Assessing the Computational Benefits of AREA-Oriented DAG-Scheduling

Gennaro Cordasco[1], Rosario De Chiara[2], and Arnold L. Rosenberg[3]

[1] Seconda Università di Napoli, Italy
gennaro.cordasco@unina2.it
[2] Università di Salerno, Italy
dechiara@dia.unisa.it
[3] Colorado State Univ. and Northeastern Univ., USA
rsnbrg@cs.umass.edu

Abstract. Many modern computing platforms, including "aggressive" multicore architectures, proposed exascale architectures, and many modalities of Internet-based computing are "task hungry"—their performance is enhanced by always having as many tasks eligible for allocation to processors as possible. The *AREA-Oriented scheduling* (*AO-scheduling*) paradigm for computations with intertask dependencies—modeled as DAGs—was developed to address the "hunger" of such platforms, by executing an input DAG so as to render tasks eligible for execution quickly. AO-scheduling is a weaker, but more robust, successor to *IC-scheduling*. The latter renders tasks eligible for execution maximally fast—a goal that is not achievable for many DAGs. AO-scheduling coincides with IC-scheduling on DAGs that admit optimal IC-schedules—and optimal AO-scheduling is possible *for all* DAG*s*. The computational complexity of optimal AO-scheduling is not yet known; therefore, this goal is replaced here by a multi-phase heuristic that produces optimal AO-schedules for *series-parallel* DAGs but possibly suboptimal schedules for general DAGs. This paper employs simulation experiments to assess the computational benefits of AO-scheduling in a variety of scenarios and on a range of DAGs whose structure is reminiscent of ones encountered in scientific computing. The experiments pit AO-scheduling against a range of heuristics, from lightweight ones such as FIFO scheduling to computationally more intensive ones that mimic IC-scheduling's *local* decisions. The observed results indicate that AO-scheduling does enhance the efficiency of task-hungry platforms, by amounts that vary according to the availability patterns of processors and the structure of the DAG being executed.

Keywords: Area-oriented DAG-scheduling; Scheduling for task-hungry platforms.

1 Introduction

Many modern computing platforms, including "aggressive" multicore architectures [24], proposed exascale architectures [9], and many modalities of Internet-based computing [11,14,15,22], are "task hungry"—their performance is enhanced by always having as many tasks eligible for allocation to processors as possible. In earlier work, we developed the master-worker *IC-scheduling* paradigm for computations with intertask dependencies—modeled as DAGs—to address the "hunger" of such platforms

[3,5,18,19,22,23]. IC-schedules attempt to execute an input DAG so as to render tasks eligible for execution as fast as possible, with a dual goal: (1) Prevent a computation's stalling pending the return of already allocated tasks. (2) Increase "parallelism" by enhancing the effective utilization of workers. Because many DAGs do not admit optimal IC-schedules [19], we have developed a new paradigm—*AREA-Oriented scheduling* (*AO-scheduling*)—to address this deficiency. Optimal AO-schedules—or, *AREA-max schedules*—coincide with optimal IC-schedules on DAGs that admit such schedules; and, AREA-max schedules exist *for every* DAG. AO-scheduling achieves its universal optimizability by weakening IC-scheduling's often-unachievable demand of maximizing the number of eligible tasks at *every* step of a DAG-execution to the always-achievable demand that this number be maximized *on average*. The foundations of AO-scheduling are developed for general DAGs in [6] and for series-parallel DAGs (*SP*-DAG*s*, see [10,13,20]) in [7]. Because optimal AO-scheduling may be computationally intractable, we develop in Sec. 3 a multiphase heuristic, AO, that produces AREA-max schedules for SP-DAGs but possibly suboptimal AO-schedules for general DAGs.

As with IC-scheduling, it is not clear *a priori* that AO-scheduling enhances the efficiency of executing a DAG. The enhancement of efficiency via *IC-scheduling* is verified experimentally in [4,12,17] for many families of DAGs that admit optimal IC-schedules. But, as noted earlier, many DAGs do not admit such schedules—which fact motivates AO-scheduling and the current study. The current paper adapts the methodology of [12] to *assess the potential computational benefits of AO-scheduling*. We model a "task-hungry" computing platform as a stream of task-seeking workers that arrive according to a random process. We focus on two populations of DAGs:

1. We study AREA-max schedules for randomly constructed *SP*-DAG*s*.
2. We study the AO-schedules produced by our multiphase heuristic AO for DAGs that are random compositions of small "building-block" DAGs [19]. The DAGs we schedule model computations each of whose subcomputations has the structure of *an expansion* (as in a search tree), *a reduction* (as in an accumulation), *a parallel-prefix* (a/k/a *scan*), *an all-to-all communication* (as in a "gossip"). Such compositions have structures reminiscent of ones that arise in scientific computing.

Thus, all of the AO-schedules that we study can be constructed *efficiently* from the DAGs being scheduled. We simulate executing each generated DAG on our platform model: (a) using the schedule produced by AO and (b) using schedules produced by several popular heuristics, ranging from lightweight ones such as a version of CONDOR's FIFO scheduling [1] to computationally intensive ones that mimic IC-scheduling's *local* decisions. The results we observe indicate that, statistically, *AO-scheduling does significantly enhance the efficiency of task-hungry platforms*, by amounts that vary according to the availability patterns of processors and the structure of the DAG being executed.

