Optimal Task Scheduling for Partially Heterogeneous Systems

Michael Orr*, Oliver Sinnen

Department of Electrical, Computer, and Software Engineering, University of Auckland, New Zealand

Abstract

Task scheduling with communication delays is a strongly NP-hard problem. Previous attempts at finding optimal solutions to this problem have used branch-and-bound state-space search, with promising results. However, the scheduling model used assumes a target system with fully homogeneous processors, which is unrealistic for many real world systems for which task scheduling might be performed. This paper presents an extension to the Allocation-Ordering (AO) state-space model for task scheduling which allows a system with related heterogeneous processors to be modeled, and optimal schedules on such a system to be found. Of particular note, the distinct allocation phase allows this model to efficiently adapt to partially heterogeneous systems, in which subsets of the processors are identical to each other, which significantly helps to reduce the search space. An extensive experimental evaluation shows that the introduction of heterogeneity certainly increases the difficulty of the problem. However, many problem instances solvable using homogeneous processors remain solvable with a heterogeneous target system, made possible by the significant benefit of this model in considering partial heterogeneity.

Keywords: task scheduling, parallel computing, branch-and-bound, combinatorial optimisation, heterogeneous processors

1. Introduction

Task scheduling with communication delays, otherwise known as $P|prec, c_{ij}|C_{\max}$ [1], is a well known problem in which a set of computational tasks must be scheduled on a set of processors with the goal of minimising the overall execution time, constrained by precedence relations between tasks and associated communication costs. A related problem, with additional complexity, is the problem of task scheduling with communication delays and related heterogeneous processors, denoted as $Q|prec, c_{ij}|C_{\max}$. In the former problem, all processors are identical, and give identical execution times. Related heterogeneous processors may have different speeds, such that the execution time of tasks on one processor is some fixed multiple of their execution time on a different processor. Both of these problems are strongly NP-hard, and therefore there is no algorithm known which will find optimal solutions in polynomial time [2]. Many polynomial-time heuristic algorithms have been proposed which provide non-optimal solutions [3, 4, 5]. Dynamic scheduling must necessarily be performed with heuristic approximation algorithms - these may help to inform lower bound heuristics for optimal scheduling algorithms, but are otherwise quite distinct. In this work we are concerned with optimal scheduling, and therefore with static scheduling in which all tasks and their properties are known *a priori*.

To acquire the maximum benefit of a parallel system for timely program execution, it is necessary to have high quality schedules. Because no α-approximation scheme is known for the problem, it is not generally possible to guarantee the quality of an approximate solution [6]. Properly analysing the performance of approximation algorithms therefore requires optimal solutions for comparison. Previous work on optimal solving for the homogeneous-processor variant of the scheduling problem has found branch-and-bound state-space search to be a promising method [7], particularly when using a state-space model called Allocation-Ordering (AO) [8]. However, there are many real world systems on which scheduling might be performed which do not have homogeneous processors.

In this paper, we describe how the AO state-space model has been adapted to allow it to provide optimal solutions to the problem of task scheduling with related heterogeneous processors. In particular, the AO model appears suited for systems exhibiting partial heterogeneity: that is, where multiple categories of mutually identical processors can be identified. One example is that of a computing cluster in which different types of processor have been added or replaced in distinct stages over time. We show that this modified AO model can successfully solve many task scheduling problems of similar size to what is achievable with homogeneous processors, and identify some trends in the relative performance on heterogeneous and homogeneous systems. In general, heterogeneous systems make task scheduling more difficult. However, we demonstrate that the AO model's adaptation for partial heterogeneity provides a significant advantage when scheduling for many possible systems. In addition, it is seen that when a target system is strongly heterogeneous, it can instead make finding solutions easier.

In Section 2 we discuss relevant background information, including the task scheduling model used and the original formulation of the AO model. Section 3 discusses how the AO model was reformulated to allow heterogeneity, and the ad-

*Corresponding author
Email address: morr010@aucklanduni.ac.nz (Michael Orr)

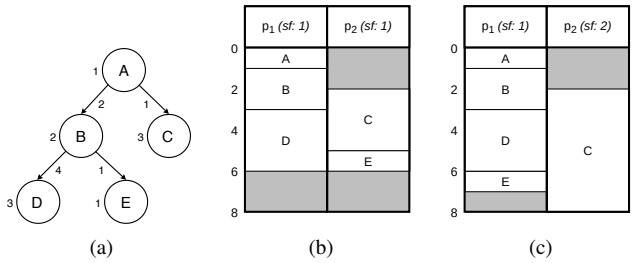

Figure 1: A simple task graph, and valid schedules on homogenous and heterogeneous processors.

ditional complexity this introduces to the state-space. Subsequently, Section 4 presents an empirical evaluation of the performance of the reformulated AO model on a variety of heterogeneous and homogeneous systems. Finally, Section 5 presents the conclusions of the paper.

2. Background

2.1. Task Scheduling Model

The problem this work is concerned with is defined as the scheduling of a task graph $G = \{V, E, w, c\}$ on a target parallel system, this being a set of processors P. G is a directed acyclic graph (DAG) which represents a program we wish to execute, and is referred to as a task graph. Nodes $n \in V$ represent distinct computational tasks which make up the program. Each task has an integer weight $w(n)$, known as the computation cost, indicating a relative number of time units necessary to complete the associated computation. Dependencies between tasks are represented by the task graph's edges. An edge $e_{ij} \in E$ indicates that task n_j is dependent on task n_i. This means that data produced by the computation in n_i is a necessary input of n_j, and therefore n_i must complete execution before n_j can begin. The weight of an edge $c(e_{ij})$, known as the communication cost, gives a number of time units necessary for the associated data to be communicated from one processor to another. An example of a task graph can be seen in Figure 1a. The set of parents (or predecessors) of task n is denoted by $pred(n)$, while the set of children (or successors) is denoted by $succ(n)$. The processors $p \in P$ of the target system are assumed to have a homogeneous communication subsystem such that data may be communicated uniformly between any pair of processors $p_i, p_j \in P$. Communication is achieved without contention and without disrupting the computational work of the processors. There is no cost associated with local communication, i.e. from p_i to p_i. This means that communication costs between tasks assigned to the same processor are ignored, and do not cause delays.

The goal of the problem is to define a schedule $S = \{proc, t_s\}$. The function $proc(n)$ maps a task $n \in V$ to a processor $p \in P$, indicating on which processor the task should be executed. The function $t_s(n)$ maps a task $n \in V$ to an integer start time, indicating the number of time units after the beginning of program execution when this task should start to be executed. A valid schedule is one in which $proc(n)$ and $t_s(n)$ are defined for all $n \in V$, subject to two additional constraining criteria. Firstly, each processor may execute at most one task at any given time. Second, a task n may only start executing after all of the tasks it is dependent on have finished execution, and the data they produce has been communicated to $proc(n)$. An optimal schedule S^* is a valid schedule which has the minimum possible total execution time. Figure 1b shows a valid schedule for the simple task graph in Figure 1a, given two homogeneous processors. Note that although the task graph specifies a communication cost of 1 time unit between tasks A and B, in this schedule B is able to be placed immediately after A with no delay. Since A and B are both assigned to processor p_1, no communication is necessary and so no cost is incurred.

The distinguishing factor between the model used in this work and that used by earlier applications of state-space search to task scheduling is in the definition of the set of processors P. In both cases, P contains a finite number of processors P. The processors in P are dedicated, meaning that tasks cannot be preempted once they begin execution. In earlier work, the processors are homogeneous: for all $p \in P$, the execution time of a task n will be the same. More specifically, given a start time $t_s(n)$ for the task, the finish time $t_f(n) = t_s(n) + w(n)$, regardless of the value of $proc(n)$.

The model used in this paper, however, allows for processors to be heterogeneous: the required execution time for a task may differ from processor to processor.

Definition 1. Time Scaling Factor

A function sf maps each processor $p \in P$ to an integer time scaling factor $sf(p)$. To find the execution time of a task n on processor p, the computational weight $w(n)$ is multiplied by the time scaling factor $sf(p)$ to give the total number of time units required to complete the task. Therefore, when given a start time $t_s(n)$ for a task in this model, the finish time $t_f(n) = t_s(n) + w(n) \times sf(proc(n))$. The lowest time scaling factor among all processors is denoted by $sf^* = \min_{p_i \in P}\{sf(p_i)\}$.

This additional variable allows processors of varying speeds to be modeled, better approximating many real world systems on which programs might be scheduled. If $sf(p_i) = 1$ and $sf(p_j) = 2$, then it will take p_j twice as long to execute any given task as it would take p_i. Figure 1c shows an example of a valid schedule for the graph in Figure 1a, with $sf(p_1) = 1$ and $sf(p_2) = 2$. The absolute values of these time scaling factors are not important. It is the ratios between them that define the characteristics of P. Integers are used in order to maintain simple and precise computation and modeling. In general, when modeling a desired target system, it is useful to give the fastest processor a baseline scaling factor such as 1, 10, or 100. The scaling factors of the remaining processors can then be easily calculated according to the desired ratio. Arbitrarily precise ratios can be modeled by setting the baseline to increasing powers of ten.

