Scheduling under Precedence, Communication, and Energy Constraints

David Felber† Adam Meyerson‡

Abstract

We consider the problem of scheduling a set of \( n \) tasks on \( m \) processors under precedence, communication, and global system energy constraints to minimize makespan. We extend existing scheduling models to account for energy usage and give convex programming algorithms that yield essentially the same results as existing algorithms that do not consider energy, while adhering to a strict energy bound.

1 Introduction

We consider the problem of scheduling a set of \( n \) tasks on \( m \) homogeneous processors under precedence, communication, and global system energy constraints to minimize makespan. This problem is of particular importance in portable and embedded systems [21][1][20][4], where the power demands of a growing number of computationally intensive applications [11][22] outmatch the growth rate of battery energy density [16]. It is also important in high-performance systems and data centers [31] where the operational costs of powering and cooling [6][17][9] and related reliability issues from power dissipation [34][17] are substantial. Because multi-core processors are the industry’s answer to the power and thermal constraints limiting clock speed [27][8], general scheduling methods that conserve energy and minimize running times are needed to fully and efficiently use these systems.

The problem of scheduling tasks on processors to minimize the makespan (overall runtime) has a long history [5]. In the most general form, we are given a number of tasks to assign to processors, such that no processor may run more than one task at a time and the goal is to minimize the makespan (latest completion time). Distributing the tasks amongst processors typically reduces the makespan; however the problem presents several issues which make the solution non-trivial. First, a system has a finite number of processors which is typically much less than the number of tasks, implying that some tasks must be allocated to the same processor (thus potentially delaying their completion). Second, there is communication between tasks in the form of precedence constraints. When one task requires the output of another, we cannot schedule them to run in parallel on different processors. Third, there is communication between processors; the implication is that if the output of task \( i \) is required by task \( j \) which is scheduled to run on a different processor, then there will be some additional time delay \( c_{ij} \) after the completion of \( i \) for the necessary information to arrive. Note that this delay can be avoided by scheduling \( i, j \) on the same processor.

We further consider a overall bound on the total energy consumed by all tasks. Our goal is to produce a smooth tradeoff between the total energy consumption and the makespan, which is achievable by varying

---

*This material is based upon work supported by the National Science Foundation under Grant No. CCF-1016540.
†University of California at Los Angeles, Department of Computer Science. Email: dvfelber@cs.ucla.edu.
‡University of California at Los Angeles, Department of Computer Science. Email: awm@cs.ucla.edu.
the energy bound. We observe that some previous scheduling models permitted re-computation of tasks (computing the same task multiple times on different processors to save on communication delays); this is energy-wasteful and our models will assume that such re-computation does not occur. Previous work on energy often tried to reduce the energy consumption without any reduction to the makespan; this approach has some merits but essentially treats energy as a “secondary” objective rather than producing a tradeoff between energy and makespan. Our model also extends previous work by allowing the tradeoff between running time and energy to be arbitrary (but convex) and task-dependent; this makes sense in the context of tasks which heavily load different system components (i.e. processor, network card, memory) and therefore may behave differently under speed-scaling.

Problem Definition We formally state the problem as follows. We are given (1) a set \( J \) of \( n \) tasks; (2) \( m \) processors; (3) a directed acyclic graph \( G \) on \( n \) nodes representing precedence constraints between tasks; (4) a communication delay \( c_{ij} \) for each edge \((i, j)\) in \( G \); (5) a set of energy/time tradeoff functions \( e_j(d_j) \); and (6) an energy bound \( E \).

The goal is to construct a schedule \( \sigma = \{(j, I, p)\} \) where task \( j \) is assigned over time interval \( I \) to processor \( p \), such that (a) if there is an edge from \( i \) to \( j \) in \( G \), then task \( i \) must finish before \( j \) can begin; (b) each processor can only work on one task at any time; (c) \( d_j \) is the total time for which task \( j \) runs over all assigned intervals; (d) the total energy used, \( \sum_j e_j(d_j) \), is bounded by \( E \); and (e) the time at which the last task completes (the makespan) is minimized.

When the communication delays are non-zero, we restrict attention to schedules that do not migrate tasks, and require the additional constraint (f) that if \((i, j)\) is an edge in \( G \) and \( i \) and \( j \) are scheduled on different processors, then \( j \) cannot start until \( c_{ij} \) time after \( i \) has completed.

Previous Work Most previous work is experimental and considers the problem in the context of dynamic voltage scaling, for which speed (operating frequency) is approximately proportional to the voltage and for which power is approximately proportional to voltage cubed \[30\]. This work includes more recent heuristics that minimize makespan given a hard energy bound \[21\] as well as heuristics that minimize energy given hard timing constraints \[31\] \[33\] \[11\] \[20\] \[6\]. For both variations, the approach generally taken is to create an initial schedule in which tasks are scheduled at the highest speed possible, and then to reduce the speed of tasks in such a way that the schedule length does not increase.

Several heuristic approaches also incorporate mathematical programming. Rountree et al \[29\] use a linear program to bound the optimal solution from below. Kang and Ranka \[19\] and Leung et al \[22\] use heuristics to approximate the optimal solutions to integer programs. Zhang et al \[35\] use a mathematical program to perform voltage scaling after the tasks have been scheduled. Unlike our methods, these approaches do not yield schedules with provable guarantees.

