Optimal Multistage Algorithm for Adjoint Computation

Guillaume Aupy, Julien Herrmann, Paul Hovland, Yves Robert

OPTIMAL MULTISTAGE ALGORITHM FOR ADJOINT COMPUTATION

GUILLAUME AUPY† JULIEN HERRMANN† PAUL HOVLAND‡ AND YVES ROBERT†§

Abstract. We reexamine the work of Stumm and Walther on multistage algorithms for adjoint computation. We provide an optimal algorithm for this problem when there are two levels of checkpoints, in memory and on disk. Previously, optimal algorithms for adjoint computations were known only for a single level of checkpoints with no writing and reading costs; a well-known example is the binomial checkpointing algorithm of Griewank and Walther. Stumm and Walther extended that binomial checkpointing algorithm to the case of two levels of checkpoints, but they did not provide any optimality results. We bridge the gap by designing the first optimal algorithm in this context. We experimentally compare our optimal algorithm with that of Stumm and Walther to assess the difference in performance.

Key words. Optimal algorithms, Checkpointing, Adjoint Computation, Program Reversal, Automatic Differentiation

AMS subject classifications. 65Y20, 68W40, 49N90, 90C30

1. Introduction. The need to efficiently compute the derivatives of a function arises frequently in many areas of scientific computing, including mathematical optimization, uncertainty quantification, and nonlinear systems of equations. When the first derivatives of a scalar-valued function are desired, so-called adjoint computations can compute the gradient at a cost equal to a small constant times the cost of the function itself, without regard to the number of independent variables. This adjoint computation can arise from discretizing the continuous adjoint of a partial differential equation [9, 3] or from applying the so-called reverse or adjoint mode of algorithmic (also called automatic) differentiation to a program for computing the function [6]. In either case, the derivative computation applies the chain rule of differential calculus starting with the dependent variables and propagating back to the independent variables. Thus, it reverses the flow of the original function evaluation. In general, intermediate function values are not available at the time they are needed for partial derivative computation and must be stored or recomputed [1].

A popular storage or recomputation strategy for functions that have some sort of natural "time step" is to save (checkpoint) the state at each time step during the function computation (forward sweep) and use this saved state in the derivative computation (reverse sweep). If the storage is inadequate for all states, one can checkpoint only some states and recompute the unsaved states as needed. Griewank and Walther prove that given a fixed number of checkpoints, the schedule that minimizes the amount of recomputation is a binomial checkpointing strategy [4, 5]. The problem formulation they used implicitly assumes that reading and writing checkpoints are essentially free, but the number of available checkpoints is limited (see Problem 1 below). In [13], Stumm and Walter consider the case where checkpoints can be written to either memory or disk. The number of checkpoints to disk is effectively unlimited but the time to read or write a checkpoint can no longer be ignored (see Problem 2). We consider the same situation. In contrast to Stumm and Walther, however, we do not restrict ourselves to a single binomial schedule but instead prove that there exists

†LIP, ENS Lyon, INRIA, France
‡Argonne National Laboratory, Illinois, USA
§University of Tennessee Knoxville, USA

a time-optimal schedule possessing certain key properties (including no checkpoints written to disk after the first checkpoint to memory has been written), and we provide a polynomial time algorithm for determining an optimal schedule.

The rest of this paper is organized as follows. Section 2 introduces terminology and the general problem framework. Section 3 establishes several properties that must hold true for some optimal schedule and provides an algorithm for identifying this schedule. Section 4 compares our checkpointing schedule with that of Stumm and Walther. We conclude with some thoughts on future research directions.

2. Framework.

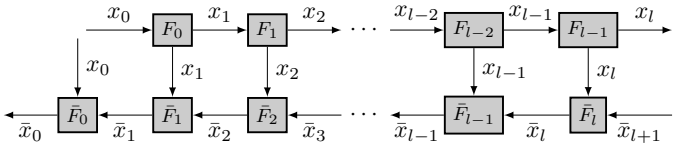


Figure 1: The AC dependence graph.

2.1. The AC problem.

DEFINITION 2.1 (Adjoint Computation (AC) [5, 13]). *An adjoint computation (AC) with l time steps can be described by the following set of equations:*

$$F_i(x_i) = x_{i+1} \text{ for } 1 \leq i < l \tag{1}$$

$$\bar{F}_i(x_i, \bar{x}_{i+1}) = \bar{x}_i \text{ for } 1 \leq i \leq l \tag{2}$$

The dependencies between these operations[1] *are represented by the graph $\mathcal{G} = (V, E)$ depicted in Figure 1.*

The F computations are called *forward* steps. The $\bar{F}$ computations are called *backward* steps. If $\bar{x}_l$ is initialized appropriately, then at the conclusion of the adjoint computation, $\bar{x}_0$ will contain the gradient with respect to the initial state (x_0).

DEFINITION 2.2 (Platform). *We consider a platform with three storage locations:*

- Buffers: *there are two buffers, the top buffer and the bottom buffer. The top buffer is used to store a value x_i for some i, while the bottom buffer is used to store a value $\bar{x}_j$ for some j. For a computation (F or $\bar{F}$) to be executed, its input values have to be stored in the buffers. Let $\mathcal{B}^\top$ and $\mathcal{B}^\bot$ denote the content of the top and bottom buffers. In order to start the execution of the graph, x_0 must be stored in the top buffer and $\bar{x}_{l+1}$ in the bottom buffer. Hence, without loss of generality, we assume that at the beginning of the execution, $\mathcal{B}^\top = \{x_0\}$ and $\mathcal{B}^\bot = \{\bar{x}_{l+1}\}$.*
- Memory: *there are c_m slots of memory where the content of a buffer can be stored. The time to write from buffer to memory is w_m. The time to read from memory to buffer is r_m. Let $\mathcal{M}$ be the set of x_i and $\bar{x}_i$ values stored in the memory. The memory is empty at the beginning of the execution ($\mathcal{M} = \emptyset$).*
- Disks: *there are c_d slots of disks where the content of a buffer can be stored.. The time to write from buffer to disk is w_d. The time to read from disk to buffer is r_d. Let $\mathcal{D}$ be the set of x_i and $\bar{x}_i$ values stored in the disk. The disk is empty at the beginning of the execution ($\mathcal{D} = \emptyset$).*

[1] In the original approach by Griewank [4], an extra F_l operation was included. It is not difficult to take this extra operation into account.

Memory and disk are generic terms for a two-level storage system, modeling any platform with a dual memory system, including (i) a cheap-to-access first-level memory, of limited size; and (ii) and a costly-to-access second-level memory, whose size is very large in comparison with the first-level memory. The pair *(memory, disk)* can be replaced by *(cache, memory)* or *(disk, tape)* or any relevant hardware combination.

Intuitively, the core of the AC problem is the following. After the execution of a forward step, its output is kept in the top buffer only. If it is not saved in memory or disk before the next forward step, it is lost and will have to be recomputed when needed for the corresponding backward step. When no disk storage is available, the problem is to minimize the number of recomputations in the presence of limited (but cheap-to-access) memory slots. When disk storage is added, the problem becomes even more challenging: saving data on disk can save some recompilation, and a trade-off must be found between the cost of disk accesses and that of recomputations.

The problem with only memory and no disk (Problem 1: $\textsc{Prob}(l, c_m)$ below) has been solved by Griewank and Walther [5], using a binomial checkpointing algorithm called Revolve. In accordance to the scheduling literature, we use the term *makespan* for the total execution time.

Problem 1 ($\textsc{Prob}(l, c_m)$). *We want to minimize the makespan of the AC problem with the following parameters:*

		Initial state:
AC graph:	*size* l	
Steps:	u_f, u_b	
Memory:	c_m, $w_m = r_m = 0$	$\mathcal{M}_{ini} = \emptyset$
Disks:	$c_d = 0$	
Buffers:	$\mathcal{B}^\top$, $\mathcal{B}^\bot$	$\mathcal{B}^\top_{ini} = \{x_0\}$, $\mathcal{B}^\bot_{ini} = \{\bar{x}_{l+1}\}$

In this paper we consider the problem with limited memory and infinite disk (Problem $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$ below). The main goal of this paper is to provide the first optimal algorithm for $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$.