Top and bottom levels of tasks are concepts useful for formulating bounds. The top level $tl(n)$ of task n is the length of the longest path through the task graph ending at n, calculated by summing the weights of the tasks in the path. Communication costs are not included, and neither is the weight $w(n)$ itself. The

top level is a lower bound on the value that could be assigned to $t_s(n)$ in a valid schedule. Analogously, the bottom level $bl(n)$ is the length of the longest path in the task graph beginning with n, also excluding communication costs but including the weight $w(n)$. This is a lower bound on the time between $t_s(n)$ and the end of the schedule. Communication costs are excluded because of the possibility that they do not need to be incurred, due to local communication - in contrast, all computation costs must always be incurred in any valid schedule.

2.2. Related Work

Many combinatorial optimisation techniques could be used in solving the task scheduling problem. Branch-and-bound search algorithms have been applied to optimal solving of small problem instances, with some success [9, 10]. Previous work using the A* search algorithm has introduced the AO [11] state-space model, whereas earlier methods [12, 7] used a state-space model that can be called exhaustive list scheduling. The AO model avoids a limitation of older models by not being able to produce duplicate states: that is, there is only one possible path through the state-space that will reach any given distinct partial solution. The AO model has been shown to significantly increase the number of problem instances that can be solved within a given time limit, and its duplicate-free nature makes it much more amenable to memory-limited and parallel branch-and-bound algorithms [13].

Integer linear programming (ILP) is another NP-hard combinatorial optimisation method which has been applied to optimal task scheduling. In this method a problem is formulated as a linear program, with the constraint that variables in a solution may only take integer values. ILP solvers use a number of algorithms to find the optimal solution, generally including highly optimised branch-and-bound search at some stage [14]. ILP formulations of the $P|prec, c_{ij}|C_{\max}$ task scheduling problem [15, 16, 17, 18] are able to efficiently solve problem instances of comparable size to those which can be handled by pure branch-and-bound approaches, and therefore neither method has yet been shown to be most effective.

For optimal task scheduling, most work has focused on solving the problem with homogeneous processors [19, 15, 18]. A* search has been used to address the optimal allocation of tasks on heterogeneous distributed systems [20], and the scheduling of independent parallel tasks on heterogeneous systems [21]. Some ILP formulations allow for optimal scheduling on heterogeneous processors [22, 23, 24, 25, 26]. Formulations have also been presented for scheduling models which include heterogeneity in the communication layer [17]. The AO model's suitability for modeling partial heterogeneity may allow it to have an advantage when scheduling for some of the most common types of heterogeneous systems.

A number of heuristic algorithms for task scheduling have been proposed to work with heterogeneous processors [3, 4, 27, 28]. In most cases, adaptation for heterogeneity can be as simple as using an "earliest finish time" metric instead of "earliest start time" when constructing a schedule. Introducing heterogeneity therefore does not tend to increase the computational complexity of heuristics by more than a small constant factor.

2.3. Branch-and-Bound

Branch-and-bound is a method of state-space search used to solve combinatorial optimisation problems. Under this method all of the possible solutions to a problem can be investigated implicitly without resorting to actual exhaustive enumeration [29]. States in the search tree represent partial solutions to the problem under consideration, with some decisions fixed and others not. A *branching* operation is defined which takes a partial solution s and transforms it into a set of new, more complete partial solutions. These new states are referred to as the child states of s. Each state s is also *bounded* using a function $f(s)$ which gives a lower bound on the cost of complete solutions which could exist as descendants of s. This bound is commonly known as the f-value of s. These bounds allow states (and simultaneously their descendants) to be disregarded by the search, as they can be used to prove that they cannot lead to a better solution than other states elsewhere in the state-space. It is required that the cost function f provides an underestimate, in which case it is said to be an admissible heuristic. Specifically, an admissible cost function guarantees that $f(s) \leq f^*(s)$, where $f^*(s)$ is the actual lowest cost of any complete solution in the sub-tree rooted at s. For a branch and bound algorithm to be guaranteed to produce an optimal solution for any problem instance, two conditions must be met - the branching procedure must allow all valid solutions (and only valid solutions) to be reached, and the cost function using in bounding must be admissible.

One of the most well known and widely used variants of branch-and-bound is the A* algorithm. This algorithm uses a best-first approach, and has the particularly desirable property of being optimally efficient [30]. This means that, using the same cost function f, there can exist no algorithm which is guaranteed to examine less states while solving a problem. A* commonly uses a data structure known as a Closed set in order to detect when duplicate states (i.e. those previously visited via another path) are encountered.

2.4. Allocation-Ordering Model for Homogeneous Processors

Allocation-Ordering (AO) is a state-space model for task scheduling in which complete schedules are constructed through two distinct phases, combined into a single search tree [11]. In the first phase, allocation, the assignment of tasks to processors is decided. In the second phase, ordering, the sequence in which tasks on each processor will be executed is decided, based on a complete allocation from the previous phase. In combination, these phases completely define both the *proc* and t_s functions and produce a valid schedule. A search algorithm may move between these phases as required. The AO model was developed in order to overcome a shortcoming of earlier state-space models: a potential for duplicate states. Previous research has shown that this allows the AO model to provide superior performance in optimal solving. The initial formulation of the AO model assumes the use of homogeneous processors.

The goal in the allocation phase is a complete allocation of tasks to processors, and each state represents a partial allocation. We model an allocation as a partition of the set of tasks

V - a set of mutually exclusive subsets, the union of which is equal to the original set. A partition of some subset V' of V may be called a partial partition of V. Each state in this phase is therefore a partial allocation A which is a partition of some V'. The branching operation produces child states which are more complete partial allocations by adding a single additional task from V. Starting with a list of the tasks $n \in V$ arranged in a topological order, at each branching step we take a new task n_i from the head of the list and insert it into a single part in the partition. It can either be added to an existing part, or used to start a new part, with the full range of decisions here giving the set of child states. Partial partitions cannot contain more parts than there are processors in P. All possible complete partitions of V can be created using this method.

States in the allocation phase are bounded using two different metrics. The first lower bound is the maximum total computational load among any of the parts $a \in A$. The second lower bound is the "allocated critical path" through the task graph, given the allocations decided in A. The allocated critical path is determined using the allocated top level, $tl_A(n)$, and the allocated bottom level, $bl_A(n)$, of the tasks $n \in V$. These are very similar to the top and bottom levels already define, but incorporate communication costs which it is now known must be incurred. If there is an edge e_{ij}, and the tasks n_i and n_j have been assigned to different parts in our allocation A, then we know that the communication cost $c(e_{ij})$ must be incurred in any valid schedule derived from this allocation. We can therefore include the weight of this edge when calculating allocated top and bottom levels. The maximum of these two bounds gives an overall lower bound for the state:

$$f_{\text{load}}(s) = \max_{a \in A} \left\{ \sum_{n \in a} w(n) \right\} \tag{1}$$

$$f_{\text{acp}}(s) = \max_{n \in V} \{ tl_A(n) + bl_A(n) \} \tag{2}$$

A state representing a complete allocation has just a single child state: the beginning of a new ordering phase. We use our complete allocation A to fully define $proc(n)$ by arbitrarily mapping each part $a \in A$ to a processor $p \in P$. In the ordering phase, each state represents a "partial ordering". For each processor, a ready list is maintained of tasks whose dependencies have been satisfied. At each branching step, a single task n is selected from the ready list of a processor and "ordered": placed next in sequence on that processor. The full range of ready tasks that could be chosen defines the set of child states. The basic criterion for "readiness" is as follows: a task n_i allocated to p_i is considered ready if there is no unordered task n_j also allocated to p_i which is an ancestor of n_i in the graph G. More complex criteria are required for the state-space model to work most efficiently [8]. In order to ensure duplicates are avoided, the next processor to have a task ordered must be decided by some deterministic scheme, such that the processor chosen can be determined solely by the depth of the state. A simple round robin scheme suffices. Once all tasks have been ordered, a valid schedule can be uniquely derived by defining $t_s(n)$ for all $n \in V$ as the earliest start time possible given the allocation and ordering decided.

To find lower bounds in the ordering phase, we use the concept of an estimated earliest start time $eest(n)$ for each task. This is the lowest value that $t_s(n)$ could take, constrained by the complete allocation and partial ordering decided so far. For an unordered task, $eest(n) = tl_A(n)$. For ordered tasks, we first define $prev(n)$ as the task which is ordered directly before n on the processor $proc(n)$. We also define the estimated data ready time $edrt$. If n_j is a source task, with no parent tasks, then we have:

$$edrt(n_j) = 0$$

Otherwise, if n_j has one or more parents, the $edrt$ is defined as:

$$edrt(n_j) = \max_{n_i \in pred(n_j)} \left\{ eest(n_i) + w(n_i) + \begin{cases} c(e_{ij}), & proc(n_i) \neq proc(n_j) \\ 0, & \text{otherwise} \end{cases} \right\} \tag{3}$$

We then have the following for $eest$:

$$eest(n) = \begin{cases} edrt(n), & prev(n) = \emptyset \\ \max(eest(prev(n)) + w(prev(n)), edrt(n)), & prev(n) \neq \emptyset \end{cases} \tag{4}$$

With these definitions, we are again able to derive two metrics for bounding, which parallel those used in the allocation phase. The first lower bound is the partially scheduled critical path, derived by finding the maximum among all $n \in V$ of the estimated earliest start time plus the bottom level. The second lower bound is the largest among all processors of the estimated finish time plus the sum of the weights of the tasks not yet ordered on that processor. The maximum of these two bounds gives an overall lower bound for the state:

$$f_{\text{scp}}(s) = \max_{n \in V} \{ eest(n) + bl_A(n) \} \tag{5}$$

$$f_{\text{ordered-load}}(s) = \max_{p \in P} \left\{ t_{\text{f}}(p) + \sum_{n \in unordered(s): proc(n) = p} w(n) \right\} \tag{6}$$

3. Heterogeneous Processors in the AO Model

In this section, we describe how we have adapted the AO state-space in order to allow the solution of task scheduling problems using heterogeneous processors. This has been achieved mostly through changes to the allocation phase, leaving the ordering phase almost entirely the same.

