision about the computation's suitability for IC—are *not* our primary concern (although we plan to address them in future work).

This paper is part of a multi-pronged assessment of IC-Scheduling theory's impact on "real" IC. The other prongs involve simulation experiments that, within the framework of a formal model of IC, compares the performace of the scheduling algorithm, ICO, of [18] against a variety of competing scheduling algorithms. In [16], an extension of ICO is compared, on four real scientific dags, against the FIFO scheduler of Condor [6]. In [12], ICO is compared, on hundreds of synthetic dags, against three natural scheduling heuristics, including FIFO. In all of the simulations ICO either matches or improves the performance of competing schedulers, suggesting that IC-Scheduling theory has a significant positive impact on the scheduling problem for IC. We are planning future work that tests ICO's performance on real, rather than simulated computations.

RELATED WORK. The fundamental notions of IC-Scheduling theory are introduced in [19, 20], which also

1-4244-0910-1/07/$20.00

characterize and specify optimal schedules for several uniform dags (some of which appear in here also). These sample schedules are developed into the seeds of the Theory in [7, 18], whose conceptual contributions we describe imminently. Major extensions to the Theory are being developed in [8]. A companion study, [17], motivated by the impossibility of scheduling many dags optimally under the Theory, pursues a scheduling regimen in which a server allocates *batches* of tasks at once, rather than allocating individual tasks as they become eligible. Optimality is always possible within the batched framework, but achieving it may entail a prohibitively complex computation. As described earlier, our assessment of the Theory's impact on real IC has begun in [16, 12], the former of which also describes the PRIO scheduling tool. Finally, our study is motivated by the many exciting systems- and/or application-oriented studies of IC in sources such as [5, 10, 11, 13, 14, 21].

2. The Rudiments of IC-Scheduling Theory

We assume familiarity with dags and related notions; cf. [18].

2.1. A Quality Model for Executing Dags. When one executes a dag $\mathcal{G}$, an unexecuted node v is ELIGIBLE (for execution) only after all of v's parents are EXECUTED; hence, every unexecuted source is always ELIGIBLE. We do not allow recomputation of nodes, so a node loses its ELIGIBLE status once it is EXECUTED. In compensation, executing a node v may render new nodes ELIGIBLE—specifically, when v is their last parent to be executed. A *schedule* for $\mathcal{G}$ is a rule for selecting which ELIGIBLE node to execute at each step of an execution of $\mathcal{G}$. We measure the quality of an execution of $\mathcal{G}$ by the number of ELIGIBLE nodes after each node-execution—the more, the better. (Note that *we measure time in an event-driven manner*, as the number of nodes that are EXECUTED at that point.) Our goal is to execute $\mathcal{G}$'s nodes in an order that maximizes the production rate of ELIGIBLE nodes *at every step of the computation*. A schedule for $\mathcal{G}$ that achieves this demanding goal is said to be **IC-optimal**. The significance of IC optimality stems from the facts that producing ELIGIBLE nodes more quickly may: (1) reduce the chance of a computation's stalling when clients are slow, so that no tasks can be allocated pending the return of already allocated ones, (2) allow the Server to satisfy more requests for tasks from a batch of (roughly) simultaneous requests, thereby increasing "parallelism."

2.2. Tools for Crafting IC-Optimal Schedules.

A. COMPOSITION-BASED TOOLS. For $i = 1, 2$, let the dag $\mathcal{G}_i$ have n_i nonsinks, and let it admit the IC-optimal schedule Σ_i. If the following inequalities hold:[1] [2]

$$\begin{aligned}&(\forall x \in [0, n_1])\ (\forall y \in [0, n_2]) : \\ &E_{\Sigma_1}(x) + E_{\Sigma_2}(y) \leq\ E_{\Sigma_1}(\min\{n_1, x+y\}) + \\ &\qquad\qquad E_{\Sigma_2}(\max\{0, x+y-n_1\}),\end{aligned} \tag{1}$$

then $\mathcal{G}_1$ *has priority over* $\mathcal{G}_2$: $\mathcal{G}_1 \triangleright \mathcal{G}_2$; one never decreases IC quality by executing a nonsink of $\mathcal{G}_1$ when possible.

The operation of *composition* is defined inductively:

- Start with a set $\mathcal{S}$ of base dags.
- To compose dags $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{S}$—which could be the same dag with nodes renamed to achieve disjointness—thereby obtaining a composite dag $\mathcal{G}$:

  Take disjoint copies of $\mathcal{G}_1$ and $\mathcal{G}_2$, with nodes renamed if necessary to ensure that $N_{\mathcal{G}} \cap (N_{\mathcal{G}_1} \cup N_{\mathcal{G}_2}) = \emptyset$.
  (1) Select some set S_1 of sinks from $\mathcal{G}_1$, and an equal-size set S_2 of sources from $\mathcal{G}_2$.
  (2) Pairwise identify (i.e., merge) the nodes in the sets S_1 and S_2 (in an arbitrary way). The resulting node-set is $N_{\mathcal{G}}$; the induced set of arcs is $A_{\mathcal{G}}$.[3]
- Add the dag $\mathcal{G}$ thus obtained to the base set $\mathcal{S}$.

We say that $\mathcal{G}$ is *composite of type* $[\mathcal{G}_1 \Uparrow \mathcal{G}_2]$. $\mathcal{G}$ is a $\triangleright$-*linear composition* of dags $\mathcal{G}_1, \ldots, \mathcal{G}_n$ if: (a) $\mathcal{G}$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$; (b) $\mathcal{G}_i \triangleright \mathcal{G}_{i+1}$, for all $i \in [1, n-1]$.

