Area-Maximizing Schedules for Series-Parallel DAGs

Gennaro Cordasco[1] and Arnold L. Rosenberg[2,⋆]

[1] University of Salerno,
ISISLab, Dipartimento di Informatica ed Applicazioni "R.M. Capocelli",
Fisciano 84084, Italy
cordasco@dia.unisa.it
[2] Colorado State University,
Electrical and Computer Engineering, Fort Collins, CO 81523, USA
rsnbrg@colostate.edu

Abstract. Earlier work introduced a new optimization goal for DAG schedules: the "AREA" of the schedule. AREA-maximizing schedules are intended for computational environments—such as Internet-based computing and massively multicore computers—that benefit from DAG-schedules that produce execution-eligible tasks as fast as possible. The earlier study of AREA-maximizing schedules showed how to craft such schedules efficiently for DAGs that have the structure of trees and other, less well-known, families of DAGs. The current paper extends the earlier work by showing how to efficiently craft AREA-maximizing schedules for *series-parallel* DAGs, a family that arises, e.g., in multi-threaded computations. The tools that produce the schedules for series-parallel DAGs promise to apply also to other large families of computationally significant DAGs.

1 Introduction

Many modern computing platforms, such as the Internet and massively multicore architectures, have characteristics that are not addressed by traditional strategies[1] for scheduling DAGs i.e., computations having inter-task dependencies that constrain the order of executing tasks. This issue is discussed at length in, e.g., [24], where the seeds of the *Internet-based computing scheduling* (*IC-scheduling*) paradigm are planted. IC-scheduling strives to meet the needs of the new platforms by crafting schedules that execute DAGs in a manner that renders new tasks eligible for execution *at the maximal possible rate*. The paradigm thereby aims to: (a) enhance the utilization of computational resources, by always having work to allocate to an available client/processor; (b) lessen the likelihood of a computation's stalling pending completion of already-allocated tasks. Significant progress in [5,6,8,21,24,25] has extended the capabilities of IC-scheduling so that it can now optimally schedule a wide range of computationally significant DAGs (cf. [4]). Moreover, simulations using DAGs that arise in real scientific computations [20] as well as structurally similar artificial DAGs [13] suggest that IC-schedules can have substantial computational benefits over schedules produced by a

⋆ Research supported in part by US NSF Grant CNS-0905399.

[1] Many traditional DAG-scheduling strategies are discussed and compared in [12,18,22].

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part II, LNCS 6272, pp. 380–392, 2010.

range of common heuristics. However, it has been known since [21] that many significant classes of DAGs do not admit schedules that are optimal within the framework of IC-scheduling. The current authors have responded to this fact with a relaxed version of IC-scheduling under which *every* DAG admits an optimal schedule [7]. This relaxed strategy strives to maximize the *average* rate at which new tasks are rendered eligible for execution. For reasons that become clear in Section 2, we call the new optimization metric for schedules the *AREA* of the schedule: the goal is an *AREA-maximizing schedule* for a DAG (an *AM-schedule*, for short). The study in [7] derived many basic properties of AM-schedules. Notable among these are: (1) Every DAG admits an AM-schedule. (2) AM-scheduling subsumes the goal of IC-scheduling, in the following sense. If a DAG $\mathcal{G}$ admits an optimal IC-schedule, then: (a) that schedule is an AM-schedule; (b) every AM-schedule is optimal under IC-scheduling. Thus, we never lose scheduling quality by focusing on achieving an AM-schedules, rather than an IC-optimal schedule.

Our Contribution. The major algorithmic results of [7] show how to craft AM-schedules efficiently for DAGs that have the structure of a *monotonic tree*, as well as for other, less common, families of DAGs. The current paper extends this contribution by showing how to efficiently craft an AM-schedule for any *series-parallel* DAG (*SP-DAG*, for short). SP-DAGs have a regularity of structure that makes them algorithmically advantageous in a broad range of applications; cf. [15,23,27]. Most relevant to our study: (1) SP-DAGs are the natural abstraction of many significant classes of computations, including divide-and-conquer algorithms. (2) SP-DAGs admit efficient schedules in parallel computing systems such as CILK [1] that employ a multi-threaded computational paradigm. (3) Arbitrary DAGs can be efficiently (in linear time) reformulated as SP-DAGs with little loss in degree of parallelism [15,23]. Our contribution addresses two facets of the existing work on DAG-scheduling: (1) The work on CILK and kindred multi-threaded systems [1,2,3] employ performance metrics that are not relevant in the computational environments that we target (as discussed earlier). (2) Many SP-DAGs do not admit optimal schedules under IC-scheduling [6,21].

Related Work. Problems related to scheduling DAGs on parallel/distributed computing systems have been studied for decades. Most versions of these problems are known to be **NP**-Hard [10], except when scheduling special classes of DAGs (cf. [11]). This has led both researchers and practitioners to seek efficient scheduling heuristics that seem to perform well in practice (cf. [14,26]). Among the interesting attempts to understand such heuristics are the comparisons in [12,18,22] and the taxonomy in [19]. Despite the disparate approaches to scheduling in sources such as those cited, virtually every algorithm/heuristic that predates IC-scheduling shared one central characteristic: they all rely on knowing (almost) exact times for each computer to execute each task and to communicate with a collaborating computer. The central premise underlying IC- and AM-scheduling is that within many modern computing environments, one cannot even apprximate such knowledge reliably. This premise is shared by sources such as [16,17], whose approaches to scheduling for IC platforms admit margins of error in time estimates of 50% or more. Indeed, IC- and AM scheduling are analytical proposals for what to do when accurate estimates are out of the question.

2 Background

Computation-DAGs and schedules. We study computations that are described by DAGs. Each DAG $\mathcal{G}$ has a set $V_{\mathcal{G}}$ of *nodes*, each representing a *task*, and a set $A_{\mathcal{G}}$ of (directed) *arcs*, each representing an intertask dependency. For arc $(u \to v) \in A_{\mathcal{G}}$:

- task v cannot be executed until task u is;
- u is a *parent* of v, and v is a *child* of u in $\mathcal{G}$.