The allocation phase of AO models allocations as partitions, which takes advantage of the homogeneity of processors on target systems for which it was originally designed. Through the recognition that these homogeneous processors are entirely interchangeable, the complexity of the state-space attributable to allocation is significantly reduced. With heterogeneous systems

(or even homogeneous systems, as with the ELS model without processor normalisation pruning), other models often treat all processors in the system as if they are distinct, leading to an allocation complexity of $O(|P|^{|V|})$. For a system in which all processors are truly different from all others, this complexity is unavoidable. However, in real life heterogeneous systems, it is often the case that subsets of the processors are identical to each other - a common example would be a system which has a number of CPUs working in tandem with one or more GPUs. We can generalise the AO model's strategy of modeling allocation with partitions to heterogeneous processors such that the complexity of the state space is minimised for any given target system. This means that every possible truly distinct allocation onto the target system can be reached by following exactly one path through the state-space.

3.1. Processor Category

To begin, we define the concept of a processor category. When scheduling, we are attempting to efficiently arrange the components of a program, represented by the task graph G, onto a target parallel system, abstracted as the set of processors P. Each processor $p \in P$ has a time scaling factor $sf(p)$, allowing the processors to have arbitrarily differing speeds from each other. However, we observe that real world systems often do not exhibit complete heterogeneity, where no two processors are the same, but instead contain several distinct classes of processor. Processors may differ greatly between these groups, but within such a group they are identical.

Definition 2. Processor Category

A function *cat* maps processors $p \in P$ to a set of processor categories $K(P)$. The function *cat* defines an equivalence class such that for processors p_i and p_j, $cat(p_i) = cat(p_j)$ **if and only if** $sf(p_i) = sf(p_j)$. Therefore, if two processors have the same time scaling factor, they belong to the same processor category $\kappa_i \in K$,

In a homogeneous system, there is only one processor category. In a fully heterogeneous system, the number of categories is equal to the number of processors. A system with a number of categories greater than one but less than the number of processors can be termed a partially heterogeneous system.

3.2. Heterogeneous Allocation

Before beginning the allocation phase, we first determine how many processor categories the target system possesses, and the number of processors contained in each category. We then proceed with constructing a partition of V as normal, but with the parts of the partition A_K each labeled by a category κ_i. This is done as follows. We start with a list of the tasks $n \in V$, arranged in some topological order. At each step, we take the next task n_i from the head of the list. We then choose a category $\kappa_i \in K$. Once we have made this choice, we choose to insert the task n_i into one of the existing parts in A_K which is labeled by κ_i, or to begin a new part labeled by κ_i with that task. The maximum number of parts which can be created with a given label κ_i is equal to the number of processors in that category, $|\kappa_i|$. Algorithm 1 gives pseudocode for this process. Figure 2 shows the possible children of a single allocation state, both with a homogeneous and a heterogeneous target system, demonstrating the various ways in which a new task D can be inserted into the partial partition. We see that more children are possible with the heterogeneous system, as when adding a new part to the partition there is a choice as to which processor category this new part will be labeled with. The following proof, adapted from work on the original AO model[8], demonstrates that all possible allocations onto heterogeneous proccesors can be produced with this bounding procedure.

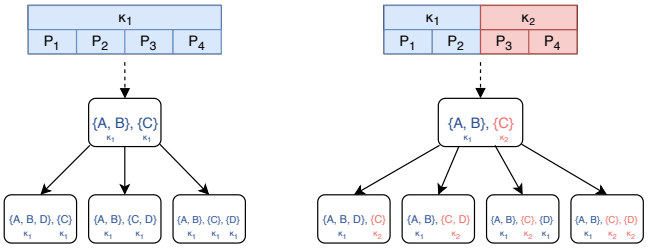

Figure 2: Branching in the allocation phase for a homogeneous and a heterogeneous system.

Lemma 1. ***The allocation phase of the AO model with heterogeneity can produce all possible labeled partitions of tasks.***

Proof. We show how any given allocation A_K^Ω, which is a complete partition of V labeled by K, is constructed with the proposed allocation procedure. We begin with an empty partial allocation $A_K = \{\}$, and are presented with the tasks in V in a fixed order $n_1, n_2, \ldots n_{|V|}$. We must always begin by placing n_1 into a new part, so that $A = \{\{n_1\}\}$. The new part $a_1$to which n_1 belongs must then be labeled with the same $\kappa_i \in K$ as the part to which n_1 belongs in A_K^Ω. Now we must place n_2. If n_1 and n_2 belong to the same part in A_K^Ω, they must also be placed in the same part in A_K, and so we must have $A_K = \{\{n_1, n_2\}\}$. Conversely, if n_1 and n_2 belong to different parts in A_K^Ω, the same must be true in A_K, and we make $A_k = \{\{n_1\}, \{n_2\}\}$. The part $a_2$to which n_2 now belongs must be given the same label as the part to which n_2 belongs in A_K^Ω. For each subsequent task n_i, if n_i in A_K^Ω belongs to the same part as any of tasks n_1 to n_{i-1}, we must place it in the same part in A_K. If n_i does not share a part in A_K^Ω with any task n_1 to n_{i-1}, it must be placed in a new part by itself, the label of which is exactly determined by the label of the part a_j to which n_i belongs in A_K^Ω. At each step, there is exactly one possible move that can be taken in order to keep A_K consistent with A_K^Ω. Since there is always at least one move, it is possible to produce any partition of V in this fashion. □

3.3. Levels and Bounds with Heterogeneity

Bounding in this new allocation phase is very similar, except that the differing time scaling factors of the processors must be taken into account when calculating the top and bottom levels and the total computational load. The total computational load of a part must simply be multiplied by the time

scaling factor $sf(\kappa_i)$ of the category with which that part is labeled. Recall that in order for the result of our search to be provably optimal, the cost function must be admissible; it must never overestimate the length of the shortest schedule that can be reached from the current state. In formulating these bounds we therefore assume that the minimum possible computation and communication costs will be incurred, in order to produce an underestimate. If a task has been allocated in the current state, we then know what its final computation cost will be, determined by the scaling factor of the part it is allocated to. All tasks which are still unallocated are instead considered to have the lowest possible scaling factor, sf^*. Looking at the top and bottom levels, we see that the minimum start time will be obtained if all tasks on the path are placed on the fastest processor. The top level $tl(n)$ in this context is then equal to the sum of the weights of the nodes in the path, multiplied by sf^*. The bottom level $bl(n)$ is changed in the same way.

$$tl(n) = \max_{n_p \in pred(n)} \left\{ tl(n_p) + w(n_p) \times sf^* \right\} \tag{7}$$

$$bl(n) = w(n) \times sf^* + \max_{n_s \in succ(n)} \{bl(n_s)\} \tag{8}$$

When calculating **allocated** top and bottom levels, the scaling factor of the part a task is allocated to must be considered when determining the task's execution time. Given a path ϕ in the task graph G, and allocation A, we now find the length of that path by summing the weights of nodes and edges as follows: if a task n has not been allocated, we add $w(n) \times sf^*$. If n has been allocated, we add $w(n) \times sf(\kappa_i)$. If n_i and n_j have each been allocated, and to different parts, we add $c(e_{ij})$. If either has not been allocated, or both are allocated to the same part, we add zero - assuming that the communication cost of the relevant edge will not be incurred, until it is proved otherwise. This definition allows both the top and bottom levels to be carried over to the heterogeneous context. The relevant weights of tasks and edges for an allocation A can be expressed as follows:

$$w_A(n) = \begin{cases} w(n) \times sf(proc(n)), & n \in A \\ w(n) \times sf^*, & n \notin A \end{cases} \tag{9}$$

$$c_A(n_p, n_i) = \begin{cases} c(e_{pi}), & n_i, n_p \in A \wedge proc(n_i) \neq proc(n_p) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

Incorporating these definitions gives us these formulas for allocated top and bottom levels under the heterogeneous model:

$$tl_A(n) = \max_{n_p \in pred(n)} \left\{ tl_A(n_p) + w_A(n_p) + c_A(n_p, n) \right\} \tag{11}$$

$$bl_A(n) = w_A(n) + \max_{n_s \in succ(n)} \{bl_A(n_s) + c_A(n, n_s)\} \tag{12}$$

Algorithm 1 Defining child states in the allocation phase with heterogeneity.