Problem 2 ($\textsc{Prob}_\infty(l, c_m, w_d, r_d)$). *We want to minimize the makespan of the AC problem with the following parameters:*

		Initial state:
AC graph:	*size* l	
Steps:	u_f, u_b	
Memory:	c_m, $w_m = r_m = 0$	$\mathcal{M}_{ini} = \emptyset$
Disks:	$c_d = +\infty, w_d, r_d$	$\mathcal{D}_{ini} = \emptyset$
Buffers:	$\mathcal{B}^\top$, $\mathcal{B}^\bot$	$\mathcal{B}^\top_{ini} = \{x_0\}$, $\mathcal{B}^\bot_{ini} = \{\bar{x}_{l+1}\}$

2.2. Algorithm model. We next detail the elementary operations that an algorithm can perform.

- F_i Execute one forward computation F_i (for $i \in \{0, \ldots, l-1\}$). Note that by definition, for F_i to occur, x_i should be in the top buffer before (i.e., $\mathcal{B}^\top = \{x_i\}$) and x_{i+1} will be in the top buffer after (i.e., $\mathcal{B}^\top \leftarrow \{x_{i+1}\}$). This operation takes a time $\mathsf{time}(F_i) = u_f$.
- $\bar{F}_i$ Execute the backward computation $\bar{F}_i$ ($i \in \{0, \ldots, l\}$). Note that by definition, for $\bar{F}_i$ to occur, x_i should be in the top buffer and $\bar{x}_{i+1}$ in the bottom buffer (i.e., $\mathcal{B}^\top = \{x_i\}$ and $\mathcal{B}^\bot = \{\bar{x}_{i+1}\}$) and $\bar{x}_i$ will be in the bottom buffer after (i.e., $\mathcal{B}^\bot \leftarrow \{\bar{x}_i\}$). This operation takes a time $\mathsf{time}(\bar{F}_i) = u_b$.
- W_i^m Write the value x_i of the top buffer into the memory. Note that by definition, for W_i^m to occur, x_i should be in the top buffer (i.e., $\mathcal{B}^\top = \{x_i\}$) and there

should be enough space for x_i in the memory (i.e., $|\mathcal{M}| < c_m$); x_i will be in the memory after (i.e., $\mathcal{M} \leftarrow \mathcal{M} \cup \{x_i\}$). This operation takes time $\mathsf{time}(W_i^m) = w_m$.

D_i^m Discard the value x_i of the memory (i.e., $\mathcal{M} \leftarrow \mathcal{M} \backslash \{x_i\}$). This operation takes a time $\mathsf{time}(D_i^m) = 0$. This operation is introduced only to clarify the proofs, since a D_i^m operation is always immediately followed by a W_i^m operation. In other words, all write operations overwrite the content of some memory slot, and we simply decompose an overwrite operation into a discard operation followed by a write operation.

R_i^m Read the value x_i in the memory, and put it into the top buffer. Note that by definition, for R_i^m to occur, x_i should be in the memory (i.e., $x_i \in \mathcal{M}$) and x_i will be in the top buffer after (i.e., $\mathcal{B}^\top = \{x_i\}$). This operation takes a time $\mathsf{time}(R_i^m) = r_m$.

W_i^d Write the value x_i of the top buffer into the disk. Note that by definition, for W_i^d to occur, x_i should be in the top buffer (i.e., $\mathcal{B}^\top = \{x_i\}$) and x_i will be in the disk after (i.e., $\mathcal{D} \leftarrow \mathcal{D} \cup \{x_i\}$). This operation takes a time $\mathsf{time}(W_i^d) = w_d$.

R_i^d Read the value x_i in the disk and puts it into the top buffer. Note that by definition, for R_i^d to occur, then x_i should be in the disk (i.e., $x_i \in \mathcal{D}$) and x_i will be in the top buffer after (i.e., $\mathcal{B}^\top = \{x_i\}$). This operation takes a time $\mathsf{time}(R_i^d) = r_d$.

D_i^d Discard the value x_i of the disk (i.e., $\mathcal{D} \leftarrow \mathcal{D} \backslash \{x_i\}$). This operation takes a time $\mathsf{time}(D_i^d) = 0$. The same comment as for D_i^m operations holds: all disk writes, just as memory writes, are overwrite operations, which we decompose as indicated above. Both discard operations are introduced for the clarity of the proofs.

For conciseness, we let $F_{i \to i'}$ denote the sequence $F_i \cdot F_{i+1} \cdot \ldots \cdot F_{i'}$.

Because of the shape of the AC dependence graph (see Figure 1), any algorithm solving Problem 2 will have the following unique structure,

$$\mathfrak{S}_l \cdot \bar{F}_l \cdot \mathfrak{S}_{l-1} \cdot \bar{F}_{l-1} \cdot \ldots \cdot \mathfrak{S}_0 \cdot \bar{F}_0, \tag{3}$$

where for all i, $\mathfrak{S}_i$ is a sequence of operations that does not contain $\bar{F}_i$ (hence the $\bar{F}_i$ following $\mathfrak{S}_i$ is the first occurrence of $\bar{F}_i$).

DEFINITION 2.3 (iteration i). *Given an algorithm that solves the AC problem, we let* iteration i *(for $i = l \ldots 0$) be the sequence of operations $\mathfrak{S}_i \cdot \bar{F}_i$. Let l_i be the execution time of iteration i.*

With this definition, the makespan of an algorithm is $\sum_{i=1}^{l} l_i$.

3. Solution of $\mathbf{Prob}_\infty(l, c_m, w_d, r_d)$. In this section we show that we can compute in polynomial time the solution to $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$. To do so, we start by showing some properties on an optimal algorithm.

- We show that iteration l starts by writing some values x_i on disk checkpoints before doing any memory checkpoints (Lemma 3.7).
- Once this is done, we show that we can partition the initial AC graph into connected subgraphs by considering the different subgraphs between consecutive disk checkpoints (Proposition 3.8). Each of these subgraphs can be looked at (and solved) independently.
- We give some details on how to solve the problem on all subgraphs. In particular we show that (i) we do not write any additional values to disks in order to solve them (Lemma 3.2); and (ii) we show that similarly to the first

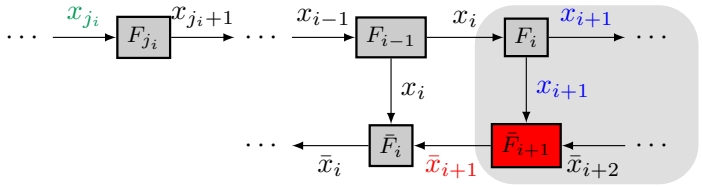


Figure 2: After executing $\bar{F}_{i+1}$, x_{i+1} (blue) is in the top buffer, and $\bar{x}_{i+1}$ (red) is in the bottom buffer. For the remainder of the execution, the algorithm will not need the grey area anymore; hence it will need to fetch x_{j_i} (green) from a checkpoint slot.

iteration, the algorithm writes some values to memory checkpoints and we can partition these subgraphs by considering the different subgraphs between memory checkpoints (Lemma 3.3).
- To solve these subgraphs, we introduce new problems (Problems 3 and 4) that inherit the properties of the general problem.
- We show how to compute the size of the different connected subgraphs through a dynamic programming algorithm (§ 3.2).

3.1. Properties of an optimal algorithm. We show here some dominance properties. That is, there exist optimal algorithms that obey these properties, even though not all optimal algorithms do.

LEMMA 3.1. *There exists an optimal solution that has the following structure,* $\mathfrak{S}_l \cdot \bar{F}_l \cdot \mathfrak{S}_{l-1} \cdot \bar{F}_{l-1} \cdot \ldots \cdot \mathfrak{S}_0 \cdot \bar{F}_0$*, and that satisfies*
(P0):

(i) There are no D^d-type operations (we do not discard from disks).
(ii) Each D^m-type operation is immediately followed by a W^m-type operation (we discard a value from memory only to overwrite it).
(iii) Each R-type operation is not immediately followed by another R-type operation.
(iv) Each W^m-type operation (resp. W^d-type operation) is not immediately followed by another W^m-type operation (resp. W^d-type operation).
(v) Each W^m-type operation is not immediately followed by another D^m-type operation.
(vi) There are no $\bar{F}$-type operations in any $\mathfrak{S}_i$ sequence (backward steps are not recomputed)*;*
(vii) During $\mathfrak{S}_i$ $(i < l)$, there are no F_i to F_{l-1} operations (nor actions involving x_{i+1} to x_l);
(viii) In particular, for all $i < l$, the first operation of sequence $\mathfrak{S}_i$ is a R-type operation; in other words, there exist j_i and $s \in \{m, d\}$ such that $\mathfrak{S}_i = R^s_{j_i}\tilde{\mathfrak{S}}_i$;
(ix) $\forall i$, there is at least one R^m_i operation between a W^m_i and a D^m_i operations.
(x) If $l > 0$, the first operation of the algorithm is a W-type operation.