A parentless node is a *source*; a childless node is a *target*. $\mathcal{G}$ is *connected* if it is so when one ignores arc orientations. When $V_{\mathcal{G}_1} \cap V_{\mathcal{G}_2} = \emptyset$, the *sum* $\mathcal{G}_1 + \mathcal{G}_2$ of DAGs $\mathcal{G}_1$ and $\mathcal{G}_2$ is the DAG with node-set $V_{\mathcal{G}_1} \cup V_{\mathcal{G}_2}$ and arc-set $A_{\mathcal{G}_1} \cup A_{\mathcal{G}_2}$.

When one executes a DAG $\mathcal{G}$, a node $v \in V_{\mathcal{G}}$ becomes ELIGIBLE (for execution) only after all of its parents have been executed. Note that all of $\mathcal{G}$'s sources are ELIGIBLE at the beginning of an execution; the goal is to render all of $\mathcal{G}$'s targets ELIGIBLE. Informally, a *schedule* Σ for $\mathcal{G}$ is a rule for selecting which ELIGIBLE node to execute at each step of an execution of $\mathcal{G}$; formally, Σ is a *topological sort* of $\mathcal{G}$, i.e., a linearization of $V_{\mathcal{G}}$ under which all arcs point from left to right (cf. [9]). We do not allow recomputation of nodes/tasks, so a node loses its ELIGIBLE status once it is executed. In compensation, after $v \in V_{\mathcal{G}}$ has been executed, there may be new nodes that are rendered ELIGIBLE; this occurs when v is their last parent to be executed.

We measure the quality of a schedule Σ using the rate at which Σ renders nodes of $\mathcal{G}$ ELIGIBLE. Toward this end, we define[2], for $k \in [1, |V_{\mathcal{G}}|]$, the quantities $E_{\Sigma}(k)$ and $e_{\Sigma}(k)$: $E_{\Sigma}(k)$ is the number of nodes of $\mathcal{G}$ that are ELIGIBLE after Σ has executed k nodes; and $e_{\Sigma}(k)$ is the number of nodes (perforce, nonsources) of $\mathcal{G}$ that are rendered ELIGIBLE by Σ's kth node-execution. (*We measure time in an event-driven manner*, as the number of nodes that have been executed thus far, so we often refer to "step k" rather than "Σ's kth node-execution.")

The AREA metric for schedules. Focus on a DAG $\mathcal{G} = (V_{\mathcal{G}}, A_{\mathcal{G}})$ with n nontargets, N nonsources, s sources, and S targets (note: $N_{\mathcal{G}} \stackrel{\text{def}}{=} |V_{\mathcal{G}}| = s + N = S + n$). The quality of a schedule Σ for $\mathcal{G}$ at step t is given by the size of $E_{\Sigma}(t)$: the larger, the better. The goal of IC-scheduling is to execute $\mathcal{G}$'s nodes in an order that maximizes quality *at every step* $t \in [1, N_{\mathcal{G}}]$ *of the execution*. A schedule Σ^* that achieves this demanding goal is IC-optimal; formally,

$$(\forall t \in [1, N_{\mathcal{G}}]) \;\; E_{\Sigma^*}(t) \;=\; \max_{\Sigma \text{ a schedule for } \mathcal{G}} \{E_{\Sigma}(t)\}$$

The *AREA* of a schedule Σ for $\mathcal{G}$, $AREA(\Sigma)$, is the sum

$$AREA(\Sigma) \;\stackrel{\text{def}}{=}\; E_{\Sigma}(0) + E_{\Sigma}(1) + \cdots + E_{\Sigma}(N_{\mathcal{G}}). \tag{2.1}$$

The normalized AREA, $\widehat{E}(\Sigma) \stackrel{\text{def}}{=} AREA(\Sigma) \div N_{\mathcal{G}}$, is the average number of nodes that are ELIGIBLE when Σ executes $\mathcal{G}$. The term "*area*" arises by formal analogy with Riemann sums as approximations to integrals. The goal of the scheduling paradigm we develop here is to find an *AM schedule* for $\mathcal{G}$, i.e., a schedule $\Sigma^\star$ such that

$$AREA(\Sigma^\star) \;=\; \max_{\Sigma \text{ a schedule for } \mathcal{G}} AREA(\Sigma).$$

[2] $[a, b]$ denotes the set of integers $\{a, a + 1, \ldots, b\}$.

We have discovered in [7] a number of ways to simplify the quest for AM schedules. First, we can restrict the form of such schedules.

Lemma 1. [7] *Altering a schedule Σ for* DAG $\mathcal{G}$ *so that it executes all of $\mathcal{G}$'s nontargets before any of its targets cannot decrease Σ's AREA.*

Hence, we can streamline analysis by ignoring targets. Second, we can alter the AREA metric in certain ways: The only portion of $AREA(\Sigma)$ that actually depends on choices made by Σ is

$$area(\Sigma) \stackrel{\text{def}}{=} \sum_{t=1}^{n} \sum_{j=1}^{t} e_\Sigma(j) \quad ne_\Sigma(1)+(n-1)e_\Sigma(2)+\ldots+e_\Sigma(n). \tag{2.2}$$

The *eligibility profile* associated with schedule Σ for $\mathcal{G}$ is the n-entry vector $\Pi(\Sigma) = \langle e_\Sigma(1), e_\Sigma(2), \ldots, e_\Sigma(n)\rangle$.

Our view of schedules as sequences of nontargets nodes allows us to talk about *subschedules* of Σ, which are *contiguous subsequences* of Σ. Each subschedule Φ delimits a (not necessarily connected) subDAG $\mathcal{G}_\Phi$ of $\mathcal{G}$: $V_{\mathcal{G}_\Phi}$ is the subset of $V_\mathcal{G}$ whose nodes appear in Φ plus all the nodes in $V_\mathcal{G}$ which become ELIGIBLE due to the execution of Φ, and $A_{\mathcal{G}_\Phi}$ contains precisely those arcs from $A_\mathcal{G}$ that have both ends in $V_{\mathcal{G}_\Phi}$.

Let $\mathcal{G}$ be a DAG that admit a schedules Σ. The following abbreviations hopefully enhance legibility. Let Φ be a subschedule of Σ. For any sequence Φ, including (sub)schedules, $\ell_\Phi = |\Phi|$ denotes the length of Φ. Note that $E_\Phi(k)$ and $e_\Phi(k)$ are defined for $k \in [1, \ell_\Phi]$.