The only previous work that yields schedules with provable guarantees while considering energy is by Pruhs et al \[26\]. They develop a speed-scaling processor model for precedence-constrained tasks with three primary results. The first is a proof that running all processors at a single fixed speed is at best \( \Omega(poly(m)) \) approximate. The second is a proof that the total power across all processors is constant over time in any optimal solution. The third result is an algorithm that is \( O(1) \) approximate in energy and \( O(\log^{1+2/\alpha} m) \) approximate in the makespan for a model-dependent constant \( \alpha \). This algorithm reduces the problem for \( m \leq n \) processors to the problem of scheduling tasks on \( m \) related processors (considered by Chekuri and Bender in \[21\]) that run at speeds that are a function of a power level \( p \), which is chosen using a binary search to give the energy bound. We improve on this result in section \[22\] where we combine convex programming with the list scheduling result by Graham \[12\] \[13\] to obtain an algorithm that is \((2 - 1/m)\)-approximate in
the makespan and satisfies the energy bound exactly.

Algorithms for precedence-constrained task scheduling that give provable guarantees on makespan (but without accounting for energy considerations) have been an active area of research since the last millennium. Graham [12, 13] gives a \((2 - 1/m)\)-approximation algorithm for the \(m < n\) case and Fujii et al [7] and Coffman and Graham [3] give exact algorithms for the \(m = 2\) case that use preemption and migration. Papadimitriou and Yannakakis [24, 25] give a 2-approximation algorithm for the \(m \geq n\), common communication delay \(c_{ij} = \tau\) case that uses recomputation, and Jung et al [18] give an exact algorithm for this case that is exponential in fixed integer \(\tau\). Munier and König [23] give an algorithm for \(m \geq n\) and small communication delays (where the duration of a task is at least \(p \geq 1\) times any communication delay) that is \(\beta\)-approximate for \(\beta = \frac{2 + 2\rho}{1 + 2\rho}\), and Hanen and Munier [14, 15] give an algorithm for the \(m < n\) case that is \((1 + \beta(1 - 1/m))\)-approximate. A recent survey by Drozdowski [5] details algorithms and heuristics for the scheduling problem with communication delays.

Our Contributions Our primary contribution is to show that convex programming formulations of many scheduling problems permit energy constraints to be simply appended; this enables us to produce comparable provable results to the energy-blind case without substantially more complex analysis. In section 2 we consider the case where the communication delays are all zero. We obtain optimal schedules when \(m \geq n\) and for \(m = 2\); and for \(m < n\), we obtain \((2 - 1/m)\)-approximate schedules. These results are analogous to the energy-blind results of [12, 13] and [7, 3], and improve on the result in [26].

In section 3 we consider the case where the communication delays may be non-zero. For small communication delays we obtain \(\beta\)-approximate schedules for \(m \geq n\) processors, \(\beta = \frac{2 + 2\rho}{1 + 2\rho}\), and \((1 + \beta(1 - 1/m))\)-approximate schedules for arbitrary \(m\). These results extend the energy-blind results in [14, 23, 15]. For large communication delays we extend the approach of [10] to obtain \(\frac{2(R+1)}{3}\)-approximate schedules. In these cases, \(\rho\) and \(R\) are parameters to the algorithm that bound the relative size of the delays.

Discussion The variants for which we obtain approximations are NP-complete [32, 28, 24] and are therefore unlikely to have fast exact solutions. We obtain our results modulo \(\varepsilon\) error due to the finite precision used in solving convex programs. To simplify our analyses we do not mention this term further.

Except in section 2.3, our algorithms do not use preemption (stopping and starting of tasks), migration (moving tasks from one processor to another), or recomputation (computing a task more than once.) Our algorithms in section 2 are approximate even to optimal algorithms that do have these properties. Our algorithms in section 3 are approximate to optimal algorithms that do not have these properties.

We consider cases in which \(e_j\) and \(d_j\) are inversely related according to a convex function; that is, \(e_j(d_j)\) is convex and non-increasing. It is natural to make this assumption for several reasons. Convexity is a more general assumption than either the speed scaling model in [24] or the dynamic voltage scaling model assumed by the experimental work cited above; the time/energy tradeoff in both of these models is convex. We consider separate functions for each task because they may vary in their use of resources and therefore may not have the same curve, even when a measure of “work” for a task is taken into account. Lastly, because we consider a homogeneous system in which there is no contention for resources (other than processors) by tasks, the functions are independent of processors and of other tasks.

Notation We use the following notation throughout the paper. \(i, j, k\) are tasks. \(p\) and \(q\) are processors. \(p_j\) is the processor on which \(j\) is scheduled. We write \(i \rightarrow j\) if the edge \((i, j)\) exists in \(G\), \(i < j\) if there is a (directed) path from \(i\) to \(j\) in \(G\), and \(i \sim j\) if neither \(i < j\) nor \(j < i\). The duration of a task \(j\) is \(d_j\) and the energy it consumes is \(e_j\); these are inversely related in a model-dependent manner. \(t_j\) is the start time.
of task $j$. $\sigma$ is a schedule, $\mu$ is a makespan and $E$ is the energy bound. We write $t_j(\sigma_1)$, $\mu(\sigma_2)$, etc. to differentiate between schedule values. Task $j$ is a source task if there is no $i \rightarrow j$ and a sink task if there is no $j \rightarrow k$. We say a task $j$ is active if it is currently running on a processor and available if its predecessors have all completed (regardless whether it’s active.) We say a processor is active at time $t$ if it is working on a task at time $t$, else idle.

All of these parameters are required to be non-negative. We abuse notation and use $I$ to refer to both an interval $I = [a, b]$ and its length $b - a$, and we use $S$ to refer to both the set $S$ and the number of elements it contains. If $S$ is a set of disjoint intervals, we also write $S$ for $\sum_{I \in S} I$.

Organization We organize the rest of the paper as follows. Section 2 contains our results for the case in which the communication delays are all zero. In section 3 we extend the results of [14, 23, 15] to the case in which there are small communication delays and the result of [10] to the case in which the delays are large.