2 Background

A. Basics. We study computations that are described by DAGs. Each DAG $\mathcal{G}$ is specified by two sets: its *nodes* $V_{\mathcal{G}}$, each denoting a *task*,[1] and its (directed) *arcs* $A_{\mathcal{G}}$. Each arc

[1] We henceforth refer to DAG *tasks*, rather than *nodes*, to emphasize our computational focus.

$(u \rightarrow v)$ denotes a *dependency* between parent-task u and child-task v. When one executes $\mathcal{G}$, task $v \in V_{\mathcal{G}}$ becomes *eligible* (for execution) only after all of its parents have been executed; hence, all sources (= parentless tasks) are eligible at the beginning of an execution. The goal is to render all sinks (= childless tasks) eligible. Informally, a *schedule* Σ for $\mathcal{G}$ is a rule for selecting which eligible task to execute at each step of an execution; formally, Σ is a *topological sort* of $\mathcal{G}$ (see [8]).

B. Quality metrics. We measure the quality of a schedule Σ for an n-task DAG $\mathcal{G}$ via the rate at which Σ renders tasks of $\mathcal{G}$ eligible: the faster, the better. To this end, we define $E_{\Sigma}(t)$, *the quality of* Σ *at step* t, as the number of tasks that are eligible after Σ has executed t tasks[2] ($t \in [1, n]$). *IC-scheduling strives* to execute $\mathcal{G}$'s tasks in an order that maximizes $E_{\Sigma}(t)$ *at every step* $t \in [1, n]$ *of the execution*; formally: $(\forall t \in [1, n])\ E_{\Sigma^*}(t) = \max_{\Sigma \text{ a schedule for } \mathcal{G}} \{E_{\Sigma}(t)\}$. *AO-scheduling strives* to find a schedule Σ for $\mathcal{G}$ of maximum *AREA*, where $AREA(\Sigma) \overset{\text{def}}{=} E_{\Sigma}(0) + E_{\Sigma}(1) + \cdots + E_{\Sigma}(n)$. (Note the analogy with Riemann sums.) For such an *AREA-max schedule* $\Sigma^{\star}$,

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

Many simple DAGs, even tree-DAGs[3] and SP-DAGs, do not admit optimal IC-schedules [19]. Thus, even well-structured DAGs benefit from the universality of optimal AO-scheduling.

3 Finding Good AO-Schedules Efficiently

A. The complexity of AREA-maximization. Every DAG admits an AREA-max schedule. If a DAG admits an optimal IC-schedule, then every such schedule is AREA-max, and vice-versa. This good news from [6] is tempered by the fact that we do not yet know how to find AREA-max schedules for arbitrary DAGs *efficiently*. Indeed, a result in [6] makes it plausible that one *cannot* always produce AREA-max schedules efficiently. Fortunately, efficient AO-scheduling algorithms exist for several significant families of DAGs [6,7]. Most significantly (for our study): *One can find an AREA-max schedule for any* n*-task SP-*DAG $\mathcal{G}$ *in time* $O(n^2)$. The validating algorithm in [7] exploits $\mathcal{G}$'s structure by (a) decomposing $\mathcal{G}$ to produce a tree $\mathcal{T}_{\mathcal{G}}$ that exposes $\mathcal{G}$'s series-parallel structure, (b) recursively unrolling $\mathcal{T}_{\mathcal{G}}$ to craft an AREA-max schedule for $\mathcal{G}$.

B. Toward efficient AO-scheduling. We develop a four-phase heuristic AOH that AO-schedules any n-task DAG efficiently—specifically, in time $O(n^2)$. Given a DAG $\mathcal{G}$:
Phase 1: *Find* $\mathcal{G}$*'s transitive skeleton* $\mathcal{G}'$. Removing all shortcut arcs from $\mathcal{G}$ reduces the overall complexity of finding an AO-schedule. Formally, $\mathcal{G}'$ is a smallest sub-DAG of $\mathcal{G}$ that shares $\mathcal{G}$'s task-set and transitive closure. Easily, $\mathcal{G}$ and $\mathcal{G}'$ share all of their AREA-max schedules, because removing shortcuts does not impact tasks' dependencies.
Phase 2: *Convert* $\mathcal{G}'$ *to an SP-*DAG $\sigma(\mathcal{G}')$. Invoke an *SP-ization algorithm* to convert $\mathcal{G}'$ to $\sigma(\mathcal{G}')$. Choose an algorithm that: (a) maintains in $\sigma(\mathcal{G}')$ all of the intertask dependencies from $\mathcal{G}'$; (b) (approximately) retains the degree of parallelism inherent in $\mathcal{G}'$ (which precludes, e.g., having $\sigma(\mathcal{G}')$ simply linearize $\mathcal{G}'$); (c) operates within time $O(n^2)$. $\sigma(\mathcal{G}')$ will generally contain *extra* tasks that are not tasks of $\mathcal{G}'$; see Fig. 1. SP-ization algorithms that fit our requirements appear in, e.g., [10,13,20].

[2] $[a, b]$ denotes the set of integers $\{a, a + 1, \ldots, b\}$.
[3] A *tree*-DAG is a DAG that remains acyclic even ignoring arc orientations.