Input: A, a partial partition of V
Input: *unallocated*, topologically ordered list of tasks in V not contained in A
Input: C, a partition of P defining processor categories
Output: child states of A
$nextTask \leftarrow$the head of *unallocated*
$children_A \leftarrow \emptyset$
for $\kappa_i \in C$ **do** `// for each category`
1 $K_i \leftarrow a \in A \mid a$ is labeled with κ_i
`/* for each part belonging to category` κ_i `*/`
2 **for** $a \in K_i$ **do**
3 $children_A \leftarrow children_A \cup ((A \setminus a) \cup (a \cup \{nextTask\}))$
4 **if** $|K_i| < |\kappa_i|$ **then**
`/* more processors in` k_i `can be used */`
5 $children_A \leftarrow children_A \cup (A \cup \{nextTask\}_{\text{label } \kappa_i})$
return $children_A$

3.4. Ordering

In the ordering phase, branching is not affected at all by the introduction of heterogeneity. However, the scaling factors of each processor need to be factored into the calculation of the bounds. This is done very simply by substituting $w(n)$ with $w(n) \times sf(proc(n))$ wherever it is used. This, along with the definition of the allocated top and bottom levels already given in Section 3.3, is all that needs to be modified for this phase.

3.5. State-space Growth

The redefinition of the allocation phase changes the structure of the state-space, as it introduces new distinct possibilities for complete allocations.

3.5.1. Homogeneous processors

In general, for homogeneous processors, the size of the allocation state-space can be approximated using the Bell numbers, where B_x is the number of distinct possible partitions of a set of size x [31]. The precise number of complete allocations in the state-space can be found using the Stirling numbers of the second kind, denoted by $S\,tirling(x, k)$, these giving the number of ways to divide a set of x objects into k non-empty subsets [32]. It is necessary to consider the Stirling numbers in order to account for the limited number of processors in a target system. The exact number of partitions $\mathcal{A}(|V|, P)$ of a set of tasks V which can be mapped to allocations for a target system P is found as follows:

$$\mathcal{A}(|V|, P) = \sum_{k=1}^{|P|} S\,tirling(|V|, k) = \sum_{k=1}^{|P|} \sum_{r=1}^{k} (-1)^{k-r} \binom{k}{r} r^{|V|} \tag{13}$$

The value $\mathcal{A}(|V|, P)$ will in the worst case be equal to the Bell number $B_{|V|}$ when $|P| \geq |V|$. The Bell numbers, and therefore $\mathcal{A}(|V|, P)$, have been shown to have the following upper bound [31]:

$$\mathcal{A}(|V|, P) = O\left(\left(\frac{0.792|V|}{\ln(|V| + 1)} \right)^{|V|} \right) \tag{14}$$

Parameters		Permitted Allocations	
|V|	|P|	AO	Naive
10	2	512	1024
10	4	43,947	1,048,576
10	8	115,929	1,073,741,824
21	4	$\approx 1.833 \times 10^{11}$	$\approx 4.398 \times 10^{12}$
21	8	$\approx 2.454 \times 10^{14}$	$\approx 9.223 \times 10^{18}$

Table 1: Number of permitted allocations with homogeneous processors.

$\|A\|$	Selections	Permutations	Total
1	3 (1 0 0), (0 1 0), (0 0 1)	1, 1, 1	3
2	4 (1 1 0), (1 0 1), (0 1 1), (0 0 2)	2, 2, 2, 1	7
3	3 (1 1 1), (1 0 2), (0 1 2)	6, 3, 3	12
4	1 (1 1 2)	12	12

Table 2: Number of possible labelings for system P4C3.

The number of partitions $\mathcal{A}(|V|, P)$ grows exponentially in the number of tasks. However, the growth is slower than for a naive state-space that does not consider the homogeneity of processors, where the number of permitted allocations is $|P|^{|V|}$. Table 1 shows a comparison of specific values for the number of permitted allocations between AO and such a state-space. Note that each excess allocation permitted by a naive state-space is isomorphic with one which is permitted by AO.

3.5.2. Heterogeneous processors

For a heterogeneous system, the total number of distinct partitions of the tasks remains the same; we have not introduced any new ways in which tasks can be grouped together. However, the differences between processors mean that each distinct partition can now be mapped to several different distinct allocations. In order to take advantage of partial heterogeneity it is necessary to consider the way in which the parts of the partitions are assigned to processor categories. This assignment can be modeled by giving a label representing a category to each part in a partition. The number of possible ways to label a given partition corresponds to an increase in state-space size introduced by heterogeneity. This number of possible labelings will vary with the individual characteristics of the heterogeneous system, but also with the size $|A|$ of the partition to be labeled. When considering the entire state-space, we must consider how many possible partitions exist at all possible sizes $|A|$. To aid with this, we define a vector $\vec{\boldsymbol{U}}(|V|, |P|)$ such that $\vec{\boldsymbol{U}}(|V|, |P|)_i = Stirling(|V|, i)$, from $i = 1$ to $i = |P|$. As an example, consider the system P4C3 described in Table 4, along with a task graph which has $|V| = 10$. This will give us the following vector: $\vec{\boldsymbol{U}}(10, 4) = \begin{bmatrix} 1 & 511 & 9330 & 34105 \end{bmatrix}$. Here we see that, for example, the number of partitions with size $|A| = 2$ is 511.

Now we describe how to determine the number of possible labelings for a partition of size $|A|$. Given a set of categories K, and a complete partition A, we can construct a labeled partition A_K as follows. First, we will transform the set K to a multiset K', such that each element $\kappa_i \in K$ is also an element of K', with a multiplicity $\sigma(\kappa_i, K')$ equal to the number of processors in P belonging to category κ_i. Note that the multiplicity $\sigma(\kappa_i, K')$ is the number of times κ_i appears in K'. We then perform the following two operations:

1. Select a multiset K'' of $|A|$ labels from K', one for each subset in the partition A.
2. Choose some permutation of the selected labels K'' and apply them to the partition A to produce A_K.

P4C3 has four processors divided into three categories: $|P| = 4$, $|K| = 3$, $|K'| = 4$. For a given partition of tasks A_j with say three parts, i.e. $|A_j| = 3$, we select a multiset of labels of size 3 from K' . These three labels K'' are permuted in some order and the parts of A_j are assigned these labels (i.e. processors) in that order.

Both the number of possible selections of labels, and the number of possible permutations of the selected labels, factor into the total number of additional distinct possibilities that heterogeneity introduces. Unfortunately, the number of possible distinct selections from K' is difficult to express - explicit formulas for the number of k-combinations of a multiset exist, but they are very complex [33]. Treating each element as distinguishable, the number of combinations $\binom{|K'|}{|A|}$ gives an upper bound, but this is by no means tight. For a given heterogeneous system, the possible selections can be individually enumerated using an algorithm [34]. Table 2 gives a full enumeration of possible selections K'' for P4C3. We represent these selections as a tuple of the multiplicities of the elements of K in K'' e.g. (1 0 2).

For each selection K'' the number of distinct permutations (derived from the standard formula for permutations of a multiset [35]) is given exactly by:

$$\pi(K'') = \frac{|A|!}{\prod_{\kappa_i \in K} \sigma(\kappa_i, K'')!} \tag{15}$$

Table 2 also shows the number of permutations possible for each selection. The total number of labelings possible for a partition of size $|A|$ is the sum of these, which can be seen in the final column. For each heterogeneous system, we can define a vector $\vec{\boldsymbol{L}}(P)$ such that element $\vec{\boldsymbol{L}}(P)_i$ is the total number of labelings possible where $|A| = i$, e.g. $\vec{\boldsymbol{L}}(P4C3) = \begin{bmatrix} 3 & 7 & 12 & 12 \end{bmatrix}$. The total number of permitted allocations is then the dot-product of the labeling vector and the partition vector:

$$\mathcal{A}(|V|, P) = \vec{\boldsymbol{L}}(P) \cdot \vec{\boldsymbol{U}}(|V|, |P|)$$

$$\mathcal{A}(10, P4C3) = \begin{bmatrix} 3 \\ 7 \\ 12 \\ 12 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 511 \\ 9330 \\ 34105 \end{bmatrix} = 524800$$

Parameters		Permitted Allocations		
|V|	System ID	Without Partial	With Partial (Exact)	With Partial (Bound)
10	P4C1	1,048,576	43,947	43,947
10	P4C2	1,048,576	175,275	175,788
10	P4C3	1,048,576	524,800	527,364
10	P4C4	1,048,576	1,048,576	1,054,728
21	P4C1	$\approx 4.398 \times 10^{12}$	$\approx 1.833 \times 10^{11}$	$\approx 1.833 \times 10^{11}$
21	P4C2	$\approx 4.398 \times 10^{12}$	$\approx 7.33 \times 10^{11}$	$\approx 7.33 \times 10^{11}$
21	P4C3	$\approx 4.398 \times 10^{12}$	$\approx 2.199 \times 10^{12}$	$\approx 2.199 \times 10^{12}$
21	P4C4	$\approx 4.398 \times 10^{12}$	$\approx 4.398 \times 10^{12}$	$\approx 4.398 \times 10^{12}$

Table 3: Number of permitted allocations with and without considering partial heterogeneity.