Theorem 1 ([18]) *Let $\mathcal{G}$ be a $\triangleright$-linear composition of $\mathcal{G}_1, \ldots, \mathcal{G}_n$, where each $\mathcal{G}_i$ admits an IC-optimal schedule Σ_i. The following schedule Σ for $\mathcal{G}$ is IC optimal.*
1. For $i = 1, \ldots, n$, in turn, Σ executes the nodes of $\mathcal{G}$ that correspond to nonsinks of $\mathcal{G}_i$, in the order mandated by Σ_i.
2. Σ finally executes all sinks of $\mathcal{G}$ in any order.

B. DUALITY-BASED SCHEDULING TOOLS. The *dual* of dag $\mathcal{G}$ is the dag $\widetilde{\mathcal{G}}$ obtained by reversing all of $\mathcal{G}$'s arcs (thereby interchanging sources and sinks). One can infer IC-optimal schedules and $\triangleright$-priorities for a dag $\mathcal{G}$ from corresponding entities for $\widetilde{\mathcal{G}}$.

Let $\mathcal{G}$ have n nonsinks, $U = \{u_1, \ldots, u_n\}$, and N nonsources, $V = \{v_1, \ldots, v_N\}$. Let schedule Σ execute U's nodes in the order $u_{k_1}, \ldots, u_{k_n}$. Each execution of a u_{k_j} renders ELIGIBLE a (possibly empty) "packet" of nonsources, $P_j = \{v_{j,1}, \ldots, v_{j,i_j}\}$. Thus, Σ renders $\mathcal{G}$'s nonsources ELIGIBLE in a sequence of "packets:"

$$P_1 = \{v_{1,1}, \ldots, v_{1,i_1}\},\ \ldots,\ P_n = \{v_{n,1}, \ldots, v_{n,i_n}\}.$$

A schedule $\widetilde{\Sigma}$ for $\widetilde{\mathcal{G}}$ is *dual to* Σ if it executes $\widetilde{\mathcal{G}}$'s nonsinks—i.e., V's nodes—in an order of the form[4]

$$[[v_{n,1}, \ldots, v_{n,i_n}]],\ \ldots,\ [[v_{1,1}, \ldots, v_{1,i_1}]].$$

($\widetilde{\mathcal{G}}$ generally admits many schedules that are dual to Σ.)

[1] $[a, b] =_{def} \{a, a+1, \ldots, b\}$.
[2] $E_\Sigma(t)$ is the number of ELIGIBLE nonsources on $\mathcal{G}$ after step t of Σ.
[3] An arc $(u \to v)$ is *induced* if $\{u, v\} \subseteq N_{\mathcal{G}}$.
[4] $[[a, \ldots, c]]$ is a fixed, but unspecified, permutation of $\{a, \ldots, c\}$.

Theorem 2 ([7]) **(a)** *Let $\mathcal{G}$ admit the IC-optimal schedule $\Sigma_{\mathcal{G}}$. Any schedule for $\widetilde{\mathcal{G}}$ that is dual to $\Sigma_{\mathcal{G}}$ is IC optimal.* **(b)** *For all $\mathcal{G}_1$ and $\mathcal{G}_2$: $[\mathcal{G}_1 \triangleright \mathcal{G}_2]$ if, and only if, $[\widetilde{\mathcal{G}}_2 \triangleright \widetilde{\mathcal{G}}_1]$.*

3. Expansion-Reduction (E-R) Computations

3.1. The Abstract Dags. The computations exemplified in this section are built via iterated composition from the two basic building blocks in Fig. 1: the *Vee dag* $\mathcal{V}$ and the *Lambda dag* Λ (so named for the shapes of their drawings). Note that Λ and $\mathcal{V}$ are dual to one another. Via iterated composition: $\mathcal{V}$ generates bifurcating *expansive* computations (= binary out-trees)—as, e.g., in the "divide" phase of a divide-and-conquer computation; Λ generates bi-joining *reductive* computations (= binary in-trees)—as, e.g., in the recombination phase of a divide-and-conquer computation.

Figure 1. *The Vee $\mathcal{V}$* (left) *and the Lambda Λ* (right).

Our interest here is in multiphase computations, which compose alternating expansive and reductive computations.[5] Fig. 2 depicts explicitly one such computation, which performs a single composition. The out-tree at the

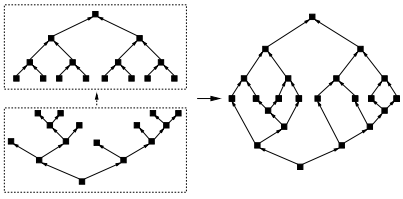

Figure 2. *An expansion-reduction computation.*

left-bottom of the figure generates values, which are accumulated by the in-tree at the left-top. The two trees are composed into the *diamond* dag at the right by merging (in this case, all) sinks of the out-tree with sources of the in-tree.

RENDERING COMPUTATIONS MULTI-GRANULAR. One can easily coarsen the tasks in an E-R computation by selectively truncating branches of the out-tree, together with mated portions of the in-tree, in a manner that leaves more of the overall computation to remote clients. We illustrate this process in Fig. 3, where we coarsen two tasks of the diamond dag of Fig. 2. We simplify the coarsening by replacing the in-tree of Fig. 2 by the dual, $\widetilde{\mathcal{T}}$, of the out-tree $\mathcal{T}$.

[5] Our focus on *binary* trees in illustrations is only for convenience.