- $\Pi(\Phi)$ denotes $\mathcal{G}_\Phi$'s eligibility profile:

$$\Pi(\Phi) \;=\; \langle e_\Phi(1), \ldots, e_\Phi(\ell_\Phi)\rangle \;=\; \langle e_\Sigma(i), \ldots, e_\Sigma(j)\rangle$$

  for some $i, j \in [1, n]$ with $\ell_\Phi = j - i + 1$.
- $SUM\,(\Phi, a, b) \;=\; \sum_{i=a}^{b} e_\Phi(i)$, and $SUM\,(\Phi) \;=\; SUM\,(\Phi, 1, \ell_\Phi)$.

For disjoint subschedules Φ and Ψ of Σ, denote by $(\Phi \cdot \Psi)$ the subschedule of Σ that concatenates Φ and Ψ. Thus: $(\Phi \cdot \Psi)$ has $\ell_\Phi + \ell_\Psi$ elements; it first executes all nodes in Φ, in the same order as Φ, and then executes all nodes in Ψ, in the same order as Ψ.

Series-parallel DAGs (SP-DAGs). A *(2-terminal) series-parallel* DAG $\mathcal{G}$ (*SP*-DAG, for short) is produced by a sequence of the following operations (cf. Fig. 1):

1. Create. Form a DAG $\mathcal{G}$ that has:
   (a) two nodes, a *source* s and a *target* t, which are jointly $\mathcal{G}$'s *terminals*,
   (b) one arc, $(s \to t)$, directed from s to t.
2. Compose SP-DAGs, $\mathcal{G}'$ with terminals s', t', and $\mathcal{G}''$, with terminals s'', t''.
   (a) Parallel composition. Form $\mathcal{G} = \mathcal{G}' \Uparrow \mathcal{G}''$ from $\mathcal{G}'$ and $\mathcal{G}''$ by identifying/merging s' with s'' to form a new source s and t' with t'' to form a new target t.
   (b) Series composition. Form $\mathcal{G} = (\mathcal{G}' \to \mathcal{G}'')$ from $\mathcal{G}'$ and $\mathcal{G}''$ by identifying/merging t' with s''. $\mathcal{G}$ has the single source s' and the single target t''.

One can use examples from [6] to craft SP-DAGs that do not admit optimal IC-schedules.

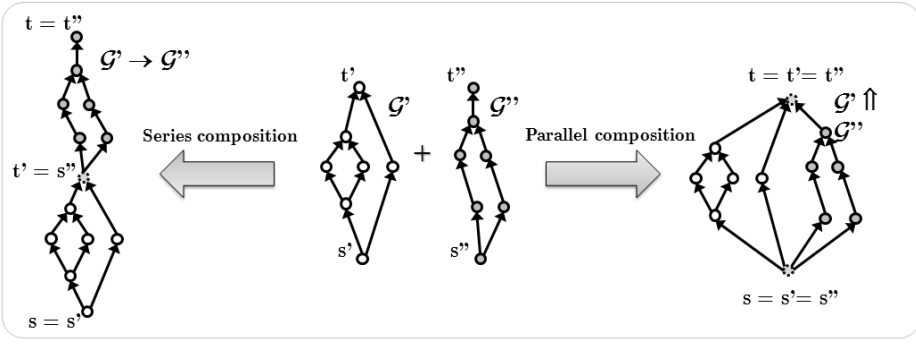

Fig. 1. Compositions of SP-DAGs

3 Maximizing Area for Series-Parallel DAGs

Theorem 1. *There exists an algorithm $\mathcal{A}_{\text{SP-DAG}}$ that finds, in time $O(n^2)$, an AM-schedule for any n-node SP-DAG.*

The remainder of the section develops Algorithm $\mathcal{A}_{\text{SP-DAG}}$, to prove Theorem 1.

3.1 The Idea Behind Algorithm $\mathcal{A}_{\text{SP-DAG}}$

The problem of recognizing when a given DAG is an SP-DAG is classic within the area of algorithm design. A decomposition-based algorithm in [27], which solves the problem in linear time, supplies the basic idea underlying Algorithm $\mathcal{A}_{\text{SP-DAG}}$. We illustrate how with the sample SP-DAG $\mathcal{G}$ in Fig. 2(left).

The fact that $\mathcal{G}$ is an SP-DAG can be verified with the help of a binary decomposition tree $\mathcal{T}_{\mathcal{G}}$ whose structure illustrates the sequence of series and parallel compositions that form $\mathcal{G}$ from a set of nodes; hence, $\mathcal{T}_{\mathcal{G}}$ ultimately takes one back to the definition of SP-DAG in Section 2. Fig. 2(right) depicts $\mathcal{T}_{\mathcal{G}}$ for the $\mathcal{G}$ of Fig. 2(left). The leaves of $\mathcal{T}_{\mathcal{G}}$ are single-arc DAGs; each of its internal nodes represents the SP-DAG obtained by composing (in series or in parallel) its two children. Importantly for the design of Algorithm $\mathcal{A}_{\text{SP-DAG}}$, one can use the algorithm of [27] to construct $\mathcal{T}_{\mathcal{G}}$ from a given DAG $\mathcal{G}$: the construction succeeds just when $\mathcal{G}$ is an SP-DAG. Algorithm $\mathcal{A}_{\text{SP-DAG}}$ uses $\mathcal{T}_{\mathcal{G}}$ to design an AM-schedule for $\mathcal{G}$ inductively.

3.2 Algorithm $\mathcal{A}_{\text{SP-DAG}}$'s Inductive Approach

Given an SP-DAG $\mathcal{G}$, Algorithm $\mathcal{A}_{\text{SP-DAG}}$ first constructs $\mathcal{T}_{\mathcal{G}}$. It then designs an AM-schedule for $\mathcal{G}$ by exploiting the structure of $\mathcal{T}_{\mathcal{G}}$.

A. Leaf-DAGs of $\mathcal{T}_{\mathcal{G}}$. Each leaf-DAG is a single-arc DAG, i.e., a series composition of degenerate one-node DAGs. Each such DAG has the form $(s \to t)$, hence admits a unique schedule (execute s; then execute t) which, perforce, is an AM-schedule.