Upper Bound for Heterogeneous Processors

Due to the potential difficulty in enumerating the number of selections of labels, it is desirable to formulate an upper bound for the total number of ways to label a partition, $\boldsymbol{L}_{\approx}(P)$.To do this we can instead disregard step 1, as in the case in which $|A| = |P|$. In this case, we select the entire multiset in step 1, so that $K'' = K'$. By taking a permutation of K', and applying the first $|A|$ labels in the sequence to our partition A, we can produce all possible values for A_K - but any given value may be reached in more than one way. The upper bound is therefore:

$$\boldsymbol{L}_{\approx}(P) = O\left(\frac{|P|!}{\prod_{\kappa_i \in K} (\sigma(\kappa_i, K'))!}\right) \tag{16}$$

We will use this value as the number of possible labelings for partitions of all sizes $|A|$, and thereby obtain an upper bound. Note that in a fully heterogeneous system, where $|K| = |P|$ and therefore $\sigma(\kappa_i, K') = 1$ for all κ_i, this bound is equal to $\boldsymbol{L}_{\boldsymbol{P}}(A) = O(|P|!)$. In other words, the denominator of (16) is the factor by which the state space is smaller when considering partial heterogeneity (i.e. categories) as we propose, in comparison to (naive) full heterogeneity. In a homogeneous system, the denominator will be equal to $|P|!$, and therefore $\boldsymbol{L}(A) = O(1)$. In general, the fewer processor categories, the lower the value of $\boldsymbol{L}(A)$.

Table 3 shows a comparison of specific values for the number of distinct allocations permitted with and without consideration of partial heterogeneity. The parallel systems referenced here are described in Table 4. The table lists both exact values, having been calculated according to their individual special cases, and the values given by the bound above. It is clear that when the grouping of processors into categories is taken into account, the number of distinct possibilities decreases dramatically - even with only one less category than the total number of processors, the number of permitted allocations is reduced by half. It can also be seen that, at least for these target systems, the bound is quite tight.

The bound will be very tight whenever $|V|$ is sufficiently larger than $|P|$. The value $\boldsymbol{L}_{\approx}(P)$ is naturally exactly correct

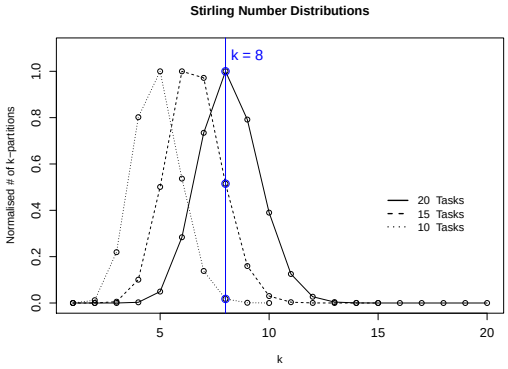

Figure 3: Normalised distribution of $S\,tirling(|V|, k)$ for some values of $|V|$.

for partitions with size $|A| = |P|$. It is also correct for partitions with size $|A| = |P| - 1$, as these will always have the same number of possible labelings. To see why this is true, consider that any $|P|$-permutation of K'can be uniquely transformed to a $(|P| - 1)$-permutation by removing its last element. Given this, the tightness of the bound will depend in large part on the proportion of the total set of possible partitions that have $|A| = |P|$ or $|P| - 1$.

To further study when the bound $\boldsymbol{L}_{\approx}(P)$ is tight, consider the following. For a given $|V|$, there is a value k^* at which $S\,tirling(|V|, k)$ has its maximum value. Remember that the total number of partitions $\mathcal{A}(|V|, P)$ is given by the sum of Sterling numbers, eq. (13). The number of possible k-partitions grows exponentially as k increases, comes to a peak at k^*, and then decreases exponentially thereafter. Figure 3 shows the distribution of $S\,tirling(|V|, k)$ for several values of $|V|$ (note that the values on the vertical axis have been scaled relative to the peaks of their respective curves to allow better comparison of the curves along the horizontal axis; by absolute values, each successive peak is exponentially higher than the last). Highlighted in this figure are the points where $k = 8$. If our $|V|$ and $|P|$ are such that $|P|$ is to the left of or exactly at the peak, i.e. $|P| \leq k^*$, our bound will be very tight. The proportion of total partitions with $|P|$ or $|P| - 1$ parts is guaranteed to be very high. This is true because even the sum all partitions with a smaller k is significantly less than at k^*, due to the initial exponential growth. When $|P|$ falls farther on the right side of the peak, i.e. $|P| > k^*$, the bound is likely to become much less tight, as the exponential decline means the proportion of total partitions with $|P|$ or $|P| - 1$ parts will be low. Figure 3 demonstrates that if we have $|P| = 8, |V| = 20$, then $|P| = k^*$ and the proportion of partitions with $|P|$ or $|P| - 1$ parts is very high. For $|P| = 8, |V| = 10$ the proportion of partitions with $|P|$ or $|P| - 1$ parts is very low, and $|P| = 8, |V| = 15$ falls between these extremes.

Figure 4 demonstrates how the accuracy of the bound varies as $|V|$ increases for three of the systems described in Table 4. Note that in the previous figure we had fixed $|V|$ and varying k (corresponding to $|P|$), while in this figure we have fixed $|P|$ and varying $|V|$. The red box in the figure highlights the region in which the k^* for those $|V|$ is equal to $|P|$. The bound accuracy decreases quickly to the *left* of this region: as $|V|$ decreases,

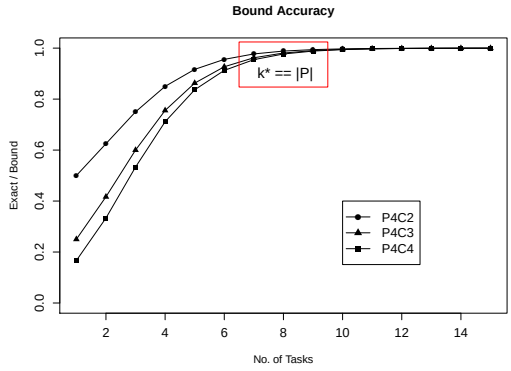

Figure 4: Accuracy of upper bound for three systems.

k^*also decreases, and the fixed $|P|$ becomes increasingly greater than k^*.

Strong Heterogeneity

It is clear that we should expect heterogeneous scheduling to be more difficult than homogeneous scheduling, and for the difficulty to increase significantly as the number of processor categories increases. However, there may be a counterbalancing factor: if one processor category is faster than another it is likely that, on average, optimal schedules will allocate more tasks to that category than to slower categories. As the disparity in speed between the categories grows, it becomes less and less likely that moving a task from a fast processor to a slow processor will be worth the trade-off between the concurrency that the slower processors allow, and the cost incurred by introducing remote communication. With a sufficiently high ratio between fast and s'low, optimal schedules can be forced to always use only the faster categories of processor. We will term systems with large differences between processor categories **strongly heterogeneous**, and systems with small differences **weakly homogeneous**.

The dividing line between these two is not strictly defined. We can, however, define a simple measure that will provide an intuition for the relative strength of heterogeneity between many systems.

Definition 3. Grade of Heterogeneity

The grade of heterogeneity $\mathcal{H}(P)$ of a target system P is defined as:

$$\mathcal{H}(P) = \frac{max_{p \in P}\{sf(p)\}}{min_{p \in P}\{sf(p)\}} \tag{17}$$

Hence $\mathcal{H}(P)$ is the ratio of the scaling factor of the slowest to that of the fastest processor. The grade of heterogeneity can be used to compare the strength of heterogeneity between different target systems. If P_i and P_j are systems, and $\mathcal{H}(P_i) > \mathcal{H}(P_j)$ then we can say that P_i is more strongly heterogeneous and P_j is more weakly heterogeneous.

When scheduling on a more strongly heterogeneous target system, we may expect that in many cases the "correct" allocation decision will be obvious enough that this will be reflected by a significant difference in f-values. From this it would follow that, although the overall state-space will be much larger, in some cases it may be necessary for the search to examine less of that state-space than if the target system was homogeneous.

4. Evaluation

This section presents an experimental evaluation of the proposed state-space model for optimal scheduling on heterogeneous processors. We performed state-space searches for optimal schedules on a large set of task graphs using a variety of different target systems, pursuing two objectives: firstly, to study whether awareness of partial heterogeneity through the use of categories can effectively reduce the search space. Further, we want to determine the effect of the grade of heterogeneity on the difficulty of solving task scheduling problems with the AO model. Lastly, we put our state-space search results into relation with ILP-based approaches to optimal scheduling with heterogeneous processors.

In order to pursue the first objective, we wished to compare the AO model considering partial heterogeneity against models which do not consider partial heterogeneity. The heterogeneous AO model described in this work can be made to ignore partial heterogeneity by treating each processor as if it belonged to its own category, regardless of the natural categories of the target parallel system. In addition, an older state-space model for optimal task scheduling, Exhaustive List Scheduling (ELS), was modified in order to allow scheduling with heterogeneous processors by substituting $w(n)$ with $w(n) \times sf(proc(n))$ wherever it is used.

4.1. Setup

The searches were performed using the A* branch-and-bound algorithm with the AO state-space (both considering partial heterogeneity, and treating each system as if it were completely heterogeneous) as well as the ELS state-space. We opted to use A* to enable a fair comparison, since the ELS state space produces duplicate states which can be most efficiently detected with a Closed set[13]. The nine target systems used are described in Table 4. Due to the length of time required to complete experiments, only a limited number of target systems could be selected for the data-set. Target systems were chosen with either 2, 4 or 6 processors in total. Scheduling on two processors often appears to be a special case, and is therefore interesting for comparison. Due to the size of the graphs to be used, it was considered that using more than six processors would have diminishing returns - it is likely that many optimal schedules would leave most processors empty, and trivial solutions may become possible. With four processors being a reasonable midpoint, most of the selected target systems fall into this category. Homogeneous systems with 2, 4 and 6 processors were used as a baseline for comparison. The systems selected provide variation in both number of processor categories and strength of heterogeneity. All possible numbers of categories with four processors were represented. Most of the target systems were considered weakly hetereogeneous, with a grade $\mathcal{H}(P) = 2$.