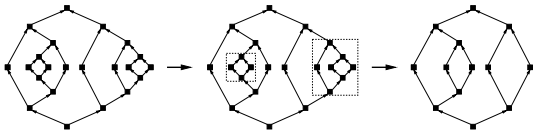

Figure 3. *Coarsening tasks in the dag of Fig. 2.*

The reader should be able to extrapolate from this example to render other diamond dags multi-granular.

IC-OPTIMAL SCHEDULES. We derive IC-optimal schedules for arbitrary alternating E-R dags in steps, beginning with out- and in-trees and progressing to compositions.

Out- and in-trees. Note first that every out-tree is composite of type $\mathcal{V} \Uparrow \cdots \Uparrow \mathcal{V}$. Using (1), one shows that $\mathcal{V} \triangleright \mathcal{V}$, so that every out-tree is a $\triangleright$-linear composition, hence admits an IC-optimal schedule (Theorem 1).[6] Since every in-tree is dual to an out-tree, Theorem 2 shows that every in-tree admits an IC-optimal schedule.[7] Indeed: *A schedule for an in-tree $\mathcal{T}$ is IC optimal iff it executes the two sources of each copy of Λ in $\mathcal{T}$ consecutively.* [20]

Diamond dags. A diamond dag $\mathcal{D}$ is a composition, of type $\mathcal{T} \Uparrow \mathcal{T}'$, of an out-tree $\mathcal{T}$ and an in-tree $\mathcal{T}'$. By associativity of composition [18], $\mathcal{D}$ is composite of type $\mathcal{V} \Uparrow \cdots \Uparrow \mathcal{V} \Uparrow \Lambda \Uparrow \cdots \Uparrow \Lambda$, for some number of $\mathcal{V}$'s (which matches the number of Λ's). Using (1), one shows that $\mathcal{V} \triangleright \Lambda$, so that $\mathcal{D}$ is a $\triangleright$-linear composition, hence admits an IC-optimal schedule (Theorem 1). Indeed, any schedule that executes $\mathcal{T}$ using an IC-optimal schedule, then executes $\mathcal{T}'$ using an IC-optimal schedule, is IC optimal for $\mathcal{D}$.

Complicated alternations. Our analysis of diamond dags applies almost verbatim to a far broader family of alternating in- and out-trees, such as those in Fig. 4. (Note that, in the rightmost dag, the numbers of leaves of a composed out-tree and in-tree do not match.) To analyze these extended dags, we note that, although $\mathcal{T} \triangleright \mathcal{T}'$ for any out-tree $\mathcal{T}$ and in-tree $\mathcal{T}'$, the converse does not hold. Nonetheless, every dag $\mathcal{G}$ of type $\mathcal{T}' \Uparrow \mathcal{T}$ (e.g., the leftmost dag in Fig. 4) admits an IC optimal schedule, because $\mathcal{G}$'s topology forces *every* schedule to execute all of $\mathcal{T}'$ before any of $\mathcal{T}$; hence, we need worry only about how to compute $\mathcal{T}$ and $\mathcal{T}'$ individually. The preceding reasoning actually

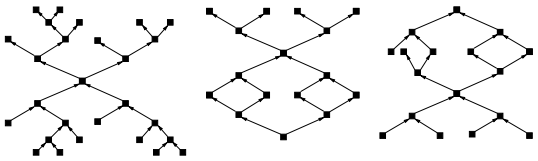

Figure 4. *Sample alternating E-R computations.*

[6] Indeed, easily, every schedule for an out-tree is IC optimal!

[7] This follows also because in-trees are iterated compositions of Λ's.

shows that any alternating composition of out-trees and in-trees of the composition-types depicted in Table 1 admits an IC-optimal schedule. (The superscript "(out)" identifies an out-tree; "(in)" identifies an in-tree.)

3.2. A Sample Computation: Numerical Integration. Alternating E-R dags arise in many divide-and-conquer computations; we describe just one significant one. Several numerical integration algorithms proceed as follows. (We specify the task residing in each node of the out-tree.) Say that one is to integrate a function F over an interval $[a_0 \leftrightarrow b_0]$.[8] One chooses a computationally simple functional form that provides a numerically adequate approximation to the area under F, at least over very small intervals. The Trapezoid Rule, e.g., uses a linear approximation, via the approximation $A(X,Y) \stackrel{\text{def}}{=} \frac{1}{2}(F(X)+F(Y))(Y-X)$. Each node computes two quantities:

$$\begin{array}{rcl} A_0 & = & A(a_0, b_0); \\ A_1 & = & A\left(a_0,\ \frac{1}{2}(a_0+b_0)\right) + A\left(\frac{1}{2}(a_0+b_0),\ b_0\right). \end{array}$$

A_0 is a linear approximation of F's area over the interval $[a_0 \leftrightarrow b_0]$; A_1 is the approximation obtained by splitting $[a_0 \leftrightarrow b_0]$ in two, thereby making some accommodation for F's curvature within the interval. If $|A_0 - A_1|$ is sufficiently small (relative to a predetermined tolerance), then the approximation A_0 is accepted, and the current task-node becomes a leaf of the out-tree; if $|A_0 - A_1|$ is too large—i.e., exceeds the tolerance—then the current task spawns two new tasks, representing the two summands of A_1, which become its children in the out-tree: the left child-task seeks to integrate F over $[a_0 \leftrightarrow \frac{1}{2}(a_0+b_0)]$, the right child over $[\frac{1}{2}(a_0+b_0) \leftrightarrow b_0]$. In Fig. 1, the variables w, x_0, x_1 represent the intervals over which the task must integrate F:

$$\begin{array}{lrcl} \textbf{if} & w & = & [a_0 \leftrightarrow b_0] \\ \textbf{then} & x_0 & = & [a_0 \leftrightarrow \frac{1}{2}(a_0+b_0)] \\ \textbf{and} & x_1 & = & [\frac{1}{2}(a_0+b_0) \leftrightarrow b_0] \end{array}$$

The initial task—the root of the out-tree—represents the entire interval $[a_0 \leftrightarrow b_0]$. (Note: Our if-then prescriptions specify intertask dependencies; they do *not* specify a computation that *we* are doing.)