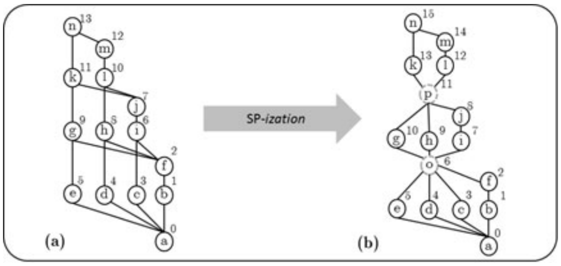

Fig. 1. A sample SP-ization of the LU-decomposition DAG. The task-numbering describes an AREA-max schedule

Phase 3: *Find an AREA-max schedule Σ' for $\sigma(\mathcal{G}')$*, using, e.g., the algorithm of [7].
Phase 4: *"Filter" the AREA-max schedule Σ' for $\sigma(\mathcal{G}')$ to obtain the AO-schedule Σ for $\mathcal{G}$*. "Filtering" Σ' removes the extra tasks added by the SP-ization algorithm. For each extra task u, we assign a *priority* to u's parents in $\mathcal{G}$, that equals the priority of u in Σ'. Σ then schedules equal-priority tasks of $\mathcal{G}$ greedily, by their *yield*—the number of eligible tasks their execution produces.

We illustrate this heuristic on the LU-decomposition DAG $\mathcal{G}$ of Fig. 1(a). $\mathcal{G}$ contains no shortcut arcs, so $\mathcal{G}' = \mathcal{G}$. One possible SP-ization $\sigma(\mathcal{G}')$ of $\mathcal{G}'$ appears in Fig. 1(b); note the two extra tasks o and p. The SP-DAG scheduling algorithm of [7] produces the Area-max schedule $\Sigma' = (a, b, f, c, d, e, o, i, j, g, h, p, l, k, m, n)$ for $\sigma(\mathcal{G}')$; note the task-numbering in Fig. 1(b). Finally, we obtain an AO-schedule Σ for $\mathcal{G}$ by simply removing tasks o and p from Σ'.

4 Experiments to Assess the Quality of AOH

4.1 Experimental Design

A. Overview. We randomly generate DAGs that share structural characteristics with a variety of "real" computation-DAGs, especially ones encountered in scientific computing. We craft five schedules for each generated DAG, one using an AO-scheduling heuristic based on AOH and four using heuristics that represent a range of sophistication and computational intensiveness. We compare the five schedules using two metrics:

1. *Batched makespan*. We overlay our DAG-scheduling with a probabilistic model that specifies the arrival patterns of "hungry" workers and the execution time of each allocated task;
2. *AREA*. We seek to verify or refute the positive correlation between larger AREA and smaller makespan observed on small examples.

B. The DAGs that we execute. We generate DAGs randomly from two populations:

1. *Random n-task SP-DAGs*. We generate a random binary tree $\mathcal{T}$ and randomly (50% uniform choice) designate each internal node of $\mathcal{T}$ either a series- or a parallel-composition node. We then view $\mathcal{T}$ as the composition tree $\mathcal{T}_{\mathcal{G}}$ of a SP-DAG $\mathcal{G}$.

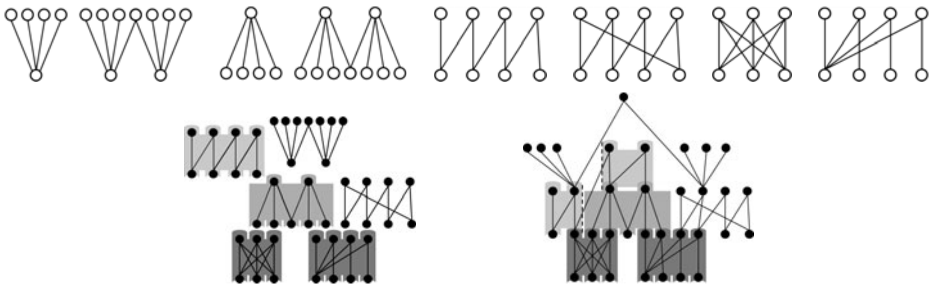

Fig. 2. (Top) A sequence of eight BBBs (All arcs point upward). (Bottom) Composing six BBBs into a LEGO®-DAG.

2. *Random n-task LEGO®*-DAG*s* (named for the toy). We select a sequence of *Bipartite Building Blocks* (*BBB*s) (see Fig. 2 (top)), randomized according to both size and structure. We *compose* the BBBs in the manner described in [19] and depicted schematically in Fig. 2 (bottom).

C. The AO **heuristic.** We compare all schedulers to the *AO-scheduler* AO, which produce an AO-schedule for an n-task DAG $\mathcal{G}$ in time $O(n^2)$, in one of two ways:

1. If $\mathcal{G}$ is an SP-DAG, then AO uses the algorithm of [7] to craft an AREA-max schedule for $\mathcal{G}$.
2. If $\mathcal{G}$ is *not* an SP-DAG, then AO uses the multi-phase heuristic AOH of Sec. 3 to craft an AO-schedule for $\mathcal{G}$.

D. The competing schedulers. The heuristics that compete against AO differ in the data structure used to store $\mathcal{G}$'s currently eligible tasks. (See [8] for specifications of the data structures.) All load newly eligible tasks in random order.