2 Zero Communication Delays

In this section we consider a model in which a task $j$ may be started as soon as all $i \rightarrow j$ have completed and there is a free processor. Our algorithms in this section are competitive against preemptive and migratory algorithms as a result of lemma 2.2. Recomputation never helps in this model; if a schedule $\sigma$ that computes a task $j$ more than once removes the second computation, the total energy will decrease, and every task can still begin at $t_j(\sigma)$. We therefore assume that no schedule uses recomputation.

2.1 $m \geq n$ Processors

We use the following convex program in the $m \geq n$ case. We assume that $e_j(d_j)$ can be computed in polynomial time.

Program 2.1.

Minimize $\mu$ subject to:

1. $t_j \geq t_i + d_i$ for all $i \rightarrow j$
2. $\mu \geq t_j + d_j$ for all $j$
3. $\sum_j e_j(d_j) \leq E$
4. $\mu \geq \frac{1}{m} \sum_j d_j$
   $\mu, t_j, d_j \geq 0$

Constraint 4 will be necessary in section 2.2. In the case where $m \geq n$, this only constrains the makespan to be at least the average duration.

We show that this convex program is equivalent to the scheduling problem.

Theorem 2.2. There is a solution $\mu, t_j, d_j$ to program 2.1 iff there is a feasible schedule $\sigma$ with makespan $\mu$.

Proof. We prove the theorem by constructing a solution to one out of a solution to the other. The following two algorithms perform these conversions. \qed
Given a solution \( \mu, t_j, d_j \) to program 2.1, schedule each task \( j \) in \( \sigma \) at time \( t_j \) with duration \( d_j \). Always schedule \( j \) on its own processor.

Precedence and energy constraints are satisfied immediately. \( j \) does not cause co-occurrence conflicts with any other task \( k \) because it is on its own processor. \( \sigma \) uses \( n \leq m \) processors.

Algorithm 2.4. Given a schedule \( \sigma \) on \( m \) processors, construct \( \mu, t_j, d_j \) as follows. Set \( \mu = \text{makespan}(\sigma) \) and \( t_j = t_j(\sigma) \). Let \( S_j \) be the set of intervals over which \( j \) is active in \( \sigma \). Set \( d_j = S_j \).

Constraint 1 is satisfied because \( t_j(\sigma) \) is at least the end of the last interval in \( S_i \) for \( i \rightarrow j \), and the first interval in \( S_i \) does not begin until \( t_i(\sigma) \). Constraint 2 is satisfied similarly. Constraint 3 is satisfied because \( \sigma \) is feasible, and constraint 4 is satisfied because \( \sigma \) uses at most \( m \) processors and the makespan is at least the average load.

Because program 2.1 is a convex optimization problem with polynomially many constraints, it can be solved in polynomial time. To generate an optimal schedule \( \sigma \) we first solve program 2.1 and then use algorithm 2.3 to construct the schedule.

### 2.2 \( m \) Processors

In the case where \( m \) is arbitrary we obtain a \((2 - 1/m)\)-approximation.

**Lemma 2.5.** Given a schedule \( \sigma_1 \) that satisfies program 2.1, algorithm 2.6 constructs a schedule \( \sigma_2 \) that has makespan \( \mu_2 \leq (2 - 1/m) \mu_1 \) and that uses at most \( E \) energy.

**Algorithm 2.6.** While there are unscheduled tasks in \( \sigma_2 \), let \( t \) be the earliest time for which there is both a processor \( p \) that has no tasks scheduled after time \( t \) and a task \( j \) whose predecessors \( i \rightarrow j \) have all completed by time \( t \). Schedule \( j \) for duration \( d_j(\sigma_1) \) starting at time \( t \) on processor \( p \) in \( \sigma_2 \).

**Proof.** We cut the time interval \((0, \mu_2)\) at each point at which some task begins or ends, and partition these sub-intervals into two sets \( A \) and \( B \), where \( A \) contains all intervals in which all \( m \) processors are active in \( \sigma_2 \) and \( B \) contains the rest. \( \mu_2 = A + B \). We define \( W = \sum_j d_j \) for convenience.

We bound \( B \leq \mu_1 \) with a potential argument. For each time \( t \) in \( \sigma_2 \) let \( F_t \) be the set of tasks finished by time \( t \) and \( J_t \) the set of available tasks at time \( t \). We define \( \phi(t) \) as the smallest time \( u \) in \( \sigma_1 \) such that \( \sigma_1 \) finishes all tasks \( F_t \) by time \( u \) and has completed at least as much work on all tasks in \( J_t \) as \( \sigma_2 \) has completed. We cannot have \( \phi(t) > \mu_1 \) since \( \sigma_1 \) completes all tasks by time \( \mu_1 \). We must also have \( \phi(0) = 0 \) and \( \phi(t_1) \leq \phi(t_2) \) whenever \( t_1 \leq t_2 \).

For each interval \( I = (a, b) \) in \( B \) it must be that all \( J_a = J_b < m \) available tasks are active; otherwise, we could have scheduled an available task on an idle processor. We must further have \( \phi(b) - \phi(a) \geq b - a \); otherwise, \( \sigma_1 \) could complete \( I \) work on these \( J_a \) tasks in less than \( I \) time. Together, these yield

\[
B = \sum_{(a,b) \in B} b - a \leq \sum_{(a,b) \in B} \phi(b) - \phi(a) \\
\leq \sum_{(a,b) \in A \cup B} \phi(b) - \phi(a) = \phi(\mu_2) - \phi(0) \leq \mu_1
\]

\( \sigma_2 \) completes at least \( I \) work over each interval \( I \) in \( B \) because in each such \( I \) there is at least one active processor; otherwise, \( t \) would be the earliest time at which we could schedule a task in \( \sigma_2 \), but we chose a time \( t' > t \) instead.
We bound \( A \leq (W - B)/m \) since there is no more than \( (W - B) \) work to do in \( A \) and all \( m \) processors are active in \( A \).