The integration procedure composes the final out-tree $\mathcal{T}$, whose leaves contain F's area over the subintervals wherein a linear approximation to F suffices, with its dual in-tree $\widetilde{\mathcal{T}}$, which accumulates these areas; hence, $\widetilde{\mathcal{T}}$'s sink ends up with the sought approximation to F's area over the entire interval. In Fig. 1, the variables y_0, y_1, z on Λ represent the areas under F over the various subintervals:

$$\begin{array}{lrcl} \textbf{if} & y_0 & = & A(a_0,\ \frac{1}{2}(a_0+b_0)) \\ \textbf{and} & y_1 & = & A(\frac{1}{2}(a_0+b_0),\ b_0) \\ \textbf{then} & z & = & y_0 + y_1 \end{array}$$

[8] $[X \leftrightarrow Y]$ denotes the closed real interval $\{Z \mid X \le Z \le Y\}$.

The described computation thus generates a (possibly irregular) binary out-tree whose leaves contain the areas under the curve in regions that are small enough for a linear approximation to provide an adequate approximation to the true area. It then uses an in-tree to accumulate these subinterval areas into the area of F over the entire interval $[a_0 \leftrightarrow b_0]$. By appropriately coarsening the diamond dag that represents this computation, one can decrease the volume of internode communication, as well as render the computation's tasks more coarse-grain.

4. Wavefront-Related Computations

4.1. The Abstract Dags. We now discuss computations having the structures of 2-dimensional meshes that are truncated along their diagonals; see Fig. 5. While *out-meshes* represent many wavefront-structured computations, in-meshes are also important, when discussing multigranularity in mesh-like dags. Moreover, results about in-meshes follow by duality from results about out-meshes.

Figure 5. *An out-mesh (left) and an in-mesh (right).*

IC-OPTIMAL SCHEDULES. Ad hoc arguments in [19, 20] show that out- and in-meshes admit IC-optimal schedules. A more interesting proof is extrapolated from Fig. 6: Every out-mesh is a composition of *W-dags*[9] having increasing numbers of sources. Since executing a W-dag's sources consecutively is IC optimal, and since smaller W-dags have $\triangleright$-priority over larger ones [18], every out-mesh is a $\triangleright$-linear composition, hence admits an IC-optimal schedule. By duality, the same is true for in-meshes.

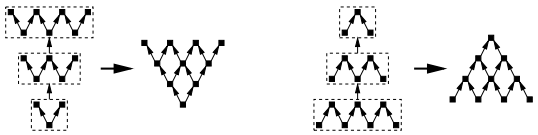

Figure 6. *Out- and in-meshes as compositions.*

RENDERING COMPUTATIONS MULTI-GRANULAR. Coarsening tasks in mesh-like dags via node-clustering is complicated by meshes' tight connectivity. The scheme in Fig. 7 coarsens an out-mesh's tasks by a factor of 4, which can be adjusted by sliding the dashed lines to form "squares" and "triangles" whose "areas" determine the factor. When tasks

[9] W-dags are named for the Latin letters suggested by their topologies.

Tree-dag notation	Diamond-dag notation
$(\mathcal{T}_0^{(\text{out})} \Uparrow \mathcal{T}_0^{(\text{in})}) \Uparrow (\mathcal{T}_1^{(\text{out})} \Uparrow \mathcal{T}_1^{(\text{in})}) \Uparrow \cdots \Uparrow (\mathcal{T}_n^{(\text{out})} \Uparrow \mathcal{T}_n^{(\text{in})})$	$\mathcal{D}_0 \Uparrow \mathcal{D}_1 \Uparrow \cdots \Uparrow \mathcal{D}_n$
$\mathcal{T}_0^{(\text{in})} \Uparrow (\mathcal{T}_1^{(\text{out})} \Uparrow \mathcal{T}_1^{(\text{in})}) \Uparrow \cdots \Uparrow (\mathcal{T}_n^{(\text{out})} \Uparrow \mathcal{T}_n^{(\text{in})})$	$\mathcal{T}_0^{(\text{in})} \Uparrow \mathcal{D}_1 \Uparrow \cdots \Uparrow \mathcal{D}_n$
$(\mathcal{T}_1^{(\text{out})} \Uparrow \mathcal{T}_1^{(\text{in})}) \Uparrow \cdots \Uparrow (\mathcal{T}_n^{(\text{out})} \Uparrow \mathcal{T}_n^{(\text{in})}) \Uparrow \mathcal{T}_0^{(\text{out})}$	$\mathcal{D}_1 \Uparrow \cdots \Uparrow \mathcal{D}_n \Uparrow \mathcal{T}_0^{(\text{out})}$

Table 1. Diamond dags that admit IC-optimal schedules.