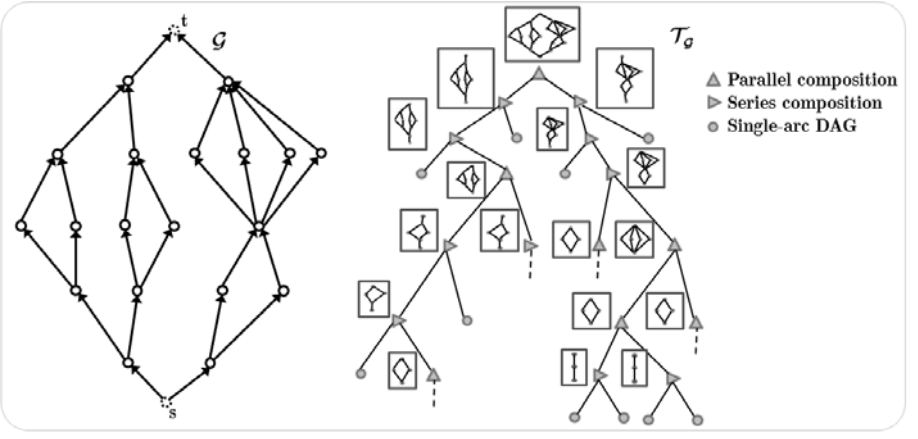

Fig. 2. An example of the decomposition of SP-DAGs

B. Series-Composing Internal Nodes of $\mathcal{T}_{\mathcal{G}}$. Focus on a node of $\mathcal{T}_{\mathcal{G}}$ that represents the series composition $(\mathcal{G}' \rightarrow \mathcal{G}'')$ of disjoint SP-DAGs $\mathcal{G}'$ and $\mathcal{G}''$.

Lemma 2. *If the disjoint SP-DAGs $\mathcal{G}'$ and $\mathcal{G}''$ admit, respectively, AM-schedules Σ' and Σ'', then the schedule $(\Sigma' \cdot \Sigma'')$ is AM for the series composition $(\mathcal{G}' \rightarrow \mathcal{G}'')$.*

Proof. Let $\mathcal{G}'$ have N' nonsources, and let $\mathcal{G}''$ have n'' nontargets. By definition of ELIGIBILITY, every node of $\mathcal{G}'$ must be executed before any node of $\mathcal{G}''$, so we need focus only on schedules for $(\mathcal{G}' \rightarrow \mathcal{G}'')$ of the form $(\Sigma_1 \cdot \Sigma_2)$, where Σ_1 is a schedule for $\mathcal{G}'$ and Σ_2 is a schedule for $\mathcal{G}''$. We then have

$$\begin{aligned} area(\Sigma_1 \cdot \Sigma_2) &= area(\Sigma_1) + area(\Sigma_2) + \ell_{\Sigma_2} SUM\,(\Sigma_1) \\ &= area(\Sigma_1) + area(\Sigma_2) + n''N'. \end{aligned}$$

Thus, choosing schedules Σ_1 and Σ_2 that maximize both $area(\Sigma_1)$ and $area(\Sigma_2)$ will maximize $area(\Sigma_1 \cdot \Sigma_2)$. The lemma follows. □

C. Parallel-Composing Internal Nodes of $\mathcal{T}_{\mathcal{G}}$. Focus on a node of $\mathcal{T}_{\mathcal{G}}$ that represents the parallel composition $(\mathcal{G}' \Uparrow \mathcal{G}'')$ of disjoint SP-DAGs $\mathcal{G}'$ and $\mathcal{G}''$. We present an algorithm that crafts an AM-schedule Σ for $(\mathcal{G}' \Uparrow \mathcal{G}'')$ from AM-schedules Σ' and Σ'' for $\mathcal{G}'$ and $\mathcal{G}''$.

Algorithm SP-AREA
Input: SP-DAGs $\mathcal{G}'$ and $\mathcal{G}''$ and respective AM schedules Σ' and Σ''.
Output: AM schedule Σ for $\mathcal{G} = (\mathcal{G}' \Uparrow \mathcal{G}'')$.

$\mathcal{G}'$ and $\mathcal{G}''$ are disjoint within $(\mathcal{G}' \Uparrow \mathcal{G}'')$, except for their shared source s and target t. Because every schedule for $(\mathcal{G}' \Uparrow \mathcal{G}'')$ begins by executing s and finishes by executing t, we can focus only on how to schedule the *sum*, $\mathcal{G}_1 + \mathcal{G}_2$, obtained by removing both s and t from $(\mathcal{G}' \Uparrow \mathcal{G}'')$: $\mathcal{G}_1$ and $\mathcal{G}_2$ are not-necessarily-connected subDAGs of, respectively, $\mathcal{G}'$ and $\mathcal{G}''$. (The parallel composition in Fig. 1 illustrates that $\mathcal{G}_1$ and/or $\mathcal{G}_2$ can be disconnected: removing s and t in the figure disconnects the lefthand DAG.)

(1) Construct the *average-eligibility profile* $AVG(\Sigma')$ from $\Pi(\Sigma')$ as follows. If $\Pi(\Sigma') = \langle e_{\Sigma'}(1), \ldots, e_{\Sigma'}(n)\rangle$, then $AVG(\Sigma') = \langle a_{\Sigma'}(1), \ldots, a_{\Sigma'}(n)\rangle$, where, for $k \in [1, n]$, $a_{\Sigma'}(k) = \frac{1}{k}\sum_{i=1}^{k} e_{\Sigma'}(k)$.
Similarly, construct the average-eligibility profile $AVG(\Sigma'')$ from $\Pi(\Sigma'')$.
(2) (a) Let j' be the smallest index of $AVG(\Sigma')$ whose value is maximum within profile $AVG(\Sigma')$. Segregate the subsequence of Σ' comprising elements $1, \ldots, j'$, to form an *(indivisible) block* of nodes of Σ' with *average eligibility value* (*AEV*) $a_{\Sigma'}(j')$. Perform a similar analysis for Σ'' to determine the value j'' and the associated block of nodes of Σ'' with AEV $a_{\Sigma''}(j'')$.
(b) Repeat procedure (a) for Σ', using indices from $j'+1$ to n, collecting blocks, until we find a block that ends with the last-executed node of Σ'. Do the analogous repetition for Σ''.