ID	Procs	Categories	Procs / Cat	Time Scaling Factors	$\mathcal{H}$
P2C1	2	1	(2)	(100)	1
P2C2	2	2	(1, 1)	(100, 200)	2
P4C1	4	1	(4)	(100)	1
P4C2	4	2	(1, 3)	(100, 200)	2
P4C2S	4	2	(1, 3)	(100, 600)	6
P4C3	4	3	(1, 1, 2)	(100, 150, 200)	2
P4C4	4	4	(1, 1, 1, 1)	(100, 120, 150, 200)	2
P6C1	6	1	(6)	(100)	1
P6C2	6	2	(3, 3)	(100, 600)	6

Table 4: The target systems used for evaluation.

Two strongly heterogeneous systems with $\mathcal{H}(P) = 6$ were also included.

An initial set of 340 task graphs were selected, each with 21 tasks. Prior experiments for homogeneous processors have shown this to be an input size of medium difficulty [8]. The graphs are a mix of the following DAG structure types: Independent, Fork, Join, Fork-Join, Out-Tree, In-Tree, Pipeline, Random, Series-Parallel, and Stencil. They also have a roughly equal mix of the following CCR values: 0.1, 1, and 10. These graphs have randomly distributed task and edge weights. Additionally, a smaller set of 90 graphs was generated with unit weights for tasks and edges - meaning that all tasks have the same weight, and all edges have the same weight. This was achieved by taking the graphs with Fork-Join, Pipeline and Stencil structures and producing copies in which all task and edge weights were replaced with the average of the existing values. The result is a set of unit-weighted graphs which preserve the structure and CCR of the random-weighted originals. The motivation for this additional set of homogeneous graphs is to observe their scheduling behaviour on heterogeneous systems, akin to scheduling heterogeneous graphs on homogeneous systems. All of the task graphs used for evaluation in this paper are available for download.[1]

A* and the heterogeneous processor AO model as described in Section 3 are implemented in Java. For each task graph, we attempted to find an optimal schedule for each target system, with a time limit of two minutes allowed for each search to complete, and a maximum heap size of 96 GB. This was performed with each of: AO with partial heterogeneity, AO without partial heterogeneity, and the older Exhaustive List Scheduling state space model without partial heterogeneity. There were a total of 11,610 trials. Each trial was run on a Linux machine with 4 Intel Xeon E7-4830 v3 @2.1GHz processors with in total 48 cores. To remove the possibility of previous trials affecting subsequent ones due to garbage collection or JIT compilation, a new JVM instance was started each time.

[1] https://parallel.auckland.ac.nz/data/HetGraphSet2021.zip

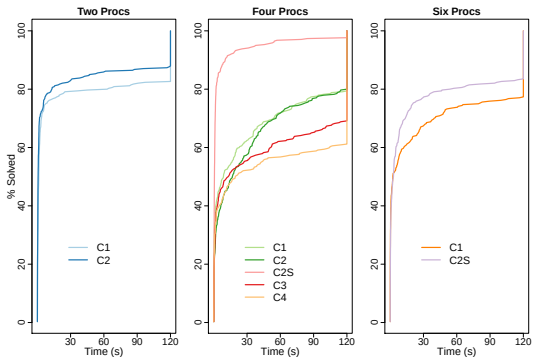

Figure 5: Performance profiles for each target system.

4.2. Results

To begin, we will look at the results of the trials for AO using partial heterogeneity, grouped by target system. These are shown in Figure 5 as performance profiles [36]: the x-axis shows time elapsed, while the y-axis shows the cumulative percentage of problem instances which were successfully solved by this time.

For the target systems with two processors, where P2C1 is a homogeneous system and P2C2 a heterogeneous system, it is clear that more problem instances are solved with the heterogeneous system than with the homogeneous. This would seem to suggest that the introduction of heterogeneity made the problem easier. However, we note that two processors may be a special case, as any heterogeneous system with two processors will have a relatively strong contrast in processor speed, as discussed in Section 3.5.

Moving on to the four processor systems, we see more variety in results. As shown in Figure 5, our baseline homogeneous system (P4C1) allows 79% of problem instances to be solved. With the strongly heterogeneous system P4C2S, we actually see better performance than with the homogeneous system, having 97% of problem instances solved. Like P2C2, the system P4C2 shows performance quite similar to the homogeneous system. On the other hand, the systems with three (P4C3) and four (P4C4) processor categories show significantly worse performance than the homogeneous system. It appears that the two category systems are, in fact, both strongly heterogeneous enough to counteract the effect of the larger state-space. The other systems, however, are not, and we see a downward trend as the number of categories increases.

The six processor target systems exhibit much the same pattern shown by P4C2S, as seen in Figure 5. The homogeneous system (P6C1) allows 77% to be solved, while the strongly heterogeneous system P6C2 has 83% solved. This is not a large difference, but the curve of the performance profile shows us that a significant proportion of problem instances were solved much earlier when using P6C2.

As expected, these results suggest that scheduling with heterogeneous processors is more difficult in general than with homogeneous processors, but also that stronger heterogeneity makes it easier. Also as expected, we see better performance when scheduling on target systems with a lower number of pro-

cessor categories. There is a downward trend in performance as the number of categories increases, and therefore as the complexity of the state-space rises.

To examine the effects of partial heterogeneity, Figure 6 shows performance profiles for each of the target systems (Table 4) when using AO while either considering partial heterogeneity (solid line), or treating each system as if it were completely heterogeneous (dashed line), and for comparison using the ELS state-space model (not considering partial heterogeneity). It is clear that considering partial heterogeneity gives the AO model a significant advantage, and as would be expected the benefit is larger as the number of processor categories (for a fixed number of processors) decreases: for P4C1, considering partial heterogeneity allows an additional 31% of schedules to be solved, for P4C2 an additional 27%, and for P4C3 an additional 7%.

For AO with complete heterogeneity (dashed lines), there is a trend of increasing performance as the number of natural processor categories in the target system increases, which is the reverse of the previously observed trend for partial heterogeneity. A particularly interesting comparison is between P4C1 with complete heterogeneity and P4C4 (which is naturally completely heterogeneous): we see that task graphs were in fact easier to schedule on P4C4 in this case, very likely due to the effects of the grade of heterogeneity already discussed. This seems to indicate that unless partial heterogeneity is specifically addressed by the state-space model, heterogeneous systems with lower numbers of natural categories do not become any easier than the completely heterogeneous case, and may in fact be more difficult.

The older ELS state-space model, which also does not consider partial heterogeneity, shows considerably worse results for all systems, which demonstrates the advantage of AO even without the consideration of partial heterogeneity. Strong heterogeneity, as demonstrated by P4C2S, has a large benefit for both partial and complete heterogeneity.

4.2.1. Results - CCR

Since we hypothesise that heterogeneous systems with a higher grade of heterogeneity make optimal scheduling easier by making the trade-off between additional concurrency and communication costs more easy to evaluate, it seems natural to examine the impact of the computation-to-communication ratio (CCR) of the task graphs. The task graphs in the test set have either low (0.1), medium (1), or high (10) CCRs, or the tasks are independent (no edges). Figure 7 shows performance profiles for three of the target systems (P4C1, P4C2S, P4C4), with results for AO considering partial heterogeneity, grouped by CCR. These three systems were chosen for further analysis as they showed the most extreme differences in performance from one another.

Graphs with independent tasks are seen to be uniformly easy for AO, and we will not analyse these further. For the homogeneous system (P4C1), CCR does not appear to have much impact; the curve of the performance profile shows that more graphs with high CCR are solved quickly, but the curves converge towards the time limit. For the weakly heterogeneous system P4C4 there is a clear difference, with low CCR graphs being most difficult, and high CCR graphs being significantly easier. For the strongly heterogeneous system P4C2S CCR makes a very significant impact, as 100% of low and 99% of medium CCR graphs are able to be solved within 60 seconds, while only 93% of high CCR graphs are solved within the two minute time limit. It seems that for the strongly heterogeneous system, the difficulty of scheduling increases with higher CCR. Even in its worst case, however, we see that scheduling on P4C2S is still easier than scheduling on P4C1 in the best case.

The reversal in difficulty of low CCR graphs between weak and strong heterogeneity seems to support the hypothesis: when heterogeneity is weak, large communication costs will dominate the schedule length, and decisions which remove these communications are more obviously beneficial. When heterogeneity is strong, deciding where to perform computations becomes more important, and communications become relatively less important; decisions regarding large communication costs therefore become less obviously beneficial or harmful.

Figure 8 again shows performance profiles comparing AO with and without partial heterogeneity, and for comparison ELS, but now grouped by CCR. Again, the benefit of considering partial heterogeneity is evident across all categories. With low and medium CCR, considering partial heterogeneity allows an additional 20% and 19% of problem instances to be solved, respectively. With high CCR, only an additional 10% of problem instances were solved when considering partial heterogeneity. It appears that the relative difficulty of low and medium CCR graphs is higher when treating systems as completely heterogeneous than when partial heterogeneity is considered. However, some of this difference could be explained by the fact that high CCR graphs were already significantly easier in general, and therefore there is less opportunity for improvement.