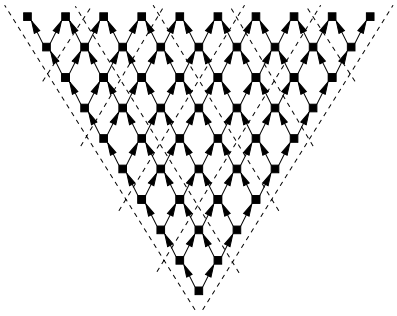

Figure 7. *Rendering an out-mesh multi-granular.*

are equi-granular, each coarsened mesh is a smaller version of the original, hence admits an IC-optimal schedule; when tasks differ in granularity, the coarsened mesh may lose its regularity and IC-optimal schedulablity. Importantly, though, under all such coarsenings, the amount of computation represented by a task grows quadratically with the task's "sidelength," while communication—a much dearer resource—grows linearly.

5. Butterfly-Structured Computations

5.1. The Abstract Dags. The computations in this section are all built via iterated composition from the *butterfly building block* $\mathcal{B}$ of Fig. 8 (so named for its shape in the drawing). A large variety of transformations effected

Figure 8. *The butterfly building block* $\mathcal{B}$.

using $\mathcal{B}$—i.e., specifications of y_0 and y_1 as functions of x_0 and x_1—lead to useful computations. Notable among the dags constructed from butterfly building blocks is the *d-dimensional butterfly dag* $\mathcal{B}_d$, for $d = 1, 2, \ldots$. The 1-dimensional butterfly dag is just $\mathcal{B}$: $\mathcal{B}_1 = \mathcal{B}$; the 2- and 3-dimensional dags, $\mathcal{B}_2$ and $\mathcal{B}_3$, are depicted in Fig. 9. (The reader can easily extrapolate to higher dimensions.)

IC-OPTIMAL SCHEDULES. The easiest way to derive an IC-optimal schedule for $\mathcal{B}_d$ is to note (as is well known) that

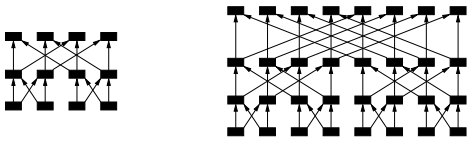

Figure 9. *The* 2*- and* 3*-dimensional butterfly dags.*

$\mathcal{B}_d$ is an iterated composition of $\mathcal{B}$. (Fig. 10 illustrates this for $\mathcal{B}_3$.) Using (1), one shows that $\mathcal{B} \rhd \mathcal{B}$, so that every iterated composition of $\mathcal{B}$'s—hence, every $\mathcal{B}_d$—is an $\rhd$-linear composition, whence $\mathcal{B}_d$ admits an IC-optimal schedule. A generalized argument from [20] shows: *A schedule for an iterated composition $\mathcal{G}$ of the butterfly building block $\mathcal{B}$ is IC optimal if, and only if, it executes the two sources of each copy of $\mathcal{B}$ within $\mathcal{G}$ in consecutive steps.*

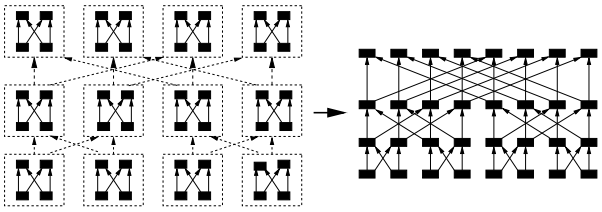

Figure 10. *The butterfly dag as a composition of* $\mathcal{B}$*'s.*

RENDERING COMPUTATIONS MULTI-GRANULAR. While most discussions of the diverse computations modeled by butterfly networks (cf. [15])—including ours in Section 5.2—focus on computations having fine-grained tasks, butterfly dags support important computations having tasks of arbitrary granularities. This is because $\mathcal{B}_{a+b}$ is always (isomorphic to) a copy of $\mathcal{B}_a$ each of whose nodes is a copy of $\mathcal{B}_b$ (cf. [1] and Fig. 10, wherein $a = 2$ and $b = 1$). This allows one both to adjust the granularities of the tasks that are allocated to remote clients and to control the volume of internode communication, while always retaining butterfly-structured dependencies.

5.2. Sample Computations. We exemplify just two useful computational transformations modeled by butterfly dags: the *comparator transformation*

$$y_0 = \min(x_0,\, x_1) \quad \text{and} \quad y_1 = \max(x_0,\, x_1)$$

and the *convolution transformation*

$y_0 = x_0 + \omega x_1$ and $y_1 = x_0 - \omega x_1$
where ω is a constant associated with this specific copy of $\mathcal{B}$. We now describe the complex computations that these transformations lead to.

SORTING. It is well known [2] that iterated compositions of $\mathcal{B}$—using the comparator transformation—will sort any sequence of keys from an ordered domain, that are presented at the sources of the composite dag. (The most efficient known such networks require a rather complicated iterated composition [9].)

CONVOLUTIONS. The product of degree-n polynomials, $f(x) = a_0 + a_1 x + \cdots + a_n x^n$, $g(x) = b_0 + b_1 x + \cdots + b_n x^n$ is the polynomial
$[f \otimes g](x) = A_0 + A_1 x + \cdots + A_{2n} x^{2n}$;
each A_k is a *convolution*, i.e., a sum of the form
$A_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0 = \sum_{i=0}^{k} a_i b_{k-i}$.
Convolutions arise in myriad computations other than polynomial multiplication, one of the most important being the Fast Fourier Transform (FFT) [9]. In fact, the data dependencies of the d-dimensional FFT are modeled by $\mathcal{B}_d$, hence can be computed IC optimally by a simple algorithm. Specifically, each copy of $\mathcal{B}$ used to construct the FFT uses the convolutional transformation with ω derived from the dth complex roots of unity. In fact, one can use the FFT to perform a large repertoire of convolutions, notably including polynomial multiplication, in sequential time $\Theta(n \log n)$. We thereby can schedule a broad range of convolutional computations IC optimally.