After procedure (a)-then-(b), each of Σ' and Σ'' is decomposed into a sequence of blocks, plus the associated sequences of AEVs.

We claim that the following schedule Σ for $(\mathcal{G}' \Uparrow \mathcal{G}'')$ is AM.

> Σ: 1. Execute s
> 2. Merge the blocks of Σ' and Σ'' in nonincreasing order of AEV. (Blocks are kept intact; ties in AEV are broken arbitrarily.)
> 3. Execute t

Claim. Σ is a valid schedule for $(\mathcal{G}' \Uparrow \mathcal{G}'')$.

This is obvious because: (1) Σ keeps the blocks of both Σ' and Σ'' intact; (2) Σ incorporates the blocks of Σ' (resp., Σ'') in their order within Σ' (resp., Σ'').

Claim. Σ is an AM schedule for $(\mathcal{G}' \Uparrow \mathcal{G}'')$.

Before verifying Σ's optimality, we digress to establish two technical lemmas.

Lemma 3. *Let Σ be a schedule for* DAG $\mathcal{G}$*, and let Φ and Ψ be disjoint subschedules of Σ.*[3] *Then* $\Big[area(\Phi \cdot \Psi) > area(\Psi \cdot \Phi)\Big]$ *iff* $\left[\dfrac{SUM(\Phi)}{\ell_\Phi} > \dfrac{SUM(\Psi)}{\ell_\Psi}\right]$.

Proof. Invoking (2.2), we find that

$$\begin{aligned} area(\Phi \cdot \Psi) = {} & (\ell_\Phi + \ell_\Psi)e_\Phi(1) + (\ell_\Phi + \ell_\Psi - 1)e_\Phi(2) + \cdots + (\ell_\Psi + 1)e_\Phi(\ell_\Phi) \\ & + \ell_\Psi e_\Psi(1) + (\ell_\Psi - 1)e_\Psi(2) + \cdots + e_\Psi(\ell_\Psi); \\ area(\Psi \cdot \Phi) = {} & (\ell_\Phi + \ell_\Psi)e_\Psi(1) + (\ell_\Phi + \ell_\Psi - 1)e_\Psi(2) + \cdots + (\ell_\Phi + 1)e_\Psi(\ell_\Psi) \\ & + \ell_\Phi e_\Phi(1) + (\ell_\Phi - 1)e_\Phi(2) + \cdots + e_\Phi(\ell_\Phi). \end{aligned}$$

Therefore, $area(\Phi \cdot \Psi) - area(\Psi \cdot \Phi) = \ell_\Psi \cdot SUM(\Phi) - \ell_\Phi \cdot SUM(\Psi)$.
The result now follows by elementary calculation. □

Lemma 4. *Let Σ be a schedule for* DAG $\mathcal{G}$*, and let Φ, Ψ, Γ, and Δ be four mutually disjoint subschedules of Σ. Then* $\Big[area(\Phi{\cdot}\Psi) > area(\Psi{\cdot}\Phi)\Big]$ *iff* $\Big[area(\Gamma{\cdot}\Phi{\cdot}\Psi{\cdot}\Delta) > area(\Gamma \cdot \Psi \cdot \Phi \cdot \Delta)\Big]$.

[3] The disjointness of Φ and Ψ ensures that both $(\Phi \cdot \Psi)$ and $(\Psi \cdot \Phi)$ are subschedules of Σ.

The import of Lemma 4 is: *One cannot change the area-ordering of the two concatenations of Φ and Ψ by appending the same fixed subschedule (Γ) before the concatenations and/or appending the same fixed subschedule (Δ) after the concatenations.*

Proof. The following two differences have the same sign:

$$\Big[area(\Phi\cdot\Psi) - area(\Psi\cdot\Phi)\Big] \quad \text{and} \quad \Big[area(\Gamma\cdot\Phi\cdot\Psi\cdot\Delta) - area(\Gamma\cdot\Psi\cdot\Phi\cdot\Delta)\Big].$$

To wit:

$$\begin{aligned}
area(\Gamma{\cdot}\Phi{\cdot}\Psi{\cdot}\Delta) = {} & (\ell_\Gamma{+}\ell_\Phi{+}\ell_\Psi{+}\ell_\Delta)e_\Gamma(1) + \cdots + (1{+}\ell_\Phi{+}\ell_\Psi{+}\ell_\Delta)e_(\ell_\Gamma) \\
& +(\ell_\Gamma{+}\ell_\Phi{+}\ell_\Delta)e_\Phi(1) + \cdots + (1{+}\ell_\Psi{+}\ell_\Delta)e_\Phi(\ell_\Phi) \\
& +(\ell_\Psi{+}\ell_\Delta)e_\Psi(1) + \cdots + (1{+}\ell_\Delta)e_\Psi(\ell_\Psi) \\
& +\ell_\Delta e_\Delta(1) + \cdots + e_\Delta(\ell_\Delta); \\
area(\Gamma{\cdot}\Psi{\cdot}\Phi{\cdot}\Delta) = {} & (\ell_\Gamma{+}\ell_\Psi{+}\ell_\Phi{+}\ell_\Delta)e_\Gamma(1) + \cdots + (1{+}\ell_\Psi{+}\ell_\Phi{+}\ell_\Delta)e_\Gamma(\ell_\Gamma) \\
& +(\ell_\Psi{+}\ell_\Phi{+}\ell_\Delta)e_\Psi(1) + \cdots + (1{+}\ell_\Phi{+}\ell_\Delta)e_\Psi(\ell_\Psi) \\
& +(\ell_\Phi{+}\ell_\Delta)e_\Phi(1) + \cdots + (1{+}\ell_\Delta)e_\Phi(\ell_\Phi)) \\
& +\ell_\Delta e_\Delta(1) + \cdots + e_\Delta(\ell_\Delta).
\end{aligned}$$

Elementary calculation now shows that

$$\Big[area(\Gamma{\cdot}\Phi{\cdot}\Psi{\cdot}\Delta) - area(\Gamma{\cdot}\Psi{\cdot}\Phi{\cdot}\Delta)\Big] \;=\; \Big[area(\Phi{\cdot}\Psi) - area(\Psi{\cdot}\Phi)\Big]. \qquad \square$$

The Optimality of Schedule Σ. We focus only on step 2 of Σ because, as noted earlier, every schedule for $(\mathcal{G}' \Uparrow \mathcal{G}'')$ begins by executing s and ends by executing t. We validate each of the salient properties of Algorithm SP-AREA.