Proof. Some of the intuitions of this lemma can be grasped from Figure 2. Note that removing an operation from the optimal solution cannot increase the makespan.

(i) We have an infinite number of disk slots available: there is no need to discard any value from it.
(ii) Discard a value from the memory is useless if the memory is not full and if we do not need to write a new value in it.
(iii) If we had two consecutive reads in the sequence, then the only action of the first read would be to put some value x_i in the top buffer. However, the second

read would immediately overwrite this value, making the first read unnecessary. Thus, the first read can be removed.

(iv) It is useless to write the same value in the same storage twice in a row.

(v) Similar to the previous point, from (ii) a D^m operation is immediately followed by a W^m operation; hence this would be writing twice in the same storage in a row.

(vi) The reason there are no $\bar{F}$-type operations in all $\mathfrak{S}_i$ is that we have a dedicated buffer for the $\bar{x}_i$ values. The only operations that use the $\bar{x}_i$ values are $\bar{F}$-type operations. Also, to execute $\bar{F}_i$, we need only the value of $\bar{x}_{i+1}$ that is already stored in the bottom buffer at the beginning of $\mathfrak{S}_i$. Hence, removing the additional $\bar{F}$-type operations from $\mathfrak{S}_i$ can only improve the execution time.

(vii) Operations involving F_i to F_{l-1} (or their output values) during $\mathfrak{S}_i$ would be useless; see Figure 2.

(viii) After the execution of $\bar{F}_i$, the content of the top buffer is x_i. The value x_i is useless for $\bar{F}_{i-1}$ (see the previous point). Hence, at the begining of $\mathfrak{S}_{i-1}$, we need to read the content of a storage slot before executing any F-type, $\bar{F}$-type, or W-type operation. Furthermore, because of property (ii), doing a D-type operation will not permit an R-type operation before the next W-type operation. Hence, the first operation of sequence $\mathfrak{S}_i$ is necessarily an R-type operation.

(ix) Assume that there exists i such that there are no R_i^m operations between a W_i^m and a D_i^m operations. It is useless to write the value x_i in the memory and discard it without reading it in between. Thus the W_i^m and D_i^m operations can be removed at no additional time delay.

(x) The first operation of the solution cannot be an R-type or a D-type operation since at the beginning of the execution the memory and the disk are empty. If $l > 0$, the forward step F_0 has to be executed before the backward step $\bar{F}_l$. Thus the first operation cannot be an $\bar{F}$-type operation. Now assume that the first operation is an F-type operation. It then has to be F_0. After the execution of F_0, the value x_1 is in the top buffer, and the value x_0 is not stored anywhere. There is no way to recompute the value x_0, thus to execute $\bar{F}_0$, which would then prevent computing $\bar{x}_0$ (absurd). Thus, at the beginning of the execution, we have to store the value x_0 either in the memory or in the disk, and the first operation of the algorithm is a W-type operation.

□

LEMMA 3.2. *There exists an optimal solution to Problem 2 that satisfies (P0) and*
(P1):

(i) All disk checkpoints are executed during the first iteration $\mathfrak{S}_l$.

(ii) For all i, $1 \leq i \leq l-1$, W_i^d operations are executed before W_i^m operations.

Proof. Suppose $\mathcal{S}$ is an optimal solution that satisfies (P0). We will show that we can transform it into a solution that satisfies (P0) and (P1):

(i) Iteration l passes through all forward computations. If a value x_i is saved on disk later on during the algorithm, we could as well save it after the first execution of F_{i-1} (in $\mathfrak{S}_l$) with no additional time delay, since we have an infinite number of slots on disk.

(ii) Let assume that there exists i, $0 \leq i \leq l-1$, such that W_i^m is executed before W_i^d in $\mathcal{S}$. By definition, when W_i^m is executed, the value x_i is stored in the top buffer. Thus we can execute W_i^d right before W_i^m, instead of later, at no additional time delay, since the amount of available disk slots is infinite. This

new solution still satisfies (P0).

□

The following lemma and its proof are inspired by Lemma 3.1 by Walther [14].

LEMMA 3.3 (Memory Checkpoint Persistence). *There exists an optimal solution to Problem 2 that satisfies (P0), (P1), and*
(P2):

(i) Let $i < l$; if W_i^m is executed, then there are no D_i^m operations until after the execution of $\bar{F}_i$ (that is, until the beginning of iteration $i-1$).

(ii) Moreover, until that time, no operation involving F_0 to F_{i-1} or values x_0 to x_{i-1} is taken.

The intuition behind this result is that if we were to discard the value x_i before executing $\bar{F}_{i+1}$, then a better solution would have stored x_{i+1} in the first place. Furthermore, because $r_m = 0$, we can show that until $\bar{F}_i$, all actions involving computations F_0 to F_{i-1} or values x_0 to x_{i-1} do not impact the actions that lead to an $\bar{F}_j$ operation, $j \geq i$, and thus can be moved to a later time at no additional time delay (and potentially reducing the makespan of the algorithm).

Proof. Let $\mathcal{S}$ be an optimal solution that satisfies (P0) and (P1) but not (P2). We can transform it into a solution that satisfies (P0), (P1), and (P2). We iteratively transform $\mathcal{S}$ to increase the number of i values respecting (P2), without increasing the makespan of the schedule. This transformation can be applied as many times as necessary to reach a schedule that satisfies (P2).

Assume, first, that $\mathcal{S}$ does not satisfy (P2(i)). Let x_i be the first value such that at some point during $\mathcal{S}$, x_i is stored in the memory and discarded before executing $\bar{F}_i$. Thus we can write

$$\mathcal{S} = \mathcal{S}_0 \cdot W_i^m \cdot \mathcal{S}_1 \cdot D_i^m \cdot \mathcal{S}_2 \cdot \bar{F}_i \cdot \mathcal{S}_3,$$

where $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ are sequences of operations that do not include $\bar{F}_i$. Since $\mathcal{S}$ satisfies (P0(ix)), there is at least one R_i^m operation in $\mathcal{S}_1$. Let us prove that all these R_i^m operations are immediately followed by an F_i operation.

- The R_i^m operations cannot be immediately followed by an $\bar{F}$-type operation, because the only $\bar{F}$-type operation allowed after an R_i^m operation is $\bar{F}_i$ (since the value x_i would be in the top buffer) and there are no $\bar{F}_i$ in $\mathcal{S}_1$.
- According to (P0(ix)), the R_i^m operations cannot be immediately followed by another R-type operation.
- The R_i^m operations cannot be immediately followed by a D^m-type operation because, according to (P0(ii)), the next operation would be a W^m-type operation. However, since the value x_i would be in the top buffer, the only W^m-type operation allowed would be W_i^m, which is useless since the value x_i is already in the memory.
- The R_i^m operations cannot be immediately followed by a D^d-type operation because, according to (P0(i)), there are no D^d-type operations in $\mathcal{S}$.
- The R_i^m operations cannot be immediately followed by a W^m-type operation, because the only W^m-type operation allowed after an R_i^m operation is W_i^m (since the value x_i would be in the top buffer), which is useless since the value x_i is already in the memory.
- The R_i^m operations cannot be immediately followed by a W^d-type operation, because the only W^d-type operation allowed after an R_i^m operation is W_i^d (since the value x_i would be in the top buffer), which is impossible according to (P1(ii)) since a W_i^m has already been executed before $\mathcal{S}_1$.