1. The FIFO (*first-in, first-out*) scheduler organizes $\mathcal{G}$'s eligible tasks in a FIFO queue. FIFO is, essentially, the scheduler used by systems such as Condor [1].
2. The LIFO (*last-in, first-out*) scheduler organizes $\mathcal{G}$'s current eligible tasks in a stack.
3. The STATIC-GREEDY scheduler organizes tasks that are newly rendered eligible in a MAX-priority queue whose entries are (partially) ordered by *outdegree*.
4. The DYNAMIC-GREEDY scheduler organizes tasks that are newly rendered eligible in a structure that is (partially) ordered by tasks' *yields*. The *yield* of an eligible task v at time t is the number of non-eligible tasks that would be rendered eligible if v were executed at this step. DYNAMIC-GREEDY thus makes the same *local* decisions as does an optimal IC-scheduler (when one exists)—but it lacks the latter's tie-breaking foresight. *Complexity:* The following all take time $O(n)$: (a) initializing the list, (b) serving a "hungry" worker (using EXTRACT-MAX), (c) adding the new eligible tasks after an outdegree-d task v has completed. Thus, DYNAMIC-GREEDY and AO have proportional worst-case computational complexities.

E. The computing platform. Our batched-makespan experiment demands a model of the computing platform in which DAGs are executed. We employ a master-centric model similar to that in [12], the IC-scheduling precursor to this paper. We model the simulated execution of a DAG $\mathcal{G}$ by scheduling heuristic[4] HEUR via a discrete time-ordered queue of "events." Each "event" is represented by the not-yet-executed *residue* of $\mathcal{G}$, together with the current set of eligible tasks, organized as mandated by HEUR. The initial residue of $\mathcal{G}$ is $\mathcal{G}$ itself; the initial set of eligible tasks comprises $\mathcal{G}$'s sources. The transition from one "event" to its successor proceeds as follows:

1. The master polls the available "hungry" workers and allocates (using HEUR's priority measure) one eligible task of $\mathcal{G}$ each to some of these workers. (Only some "hungry" workers get served when there are not enough eligible tasks to serve them all.) Once allocated, a task is no longer eligible.
2. Independently, and asynchronously, served workers execute their tasks.
3. When a worker completes its allocated task, call it v, the master removes v from the current residue of $\mathcal{G}$ and adds the tasks that v's completion renders eligible to the set of current eligible tasks, in the manner mandated by HEUR.

Our model for the computing platform is completed by specifying two probability distributions, one specifying the arrival pattern of "hungry" workers and one specifying tasks' completion times.

• *Worker arrivals.* At each step t of a simulated DAG-execution, we generate a number c_t of "hungry" workers that are seeking tasks, from an exponential distribution with rate parameters $\lambda \in \{1,\ 1/2,\ 1/4,\ 1/8,\ 1/16,\ 1/32\}$, so there are $\mu = 1/\lambda$ workers per step on average.
• *Task execution times.* The master does not know which workers are more powerful than others, so it treats all workers equally. Possible differences in worker power are modeled via the distribution of task execution times. The execution time, t, of an allocated task v is chosen randomly from the "positive half" of a normal distribution with mean 1. We have studied two distributions, one with standard deviation 0.1 and one with standard deviation 0.5. The latter parameter, in particular, allows us to observe the performance of our heuristics on platforms having a rather high level of *heterogeneity*.

4.2 Experimental Methodology

A. DAG **sizes.** Our experiments simulate the execution of DAGs that range in size from 200 tasks to 4000 tasks. We thereby observe the performance of our heuristics on DAGs that model subcomputations to those that model full computations.

B. BBB structures and sizes. We generate three families of LEGO®-DAGs by composing BBBs whose structures are chosen uniformly among the six structures depicted in Fig. 2 (top) and whose sizes are selected randomly from three distributions: (a) a *uniform* distribution from the set $[2, 20]$; (b) an *exponential* distribution and (c) a *harmonic* distribution; both of the latter generate BBBs having 10 tasks on average.

[4] HEUR $\in$ {AO, FIFO, LIFO, STATIC-GREEDY, DYNAMIC-GREEDY}.

C. Experimental procedures. Both our makespan-comparison and AREA-comparison experiments involved executing four sets of 45 DAGs each: 5 DAGs of each size $n \approx 200, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000$. Each trial executed each DAG 100 times. The results on like-sized DAGs were averaged; we used the means and variances of the schedulers' makespans or AREAs for our comparisons and analyses.

4.3 Experimental Results and Discussion

A. Makespan-comparison. This experiment evaluates AO-scheduling in a simulated "real" computational setting. We considered 120 different test settings, each identified by a triple (D, H, μ). D specifies the class of DAGs:

$D \in \{$SP-DAGs, (Uniform or Exponential or Harmonic) LEGO®-DAGs$\}$;

H specifies the scheduling heuristic:

$H \in \{$AO, FIFO, LIFO, STATIC-GREEDY, DYNAMIC-GREEDY$\}$;

μ specifies the mean number of "hungry" workers per step: $\mu \in \{1, 2, 4, 8, 16, 32\}$. For this experiment, the standard deviation of task-execution time is fixed at 0.1.