Lemma 5. *The nodes inside each block determined by Algorithm* SP-AREA *cannot be rearranged in any AM-schedule for $(\mathcal{G}' \Uparrow \mathcal{G}'')$.*

Proof. We proceed by induction on the structure of the decomposition tree $\mathcal{T}_\mathcal{G}$ of $\mathcal{G} = (\mathcal{G}' \Uparrow \mathcal{G}'')$. The base of the induction is when $\mathcal{T}_\mathcal{G}$ consists of a single leaf-node. The lemma is trivially true in this case, for the structure of the resulting block is mandated by the direction of the arc in the node. (In *any* schedule for the DAG $(s \to t)$, s must be computed before t.) The lemma is trivially true also when $\mathcal{G}$ is formed solely via parallel compositions. To wit, for any such DAG, removing the source and target leaves one with a set of independent nodes. The resulting eligibility profile is, therefore, a sequence of 0s: each block has size 1 and AEV 0. This means that *every* valid schedule is AM.

In general, as we decompose $\mathcal{G}$: (a) The subDAG schedule needed to process a *parallel composition* $(\mathcal{G}_1 \Uparrow \mathcal{G}_2)$ does not generate any blocks other than those generated by the schedules for $\mathcal{G}_1$ and $\mathcal{G}_2$. (b) The subDAG schedule needed to process a *series composition* can generate new blocks. To wit, consider SP-DAGs $\mathcal{G}_1$ and $\mathcal{G}_2$, with respective schedules Σ_1 and Σ_2, so that $(\Sigma_1 \cdot \Sigma_2)$ is a schedule for the serial composition $(\mathcal{G}_1 \to \mathcal{G}_2)$. Say that $(\Sigma_1 \cdot \Sigma_2)$ generates consecutive blocks, B_1 from Σ_1 and B_2 from Σ_2, where B_1's AEV is smaller than B_2's. Then blocks B_1 and B_2 are merged—via concatenation—by $(\Sigma_1 \cdot \Sigma_2)$ into block $B_1 \cdot B_2$. For instance, the indicated scenario occurs when B_1 is the last block of Σ_1 and B_2 is the first block of Σ_2. Assuming, for induction, that the nodes in each of B_1 and B_2 cannot be rearranged, we see that the nodes inside $B_1 \cdot B_2$ cannot be reordered. That is, every node of $\mathcal{G}_1$—including those in B_1—must be executed before any node of $\mathcal{G}_2$—including those in B_2. $\square$

Lemma 5 tells that the nodes inside each block generated by Algorithm SP-AREA cannot be reordered. We show now that blocks can also not be subdivided. Note that this will also preclude merging blocks in ways that violate blocks' indivisibility.

Lemma 6. *If a schedule Σ for a* DAG $(\mathcal{G}' \Uparrow \mathcal{G}'')$ *subdivides a block (as generated by Algorithm* SP-AREA*), then Σ is not AM.*

Proof. Assume, for contradiction, that there exists an AM schedule Σ for $\mathcal{G} = (\mathcal{G}' \Uparrow \mathcal{G}'')$ that subdivides some block A of Σ' into two blocks, B and C. (Our choice of Σ' here clearly loses no generality.) This subdivision means that within Σ, B and C are separated by some sequence D.

We claim that Σ's AREA-maximality implies that $AEV(C) > AEV(B)$. Indeed, by construction, we know that

$$(\forall j \in [1, \ell_A - 1]) \left[\frac{1}{\ell_A} SUM\,(A, 1, \ell_A) \;>\; \frac{1}{j} SUM\,(A, 1, j) \right].$$

Therefore, we have

$$\frac{1}{\ell_A} SUM\,(A, 1, \ell_A) \;>\; \frac{1}{\ell_B} SUM\,(A, 1, \ell_B)\,,$$

which means that

$$\ell_B \sum_{i=1}^{\ell_A} e_A(i) \;>\; \ell_A \sum_{i=1}^{\ell_B} e_A(i).$$

It follows, noting that $\ell_C = \ell_A - \ell_B$, that

$$\ell_B \sum_{i=\ell_B+1}^{\ell_A} e_A(i) \;>\; (\ell_A - \ell_B) \sum_{i=1}^{\ell_B} e_A(i).$$

This, finally, implies that $AEV(C) > AEV(B)$.

Now we are in trouble: Either of the following inequalities,

$$(a)\ [AEV(D) < AEV(C)] \quad \text{or} \quad (b)\ [AEV(D) \geq AEV(C)] \tag{3.3}$$

would allow us to increase Σ's AREA, thereby contradicting Σ's alleged AREA-maximality! If inequality (3.3(a)) held, then Lemmas 3 and 4 would allow us to increase Σ's AREA by interchanging blocks D and C. If inequality (3.3(b)) held, then because $[AEV(C) > AEV(B)]$, we would have $[AEV(D) > AEV(B)]$. But then Lemmas 3 and 4 would allow us to increase Σ's AREA by interchanging blocks B and D. We conclude that schedule Σ cannot exist. □

We summarize the results of this section in the following result.

Theorem 2. *Let $\mathcal{G}'$ and $\mathcal{G}''$ be disjoint SP-*DAG*s that, respectively, admit AM-schedules Σ' and Σ'', and have n' and n'' nontargets. If we know the block decompositions of Σ' and Σ'', ordered by AEV, then Algorithm* SP-AREA *determines, within time $O(n'+n'')$ an AM-schedule for $\mathcal{G} = (\mathcal{G}' \Uparrow \mathcal{G}'')$.*

Proof. By Lemma 6, any AM schedule for $\mathcal{G}$ must be a permutation of the blocks obtained by decomposing Σ' and Σ''. Assume, for contradiction, that some AM schedule Σ^* is not obtained by selecting blocks in decreasing order of AEV. There would then exist (at least) two blocks, A and B, that appear consecutively in Σ^*, such that

$AEV(A) < AEV(B)$. We would then have $SUM(A)/\ell_A < SUM(B)/\ell_B$, so by Lemma 3, $area(A \cdot B) > area(B \cdot A)$. Let C (resp., D) be the concatenation of all blocks that come before A (resp., after B) in Σ^*. An invocation of Lemma 4 shows that Σ^* cannot be AM, contrary to assumption.