By constraint [4] in program 2.1 \( \mu_1 \geq W/m \). Together with our bounds for \( A \) and \( B \) we have \( \mu_2 = A + B \leq W/m + (1 - 1/m)B \leq (2 - 1/m)\mu_1 \). \( \square \)

**Theorem 2.7.** We can construct a \((2 - 1/m)\)-approximation for the case when \( m \) is arbitrary.

**Proof.** Let \( \mu^* \) be the optimal makespan obtainable for \( m \) processors on a given task graph \( G \) with energy bound \( E \), using an optimal schedule \( \sigma^* \). Program 2.1 has an objective value \( \mu_1 \) that is no larger than \( \mu^* \); otherwise, we could set \( t_j = t_j(\sigma^*) \) and \( d_j = d_j(\sigma^*) \), and all constraints would remain satisfied.

Since \( \mu_1 \leq \mu^* \), we can use algorithm 2.3 to create a schedule \( \sigma_1 \) that uses \( n \) processors, and algorithm 2.6 to generate a schedule \( \sigma_2 \) that uses \( m \) processors. By the lemmas, we have \( \mu(\sigma_2) \leq (2 - 1/m)\mu_1 \leq (2 - 1/m)\mu^* \). \( \square \)

### 2.3 Two Processors

In the case where \( m = 2 \) we can obtain an optimal schedule if migration and preemption are permitted. We combine convex programming with a fractional version of the matching-based algorithm in [7].

We consider tasks \( j \) according to any linear ordering \( \prec_L \) that is an extension of \( \prec_G \). The duration \( d_j \) of a task \( j \) is broken into components. The time when task \( j \) is the only task running is \( \ell_j \). The time when task \( j \) is running at the same time as another task \( i \prec_L j \) is \( \ell_{ij} \).

**Program 2.8.**

Minimize \( \mu = \sum_j \ell_j + \sum_i \sum_{j \succ L, i} \ell_{ij} \) subject to:

1. \( d_j = \ell_j + \sum_{i \prec_L j} \ell_{ij} + \sum_{k \succ L, j} \ell_{jk} \) for all \( j \)
2. \( \ell_{ij} = 0 \) for all \( i \prec_G j \)
3. \( \ell_j, \ell_{ij} \geq 0 \)

We extend this linear program with the addition of a convex energy constraint. We consider \( d_j \) to be a fixed value in program 2.8 and a variable in program 2.9.

**Program 2.9.**

Program 2.8 subject to the additional constraint:

3. \( \sum_j e_j(d_j) \leq E \)

We can construct a solution to program 2.9 from a feasible schedule \( \sigma \) by setting \( \ell_j (\ell_{ij}) \) as the sum over intervals in which only task \( j \) (tasks \( i \) and \( j \)) are active.

To construct a feasible schedule from a solution to the program, we use the following lemma as a subroutine.

**Lemma 2.10.** Let \( S \) be the set of source tasks in \( G \). Given a solution to program 2.8 with objective value \( \mu \), we can construct a new solution with the same objective value that satisfies the condition \( \exists s \in S \forall j \notin S (\ell_{sj} = 0) \); call this \( s \) the next task.

6
Proof. Let \( Z = \{ (i, j) \mid i \in S, j \notin S, \ell_{ij} > 0 \} \). We say \( Z \) has inductive pairs \((i, j)\) and \((h, k)\) if these pairs exist in \( Z \) and \( j \sim k \). The proof is by complete induction on the set \( Z \).

If \( Z \) is empty, then the lemma holds for all \( j \) in \( S \). If \( Z \) does not have inductive pairs, then we define \( T = \{ j \mid \exists i ((i, j) \in Z) \} \). It must be that the tasks in \( T \) are fully ordered by \( <_G \) (and not just \( <_L \)); otherwise, we would have \( j \sim k \) for some pairs. Let \( j \) be the least task in \( T \). Since \( j \) is in \( T \), it is not in \( S \), and therefore there is some \( s \) in \( S \) such that \( s <_G j \). There can be no \( k \in T \) for which \((s, k) \in Z \); otherwise, \( s <_G j <_G k \) would violate constraint 2. Because \( s \) does not appear in \( Z \), the lemma holds for \( s \).

For the inductive case, let \((i, j)\) and \((h, k)\) be inductive pairs in \( Z \), so that \( j \sim k \). We also have that \( i \sim h \) because \( i \) and \( h \) are both in \( S \). We define \( \Delta = \min \{ \ell_{ij}, \ell_{hk} \} \), and update \( \ell_{ih} := \ell_{ih} + \Delta, \ell_{jk} := \ell_{jk} + \Delta \), \( \ell_{ij} := \ell_{ij} - \Delta \), and \( \ell_{hk} := \ell_{hk} - \Delta \).

Neither \( d_j \) nor \( \mu \) changes as a result of these updates. No \( \ell \) is updated to be less than zero. Constraint 2 still holds because \( j \sim k \) and \( i \sim h \). Therefore, this new solution is also feasible and has the same objective value. Because \( \Delta = \min \{ \ell_{ij}, \ell_{hk} \} \), either \( \ell_{ij} = 0 \) or \( \ell_{hk} = 0 \), so \( Z \) decreases and we continue inductively.

The following algorithm constructs a feasible schedule \( \sigma \) iteratively. Program 2.9 is solved once initially to determine \( d_j \) that satisfy constraint 3. In each iteration, a solution is maintained for program 2.8 using fixed values chosen so that the total durations are \( d_j \).