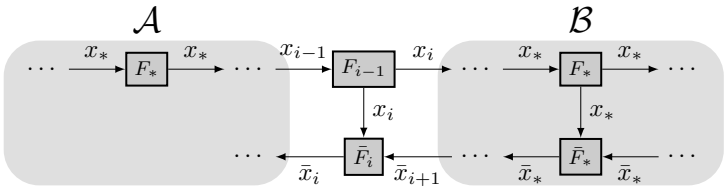


Figure 3: Consider a subsequence of $\mathcal{S}_1$ comprised between two consecutive R-type operations: R_j^s and $R_{j'}^{s'}$. If $j \geq i$, then by definition the subsequence will activate only parts of area $\mathcal{B}$ (and not overwrite any checkpoint from area $\mathcal{A}$). If $j < i$, then we have shown that the subsequence will activate only parts of the area $\mathcal{A}$ (no forward sweep through F_{i-1}).

So, all these R_i^m operations are immediately followed by an F-type operation; since the value x_i is in the top buffer, this operation is F_i. Thus, any R_i^m in $\mathcal{S}_1$ is followed by F_i.

Let us now focus on the first operation in $\mathcal{S}_1$. It cannot be a $\bar{F}$-type (the only possible $\bar{F}$-type is $\bar{F}_i$), W^m-type ((P0(iv))), W^d-type ((P1(ii))), or D-type ((P0(v))). Hence, the first operation in $\mathcal{S}_1$ is either a R-type operation or a F-type operation (in which case, it is F_i since $\mathcal{B}^\top = \{x_i\}$ at the beginning of $\mathcal{S}_1$).

- Assume that $\mathcal{S}_1 = F_i \cdot \mathcal{S}_1'$. Let $\mathcal{S}_1''$ be the sequence $\mathcal{S}_1'$ where every occurrence of $R_i^m \cdot F_i$ has been replaced by R_{i+1}^m. We know that the schedule $\mathcal{S}' = \mathcal{S}_0 \cdot F_i \cdot R_{i+1}^m \cdot \mathcal{S}_1'' \cdot D_{i+1}^m \cdot \mathcal{S}_2 \cdot \bar{F}_i \cdot \mathcal{S}_3$ is correct and has a makespan at least as good as $\mathcal{S}$ (since there is at least one occurrence of $R_i^m \cdot F_i$ in $\mathcal{S}_1'$).
- Assume that there exist j and s (either equal to m or d) such that $\mathcal{S}_1 = R_j^s \cdot \mathcal{S}_1'$. Let $\mathcal{S}_1''$ be the sequence $\mathcal{S}_1'$ where every occurrence of $R_i^m \cdot F_i$ has been replaced by R_{i+1}^m. We know that the schedule $\mathcal{S}' = \mathcal{S}_0 \cdot F_i \cdot R_{i+1}^m \cdot R_j^s \cdot \mathcal{S}_1'' \cdot D_{i+1}^m \cdot \mathcal{S}_2 \cdot \bar{F}_i \cdot \mathcal{S}_3$ is correct and has a makespan at least as good as $\mathcal{S}$ (since there is at least one occurrence of $R_i^m \cdot F_i$ in $\mathcal{S}_1'$).

Hence we were able to transform $\mathcal{S}$ into $\mathcal{S}$' without increasing the makespan, so that the number of values i, $0 \leq i < l$, that does not respect (P2(i)) decreases. We repeat this transformation until the new schedule satisfies (P2(i)).

Let us now consider (P2(ii)). Let $\mathcal{S}$ be an optimal solution that satisfies (P0), (P1), and (P2(i)). Let i, $0 \leq i < l$, such that there exists W_i^m in $\mathcal{S}$. We can write

$$\mathcal{S} = \mathcal{S}_0 \cdot W_i^m \cdot \mathcal{S}_1 \cdot \bar{F}_i \cdot \mathcal{S}_2 \cdot D_i^m \cdot \mathcal{S}_3,$$

where $\mathcal{S}_1$ is a sequence of operations that do not include D_i^m. There are no F_{i-1} in $\mathcal{S}_1$ because their only impact on the memory is to put the value x_i in the top buffer which could be done with R_i^m for no time delay.

Consider now two consecutive R-type operations of $\mathcal{S}_1$. Because there are no F_{i-1} operations in $\mathcal{S}_1$, we know that between these two R-type operations, and with the definitions of $\mathcal{A}$ and $\mathcal{B}$ in Figure 3, either only elements of $\mathcal{A}$ are activated (and no element of $\mathcal{B}$) or only elements of $\mathcal{B}$ are activated (and no element of $\mathcal{A}$).

Consider now the last R-type operation R_j^s of $\mathcal{S}_1$ such that $j < i$ (s being equal to m or d). All W^m-type operations written after this operation and before the next R-type operation of $\mathcal{S}_1$ involve some values in $\mathcal{A}$. Hence by (P2(i)), they are not discarded until after $\bar{F}_i$. Furthermore, because R_j^s is the last such operation, we know

that they are not used in $\mathcal{S}_1$ either. Hence we can move this sequence of operations (the sequence between R_j^s and the next R-type operation) right after $\bar{F}_i$ at no additional time delay. This operation can be repeated until there are no more such operations in $\mathcal{S}_1$. We then proceed with these operations recursively in the appearance order of the W_i^m. This shows that we can construct an optimal schedule that satisfies (P2(ii)). ∎

LEMMA 3.4. *There exists an optimal algorithm for Problem 2 that satisfies (P0), (P1), (P2), and*

(P3): *There is only one R-type operation (the first one) in every iteration $\mathfrak{S}_i$, where $i < l$.*

Proof. Let $\mathcal{S}$ be an optimal schedule that satisfies (P0), (P1), and (P2). We show that we can transform it into an optimal schedule that satisfies (P0–3).

To show this result, for any $i \geq 0$, we inductively show that if we have a solution such that the property is true in iterations l to $i+1$, then we can transform it into a solution such that this property in true in iterations l to i.

Assume that $\mathcal{S}$ does not satisfies (P3). Let $\mathfrak{S}_i$ be the first iteration of $\mathcal{S}$ that includes more than one R-type operation.

If $i = 0$, then all R-type operations are $R_0^{\bar{}}$ by (P2). Hence we can remove any of them until there is only one.

Otherwise assume that $i \geq 1$. Let $R_{j_1}^{s_1}$ and $R_{j_2}^{s_2}$ (where $s_1, s_2 \in \{m, d\}$) be the last two R-type operations in $\mathfrak{S}_i$. According to (P0(iv)), we know that step i does not involve x_{i+1} to x_l, so $j_1 \leq i$ and $j_2 \leq i$. We can write

$$\mathfrak{S}_i = \mathfrak{S}^{(1)} \cdot R_{j_1}^{s_1} \cdot \mathfrak{S}^{(2)} \cdot R_{j_2}^{s_2} \cdot \mathfrak{S}^{(3)},$$

where $\mathfrak{S}^{(2)}$ and $\mathfrak{S}^{(3)}$ are sequences of operations that do not include any R-type operation. According to (P0(vi)), there are no $\bar{F}$-type operations in $\mathfrak{S}^{(2)}$ and $\mathfrak{S}^{(3)}$ either. Since the value x_{j_2} is in the top buffer at the beginning of $\mathfrak{S}^{(3)}$, we know that the first F-type operation of $\mathfrak{S}^{(3)}$ has to be F_{j_2}. We know that the first operation after $\mathfrak{S}_i$ is $\bar{F}_i$. Thus the last F-type operation of $\mathfrak{S}^{(3)}$ is F_{i-1}. Since there are no R-type operations in $\mathfrak{S}^{(3)}$, we know that the sequence $\mathfrak{S}^{(3)}$ includes all F-type operations from F_{j_2} to F_{i-1}.

Similarly, the sequence $\mathfrak{S}^{(2)}$ includes all F-type operations from F_{j_1} to $F_{j_{\max}}$ operations with $j_{\max} \leq i-1$.