6. A Complex E-R Paradigm

6.1. The Parallel-Prefix/Scan Operator. The *parallel-prefix* (or, *scan*) operator provides myriad examples of important computations that our theory can schedule IC optimally. One sees in, e.g., [3, 15], that the ability to compute parallel-prefixes efficiently enables one to compute a large variety of other computations efficiently, ranging from microscopic ones such as carry-lookahead addition, to large ones such as we exemplify imminently. The operator is defined for any binary associative operation (such as $+$, $\times$, $\min$, $\max$, "concatenate"). Denoting such an operation by $*$, the $*$-parallel prefix of input vector $\langle x_1, \ldots, x_n \rangle$ is output vector $\langle y_1, \ldots, y_n \rangle$:

$$y_1 = x_1; \quad y_{i+1} = y_i * x_{i+1} = x_1 * \cdots * x_i * x_{i+1}.$$

Among the many algorithms for the $*$-parallel prefix computation, the following is attractive because it operates in $O(\log n)$ parallel steps (under n-fold parallelism).

```
for   j = 0   to ⌊log2(n − 1)⌋ do
      for     i = 2^j  to  n − 1  do in parallel
              x_i ← x_{i−2^j} * x_i
do in parallel  y_i ← x_i
```

Fig. 11 depicts the 8-input parallel-prefix dag $\mathcal{P}_8$.

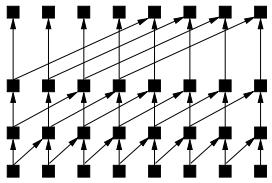

Figure 11. *The 8-input parallel-prefix dag $\mathcal{P}_8$.*

IC-OPTIMAL SCHEDULES. The n-input parallel-prefix dag $\mathcal{P}_n$ is an iterated composition of *N-dags*.[10] Fig. 12 illustrates that $\mathcal{P}_8$ is composite of type $\mathcal{N}_8 \Uparrow \mathcal{N}_4 \Uparrow \mathcal{N}_4 \Uparrow \mathcal{N}_2 \Uparrow \mathcal{N}_2 \Uparrow \mathcal{N}_2 \Uparrow \mathcal{N}_2$. One sees in [18] that: every $\mathcal{N}_s$ admits an

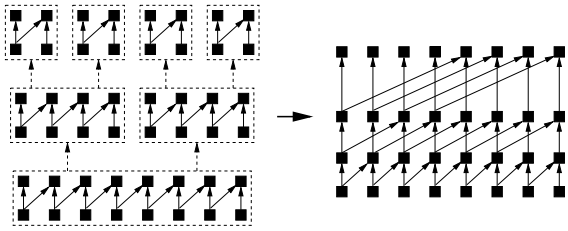

Figure 12. *Exemplifying parallel-prefix dags as compositions of N-dags.*

IC-optimal schedule; $\mathcal{N}_s \triangleright \mathcal{N}_t$ for all s and t. Thus, $\mathcal{P}_n$ is a $\triangleright$-linear composition, hence admits an IC-optimal schedule. Indeed: *Any schedule for $\mathcal{P}_n$ that executes the N-dags $\mathcal{N}_s$ in nonincreasing order of s is IC optimal.*

RENDERING COMPUTATIONS MULTI-GRANULAR. The $*$-parallel-prefix operator requires only the associativity of operation $*$, so one can employ the operator with a wide range of operations, to:

- generate the first n powers of an integer;
- generate the first n powers of a complex number;
- compute all paths in a graph $\mathcal{G}$, by generating the first n "logical" powers of $\mathcal{G}$'s adjacency matrix.

One can often manipulate tasks to change granularities; e.g., expand a matrix multiplication to its (fine-grain) constituent scalar operations, or cluster dag-nodes to coarsen tasks.

6.2. Two Sample Computations.

A. THE n-DIMENSIONAL DISCRETE LAPLACE TRANSFORM (*DLT*)—a/k/a the *Z-Transform*—transforms an n-dimensional vector $\langle x_0, \ldots, x_{n-1} \rangle$ to an m-dimensional vector of complex functions $\langle y_0(\omega), \ldots, y_{m-1}(\omega) \rangle$. The value $y_k(\omega)$ is given (cf. [4]) by $y_k(\omega) = \sum_{i=0}^{n-1} x_i \omega^{ik}$. Our algorithm for the DLT uses an n-source in-tree to accumulate the terms of the sum; each source v_i first multiplies x_i times the power of ω that v_i has received. In Fig. 1, the

[10] The *N-dag* $\mathcal{N}_s$ has s sources and s sinks; its arcs connect each source v to sink v and sink $v+1$ if the latter exists.

variables y_0, y_1, z represent subsums:

$$\begin{array}{ll} \textbf{if} & y_0 \;=\; x_i \cdot \omega^{ik} \\ \textbf{and} & y_1 \;=\; x_j \cdot \omega^{jk} \\ \textbf{then} & z \;=\; y_0 + y_1 \end{array}$$

Our algorithm uses $\mathcal{P}_n$ to generate the terms of the DLT-sum. (Of course, other algorithms may be preferable on some platforms.) For any complex number ω, one can compute $y_k(\omega)$ using $\mathcal{P}_n$ to transform the input vector $\langle \omega^k, \ldots, \omega^k \rangle$ into the vector $\langle 1, \omega^k, \ldots, \omega^{(n-1)k} \rangle$. The resulting 8-input composite DLT dag, $\mathcal{L}_8$, appears on the left-hand side of Fig. 13.