**Algorithm 2.11.** Initially, solve program 2.9 for the input graph \( G \) to determine durations \( d_j \) for each task \( j \). Define \( G_0 = G \), \( d^0_j = d_j \), and \( \sigma_0 \) to be an empty schedule.

Then, for each \( l = 0, \ldots, n - 1 \), do the following. Let \( S_l \) be the set of sources in \( G_l \). Solve program 2.8 on \( G_l \) using fixed durations \( d^l_j \). Run lemma 2.10 as a subroutine to find the next task \( s_l \in S_l \) and updated values \( \ell_{ij}^l, \ell_{ij}^l \). To get \( \tilde{\sigma}_{l+1} \), begin with \( \tilde{\sigma}_l \), schedule task \( s_l \) alone for \( \ell_{si}^l \) time, and then for each other \( i \in G_l \) schedule tasks \( s_l \) and \( i \) together for \( \ell_{si}^l \) time. Define \( G_{l+1} \) as \( G_l - \{ s \} \) and \( d^l_{ij} = d^l_{ij} - \ell_{si}^l \).

Our resulting schedule \( \sigma \) is \( \sigma_n \).

We note that the updated values \( \ell_{ij}^l, \ell_{ij}^l \) for all tasks except \( s \) constitute a feasible optimum for program 2.8 on \( G_{l+1}, d^l_{ij} \), so by saving these values we do not need to recompute a solution to this program.

**Theorem 2.12.** Let \( \Pi(G) \) be program 2.8 instantiated for graph \( G \). \( \sigma \) is feasible and has makespan \( \mu(\sigma) \) equal to the objective value \( \mu^\Pi(\sigma) \) of \( \Pi(G) \).

**Proof.** Let \( \mu_0 = \mu(\tilde{\sigma}_0) \). We first prove that \( \mu_{l+1} + \mu^\Pi_{l+1} \leq \mu_l + \mu^\Pi_l \) for all \( l \). We have \( \mu_{l+1} = \mu_l + \ell_{ij}^l + \sum_{i > L, s} \ell_{si}^l \) because we schedule \( s \) and \( (s, i) \) to get \( \tilde{\sigma}_{l+1} \). We also have that \( \mu^\Pi_{l+1} \geq \mu^\Pi_l - \ell_{ij}^l - \sum_{i > L, s} \ell_{si}^l \); otherwise, we could use \( \ell_{ijs}^l, \ell_{ij}^l, \ell_{si}^l \) instead of \( \ell_{ij}^l, \ell_{ij}^l \) and get a better solution to \( \Pi(G_l) \).

Inductively, we must have \( \mu_{l+1} + \mu^\Pi_{l+1} \leq \mu^\Pi_0 \) because \( \mu_0 = 0 \). We further have that \( \mu^\Pi_n = 0 \) because \( G_n \) is empty. Therefore, \( \mu(\sigma) \leq \mu^\Pi_0 \). Because we can construct a feasible solution to \( \Pi(G_0) \) from \( \sigma \), we also have \( \mu(\sigma) \geq \mu^\Pi_0 \).

We also must prove that \( \sigma \) uses at most \( E \) energy. The \( d_j \) were chosen by program 2.9 to satisfy constraint 3. Program 2.8 satisfies constraint 1 with equality. Because we update \( d^l_{ij} = d^l_{ij} - \ell_{si}^l \), the total duration for which each task \( j \) runs is \( d_j \), and therefore the total energy used is at most \( E \).

## 3 Small Communication Delays

In this section we consider a model in which each edge \( i \rightarrow j \) in \( G \) has an associated communication delay of \( c_{ij} \). When task \( i \) finishes, if task \( j \) is scheduled on a different processor, it may not begin until at least \( c_{ij} \)
time has passed. (If \( j \) is started on the same processor, it may begin immediately after \( i \) finishes, as long as it is not otherwise constrained.) This models a system in which there is a time/energy trade-off for processors and a constant-speed interconnect running at \( s \) data per time that must transfer \( s \ c_{ij} \) data from \( p_i \) to \( p_j \) and can make any number of point-to-point transfers at a time.

Because we consider small communication delays, we assume a given \( \rho \geq 1 \) for which \( c_{ij} \leq d_k / \rho \) for all \( i \rightarrow j \) and \( k \), and we compare against optimal solutions that also satisfy this constraint. Our results improve with increasing \( \rho \) (and smaller communication delays.)

3.1 \( m \geq n \) Processors

We extend the approach of [14, 23] to account for energy. We first show that the following non-convex program is equivalent to the scheduling problem. \( x_{ij} \) is an indicator variable that is 1 if \( j \) follows \( i \) on processor \( p_i \) without waiting the \( c_{ij} \) communication delay time, and 0 otherwise.

Program 3.1.

\[
\text{Minimize } \mu \text{ subject to:}
\]

1. \( t_j \geq t_i + d_i + (1 - x_{ij}) \ c_{ij} \) for all \( i \rightarrow j \)
2. \( \sum_{j:i 

3. \( \sum_{i:j} x_{ij} \leq 1 \) for all \( i \)
4. \( \mu \geq t_j + d_j \) for all \( j \)
5. \( x_{ij} \in \{0, 1\} \) for all \( i \rightarrow j \)
6. \( \sum_{j} c_j(d_j) \leq E \)
7. \( c_{ij} \leq d_k / \rho \) for all \( i \rightarrow j, k \)
8. \( \mu \geq \frac{1}{m} \sum_j d_j \)

\( \mu, t_j, d_j \geq 0 \)

Lemma 3.2. There is a solution \( \mu, t_j, d_j, x_{ij} \) to program 3.1 if there is a feasible schedule \( \sigma \) with makespan \( \mu \) that satisfies \( d_i / \rho \geq c_{ij} \) and that does not use preemption, migration, or recomputation.