We compared the performance of heuristic AO against its competitors via the *timing-ratios* $T(H) \div T(\text{AO})$, where $T(H)$ denotes the simulated makespan observed using heuristic $H \in \{$FIFO, LIFO, STATIC-GREEDY, DYNAMIC-GREEDY$\}$. Note that larger values of the ratio favor heuristic AO. We present both means and 95% confidence intervals of the results in Fig. 3. To enhance legibility, we present a separate plot for each value of D and μ; to conserve space, we present results for random SP-DAGs and only *uniformly* distributed LEGO®-DAGs. (The three families of LEGO®-DAGs exhibit very similar behaviors; see [2].) In each plot, the X-axis indicate the size of DAG instances, while the Y-axis indicates the timing-ratios for AO's four competitors.

Our first observation is that AO-scheduling decreases makespans only for "intermediate" arrival rates μ of "hungry" workers. This is not surprising. When workers arrive very infrequently, i.e., $\mu \approx 1$, *any* heuristic will require roughly n steps to execute an n-task DAG; one observes this in the top plots of Fig. 3. At the other extreme, when workers "flood" the system, there is so much "parallelism" that the only hard limitation for *any* heuristic is the length of a DAG's inherently sequential "critical path." In neither extreme does makespan depend on the scheduling heuristic. Between these extremes, though, there is a range of values of μ where the scheduling heuristic strongly influences makespan: In our trials, when $1 < \mu \leq 32$, AO always completed executing the DAG in less (simulated) time than its competitors. Importantly, we observed that:

Within a broad range of worker arrivals, the makespan of a heuristic, as exposed in Fig. 3, has a strong positive correlation with the AREAs of heuristics' schedules, as exposed in Fig. 5. In other words, schedules with higher AREAs executed DAG*s with smaller makespans.*

The *amount* of observed advantage in makespan depended on three factors: the value of μ, the size of the DAG being executed, and the family of DAGs. Several cases (e.g., $\mu = 8, 16$) show an improvement in the range of 7–12% for LEGO®-DAGs and 10–14% for SP-DAGs. Recall that AO always provides an *AREA-max* schedule for each SP-DAG but not necessarily for each LEGO®-DAG.

Comparing the performance of AO's competitors, we observe first that DYNAMIC-GREEDY always outperforms the other competitors by a considerable margin. This is

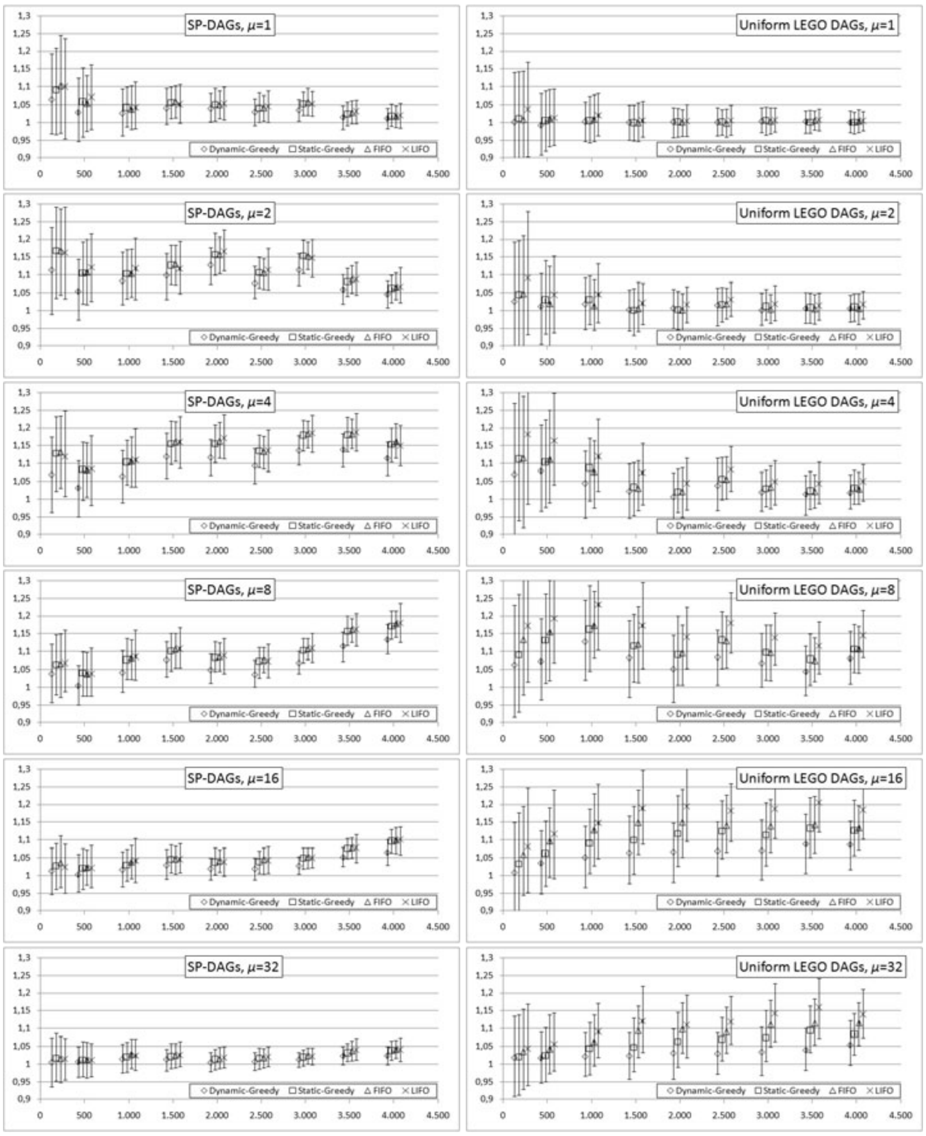

Fig. 3. Timing-ratios for (left) *random SP*-DAG*s* and (right) *Uniform LEGO*$^{\circledR}$-DAG*s* when the average number of "hungry" workers is $\mu = 1, 2, 4, 8, 16, 32$ (top to bottom).