- If $j_{\max} \geq j_2$, we note $j_{\min} = \min(j_1, j_2)$ and $s_{\min}$ the corresponding value of s_1 or s_2. Iteration $\mathfrak{S}_i$ includes each F-type operations from $F_{j_{\min}}$ to F_i (possibly twice). Let us build the sequence of operations $\mathfrak{S}^{(4)}$ from the sequence $F_{j_{\min} \to i}$ where each operation F_k if immediately followed by W_k^m if W_k^m is present in either $\mathfrak{S}^{(2)}$ or $\mathfrak{S}^{(3)}$ (see Figure 4 for this transformation). Thus we know that the sequence $\mathfrak{S}_i' = \mathfrak{S}^{(1)} \cdot R_{j_{\min}}^{s_{\min}} \cdot \mathfrak{S}^{(4)}$ will have the exact same impact on the memory as $\mathfrak{S}_i$ without increasing the makespan. Transforming iteration $\mathfrak{S}_i$ into sequence $\mathfrak{S}_i'$ reduces by one the number of readings in iteration i.
- If $j_{\max} < j_2$, sequences of operations $\mathfrak{S}^{(2)}$ and $\mathfrak{S}^{(3)}$ are disjoint. Thus $\mathfrak{S}^{(2)}$ has no impact on iteration i and can be moved to the beginning of the next operation. Thus transforming iteration $\mathfrak{S}_i$ into $\mathfrak{S}_i' = \mathfrak{S}^{(1)} \cdot R_{j_2}^{s_2} \cdot \mathfrak{S}^{(3)}$ and moving $R_{j_1}^{s_1} \cdot \mathfrak{S}^{(2)}$ to the beginning of iteration $\mathfrak{S}_{i+1}$ will not increase the makespan of $\mathcal{S}$ and will reduce by one the number of readings in iteration i.

Hence we have shown that until there is only one R-type operation in iteration i, we can reduce by one the number of R-type operations in iteration i (and leave as they were iterations l to $i+1$). Thus, if the property is true in iteration l to $i+1$, then

$$\begin{aligned}
\tilde{\mathfrak{S}}^{(2)} &= R_0^s F_1 W_1^m F_2 F_3 F_4 F_5 W_5^m F_6 W_6^m \\
\tilde{\mathfrak{S}}^{(3)} &= \qquad\qquad R_3^{s'} F_4 F_5 \qquad F_6 W_6^m F_7 F_8 F_9 W_9^m \\
\hline
\tilde{\mathfrak{S}}^{(4)} &= R_0^s F_1 W_1^m F_2 F_3 F_4 F_5 W_5^m F_6 W_6^m F_7 F_8 F_9 W_9^m
\end{aligned}$$

Figure 4: Example of the merging operation.

we can transform it into a solution such that this property in true in iteration l to i. This concludes the proof. □

COROLLARY 3.5 (Description of iteration $i < l$). *Given an optimal algorithm for Problem 2 that satisfies (P0–3), each iteration $i < l$ can be written as*

$$\mathfrak{S}_i = R_{j_i}^{s_i} \tilde{\mathfrak{S}}_i \tag{4}$$

for some j_i and for some $s_i \in \{m, d\}$, where $\tilde{\mathfrak{S}}_i$ is composed only of F-type, W^m-type, and D^m-type operations (possibly empty). Furthermore, it goes through all operations F_j for j from j_i to $i-1$.

LEMMA 3.6 (Description of iteration l).

There exists an optimal algorithm for Problem 2 that satisfies (P0–3) and
(P4): *There are no R-type operations in iteration $\mathfrak{S}_l$. Hence, every F-type operation is executed once and only once in iteration $\mathfrak{S}_l$.*

Proof. Let $\mathcal{S}$ be an optimal schedule that satisfies (P0–3). We can write

$$\mathcal{S} = \mathfrak{S}_l \cdot \bar{F}_l \cdot \mathfrak{S}_{l-1} \cdot \bar{F}_{l-1} \cdot \ldots \cdot \mathfrak{S}_0 \cdot \bar{F}_0.$$

We know that at the end of the execution of $\mathfrak{S}_l$, the top buffer contains the value x_l and the bottom buffer contains the value $\bar{x}_{l+1}$. Let $\mathcal{M}_l$ and $\mathcal{D}_l$ be the state of the memory and the disk at the end of the execution of $\mathfrak{S}_l$.

Let $\mathfrak{S}'_l$ be the sequence of operations $F_{0 \to l-1}$ where every operation F_i is immediately followed by (i)$W_i^d W_i^m$ if $x_i \in \mathcal{M}_l \cap \mathcal{D}_l$; (ii) else, W_i^m if $x_i \in \mathcal{M}_l$; (iii) else W_i^d if $x_i \in \mathcal{D}_l$. At the end of the execution of $\mathfrak{S}'_l$, the memory and the disk will be in the states $\mathcal{M}_l$ and $\mathcal{D}_l$. Furthermore, if $x_i \in \mathcal{M}_l$ (resp. if $x_i \in \mathcal{D}_l$), the sequence $\mathfrak{S}_l$ contains the operation W_i^m (resp. W_i^d). Thus all W-type operations of $\mathfrak{S}'_l$ are in $\mathfrak{S}_l$. Moreover, $\mathfrak{S}_l$ has to contain all the forward steps from F_0 to F_{l-1}. Thus, every operation in $\mathfrak{S}'_l$ is included in $\mathfrak{S}_l$. Hence, the sequence $\mathcal{S}' = \mathfrak{S}'_l \cdot \bar{F}_l \cdot \mathfrak{S}_{l-1} \cdot \bar{F}_{l-1} \cdot \ldots \cdot \mathfrak{S}_0 \cdot \bar{F}_0$ is valid and has a makespan not larger than $\mathcal{S}$. $\mathcal{S}'$ is then optimal and satisfies (P0–4). □

LEMMA 3.7. *There exists an optimal algorithm for Problem 2 that satisfies (P0–4) and*
(P5): *Given i, j, $0 \leq i, j \leq l-1$, all W_i^d operations are executed before any W_j^m operation.*

Hence, during the first iteration $\mathfrak{S}_l$, we first assign the disk checkpoints before assigning the memory checkpoints. The idea is that the farther away in the graph a checkpoint is set, the more times it is going to be read during the execution.

Proof. Let $\mathcal{S}$ be an optimal algorithm that satisfies (P0–4), but not (P5). We show that we can transform it into an optimal algorithm that also satisfies (P5) without increasing the makespan.

By contradiction, assume that there exist i and j such that W_i^m is executed before W_j^d. According to (P1(i)), every write on the disk occurs during the first iteration $\mathfrak{S}_l$. Thus W_i^m also occurs in iteration $\mathfrak{S}_l$. According to (P4), the F-type operations are not re-executed in iteration $\mathfrak{S}_l$. Thus, necessarily, $i < j$.

According to (P0(ix)), since the algorithm wrote the value x_i in the memory and the value x_j in the disk, they are read later in the schedule. Let $\mathfrak{S}_{i_m}$ be a step when R_i^m occurs and $\mathfrak{S}_{i_d}$ a step when R_j^d occurs. Then we have: $i \leq i_m$ by (P0(iv)) and $j \leq i_d$ by (P0(iv)). Finally, there is only one R-type operation per step (Corollary 3.5); thus $i_m \neq i_d$.

Assume first that $i_m > i_d$. From Corollary 3.5, the first operation of $\mathfrak{S}_{i_m}$ is R_i^m. Thus the value x_i is in the top buffer at the beginning of $\mathfrak{S}_{i_m}$. Furthermore, by definition of the steps, the value x_{i_m} has to be in the top buffer at the end of $\mathfrak{S}_{i_m}$. Since there are no other R-type operations in $\mathfrak{S}_{i_m}$, all forward steps from F_i to F_{i_m} are executed in $\mathfrak{S}_{i_m}$. In particular F_{j-1} is executed in $\mathfrak{S}_{i_m}$. Let n be the number of consecutive F-type operations right before F_{j-1}. Thus $\mathfrak{S}_{i_m}$ has the shape: $\mathfrak{S}_{i_m} = R_i^m \cdot \mathfrak{S}^1 \cdot F_{(j-n)\to(j-1)} \cdot \mathfrak{S}^2$ where the last operation of $\mathfrak{S}^1$ is not a F-type operation. Recall that the time for a disk read is r_d and for a forward step is u_f.