4.2.2. Results - Unit weights

For target systems with homogeneous processors, it is easier to schedule task graphs with unit-weighted tasks (having a single value for all task weights, and a single value for all edge weights) than to schedule task graphs with randomly varying weights. We included unit-weighted graphs in this evaluation in order to investigate whether they would also be easier to schedule on heterogeneous systems as they are on homogeneous systems, and whether any differences in trends might be observed.

Figure 9 shows performance profiles for all target systems on the set of task graphs with unit weights. It is clear that all of these graphs are very easy to solve in combination with the two processor systems, as 100% of problem instances are solved within 10 seconds. Among the four processor systems, we see that the strongly heterogeneous P4C2S system is able to solve 76% of problem instances. It is difficult to discern trends among the other systems, with each solving 67%. With the six processor systems, we see that the strongly heterogeneous P6C2 has 89% of instances solved quickly, while only 67% are solved within two minutes with the homogeneous system. With the unit-weighted graphs, as with the random-weighted graphs, we see that strong heterogeneity has a substantial effect in decreasing the difficulty of optimal scheduling. It is also clear

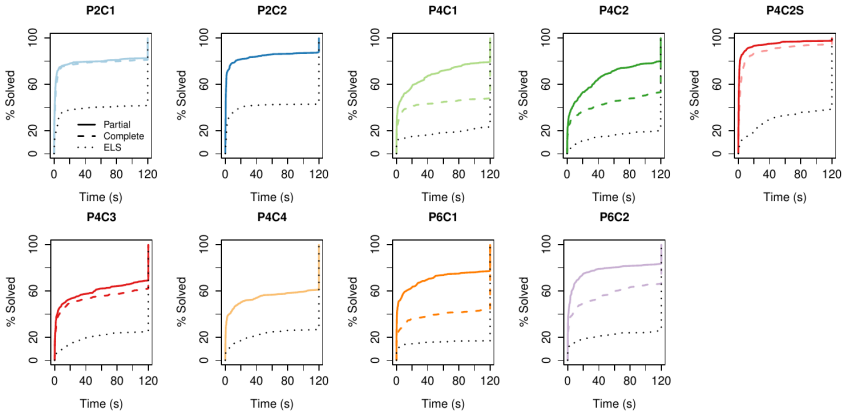

Figure 6: Performance profiles with and without considering partial heterogeneity (target systems: Table 4.)

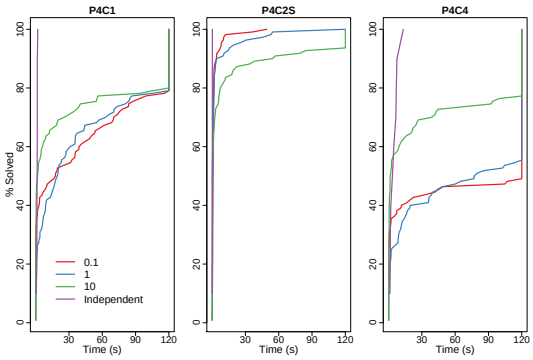

Figure 7: Performance profiles by CCR for selected target systems.

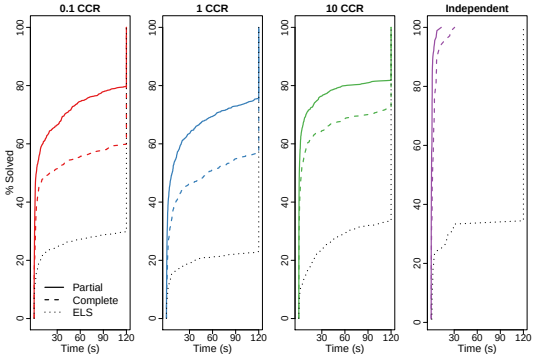

Figure 8: Performance profiles with and without considering partial heterogeneity, grouped by CCR.

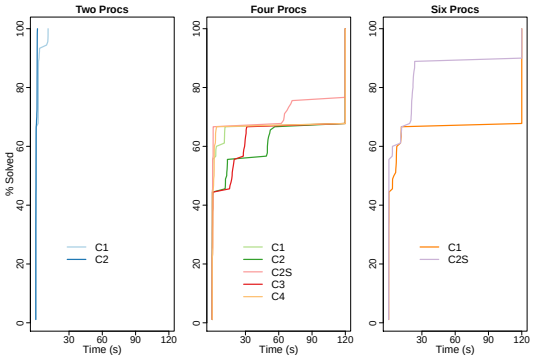

Figure 9: Performance profiles with unit weight graphs for selected systems.

that unit-weighted graphs are easier to schedule optimally than random-weighted graphs, both on homogeneous and heterogeneous processors.

4.3. Larger Graphs

n order to verify the so far observed results with larger graphs and graphs for real-world scientific applications we performed some additional trials. An small set of graphs with a greater number of tasks was selected for additional evaluation. These graphs were generated with the aid of the Python package WfCommons, the WfGen module of which aims to generate realistic synthetic workflow traces. WfGen was used to generate one workflow each with structure, job runtime, and input/output file sizes corresponding to these real-world scientific applications: Cycles, BLAST, SoyKB, Epigenomics, and Seismology. For each, one workflow was generated with approximately 25 jobs, and other with approximately 30 jobs, with variation depending on the structure dictated by the applications. The workflows generated are represented by a list of jobs (with an associated run time), and a list of files associated with each job (with a file size, and marked as input or output). These workflow files were transformed to task graphs appropriate for our task scheduling model. Each job defines a task, with job run time giving the weight of the task. The files define edges, with file size as weight. The edge weights of the task graph were then scaled uniformly to produce a medium CCR of 1.0.

The 10 task graphs produced by this process were used for trials with the A* algorithm, again comparing AO with partial (AO-P) and complete (AO-C) heterogeneity - as well as ELS for the smaller group. The set of target systems with 4 processors were used. For each task graph, we attempted to find an optimal schedule for each target system. A longer time limit of two hours was allowed for each search to complete. The same Linux machine was used to run these new trials, with a maximum heap size of 96 GB.

Table 5 shows the full results of these trials. For each combination of graph, algorithm, and target system, the table reports either the time in seconds taken for the search to complete successfully, or the type of failure: T signifies a timeout, which M signifies that the memory limit was reached. The majority of

Graph (# tasks)	Algorithm	Target System				
		P4C1	P4C2	P4C2S	P4C3	P4C4
Cycles (23)	AO-P	0.11	164.1	2.31	T	218.8
	AO-C	0.26	681.7	2.45	M	N/A
	ELS	T	T	T	M	T
Cycles (46	AO-P	M	M	M	T	T
	AO-C	T	M	T	T	N/A
BLAST (25)	AO-P	M	T	M	T	T
	AO-C	T	T	M	T	N/A
	ELS	M	T	T	M	M
BLAST (30)	AO-P	M	T	M	M	M
	AO-C	T	M	T	T	N/A
SoyKB (24)	AO-P	0.10	0.01	0.07	0.11	0.11
	AO-C	0.15	0.12	0.08	0.12	N/A
	ELS	0.87	0.53	6.55	0.63	0.41
SoyKB (30)	AO-P	M	M	0.11	M	M
	AO-C	T	T	0.16	M	N/A
Epigenomics (23)	AO-P	299.1	T	M	0.1	5.96
	AO-C	T	T	M	0.1	N/A
	ELS	T	T	T	2,006.9	M
Epigenomics (29)	AO-P	14.75	M	1099.4	M	T
	AO-C	62.86	T	T	M	N/A
Seismology (25)	AO-P	T	T	636.3	T	M
	AO-C	T	T	T	M	N/A
	ELS	M	T	M	M	M
Seismology (30)	AO-P	T	T	T	T	T
	AO-C	T	T	T	T	N/A

Table 5: Results of trials with larger graphs, given as seconds taken for completion, T (meaning timed out) or M (meaning out of memory).

trials with these larger graphs were unsuccessful. However, the major trends that were present in the earlier evaluation can still be observed here. ELS has the least successes, and where it is successful almost always has the largest solving time. AO with partial heterogeneity has the most successes, and where multiple algorithms succeeded at the same configuration, it almost uniformly has the lowest solving time. Although the sample size is smaller, this data suggests that AO with partial heterogeneity continues to have an advantage in optimal solving for these larger, more realistic graphs.

4.4. Comparison with ILP

As discussed in the Related Work, ILP formulations for the task scheduling problem have been proposed in the literature. Some of them are applicable to heterogeneous systems and in Table 6, we have compiled a comparison of such ILP formulations from the literature, and their reported success in optimal solving of task scheduling problems with heterogeneous processors. Many formulations are not directly comparable to the problem addressed in this work, having significant differences in objective function, environment, etc. While bearing in mind these complications in comparison, it is clear that the size of problem instances solved by these formulations is quite similar to that used in this work - the largest graphs solved have 100 tasks, and most are much smaller. This shows that despite the general advantages of (highly optimised) ILP solvers in speed, the branch-and-bound approach is still very competitive.