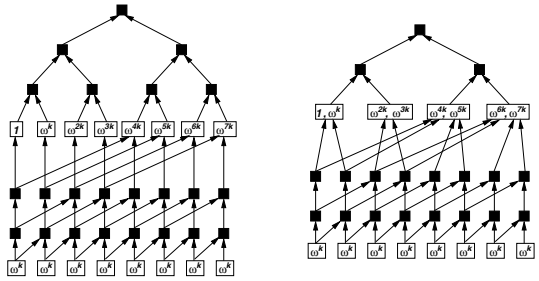

Figure 13. (left) *The* 8*-input DLT dag* $\mathcal{L}_8$*;* (right) *a coarsened version of* $\mathcal{L}_8$.

We simplify the argument that every $\mathcal{L}_n$ admits an IC-optimal schedule by assuming that $n = 2^p$ is a power of 2. Focus on the building blocks of $\mathcal{L}_n$, and note: (a) $\mathcal{L}_n$ is composite of type $\mathcal{P}_n \Uparrow \mathcal{T}_n$, where $\mathcal{T}_n$ is the n-source in-tree; (b) $\mathcal{P}_n$ is composite of type[11]

$$\begin{array}{l} \mathcal{N}_{2^p} \Uparrow (\mathcal{N}_{2^{p-1}} \Uparrow \mathcal{N}_{2^{p-1}}) \\ \quad \Uparrow (\mathcal{N}_{2^{p-2}} \Uparrow \mathcal{N}_{2^{p-2}} \Uparrow \mathcal{N}_{2^{p-2}} \Uparrow \mathcal{N}_{2^{p-2}}) \\ \quad \Uparrow \cdots \Uparrow (\mathcal{N}_2 \Uparrow \mathcal{N}_2 \Uparrow \cdots \Uparrow \mathcal{N}_2) \end{array}$$

(2^{p-1} copies of $\mathcal{N}_2$); (c) $\mathcal{T}_n$ is composite of type $\Lambda \Uparrow \cdots \Uparrow \Lambda$ ($2^p - 1$ copies of Λ). Every $\mathcal{L}_n$ admits an IC-optimal schedule because (cf. [18]): (1) $(\forall s, t)$ $[\mathcal{N}_s \rhd \mathcal{N}_t]$; (2) $(\forall s)$ $[\mathcal{N}_s \rhd \Lambda]$; (3) $[\Lambda \rhd \Lambda]$. Indeed: *any schedule that executes* $\mathcal{L}_n$ *by executing its copy of* $\mathcal{P}_n$ *IC optimally, then executing its copy of* $\mathcal{T}_n$ *IC optimally, is IC optimal.* The demonstration that the coarsened version of $\mathcal{L}_8$ on the right-hand side of Fig. 13 admits an IC-optimal schedule combines the preceding priority-related reasoning with the topological fact that the righthand portion of the in-tree cannot be executed until its sources have been executed.

B. COMPUTING PATHS IN A GRAPH. Our next computation, while not familiar, is quite natural; it exemplifies a coarse-grained computation that employs $\mathcal{P}_n$. Consider Fig. 14 as we describe the computation. Say that we have

[11] We add parentheses to the type expression to enhance legibility.

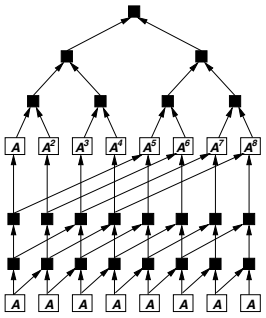

Figure 14. *Computing the paths in a* 9*-node graph.*

a 9-node graph $\mathcal{G}$, presented via its 9×9 adjacency matrix A. (The integer 9 makes Fig. 14 attractive.) We compute a 9×9 matrix M of integers whose (i, j) entry is a vector $\vec{v}_{i,j} = \langle \beta_{i,j}^{(1)}, \ldots, \beta_{i,j}^{(8)} \rangle$, where

$$\beta_{i,j}^{(k)} \;=\; \begin{cases} 1 & \text{if there is a path of length } k \text{ in } \mathcal{G} \text{ between} \\ & \text{nodes } i \text{ and } j \\ 0 & \text{if there is no such path} \end{cases}$$

The resulting intertask dependencies M appear in Fig. 14.

1. We use an 8-input parallel-prefix to compute all logical powers of A. Within each power A^k, the (i, j) entry is 1 or 0, indicating whether or not there is a path of length k in $\mathcal{G}$ between nodes i and j.
2. We use an in-tree to accumulate the information from the eight power matrices A^k into M's 64 vectors $\vec{v}_{i,j}$.

A large range of applications yield to the theme of Fig. 14.

7. Matrix Multiplication

Our final sample computation is matrix multiplication. The recursive algorithm that multiplies $n \times n$ matrices via 2×2 ones (cf. [9]) gives one control of task granularities.

7.1. Multiplying 2×2 **Matrices**. Consider the product

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + BH \end{pmatrix}$$

Importantly, the righthand expression does not invoke the commutativity of multiplication, so the equation holds when the elements of $\mathcal{A}$ and $\mathcal{B}$ are matrices, thus yielding a recursive algorithm for multiplying $n \times n$ matrices. Each level of recursion has the structure exposed in Fig. 15.

7.2. IC-Optimal Schedules. The structure of the dag $\mathcal{M}$ can be elucidated in terms of *(bipartite) cycle-dags*.[12] One notes in Fig. 15 that $\mathcal{M}$ contains two copies

[12] The s-source *(bipartite) cycle-dag* $\mathcal{C}_s$ is obtained from $\mathcal{N}_s$ by adding an arc from the rightmost source to the leftmost sink.