Proof. We prove the lemma by constructing a solution to one out of a solution to the other. The following two algorithms perform these conversions.

Algorithm 3.3. Given a solution \( \mu, t_j, d_j, x_{ij} \) to program 3.1 construct \( \sigma \) as follows. Consider tasks \( j \) according to any linear extension \( <_L \) of \( <_G \). Schedule each task \( j \) at time \( t_j \) with duration \( d_j \). If there is some \( i \rightarrow j \) for which \( x_{ij} = 1 \) schedule \( j \) on \( p_i \) and on a new processor otherwise.

Precedence, communication, and energy constraints are satisfied immediately. If \( j \) is scheduled on a new processor, then \( j \) does not cause co-occurrence conflicts with any prior task \( k <_L j \) because it is on a new processor. If \( j \) is scheduled on \( p_i \), then it could co-occur with a prior task \( k <_L j \) only if \( k \) is also scheduled on \( p_i \) after \( i \). But \( k \) is not scheduled on \( p_i \) after \( i \) because \( x_{ij} = 1 \) and therefore \( x_{ik} = x_{kj} = 0 \) by constraints 2 and 3, so in this case \( j \) also does not cause co-occurrence conflicts.

Algorithm 3.4. Given a schedule \( \sigma \), construct \( \mu, t_j, d_j, x_{ij} \) as follows. Set \( \mu = \text{makespan}(\sigma) \), \( t_j = t_j(\sigma) \), \( d_j = d_j(\sigma) \). Set \( x_{ij} = 1 \) if \( t_j < t_i + d_i + c_{ij} \) and \( x_{ij} = 0 \) otherwise.
All constraints except for 2, 3, and 8 are satisfied immediately. Suppose \( x_{ij} = 1 \). Since \( t_j < t_i + d_i + c_{ij} \) and \( c_{ij} \leq d_k \), it must be that no other task \( k \) is scheduled between \( t_i + d_i \) and \( t_j \) on processor \( p_i \). Therefore, for any other task \( k \) where \( i \rightarrow k \) or \( k \rightarrow j \), \( x_{ik} = x_{kj} = 0 \), so constraints 2 and 3 hold. Lastly, makespan(\( \sigma \)) is at least the average load on each processor, and therefore constraint \( 8 \) holds.

We note that we need \( \rho \geq 1 \) to enforce that no task \( k \) can be between \( i \) and \( j \) on \( p_i \); if \( \rho < 1 \) then we could possibly have \( d_k < c_{ij} \), which would break our argument.

We relax program (3.1) to a convex program by requiring only that \( 0 \leq x_{ij} \leq 1 \) rather than \( x_{ij} \in \{0, 1\} \). We show that the following deterministic rounding algorithm gives a feasible schedule \( \sigma \) that satisfies the energy bound \( E \) and is \( \beta \)-approximate in the makespan for \( \beta = \frac{2+\rho}{1+\rho} \).

**Algorithm 3.5.** Solve the convex relaxation of program (3.1) For each \( x_{ij} \), Define \( \hat{x}_{ij} = 1 \) if \( x_{ij} > 1/2 \) and \( \hat{x}_{ij} = 0 \) otherwise. Take any linear extension \( \sigma_L \) of \( \sigma_G \). For each \( j \) in order of \( \sigma_L \), if there is an \( i \rightarrow j \) such that \( \hat{x}_{ij} = 1 \), schedule \( j \) on \( p_i \) to begin at time \( t_i(\sigma) + d_i \). If there is no such \( i \), schedule \( j \) on its own processor at time \( \max_{i \rightarrow j} \{ t_i(\sigma) + d_i + c_{ij} \} \). Always schedule \( j \) for duration \( c_{ij} \).

**Proof.** We first prove the stronger claim that for every task \( j \), \( t_j(\sigma) \leq \beta t_j \). We prove this inductively on \( \sigma \).

If \( j \) is a source task, then there are no \( i \rightarrow j \), so \( t_j(\sigma) = 0 \leq \beta t_j \).

If \( j \) is not a source task, then \( j \) is scheduled either on \( p_i \) for some \( i \rightarrow j \) or on its own processor. If \( j \) is scheduled on \( p_i \), then \( \hat{x}_{ij} = 1 \), and therefore \( t_j(\sigma) = t_i(\sigma) + d_i \leq \beta t_i + d_i \leq \beta t_j \) by induction and constraint 1. Otherwise, there is some \( i \rightarrow j \) for which \( j \) is scheduled at time \( t_j(\sigma) = t_i(\sigma) + d_i + c_{ij} \leq \beta t_i + d_i + c_{ij} \).

\( \hat{x}_{ij} = 0 \), so \( x_{ij} \leq 1/2 \), and therefore \( 1 - x_{ij} \geq 1/2 \) and \( t_j \geq t_i + d_i + c_{ij} / 2 \). With \( \beta = \frac{2+\rho}{1+\rho} \) and \( c_{ij} \leq d_i / \rho \), algebraic manipulation yields \( t_j(\sigma) \leq \beta t_j \).

With this stronger claim, we can bound the makespan by

\[
\mu(\sigma) = \max_j \{ t_j(\sigma) + d_j \} \leq \max_j \{ \beta t_j + d_j \} \leq \beta \mu
\]

We now prove that \( \sigma \) is a feasible schedule. Precedence, communication, and energy constraints are satisfied immediately. If \( j \) is scheduled on a new processor, then \( j \) does not cause co-occurrence conflicts with any prior task \( k <_L j \) because it is on a new processor. If \( j \) is scheduled on \( p_i \), then \( \hat{x}_{ik} = \hat{x}_{kj} = 0 \) for any other task \( k \), so in this case \( j \) also does not cause co-occurrence conflicts.