not surprising because DYNAMIC-GREEDY makes the same *local* decision as an optimal IC-schedule. DYNAMIC-GREEDY "pays for" its superiority among the competitors by its much greater computational expense. Among the other three competitors: LIFO is always the worst heuristic; STATIC-GREEDY and FIFO perform roughly equivalently much of the time, but STATIC-GREEDY sometimes significantly outperforms FIFO; cf., (LEGO$^{\circledR}$-DAGs, STATIC-GREEDY, 16) vs. (LEGO$^{\circledR}$-DAGs, FIFO, 16). For SP-DAGs, the three static heuristics: STATIC-GREEDY, FIFO, and LIFO, do not differ substantially.

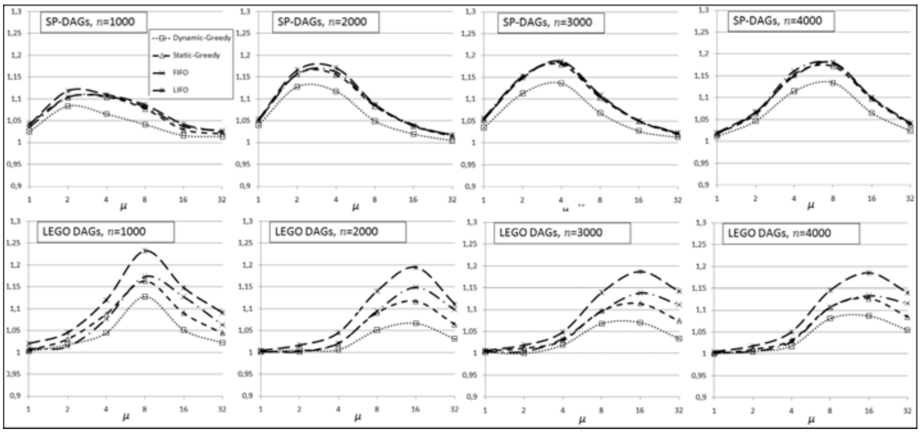

Fig. 4. Timing-ratios for random (top) SP-DAGs, (bottom) Uniform LEGO®-DAGs of different sizes. Left-to-right: 1000 tasks, 2000 tasks, 3000 tasks, 4000 tasks. The X-axes indicate the average number of "hungry" workers at each poll.

The impact of worker-arrival rates. We have just noted that average worker-arrival rate μ influences the performance of AO relative to its competitors. In order to refine this observation, with an eye toward better understanding how μ influences the relative qualities of schedules, we provide, in Fig. 4, plots that show the performance advantage of AO (in terms of timing-ratios) as a function of μ; the values of μ appear logarithmically along the X-axes of the plots. The most notable similarity in the plots is that all are unimodal: as μ increases, AO's relative performance improves up to a unique peak and thereafter degrades. Moreover, AO's peak advantage is comparable for all DAGs of similar sizes, both LEGO®-DAGs and SP-DAGs. However, there are also notable differences in the plots, particularly between LEGO®-DAGs as a class and SP-DAGs as a class. Specifically, we observe the advantage of AO peaking at a higher value of μ for LEGO®-DAGs than for SP-DAGs. Moreover, while the value of μ that maximizes AO's advantage for SP-DAGs grows roughly linearly with DAG-size (the maximizing values range from 2 for 1000-task DAGs to 8 for 4000-task DAGs), this does not appear to happen with LEGO®-DAGs (for which the maximizing values start at 8, for 1000-task DAGs, then jump to 16 for the other three DAG-sizes).

In an attempt to understand why our two DAG families' makespans react differently to the average worker-arrival rate, we analyzed certain characteristics of DAGs from these families. Based on the data in the following table, we conjecture that *the maximizing value of μ depends on the inherent degree of parallelism in the* DAG *being executed.*

	SP-DAGS			LEGO-DAGS		
DAG-size (nodes)	DAG-size (arcs)	Normalized AREA	Critical path lenght	DAG-size (arcs)	Normalized AREA	Critical path lenght
1000	1219	70	150	2885	76	58
2000	2429	75	328	5644	132	74
3000	3666	106	411	8332	189	92
4000	4920	181	445	11114	255	119

We observe that LEGO®-DAGs have smaller critical-path lengths and higher normalized AREAs than do SP-DAGs. (The observed difference would be even larger if we used AREA-max schedules for LEGO®-DAGs rather than the often-suboptimal schedules provided by heuristic AO.) Thus, the values of normalized AREA and critical-path length suggest that LEGO®-DAGs are more "parallelizable" than SP-DAGs.