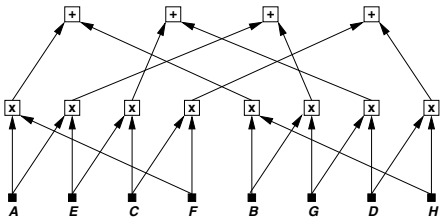

Figure 15. *The dag $\mathcal{M}$: Multiplying 2×2 matrices, or, via recursion, $n \times n$ matrices.*

of $\mathcal{C}_4$, one for products AE, AF, CE, CF and one for BG, BH, DG, DH. These copies are composed with four copies of Λ that compute the required sums of these products. Thus, $\mathcal{M}$ is composite of type $\mathcal{C}_4 \Uparrow \mathcal{C}_4 \Uparrow \Lambda \Uparrow \Lambda \Uparrow \Lambda \Uparrow \Lambda$. Using (1) one shows that $\mathcal{C}_4 \rhd \mathcal{C}_4 \rhd \Lambda \rhd \Lambda$ (cf. [18]). Thus, $\mathcal{M}$ is a $\rhd$-linear composition, hence admits an IC-optimal schedule. Indeed, *the following IC-optimal schedule multiplies 2×2 matrices. Compute the eight required products in the order $AE, CE, CF, AF, BG, DG, DH, BH$. Then compute the four required sums involving these products, in any order.* As with previous sample computations, refining or clustering the dependency dag allows one to adjust the granularity of the computation's constituent tasks

8. Where We Are, and Where We're Going

We have presented a broad range of dag structures that arise in a broad variety of disparate significant real computations, that can all be scheduled optimally—i.e., so as to maximize the rate of rendering nodes eligible for execution—by IC-Scheduling theory. The illustrated computations range from the Discrete Laplace and Fourier Transforms, to matrix multiplication, to generic wavefront and divide-and-conquer algorithms, to computations involving the generic parallel-prefix operator. Thus, IC-Scheduling theory is seen to provide computational solutions for a broad range of real computations of quite distinct structures.

References

[1] A. Avior, T. Calamoneri, S. Even, A. Litman, A.L. Rosenberg (1998): A tight layout of the butterfly network. *Theory of Computing Systs. 31*, 475–487.

[2] V.E. Beneš (1964): Optimal rearrangeable multistage connecting networks. *Bell Syst. Tech. J. 43*, 1641–1656.

[3] G.E. Blelloch (1989): Scans as primitive parallel operations. *IEEE Trans. Comput. 38*, 1526–1538.

[4] L.I. Bluestein (1970): A linear filtering approach to the computation of the Discrete Fourier Transform. *IEEE Trans. Audio Electroacoustics, AU-18*, 451–455.

[5] R. Buyya, D. Abramson, J. Giddy (2001): A case for economy Grid architecture for service oriented Grid computing. *10th Heterogeneous Computing Wkshp.*

[6] Condor Project, University of Wisconsin. http://www.cs.wisc.edu/condor

[7] G. Cordasco, G. Malewicz, A.L. Rosenberg (2007): Advances in IC-scheduling theory: scheduling expansive and reductive dags and scheduling dags via duality. *IEEE Trans. Parallel and Distributed Systs.*, to appear.

[8] G. Cordasco, G. Malewicz, A.L. Rosenberg (2006): Extending IC-Scheduling Theory via the Sweep Algorithm. Typescript, Univ. Massachusetts.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein (1999): *Introduction to Algorithms* (2nd Ed.) MIT Press, Cambridge.

[10] I. Foster and C. Kesselman [eds.] (2004): *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*. Morgan-Kaufmann, San Francisco.

[11] I. Foster, C. Kesselman, S. Tuecke (2001): The anatomy of the Grid: enabling scalable virtual organizations. *Intl. J. Supercomputer Applications*.

[12] R. Hall, A.L. Rosenberg, A. Venkataramani (2007): A comparison of dag-scheduling strategies for Internet-based computing. *IEEE Intl. Parallel and Distributed Processing Symp.*

[13] D. Kondo, H. Casanova, E. Wing, F. Berman (2002): Models and scheduling mechanisms for global computing applications. *Intl. Parallel and Distributed Processing Symp.*

[14] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, M. Lebofsky (2000): SETI@home: massively distributed computing for SETI. In *Computing in Science and Engineering* (P.F. Dubois, Ed.) IEEE Computer Soc. Press, Los Alamitos.

[15] F.T. Leighton (1992): *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, San Francisco.

[16] G. Malewicz, I. Foster, A.L. Rosenberg, M. Wilde (2007): A tool for prioritizing DAGMan jobs and its evaluation. *J. Grid Computing*, to appear.

[17] G. Malewicz and A.L. Rosenberg (2005): On batch-scheduling dags for Internet-based computing. *Euro-Par 2005*. In *LNCS 3648*, Springer-Verlag, Berlin, 262–271.

[18] G. Malewicz, A.L. Rosenberg, M. Yurkewych (2006): Toward a theory for scheduling dags in Internet-based computing. *IEEE Trans. Comput. 55*, 757–768.

[19] A.L. Rosenberg (2004): On scheduling mesh-structured computations for Internet-based computing. *IEEE Trans. Comput. 53*, 1176–1186.

[20] A.L. Rosenberg and M. Yurkewych (2005): Guidelines for scheduling some common computation-dags for Internet-based computing. *IEEE Trans. Comput. 54*, 428–438.

[21] X.-H. Sun and M. Wu (2003): GHS: A performance prediction and node scheduling system for Grid computing. *IEEE Intl. Parallel and Distributed Processing Symp.*