- Assume that $(n+1)u_f > r_d$. Then the sequence $R_i^m \cdot \mathfrak{S}^1 \cdot R_j^d \cdot \mathfrak{S}^2$ has a smaller execution time than $\mathfrak{S}_{i_m}$ contradicting the optimality.
- Assume that $(n+1)u_f \leq r_d$. According to Corollary 3.5, if $\mathfrak{S}^1$ is not empty, then its last operation is W_{j-n-1}^m (if it is empty, then $i = j-n-1$). (P2) ensures that x_{j-n-1} will not be discarded until after x_j has become useless (after $\bar{F}_j$). Since $(n+1)u_f \leq r_d$, we could replace all future instances of R_j^d by $R_{j-n-1}^m \cdot F_{(j-n)\to j}$. Hence W_j^d would be useless, which would contradict the optimality of the schedule.

Hence, $i_m < i_d$. In particular, this is true for any i_m and i_d. Then we can show with similar arguments that this is true until $\bar{F}_i$. Hence, there are not any R_j^d until after $\bar{F}_i$. Since $j > i$, this means that there will not be anymore R_j^d operations at all. Finally, this shows that the execution of W_j^d is useless and can be removed. □

PROPOSITION 3.8 (Disk Checkpoint Persistence). *Given an optimal algorithm for Problem 2 that satisfies (P0–5), then after any operation W_i^d, there are no F_j operations for $0 \leq j \leq i-1$ (nor actions involving the values x_j for $0 \leq j \leq i-1$) until after the execution of $\bar{F}_i$.*

Proof. Let $\mathcal{S}$ be an optimal algorithm that satisfies (P0–5). Assume by contradiction that $\mathcal{S}$ includes W_i^d and there exists $j \geq i$ and $i' < i$ such that iteration $\mathfrak{S}_j$ involves $F_{i'}$. In particular, according to Corollary 3.5, it involves F_{i-1}. According to Corollary 3.5, there exist k, $s \in \{m, d\}$, and n maximum such that

$$\mathfrak{S}_j = R_k^s \mathfrak{S}^1 \cdot F_{(i-1-n)\to(i-1)} \cdot \mathfrak{S}^2.$$

We can first show that $r_d > n$. Indeed, otherwise, $\mathfrak{S}^1 \cdot R_{i'}^d \cdot \mathfrak{S}^2$ has a smaller execution time than does $\mathfrak{S}_j$ for the same result, contradicting the optimality. Let us now show that we can remove all appearances of R_i^d in the schedule, hence decreasing the execution time, which would contradict the optimality of the algorithm.

- Consider the occurrence of R_i^d after $\mathfrak{S}_j$. By maximality of n, if $\mathfrak{S}^1$ is not empty, then the last operation of $\mathfrak{S}^1$ is W_{i-1-n}^m (Corollary 3.5). If $\mathfrak{S}^1$ is empty, then $k = i-1-n$ and $s = m$ (otherwise we could replace $R_k^d F_{(i-1-n)\to(i-1)}$ by R_i^d which would contradict the optimality of the algorithm). Thus, in both cases the value x_{i-1-n} is stored in the memory during $\mathfrak{S}_j$, and (P2) ensures that it will not be discarded until after x_{i-1-n} has become useless

(after $\bar{F}_i$). Because $r_d > n$, we can replace all later appearances of R_i^d by $R_{i-1-n}^m F_{(i-1-n)\to(i-1)}$ at no additional time delay.

- Let us now consider the eventual occurrences of R_i^d anterior to iteration j. Necessarily, all R_i^d anterior to $\mathfrak{S}_j$ are followed by F_i (otherwise one of them is followed by W_i^m and it is not permitted by the memory checkpoint persistence property (P2)). Hence, we can store the value x_{i+1} in the disk instead of the value x_i during $\mathfrak{S}_l$ ($\mathfrak{S}_l$ goes through all forward operation according to (P4)). Then, it is possible to replace all $R_i^d F_i$ operations by R_{i+1}^d reducing the execution time.

Hence, we can decrease the execution time by removing all appearances of R_i^d in the schedule, which shows the contradiction. □

3.2. Optimal execution times. We construct here a dynamic program that solves Problem 2 optimally. To do so we introduce two auxiliary dynamic programs during the construction. The time complexity of our optimal algorithm is $O(l^2)$.

DEFINITION 3.9 ($\mathrm{Opt}_0(l, c_m)$). *Let $l \in \mathbb{N}$ and $c_m \in \mathbb{N}$, $Opt_0(l, c_m)$ is the execution time of an optimal solution to* $\textsc{Prob}(l, c_m)$.

Note that $\mathrm{Opt}_0(l, c_m)$ is the execution time of the routine $\textsc{Revolve}(l, c_m)$, from Griewank and Walther [5].

DEFINITION 3.10 ($\mathrm{Opt}_\infty(l, c_m, w_d, r_d)$). *Let $l \in \mathbb{N}$, $c_m \in \mathbb{N}$, $w_d \in \mathbb{R}$ and $r_d \in \mathbb{R}$, $Opt_\infty(l, c_m, w_d, r_d)$ is the execution time of an optimal solution to* $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$.

To compute $\mathrm{Opt}_\infty(l, c_m, w_d, r_d)$, we first focus on the variant of Problem 2 where the input value x_0 is initially in both the top buffer and the disk:

PROBLEM 3 ($\textsc{Prob}_\infty^{(d)}(l, c_m, w_d, r_d)$). *We want to minimize the makespan of the AC problem with the following parameters:*

		Initial state:
AC graph:	*size* l	
Steps:	u_f, u_b	
Memory:	c_m, $w_m = r_m = 0$	$\mathcal{M}_{ini} = \emptyset$
Disks:	$c_d = +\infty, w_d, r_d$	$\mathcal{D}_{ini} = \{x_0\}$
Buffers:	$\mathcal{B}^\top$, $\mathcal{B}^\bot$	$\mathcal{B}^\top_{ini} = \{x_0\}$, $\mathcal{B}^\bot_{ini} = \{\bar{x}_{l+1}\}$

DEFINITION 3.11 ($\overline{\textsc{Prob}}_\infty^{(d)}(l, c_m, w_d, r_d)$). *Let $l \in \mathbb{N}$, $c_m \in \mathbb{N}$, $w_d \in \mathbb{R}$ and $r_d \in \mathbb{R}$, $\overline{\textsc{Prob}}_\infty^{(d)}(l, c_m, w_d, r_d)$ is the subproblem of $\textsc{Prob}_\infty^{(d)}(l, c_m, w_d, r_d)$ where the space of solution is restricted to the schedules that satisfy the following properties:*
(P6):

(i) (P0(i)), (P0(ii)), (P0(iii)), (P0(iv)), and (P0(v)) are matched.
(ii) Given i, there are no operations F_j for $i \le j \le l-1$ after the execution of $\bar{F}_i$.
(iii) (Memory and disk checkpoint persistence) Let $s \in \{m, d\}$ and $i < l$, if W_i^s is executed, then there are no D_i^s until after the execution of $\bar{F}_i$. Moreover no operations involving F_0 to F_{i-1} or values x_0 to x_{i-1} are taken until after the execution of $\bar{F}_i$.
(iv) Let $0 \le i, j \le l-1$. All W_i^d operations are executed before any W_j^m operation. Moreover, all W_i^d operations are executed during iteration l.

Note that (P6) is a subset of (P0–5). Let $\mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d)$ be the execution time of an optimal solution to $\overline{\textsc{Prob}}_\infty^{(d)}(l, c_m, w_d, r_d)$.