Source	Model Differences	Max Size Solved	Time To Solve
Yang 2008 [37]	Unrelated procs, deadlines, data parallelism	14-16 tasks	Up to 1221 secs
Tosun 2012 [25]	Objective: energy/reliability, no comm costs	2-16 tasks, 2-8 procs	Up to 400 secs
Singh 2012 [24]	Unrelated processors, duplication	Up to 20 tasks	1000 sec limit
Khajekarimi 2013 [38]	Objective: energy cost	53 tasks	Not reported
Kinsy 2014 [39]	N/A	28 tasks	Not reported
Mallach 2018 [16]	Homogeneous procs	100 tasks	10 minutes
Roy 2020 [17]	Unrelated procs, heterogeneous comms with contention	16 task nodes, 30 message nodes	6h 40 mins

Table 6: Comparison of reported successful optimal solving between ILP formulations with heterogeneous processors.

5. Conclusions

Previous applications of branch-and-bound state-space search to optimal task scheduling have used a scheduling model which allowed only for homogeneous processors. We have presented an adaption to the AO state-space model for optimal task scheduling which allows it to find optimal solutions for target systems with related heterogeneous processors. Of particular note, the model recognises partial heterogeneity in parallel systems and uses this to reduce the size of the search space in comparison to a plain fully heterogeneous model. The additional complexity of this task scheduling model allows a wider variety of real world target systems to be more closely approximated.

We have demonstrated that this adapted AO model can be used to solve a large number of problem instances with heterogeneous target systems, although often significantly less than can be solved for a homogeneous system. Importantly, we have shown that the AO model gains significant benefit from its specific adaptation to partial heterogeneity. We also demonstrated how strongly heterogeneous systems can make optimal schedules easier to find, sometimes so much so that they are easier than homogeneous systems.

While this work has focused on related heterogeneous processors, the state-space model presented would also allow optimal scheduling with unrelated heterogeneous processors. Incorporating this would however require developing a suitable input format, and a suitable and realistic experimental regime for evaluation.

Similarly, this state-space model could be extended to operate on a task scheduling model which includes heterogeneity in the communication subsystem. Of particular interest would be a model in which communication links are also treated as resources onto which communications must be scheduled[40]. The AO state-space model could potentially be adapted with the addition of a third phase in which the scheduling of communications is decided.

References

[1] B. Veltman, B. J. Lageweg, and J. K. Lenstra, "Multiprocessor Scheduling with Communication Delays," *Parallel Computing*, vol. 16, no. 2-3, pp. 173–182, 1990.

[2] V. Sarkar, *Partitioning and scheduling parallel programs for multiprocessors*. MIT press, 1989.

[3] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," *Parallel Computing*, vol. 31, no. 7, pp. 653–670, 2005.

[4] T. Yang and A. Gerasoulis, "List scheduling with and without communication delays," *Parallel Computing*, vol. 19, no. 12, pp. 1321–1344, 1993.

[5] A. Radulescu and A. van Gemund, "Low-cost task scheduling for distributed-memory machines," *Parallel and Distributed Systems, IEEE Transactions on*, vol. vol 13, pp. 648–658, jun 2002.

[6] M. Drozdowski, *Scheduling for Parallel Processing*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[7] A. Z. Semar Shahul and O. Sinnen, "Scheduling task graphs optimally with A*," *Journal of Supercomputing*, vol. 51, pp. 310–332, Mar. 2010.

[8] M. Orr and O. Sinnen, "Optimal task scheduling benefits from a duplicate-free state-space," *Journal of Parallel and Distributed Computing*, vol. 146, pp. 158–174, 2020.

[9] A. Krämer, "Branch and bound methods for scheduling problems with multiprocessor tasks on dedicated processors," *Operations-Research-Spektrum*, vol. 19, no. 3, pp. 219–227, 1997.

[10] S. Fujita, "A branch-and-bound algorithm for solving the multiprocessor scheduling problem with improved lower bounding techniques," *IEEE Transactions on computers*, vol. 60, no. 7, pp. 1006–1016, 2011.

[11] M. Orr and O. Sinnen, "A duplicate-free state-space model for optimal task scheduling," in *Proc. of 21st Int. European Conference on Parallel and Distributed Computing (Euro-Par 2015)*, vol. 9233 of *Lecture Notes in Computer Science*, (Vienna, Austria), Springer, 2015.

[12] Y.-K. Kwok and I. Ahmad, "On multiprocessor task scheduling using efficient state space search approaches," *Journal of Parallel and Distributed Computing*, vol. 65, no. 12, pp. 1515–1532, 2005.

[13] M. Orr and O. Sinnen, "Parallel and Memory-limited Algorithms for Optimal Task Scheduling Using a Duplicate-Free State-Space," *arXiv e-prints*, p. arXiv:1905.05568, May 2019.

[14] J. E. Mitchell, "Branch-and-cut algorithms for combinatorial optimization problems," *Handbook of applied optimization*, vol. 1, pp. 65–77, 2002.

[15] A. A. El Cadi, R. B. Atitallah, S. d. Hanafi, N. Mladenović, and A. Artiba, "New MIP model for multiprocessor scheduling problem with communication delays," *Optimization Letters*, pp. 1–17, 2014.

[16] S. Mallach, "Improved mixed-integer programming models for the multiprocessor scheduling problem with communication delays," *J Comb Optim*, vol. 36, pp. 871–895, 2018.

[17] S. K. Roy, R. Devaraj, A. Sarkar, K. Maji, and S. Sinha, "Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems," *Journal of Systems Architecture*, vol. 105, p. 101706, 2020.

[18] Q. Tang, L. H. Zhu, L. Zhou, J. Xiong, and J. B. Wei, "Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems," *Journal of Parallel and Distributed Computing*, vol. 138, pp. 115–127, 2020.

[19] S. Mallach, "Improved mixed-integer programming models for multiprocessor scheduling with communication delays," 2016.

[20] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency (out of print)*, vol. 6, pp. 42–51, 07 1998.

[21] R. Dietze and G. RÃŒnger, "The search-based scheduling algorithm HP* for parallel tasks on heterogeneous platforms," *Concurrency and Computation: Practice and Experience*, p. e5898, 2020.

[22] A. Davare, J. Chong, Q. Zhu, D. M. Densmore, and A. L. Sangiovanni-Vincentelli, "Classification, customization, and characterization: Using MILP for task allocation and scheduling," *Systems Research*, 2006.

[23] N. Maculan, S. C. Porto, C. C. Ribeiro, and C. C. de Souza, "A new formulation for scheduling unrelated processor under precedence constraints," *RAIRO-Operations Research*, vol. 33, no. 1, pp. 87–92, 1999.

[24] J. Singh, B. Mangipudi, S. Betha, and N. Auluck, "Restricted duplication based MILP formulation for scheduling task graphs on unrelated parallel machines," in *2012 Fifth International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 202–209, IEEE, 2012.

[25] S. Tosun, "Energy-and reliability-aware task scheduling onto heterogeneous MPSoC architectures," *The Journal of Supercomputing*, vol. 62, no. 1, pp. 265–289, 2012.

[26] A. Bender, "MILP based task mapping for heterogeneous multiprocessor systems," in *Proceedings EURO-DAC'96. European Design Automation Conference with EURO-VHDL'96 and Exhibition*, pp. 190–197, IEEE, 1996.

[27] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM J. Comput.*, vol. 18, no. 2, pp. 244–257, 1989.

[28] O. Sinnen, *Task Scheduling for Parallel Systems (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2007.

[29] A. Bundy and L. Wallen, "Branch-and-bound algorithms," in *Catalogue of Artificial Intelligence Tools* (A. Bundy and L. Wallen, eds.), Symbolic Computation, pp. 12–12, Springer Berlin Heidelberg, 1984.

[30] N. J. N. P. E. Hart and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Science, and Cybernetics*, vol. SSC-4, no. 2, pp. 100–107, 1968.

[31] D. Berend and T. Tassa, "Improved bounds on bell numbers and on moments of sums of random variables," *Probability and Mathematical Statistics*, vol. 30, no. 2, pp. 185–205, 2010.

[32] B. C. Rennie and A. J. Dobson, "On stirling numbers of the second kind," *Journal of Combinatorial Theory*, vol. 7, no. 2, pp. 116–121, 1969.

[33] Z. Juric and H. Siljak, "A new formula for the number of combinations and permutations of multisets," *Applied Mathematical Sciences (Ruse)*, vol. 5, 01 2011.

[34] F. Ruskey and C. D. Savage, "A gray code for combinations of a multiset," *European Journal of Combinatorics*, vol. 17, no. 5, pp. 493–500, 1996.

[35] A. Brualdi Richard, "Introductory combinatorics," 2010.

[36] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.

[37] H. Yang and S. Ha, "ILP based data parallel multi-task mapping/scheduling technique for MPSoC," in *2008 International SoC Design Conference*, vol. 1, pp. I–134, IEEE, 2008.

[38] E. Khajekarimi and M. R. Hashemi, "Energy-aware ILP formulation for application mapping on NoC based MPSoCs," in *2013 21st Iranian conference on electrical engineering (ICEE)*, pp. 1–5, IEEE, 2013.

[39] M. A. Kinsy and S. Devadas, "Algorithms for scheduling task-based applications onto heterogeneous many-core architectures," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, IEEE, 2014.

[40] O. Sinnen and L. A. Sousa, "Communication Contention in Task Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, pp. 503–515, 2005.