Timing: Because the blocks obtained by the decomposition are already ordered, completing $\mathcal{G}$'s schedule requires only merging two ordered sequences of blocks, which can be accomplished in linear time. □

Observation 1. *It is worth noting that Algorithm $\mathcal{A}_{\text{SP-DAG}}$'s decomposition phase, as described in Section 3.2, may require time quadratic in the size of $\mathcal{G}$. However, we use Algorithm* SP-AREA *in a recursive manner, so the decomposition for an SP-*DAG $\mathcal{G}$ *is simplified by the algorithm's (recursive) access to the decompositions of $\mathcal{G}$'s sub*DAGs*.*

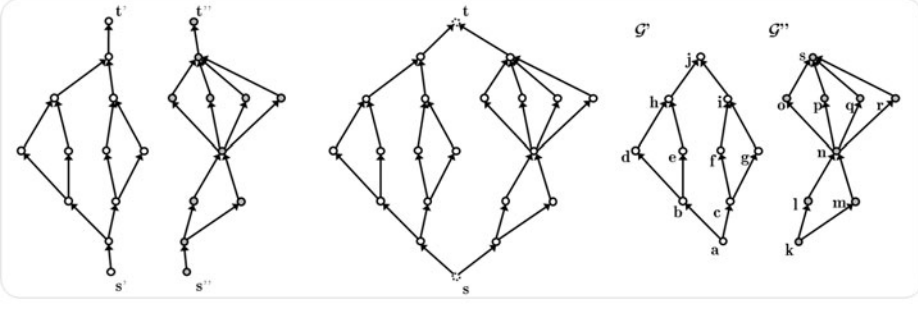

Fig. 3. An example of composition

An Example. The DAG $\mathcal{G}$ of Fig. 3(center) is the parallel composition of the DAGs of Fig. 3(left). ($\mathcal{G}$ appears at the root of $\mathcal{T}_{\mathcal{G}}$ in Fig. 2.) By removing nodes s and t from $\mathcal{G}$, we obtain the disjoint DAGs $\mathcal{G}'$ and $\mathcal{G}''$ of Fig. 3(right). We note the AM-schedules:

$$\Sigma' = \langle a, b, c, d, e, f, g, h, i \rangle \text{ for } \mathcal{G}'; \quad \Sigma'' = \langle k, l, m, n, o, p, q, r \rangle \text{ for } \mathcal{G}''.$$

We use $\mathcal{G}'$ and $\mathcal{G}''$ to illustrate Algorithm SP-AREA. *For schedule* Σ': $\Pi(\Sigma') = \langle 2, 2, 2, 0, 1, 0, 1, 0, 1 \rangle$, and profile $AVG(\Sigma') = \langle 2, 2, 2, 3/2, 7/5, 7/6, 8/7, 1, 1 \rangle$. The position of the maximum element of $AVG(\Sigma')$ is 3 (choosing the rightmost element in case of ties), so the first block is $\langle a, b, c \rangle$, with AEV 2. Continuing: the new eligibility profile is $\langle 0, 1, 0, 1, 0, 1 \rangle$, the average eligibility profile is $\langle 0, 1/2, 1/3, 1/2, 2/5, 1/2 \rangle$. The maximum is the last element, so the new block is $\langle d, e, f, g, h, i \rangle$, with AEV $1/2$. *For schedule* Σ'': $\Pi(\Sigma'') = \langle 2, 0, 1, 4, 0, 0, 0, 1 \rangle$, and $AVG(\Sigma'') = \langle 2, 1, 1, 7/4, 7/5, 7/6, 1, 1 \rangle$. The maximum is at the first element, so the first block is $\langle k \rangle$, with AEV 2. The next eligibility profile is $\langle 0, 1, 4, 0, 0, 0, 1 \rangle$, and the average eligibility profile is $\langle 0, 1/2, 5/3, 5/4, 1, 5/6, 1 \rangle$, which has its maximum at the 3rd element. The new block is $\langle l, m, n \rangle$, with AEV $5/3$. The new average eligibility profile is $\langle 0, 0, 0, 1/4 \rangle$, so the last block is $\langle o, p, q, r \rangle$ with AEV $1/4$. Thus, the two schedules are split into five blocks:

Σ' Blocks	AEV
$\langle a, b, c\rangle$	2
$\langle d, e, f, g, h, i\rangle$	1/2

Σ'' Blocks	AEV
$\langle k\rangle$	2
$\langle l, m, n\rangle$	5/3
$\langle o, p, q, r\rangle$	1/4

The AM-schedule obtained by ordering the blocks in order of decreasing AEV is $\langle a, b, c, k, l, m, n, d, e, f, g, h, i, o, p, q, r\rangle$.

3.3 The Timing of Algorithm $\mathcal{A}_{\text{SP-DAG}}$

Let $\mathcal{G}$ be an N-node SP-DAG, and let $T(N)$ be the time that Algorithm $\mathcal{A}_{\text{SP-DAG}}$ takes to find an AM-schedule Σ for $\mathcal{G}$, plus the time it takes to decompose Σ into (indivisible) ordered blocks. Let us see what goes into $T(N)$.
(1) Algorithm $\mathcal{A}_{\text{SP-DAG}}$ invokes [27] to decompose $\mathcal{G}$ in time $O(N)$.
(2) The algorithm finds an AM-schedule Σ for $\mathcal{G}$ by recursively unrolling $\mathcal{G}$'s decomposition tree $\mathcal{T}_{\mathcal{G}}$.

($2.a$) If $\mathcal{G}$ is a series composition ($\mathcal{G}' \rightarrow \mathcal{G}''$) of an n-node SP-DAG $\mathcal{G}'$ and an $(N-n+1)$-node SP-DAG $\mathcal{G}''$, then, by induction, $T(N) = T(N-n+1)+T(n)+O(N)$ for some $n \in [2, N-1]$. (The "extra" node is the merged source of $\mathcal{G}'$ and target of $\mathcal{G}''$.) The algorithm: (i) recursively generates AM-schedules, Σ' for $\mathcal{G}'$ and Σ'' for $\mathcal{G}''$, in time $T(N-n+1)+T(n)$; (ii) generates Σ by concatenating Σ' and Σ'' in time $O(N)$; (iii) decomposes Σ into blocks in time $O(N)$, using the $O(N)$ blocks obtained while generating Σ' and Σ''.