Modeling heterogeneity via large variance in task execution-times. A major motivation for the development of IC-scheduling (see [22])—hence also of AO-scheduling—was the observed *temporal unpredictability* of many modern computing platforms, which precludes the reliable use of classical, critical-path based, DAG-scheduling strategies as in, e.g., [16]. As noted in sources such as [14,22], we seldom know literally *nothing* quantitative about the computing platform; it is more that our knowledge is very indefinite. A basic tenet of both IC-scheduling and AO-scheduling is that one does not have to deal explicitly with the temporal unpredictability of task execution-times when scheduling a DAG—as long as one enhances the rate of rendering tasks eligible for execution. We test this tenet experimentally by allowing greater variability in task execution-times, expressed via a larger standard deviation in these times. Our primary model allows 10% standard deviation in task execution-times: a mean time of 1 time-unit/task and a standard deviation of 0.1 time-units. How would our results change if we allowed 50% deviation, by raising the allowed standard deviation to 0.5 time-units? We repeated the experiments presented in earlier sections with this larger standard deviation—with rather surprising results. Increasing the allowable standard deviation from 0.1 to 0.5—a truly significant change!—produces a negligible change in the observed advantage of heuristic AO! The observed differences in the average timing-ratios obtained with the two standard deviations in task execution-times, 0.1 and 0.5, do not exceed 0.05%. This suggests that the quality of AO-schedules—as generated by heuristic AO—relative to the four competing heuristics is virtually unaffected by both heterogeneity and temporal unpredictability in "task-hungry" platforms.

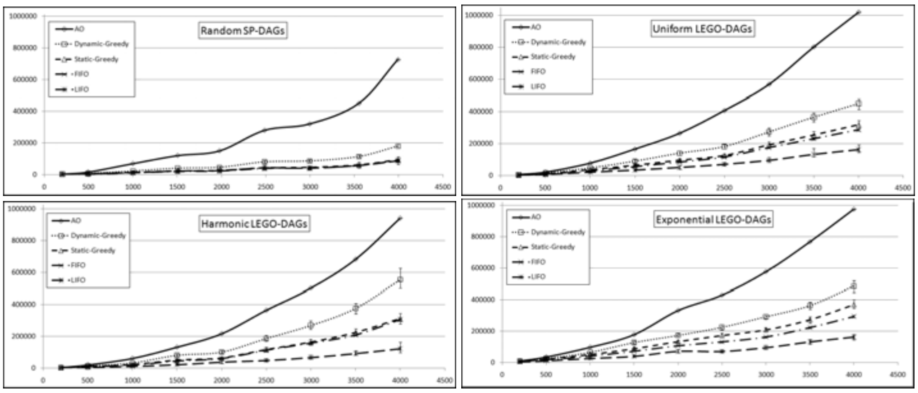

Fig. 5. AREA comparison. Clockwise from the top-left: Random SP-DAGs, Uniform LEGO®-DAGs, Exponential LEGO®-DAGs, Harmonic LEGO®-DAGs.

B. AREA-comparison. Our makespan-oriented experiment suggest that AO-scheduling, as implemented by heuristic AO, has a benign impact on computational performance. This inference has led us to wonder: (a) How do the AREAs of the schedules produced by heuristic AO compare to those of the schedules produced by its four heuristic competitors? (b) How well do the observed differences in makespan of the four competitors track the differences in the AREAs of schedules produced by these heuristics?

We have studied these questions via an experiment that compared the AREAs of AO's schedules to those of schedules for the same DAGs that are produced by the four competing heuristics. We considered 20 test settings, each characterized by the class of DAGs considered and the scheduling heuristic analyzed. For each setting, we executed each DAG 100 times. Fig. 5 presents the mean observed AREA values, as well as their ranges $[\min, \max]$. There is one plot for different-size DAGs from each family indicated in the caption; the sizes of DAG-instances appear along the X-axes. Notable among the observed results: As expected, the schedules provided by heuristic AO for SP-DAGs, being AREA-max, always have the largest AREAs. But, not obviously, the AREA-superiority of AO's schedules persist for general DAGs—which suggests that AO produces high-AREA schedules. This suggestion is reinforced by the fact that difference between the AREAs of AO's schedules and those produced by the competitor heuristics grows more than linearly with the size of the DAG being scheduled.

C. Summing up the experiments. When we consider the results of both the makespan-comparison experiment and the AREA-comparson experiments, as exposed in Figs. 3, 4, and 5, we observe three factors that support our hypothesis that *there is a strong positive correlation between the AREA of a schedule and its makespan.*

- *The schedules provided by all five heuristics*—AO and its four competitors—*have the same relative ordering in the makespan-comparison experiment as in the AREA-comparison experiment.*
- *When the three lightweight competitor heuristics,* FIFO, LIFO, *and* STATIC-GREEDY, *produce schedules for SP-*DAG*s, these schedules have roughly the same AREAs and roughly the same makespans.*
- *The ratio of the AREAs of* AO*'s schedules to those produced by the four competitor heuristics is roughly* 4 *for SP-*DAG*s and only roughly* 2 *for LEGO®-*DAG*s. This correlates positively with the relative improvements in makespan for the same families of* DAG*s.*

In the interest of full disclosure, we do not yet know if the observed differences between results for SP-DAGs and for LEGO®-DAGs are *inherent*, due to the different characteristics of such DAGs (such as degree of inherent parallelism), or algorithmic, due to a possible loss of quality introduced by the heuristics of Sec. 3.