### 3.2 \( m \) Processors

We use the following lemma from [15] as a black-box to obtain a bound for \( m \) processors.

**Lemma 3.6.** In the case where \( c_{ij} \leq d_k / \rho \), given a schedule \( \sigma_1 \) with makespan \( \mu_1 \) that uses more than \( m \) processors we can construct a new schedule \( \sigma_2 \) with makespan \( \mu_2 \leq \frac{1}{m} \sum_j (d_j) + (1 - 1/m) \mu_1 \) that uses only \( m \) processors.

The model considered in [15] assumes fixed durations, so \( d_j(\sigma_1) = d_j(\sigma_2) \) and therefore \( \sigma_2 \) uses at most \( E \) energy as well.

**Theorem 3.7.** We can construct a \((1 + \beta(1 - 1/m))\)-approximate schedule \( \sigma \) for the case in which \( m < n \).

**Proof.** We use algorithm (3.5) to generate a schedule \( \sigma_1 \) with makespan \( \mu_1 \leq \beta \mu \), where \( \mu \) is the optimal objective value of program (3.1).

Let \( \sigma^* \) be an optimal schedule for \( m \) processors with makespan \( \mu^* \). We must have \( \mu^* \geq \mu \); otherwise, we could use algorithm (3.4) to convert \( \sigma^* \) into a feasible solution to program (3.1) with smaller \( \mu \).

\( \sigma^* \) satisfies constraint \( 8 \) of program (3.1) and therefore \( \mu^* \geq \frac{1}{m} \sum_j d_j \). These three bounds plus the bound in lemma (3.6) yield the theorem. 

\[ \square \]
3.3 Large Communication Delays

We use the approach of [10] to obtain a $\frac{2(R+1)}{3}$-approximate schedule $\sigma_2$ in the case of large communication delays where $c_{ij}/d_k \leq R$.

Algorithm 3.8. Define $\hat{c}_{ij} = c_{ij}/R$. Run algorithm 3.5 on $G, \hat{c}_{ij}$, and $\rho = 1$ to get a schedule $\sigma_1$. Scale up each communication delay $\hat{c}_{ij}$ in $\sigma_1$ by $R$ to get $\sigma_2$.

Theorem 3.9. $\sigma_2$ is a $\frac{2(R+1)}{3}$-approximate schedule on $m \geq n$ processors.

Proof. Let $\mu_1^*$ be the optimal makespan for $\hat{c}_{ij}$ and $\mu^*$ the optimal makespan for $c_{ij}$. It is clear that $\mu_1^* \leq \mu^*$; otherwise, when we scale down by $R$ we could get a better solution.

Let $\mu_1 = \mu(\sigma_1)$ and $\mu = \mu(\sigma_2)$. We have $\mu_1 \leq \frac{4}{3}\mu_1^*$ by algorithm 3.5. $\mu_1$ is the length of some critical path $P$ of tasks in $\sigma_1$; $P$ has $g$ tasks and $g - 1$ communication delays (some of which may be 0.)

Let $A$ be the contribution of task durations to $\mu_1$ and $B$ be the contribution of delays. For each task $k$ in $P$ and each delay $\hat{c}_{ij}$ in $P$, we have $\hat{c}_{ij} \leq d_k$. Therefore, $A \geq \mu_1/2$. When we scale the $\hat{c}_{ij}$ up by $R$ to get $c_{ij}$ in $\sigma_2$, we get $\mu \leq A + RB \leq \frac{1}{2}\mu_1 + \frac{R}{2}\mu_1 \leq \frac{2(R+1)}{3}\mu_1^*$, so $\mu \leq \frac{2(R+1)}{3}\mu^*$.

4 Conclusions

We have shown that for several scheduling problems, using convex programming, we can obtain approximation bounds when energy constraints are present that are no worse than the existing bounds obtained when energy is not considered. Our analyses for the most part used the analyses of algorithms for the corresponding energy-blind cases, adjusted where needed to fit the convex programming formulation.

One possible direction for future work is to characterize the conditions necessary for the approach in this paper to be applicable. We rely heavily on the fact that, once the time/energy allocations are determined, our problem is essentially an instance of the energy-blind problem, and can be solved using energy-blind methods.

As an example, recomputation is a consideration for which our approach appears to break. In scheduling models that permit multiple copies of tasks to be computed on different processors, such as the model considered by Papadimitriou and Yannakakis in [25], the energy for each copy should be taken into account. More work is necessary to determine the extent to which our methods are applicable to these models. In particular, a convex programming formulation that would permit energy constraints to be appended is not obvious and would be interesting.

References

[1] Ishfaq Ahmad, Roman Arora, Derek White, Vangelis Metsis, and Rebecca Ingram. Energy-constrained scheduling of dags on multi-core processors. In Sanjay Ranka, Srinivas Aluru, Rajkumar Buyya, Yeh-Ching Chung, Sumeet Dua, Ananth Grama, Sandeep K. S. Gupta, Rajeev Kumar, and Vir V. Phoha, editors, Contemporary Computing, volume 40 of Communications in Computer and Information Science, pages 592–603. Springer Berlin Heidelberg, 2009.

[2] Chandra Chekuri and Michael Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. J. Algorithms, 41:212–224, November 2001.
[3] E. G. Coffman and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1:200–213, 1972. 10.1007/BF00288685.

[4] Srinivas Devadas and Sharad Malik. A survey of optimization techniques targeting low power VLSI circuits. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, DAC ’95, pages 242–247, New York, NY, USA, 1995. ACM.