($2.b$) If $\mathcal{G}$ is a parallel composition ($\mathcal{G}' \Uparrow \mathcal{G}''$) of an $(n+1)$-node SP-DAG $\mathcal{G}'$ and an $(N-n+1)$-node SP-DAG $\mathcal{G}''$, then, by induction, $T(N) = T(N-n+1)+T(n+1)+O(N)$, for some $n \in [2, N-2]$. (The two "extra" nodes are the merged sources and targets of $\mathcal{G}'$ and $\mathcal{G}''$.) The algorithm: (i) recursively generates AM-schedules, Σ' for $\mathcal{G}'$ and Σ'' for $\mathcal{G}''$, in time $T(N-n+1)+T(n)$; (ii) generates Σ in time $O(N)$ using Algorithm SP-AREA; (iii) decomposes Σ into blocks in time $O(N)$, by taking the union of the blocks obtained while generating Σ' and Σ''. (Recall that the parallel composition does not generate additional blocks.)

Overall, then, $T(N) = O(N^2)$, as claimed. □

4 Conclusion

In furtherance of our extension of IC-Scheduling [4,5,6,8,21] to a scheduling paradigm that applies to all DAGs, we have expanded our initial study [7] of AREA-maximizing (AM) DAG-scheduling so that we can now find optimal schedules of all series-parallel DAGs efficiently. We are in the process of studying whether AM-scheduling shares the computational benefits of IC-Scheduling [13,20]. We are also seeking (possibly heuristic) algorithms that will allow us to *provably efficiently* find schedules that are (approximately) AREA-maximizing for arbitrary DAGs.

References

1. Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: An efficient multithreaded runtime system. In: 5th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming, PPoPP 1995 (1995)

2. Blumofe, R.D., Leiserson, C.E.: Space-efficient scheduling of multithreaded computations. SIAM J. Comput. 27, 202–229 (1998)
3. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. J. ACM 46, 720–748 (1999)
4. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Applying IC-scheduling theory to some familiar computations. In: Wkshp. on Large-Scale, Volatile Desktop Grids, PCGrid 2007 (2007)
5. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Advances in IC-scheduling theory: scheduling expansive and reductive DAGs and scheduling DAGs via duality. IEEE Trans. Parallel and Distributed Systems 18, 1607–1617 (2007)
6. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Extending IC-scheduling via the Sweep algorithm. J. Parallel and Distributed Computing 70, 201–211 (2010)
7. Cordasco, G., Rosenberg, A.L.: On scheduling DAGs to maximize area. In: 23rd IEEE Int. Symp. on Parallel and Distributed Processing, IPDPS 2009 (2009)
8. Cordasco, G., Rosenberg, A.L., Sims, M.: Accommodating heterogeneity in IC-scheduling via task fattening. In: On clustering tasks in IC-optimal DAGs, 37th Intl. Conf. on Parallel Processing, ICPP 2008 (2008) (submitted for publication)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (1999)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability. W.H. Freeman and Co., San Francisco (1979)
11. Gao, L.-X., Rosenberg, A.L., Sitaraman, R.K.: Optimal clustering of tree-sweep computations for high-latency parallel environments. IEEE Trans. Parallel and Distributed Systems 10, 813–824 (1999)
12. Gerasoulis, A., Yang, T.: A comparison of clustering heuristics for scheduling DAGs on multiprocessors. J. Parallel and Distributed Computing 16, 276–291 (1992)
13. Hall, R., Rosenberg, A.L., Venkataramani, A.: A comparison of DAG-scheduling strategies for Internet-based computing. In: Intl. Parallel and Distr. Processing Symp. (2007)
14. Hwang, K., Xu, Z.: Scalable Parallel Computing: Technology, Architecture, Programming. McGraw-Hill, New York (1998)
15. Jayasena, S., Ganesh, S.: Conversion of NSP DAGs to SP DAGs. MIT Course Notes 6.895 (2003)
16. Kondo, D., Casanova, H., Wing, E., Berman, F.: Models and scheduling mechanisms for global computing applications. In: Intl. Parallel and Distr. Processing Symp. (2002)
17. Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Lebofsky, M.: SETI@home: massively distributed computing for SETI. In: Dubois, P.F. (ed.) Computing in Science and Engineering. IEEE Computer Soc. Press, Los Alamitos (2000)
18. Kwok, Y.-K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. J. Parallel and Distributed Computing 59, 381–422 (1999)
19. Kwok, Y.-K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys 31, 406–471 (1999)
20. Malewicz, G., Foster, I., Rosenberg, A.L., Wilde, M.: A tool for prioritizing DAGMan jobs and its evaluation. J. Grid Computing 5, 197–212 (2007)
21. Malewicz, G., Rosenberg, A.L., Yurkewych, M.: Toward a theory for scheduling dags in Internet-based computing. IEEE Trans. Comput. 55, 757–768 (2006)
22. McCreary, C.L., Khan, A.A., Thompson, J., Mcardle, M.E.: A comparison of heuristics for scheduling DAGs on multiprocessors. In: 8th Intl. Parallel Processing Symp., pp. 446–451 (1994)
23. Mitchell, M.: Creating minimal vertex series parallel graphs from directed acyclic graphs. In: 2004 Australasian Symp. on Information Visualisation -, vol. 35, pp. 133–139 (2004)
24. Rosenberg, A.L.: On scheduling mesh-structured computations for Internet-based computing. IEEE Trans. Comput. 53, 1176–1186 (2004)

25. Rosenberg, A.L., Yurkewych, M.: Guidelines for scheduling some common computation-dags for Internet-based computing. IEEE Trans. Comput. 54, 428–438 (2005)
26. Sarkar, V.: Partitioning and Scheduling Parallel Programs for Multiprocessors. MIT Press, Cambridge (1989)
27. Valdes, J., Tarjan, R.E., Lawler, E.L.: The recognition of series-parallel digraphs. SIAM J. Comput. 11, 289–313 (1982)