5 Conclusion

Our contributions. Building on the novel *AREA-oriented* (*AO*) scheduling paradigm of [6], we have assessed the quality of AO-schedules for a variety of artificially generated DAGs whose structures are reminiscent of those encountered in real scientific computations. AO-schedules strive to maximize the rate at which DAG-tasks are rendered eligible for allocation to workers with the hope that this will make such schedules computationally advantageous for modern “task-hungry” computing platforms, such as Internet-based, aggressively multi-core, and exascale platforms. Our assessment pitted our new efficient AO-scheduling heuristic AO against four common scheduling heuristics that represent different points in the sophistication-complexity space of schedulers. We have shown via simulation experiments that:

– The schedules produced by AO have *AREAs that are closer to optimality* than are the schedules produced by the four competing heuristics.
– The schedules produced by AO have *lower makespans* than do the four competing heuristics, based on a probabilistic model of the computing platform and the DAG-executing process.

Importantly, our experiments suggest a strong positive relationship between the AREA of a DAG-schedule and the schedule’s performance, as measured by its makespan.

We view the new scheduling heuristic, AO, which operates within time quadratic in the size of the DAG being scheduled, as an important advance because AO *represents the first efficient scheduling mechanism that provably enhances the rate of producing allocation-eligible tasks for* every *computation-*DAG [6].

Finally our experiments have a high degree of *robustness*. The demonstrated computational benefits of AO-scheduling persist even when the “task-hungry” platforms have a high degree of heterogeneity and/or a high degree of temporal unpredictability.

Where we are going. Our demonstration of the computational benefits of AO-scheduling reinforces the importance of two algorithmic questions.

– Does there exist an algorithm for crafting AREA-max schedules for SP-DAGs that is more efficient than the quadratic-time algorithm of [7]?
– Does there exist an algorithm for SP-izing arbitrary DAGs whose use would improve the AREAs and makespans of schedules provided by heuristic AO?

Additionally, the “success” of our experiments suggests the desirability of assessing the value of AO-scheduling via experiments with *real* computations rather than simulated artificial ones. We hope to follow this path in the not-distant future, beginning with experiments using actual traces.

Acknowledgement. The research of A. Rosenberg was supported in part by NSF Grant CNS-0905399. The authors thank A. Gonzáles-Escribano and his team for providing access to their DAG-SP-ization code.

References

1. The Condor Project, Univ. of Wisconsin condor, `http://www.cs.wisc.edu/`
2. Cordasco, G., De Chiara, R., Rosenberg, A.L.: Assessing the computational benefits of AREA-oriented DAG-scheduling. Tech. Rpt., U. Salerno (2011), `http://www.isislab.it/papers/TR0711.pdf`
3. 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)
4. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Applying IC-scheduling theory to some familiar computations. Wkshp. on Large-Scale, Volatile Desktop Grids (PCGrid 2007) (2007)

5. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Extending IC-scheduling via the Sweep algorithm. J. Parallel and Distributed Computing 70, 201–211 (2010)
6. Cordasco, G., Rosenberg, A.L.:: On scheduling dags to maximize area. In: 23rd IEEE Int'l Par. and Distr. Processing Symp. IPDPS 2009 (2009)
7. Cordasco, G., Rosenberg, A.L.: Area-maximizing schedules for series-parallel dAGs. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010. LNCS, vol. 6272, pp. 380–392. Springer, Heidelberg (2010)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (1999)
9. Dongarra, J., et al.: International Exascale Software Project Roadmap. Tech. Rpt. UT-CS-10-652, Univ. Tennessee (2010)
10. González-Escribano, A., van Gemund, A., Cardeñoso-Payo, V.: Mapping unstructured applications into nested parallelism. In: Palma, J.M.L.M., Sousa, A.A., Dongarra, J., Hernández, V. (eds.) VECPAR 2002. LNCS, vol. 2565. Springer, Heidelberg (2003)
11. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure, 2nd edn. Morgan Kaufmann, San Francisco (2004)
12. Hall, R., Rosenberg, A.L., Venkataramani, A.: A comparison of dag-scheduling strategies for Internet-based computing. In: 21st IEEE Int'l Par. and Distr. Proc. Symp. (IPDPS 2007) (2007)
13. Jayasena, S., Ganesh, S.: Conversion of NSP DAGs to SP DAGs. MIT Course Notes 6.895 (2003)
14. Kondo, D., Casanova, H., Wing, E., Berman, F.: Models and scheduling mechanisms for global computing applications. Int'l Par. and Distr. Processing Symp., IPDPS 2002 (2002)
15. 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)
16. Kwok, Y.-K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys 31, 406–471 (1999)
17. 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)
18. Malewicz, G., Rosenberg, A.L.: On batch-scheduling dags for Internet-based computing. In: 11th Int'l Conf. on Parallel Computing, Euro-Par 2005 (2005)
19. Malewicz, G., Rosenberg, A.L., Yurkewych, M.: Toward a theory for scheduling dags in Internet-based computing. IEEE Trans. Comput. 55, 757–768 (2006)
20. 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)
21. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Computer and System Scis. 43, 425–440 (1991)
22. Rosenberg, A.L.: On scheduling mesh-structured computations for Internet-based computing. IEEE Trans. Comput. 53, 1176–1186 (2004)
23. Rosenberg, A.L., Yurkewych, M.: Guidelines for scheduling some common computation-dags for Internet-based computing. IEEE Trans. Comput. 54, 428–438 (2005)
24. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Computing 36(5-6), 232–240 (2010)