THEOREM 3.12 (Optimal solution to $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$). *Let $l \in \mathbb{N}$, $c_m \in \mathbb{N}$,*

$w_d \in \mathbb{R}$, *and* $r_d \in \mathbb{R}$.

$$Opt_\infty(l, c_m, w_d, r_d) = \min \begin{cases} Opt_0(l, c_m) \\ w_d + Opt_\infty^{(d)}(l, c_m, w_d, r_d) \end{cases}$$

Proof. Let

$$A = \mathrm{Opt}_\infty(l, c_m, w_d, r_d)$$
$$B = \min \begin{cases} \mathrm{Opt}_0(l, c_m) \\ w_d + \mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \end{cases}$$

Let us show that $A \leq B$. Every solution to $\textsc{Prob}(l, c_m)$ is also a solution to $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$. Hence,

$$\mathrm{Opt}_\infty(l, c_m, w_d, r_d) \leq \mathrm{Opt}_0(l, c_m).$$

Let $\bar{\mathcal{S}}_\infty^{(d)}$ be a solution to $\overline{\textsc{Prob}}_\infty^{(d)}(l, c_m, w_d, r_d)$. Then the sequence $W_0^d \cdot \bar{\mathcal{S}}_\infty^{(d)}$ is a solution to $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$. Indeed, the only difference between $\overline{\textsc{Prob}}_\infty^{(d)}(l, c_m, w_d, r_d)$ and $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$ is that in $\overline{\textsc{Prob}}_\infty^{(d)}(l, c_m, w_d, r_d)$, the value x_0 is stored in the disk initially. Thus

$$\mathrm{Opt}_\infty(l, c_m, w_d, r_d) \leq w_d + \mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d)$$

and $\mathrm{Opt}_\infty(l, c_m, w_d, r_d) \leq \min\{\mathrm{Opt}_0(l, c_m);\ w_d + \mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d)\}$.

Let us show that $A \geq B$. According to § 3.1, there exists at least an optimal algorithm $\mathcal{S}_\infty$ to solve $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$ that satisfies (P0–5). According to (P0(x)), the first operation of $\mathcal{S}_\infty$ is a W-type operation.

- If it is a W^m-type operation, according to (P5), there are no W^d-type operations is $\mathcal{S}_\infty$. Hence the disk is not used at all in $\mathcal{S}_\infty$, and $\mathcal{S}_\infty$ is also a solution to $\textsc{Prob}(l, c_m)$. Thus

$$\mathrm{Opt}_\infty(l, c_m, w_d, r_d) \geq \mathrm{Opt}_0(l, c_m).$$

- If it is a W^d-type operation, $\mathcal{S}_\infty$ has the shape $\mathcal{S}_\infty = W_0^d \cdot \mathcal{S}'_\infty$, where $\mathcal{S}'_\infty$ is a solution to $\textsc{Prob}_\infty^{(d)}(l, c_m, w_d, r_d)$. Since $\mathcal{S}_\infty$ satisfies (P0), (P1), (P2), (P3), (P4), and (P5), $\mathcal{S}'_\infty$ satisfies (P6). Hence $\mathcal{S}'_\infty$ is a solution to $\overline{\textsc{Prob}}_\infty^{(d)}(l, c_m, w_d, r_d)$. Thus

$$\mathrm{Opt}_\infty(l, c_m, w_d, r_d) \geq w_d + \mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d).$$

We get that $\mathrm{Opt}_\infty(l, c_m, w_d, r_d) \geq \min\{\mathrm{Opt}_0(l, c_m);\ w_d + \mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d)\}$, which concludes the proof. □

To compute $\mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d)$, we need to consider the problem with only one disk slot containing x_0 at the beginning of the execution:

PROBLEM 4 ($\textsc{Prob}_1^{(d)}(l, c_m, w_d, r_d)$). *We want to minimize the makespan of the AC problem with the following parameters.*

		Initial state:
AC graph:	*size* l	
Steps:	u_f, u_b	
Memory:	c_m, $w_m = r_m = 0$	$\mathcal{M}_{ini} = \emptyset$
Disks:	$c_d = 1, w_d, r_d$	$\mathcal{D}_{ini} = \{x_0\}$
Buffers:	$\mathcal{B}^\top$, $\mathcal{B}^\perp$	$\mathcal{B}^\top_{ini} = \{x_0\}$, $\mathcal{B}^\perp_{ini} = \{\bar{x}_{l+1}\}$

DEFINITION 3.13 ($\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$). *Let $l \in \mathbb{N}$, $c_m \in \mathbb{N}$ and $r_d \in \mathbb{R}$, $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$ is the subproblem of $\text{PROB}_1^{(d)}(l, c_m, +\infty, r_d)$, where the space of solution is restricted to the schedules that satisfy (P6). Note that we put $w_d = +\infty$, to enforce the property that there are no W^d-type operation (and, therefore, the value x_0 is never discarded from the disk) in this schedule.* Let $\text{Opt}_1^{(d)}(l, c_m, r_d)$ be the execution time of an optimal solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$.

THEOREM 3.14 (Optimal solution to $\overline{\text{PROB}}_\infty^{(d)}(l, c_m, w_d, r_d)$). *Given $l \in \mathbb{N}$, $c_m \in \mathbb{N}$, $w_d \in \mathbb{R}$ and $r_d \in \mathbb{R}$:*
If $l = 0$, then

$$Opt_\infty^{(d)}(0, c_m, w_d, r_d) = u_b$$

else

$$Opt_\infty^{(d)}(l, c_m, w_d, r_d) = \min_{1 \le j \le l-1} \begin{cases} Opt_1^{(d)}(l, c_m, r_d) \\ ju_f + Opt_\infty(l-j, c_m, w_d, r_d) + r_d + Opt_1^{(d)}(j-1, c_m, r_d) \end{cases}$$

Proof. For $l = 0$, the result is immediate. Let us prove the result for $l \ge 1$. Let

$$A = \text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d)$$

$$B = \min_{1 \le j \le l-1} \begin{cases} \text{Opt}_1^{(d)}(l, c_m, r_d) \\ ju_f + \text{Opt}_\infty(l-j, c_m, w_d, r_d) + r_d + \text{Opt}_1^{(d)}(j-1, c_m, r_d) \end{cases}$$

Let us show that $A \le B$. Every solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$ is also a solution to $\overline{\text{PROB}}_\infty^{(d)}(l, c_m, w_d, r_d)$. Hence,

$$\text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \le \text{Opt}_1^{(d)}(l, c_m, r_d).$$

Given j, $1 \le j \le l-1$, let $\mathcal{S}_1^{(d)}$ be an optimal solution to $\overline{\text{PROB}}_1^{(d)}(j-1, c_m, r_d)$. Let $\mathcal{S}_\infty$ be an optimal solution to $\text{PROB}_\infty(l-j, c_m, w_d, r_d)$ that satisfies (P0–5). Let $\mathcal{S}'_\infty$ be the sequence $\mathcal{S}_\infty$ where every index of the operations are increased by j (F_i becomes F_{i+j}, $W_i^{(m)}$ becomes $W_{i+j}^{(m)} \dots$). $\mathcal{S}'_\infty$ is still valid and has the same makespan as $\mathcal{S}_\infty$. Then, the sequence $F_{0\to j} \cdot \mathcal{S}'_\infty \cdot R_0^d \cdot \mathcal{S}_1^{(d)}$ is a solution to $\text{PROB}_\infty^{(d)}(l, c_m, w_d, r_d)$. By construction this sequence also satisfies (P6). Its execution time is $ju_f + \text{Opt}_\infty(l-j, c_m, w_d, r_d) + r_d + \text{Opt}_1^{(d)}(j-1, c_m, r_d)$. Thus, for all $1 \le j \le l-1$:

$$\text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \le ju_f + \text{Opt}_\infty(l-j, c_m, w_d, r_d) + r_d + \text{Opt}_1^{(d)}(j-1, c_m, r_d).$$

In particular it is smaller than the minimum over all j, hence the result.

Let us show that $A \ge B$. Let $\mathcal{S}_\infty^{(d)}$ be an optimal solution to $\overline{\text{PROB}}_\infty^{(d)}(l, c_m, w_d, r_d)$. $\mathcal{S}_\infty^{(d)}$ satisfies (P6) and its makespan is $\text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d)$.

Assume first that there is at least one W-type operation in $\mathcal{S}_\infty^{(d)}$. Consider the first one. We can prove that it occurs before the first $\bar{F}$-type operation.

- If it is a W^d-type, then it occurs before the first $\bar{F}$-type according to (P6(iv))
- If it is a W^m-type, then obviously $c_m > 0$. If no W^m-operation occured during iteration l, then at the beginning of iteration $l-1$, R_0^d is executed. Hence a better solution would be better to start with W_0^m and D_0^m at the beginning of iteration $l-1$, which contradicts the optimality of $\mathcal{S}_\infty^{(d)}$.