[5] Maciej Drozdowski. Scheduling with communication delays. In *Scheduling for Parallel Processing*, Computer Communications and Networks, pages 209–299. Springer London, 2009.

[6] Vincent W. Freeh, Nandini Kappiah, David K. Lowenthal, and Tyler K. Bletsch. Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. *Journal of Parallel and Distributed Computing*, 68(9):1175 – 1185, 2008.

[7] M. Fujii, T. Kasami, and K. Ninomiya. Optimal sequencing of two equivalent processors. *SIAM Journal on Applied Mathematics*, 17(4):784–789, 1969.

[8] D. Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11 – 13, may 2005.

[9] R. Gioiosa. Towards sustainable exascale computing. In *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, pages 270 –275, sept. 2010.

[10] Rodolphe Giroudeau, Jean-Claude Konig, Farida Kamila Moulai, and Jérôme Palaysi. Complexity and approximation for precedence constrained scheduling problems with large communication delays. *Theor. Comput. Sci.*, 401:107–119, July 2008.

[11] Lee Kee Goh, B. Veeravalli, and S. Viswanathan. Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, 20(1):1 –12, jan. 2009.

[12] R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, XLV:1563–1581, 1969.

[13] R. L. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[14] Claire Hanen and Alix Munier. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. In *IEEE Symposium on Emerging Technologies and Factory Automation 167-190*, 1995.

[15] Claire Hanen and Alix Munier. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. In *Discrete Applied Mathematics 108:239-257 Elsevier*, 2001.

[16] I. Hong, D. Kirovski, Gang Qu, M. Potkonjak, and M.B. Srivastava. Power optimization of variable-voltage core-based systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(12):1702 –1714, dec 1999.

[17] Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC ’05, pages 1–, Washington, DC, USA, 2005. IEEE Computer Society.
[18] Hermann Jung, Lefteris M. Kirousis, and Paul Spirakis. Lower bounds and efficient algorithms for multiprocessor scheduling of directed acyclic graphs with communication delays. *Inf. Comput.*, 105:94–104, July 1993.

[19] Jaeyeon Kang and S. Ranka. Dvs based energy minimization algorithm for parallel machines. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12, april 2008.

[20] Jaeyeon Kang and S. Ranka. Assignment algorithm for energy minimization on parallel machines. In *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, pages 484–491, sept. 2009.

[21] David King, Ishfaq Ahmad, and Hafiz Fahad Sheikh. Stretch and compress based re-scheduling techniques for minimizing the execution times of dags on multi-core processors under energy constraints. In *Proceedings of the International Conference on Green Computing, GREENCOMP ’10*, pages 49–60, Washington, DC, USA, 2010. IEEE Computer Society.

[22] Lap-Fai Leung, Chi-Ying Tsui, and Wing-Hung Ki. Minimizing energy consumption of multi-processors-core systems with simultaneous task allocation, scheduling and voltage assignment. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference, ASP-DAC ’04*, pages 647–652, Piscataway, NJ, USA, 2004. IEEE Press.

[23] Alix Munier and Jean-Claude Konig. A heuristic for a scheduling problem with communication delays. *OPERATIONS RESEARCH*, 45(1):145–147, 1997.

[24] Christos Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. In *Proceedings of the twentieth annual ACM symposium on Theory of computing, STOC ’88*, pages 510–513, New York, NY, USA, 1988. ACM.

[25] Christos H. Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comput.*, 19:322–328, April 1990.

[26] Kirk Pruhs, Rob van Stee, and Patchrawat Uthaisombut. Speed scaling of tasks with precedence constraints. *Theor. Comp. Sys.*, 43:67–80, March 2008.

[27] Ravishankar Rao, Sarma Vrudhula, and Chaitali Chakrabarti. Throughput of multi-core processors under thermal constraints. In *Proceedings of the 2007 international symposium on Low power electronics and design, ISLPED ’07*, pages 201–206, New York, NY, USA, 2007. ACM.

[28] V. J. Rayward-Smith. The complexity of preemptive scheduling given interprocessor communication delays. *Information Processing Letters*, 25(2):123 – 125, 1987.

[29] Barry Rountree, David K. Lowenthal, Shelby Funk, Vincent W. Freeh, Bronis R. de Supinski, and Martin Schulz. Bounding energy consumption in large-scale mpi programs. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing, SC ’07*, pages 49:1–49:9, New York, NY, USA, 2007. ACM.

[30] Marcus T. Schmitz and Bashir M. Al-Hashimi. Considering power variations of dvs processing elements for energy minimisation in distributed systems. In *Proceedings of the 14th international symposium on Systems synthesis, ISSS ’01*, pages 250–255, New York, NY, USA, 2001. ACM.
[31] Venkateswaran Shekar and Baback Izadi. Energy aware scheduling for dag structured applications on heterogeneous and dvs enabled processors. In Proceedings of the International Conference on Green Computing, GREENCOMP ’10, pages 495–502, Washington, DC, USA, 2010. IEEE Computer Society.

[32] J. D. Ullman. Np-complete scheduling problems. J. Comput. Syst. Sci., 10:384–393, June 1975.

[33] Lizhe Wang, Jie Tao, Gregor von Laszewski, and Dan Chen. Power aware scheduling for parallel tasks via task clustering. Parallel and Distributed Systems, International Conference on, 0:629–634, 2010.

[34] Lizhe Wang, Gregor von Laszewski, Jay Dayal, and Fugang Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGRID ’10, pages 368–377, Washington, DC, USA, 2010. IEEE Computer Society.

[35] Yumin Zhang, Xiaobo Sharon Hu, and Danny Z. Chen. Task scheduling and voltage selection for energy minimization. In Proceedings of the 39th annual Design Automation Conference, DAC ’02, pages 183–188, New York, NY, USA, 2002. ACM.