Hence, the first W-type operation in $\mathcal{S}_\infty^{(d)}$ occurs before the first $\bar{F}$-type operation. Then two possibilities exist.

- The first operation in $\mathcal{S}_\infty^{(d)}$ is W_0^m. Since $\mathcal{S}_\infty^{(d)}$ satisfies (P6(iv)), there are no W^d-type operations in $\mathcal{S}_\infty^{(d)}$. Thus no other value than x_0 will be stored in the disk during the execution. Furthermore, because x_0 is stored in memory, it will not be read from the disk (otherwise $\mathcal{S}_\infty^{(d)}$ will not be optimal). Thus $\mathcal{S}_\infty^{(d)}$ is also a solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$ and
$$\text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \geq \text{Opt}_1^{(d)}(l, c_m, r_d).$$
- Otherwise, the first operation in $\mathcal{S}_\infty^{(d)}$ is not W_0^m. Consider the first W-type operation. Because it occurs before the first $\bar{F}$-type operation, it cannot be W_0^m (otherwise it would be the first operation in $\mathcal{S}_\infty^{(d)}$), nor W_0^d (x_0 is already stored on disk). Let W_j^s ($s \in \{m, d\}$) be the first W-type operation in $\mathcal{S}_\infty^{(d)}$, then $j > 0$.
  ⋆ We proved that there are no $\bar{F}$-type operations before W_j^s in $\mathcal{S}_\infty^{(d)}$. By definition, there are no W-type operations before W_j^s in $\mathcal{S}_\infty^{(d)}$. Since before W_j^s the memory is empty, there are no D-type operations in $\mathcal{S}_\infty^{(d)}$. Since the only value in one of the storage is x_0 (in the disk), the only possible R-type operation before W_j^s would be R_0^d. The only reason to execute R_0^d would be to perform F_0. However, F_0 can be executed at the beginning of $\mathcal{S}_\infty^{(d)}$ at no cost, since the value x_0 is already in the top buffer. Thus, there are only F-type operations before W_j^s in $\mathcal{S}_\infty^{(d)}$ (P6(i)).
  ⋆ According to (P6(iii)), after W_j^s, there are no operations involving values x_0 to x_{j-1} until after the operation $\bar{F}_j$.
  ⋆ According to (P6(ii)), there are no operations involving values x_j to x_l after the operation $\bar{F}_j$.
  ⋆ Since after the operation $\bar{F}_j$ the content of the top buffer is useless, the first operation after $\bar{F}_j$ has to be an R-type operation.
  ⋆ Moreover, since the only value from x_0 to x_{j-1} in one of the storage after $\bar{F}_j$ is x_0 (in the disk), it has to be R_0^d. Thus, based on all these considerations, $\mathcal{S}_\infty^{(d)}$ has the following shape:
$$\mathcal{S}_\infty^{(d)} = F_{0\to j} \cdot W_j^s \cdot \mathcal{S}_1 \cdot R_0^d \cdot \mathcal{S}_2$$
  where (i) no operations involve values x_0 to x_{j-1} in $\mathcal{S}_1$ and (ii) no operations involve values x_j to x_l in $\mathcal{S}_2$.
  Let $\mathcal{S}_1'$ be the sequence $W_j^s \cdot \mathcal{S}_1$, where every index of the operations is decreased by j (F_i becomes F_{i-j}, W_i^m becomes W_{i-j}^m,...). Then $\mathcal{S}_1'$ is a solution to $\text{PROB}_\infty(l-j, c_m, w_d, r_d)$, whose makespan is necessarily not smaller than $\text{Opt}_\infty(l-j, c_m, w_d, r_d)$.
  On the other hand, the sequence $\mathcal{S}_2$ executes all the $\bar{F}$-type operations from $\bar{F}_{j-1}$ down to $\bar{F}_0$ with no operations involving values x_j to x_l. Furthermore, $\mathcal{S}_2$ does not use disk slots, except the one already used by value x_0. Since $\mathcal{S}_\infty^{(d)}$ satisfies (P0(5)), $\mathcal{S}_2$ satisfies (P6). Hence $\mathcal{S}_2$ is a solution to $\overline{\text{PROB}}_\infty^{(d)}(j-1, c_m, w_d, r_d)$, and its makespan is greater than $\text{Opt}_\infty^{(d)}(j-1, c_m, w_d, r_d)$. Finally, we have
$$\text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \geq ju_f + \text{Opt}_\infty(l-j, c_m, w_d, r_d) + r_d + \text{Opt}_1^{(d)}(j-1, c_m, r_d);$$

in particular it is smaller than B.

We note that if there is no W-type operation, because we do not use any additional disk slot, $\mathcal{S}_\infty^{(d)}$ is also a solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$ and $\text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \geq \text{Opt}_1^{(d)}(l, c_m, r_d)$. This shows that $A \geq B$ and concludes the proof. □

THEOREM 3.15 (Optimal solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$). *Let $l \in \mathbb{N}$, $c_m \in \mathbb{N}$ and $r_d \in \mathbb{R}$:*
If $l = 0$, then

$$Opt_1^{(d)}(0, c_m, r_d) = u_b$$

else

$$Opt_1^{(d)}(l, c_m, r_d) = \min_{1 \leq j \leq l-1} \begin{cases} Opt_0(l, c_m) \\ ju_f + Opt_0(l-j, c_m) + r_d + Opt_1^{(d)}(j-1, c_m, r_d) \end{cases}$$

Proof. For $l = 0$, the result is immediate. Let us prove the result for $l \geq 1$.
Let

$$A = \text{Opt}_1^{(d)}(l, c_m, r_d)$$

$$B = \min_{1 \leq j \leq l-1} \begin{cases} \text{Opt}_0(l, c_m) \\ ju_f + \text{Opt}_0(l-j, c_m) + r_d + \text{Opt}_1^{(d)}(j-1, c_m, r_d) \end{cases}$$

Let us show that $A \leq B$. Every solution to $\text{PROB}(l, c_m)$ is also a solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$. Hence,

$$\text{Opt}_1^{(d)}(l, c_m, r_d) \leq \text{Opt}_0(l, c_m).$$

Given j, $1 \leq j \leq l-1$, let $\mathcal{S}_2$ be an optimal solution to $\overline{\text{PROB}}_1^{(d)}(j-1, c_m, r_d)$. Let $\mathcal{S}_1$ be an optimal solution to $\text{PROB}(l-j, c_m)$. Let $\mathcal{S}_1'$ be the sequence $\mathcal{S}_1$ where every index of the operations are increased by j (F_i becomes F_{i+j}, $W_i^{(m)}$ becomes $W_{i+j}^{(m)} \dots$). $\mathcal{S}_1'$ is still valid and has the same makespan as $\mathcal{S}_1$. Then, the sequence $F_{0 \to j} \cdot \mathcal{S}_1' \cdot R_0^d \cdot \mathcal{S}_2$ is a solution to $\text{PROB}_1^{(d)}(l, c_m, w_d, r_d)$. By construction this sequence does not contain any W^d-type operation since neither $\mathcal{S}_1'$ nor $\mathcal{S}_2$ do. Its execution time is $ju_f + \text{Opt}_0(l-j, c_m) + r_d + \text{Opt}_1^{(d)}(j-1, c_m, r_d)$. Thus, for all $1 \leq j \leq l-1$:

$$\text{Opt}_1^{(d)}(l, c_m, r_d) \leq ju_f + \text{Opt}_0(l-j, c_m) + r_d + \text{Opt}_1^{(d)}(j-1, c_m, r_d)\}.$$

In particular it is smaller than the minimum over all j, hence the result.

Let us show that $A \geq B$. Let $\mathcal{S}_1^{(d)}$ be an optimal solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$. $\mathcal{S}_1^{(d)}$ satisfies (P6) and does not contain any W^d-type operations. Its makespan is $\text{Opt}_1^{(d)}(l, c_m, r_d)$.

First, note that if $c_m > 0$, then there is a W^m-type operation in iteration l of $\mathcal{S}_1^{(d)}$. Otherwise, if no W^m-operation occured during iteration l, then at the beginning of iteration $l-1$, R_0^d is executed. Hence a better solution would be better to start with W_0^m and D_0^m at the beginning of iteration $l-1$, which contradicts the optimality of $\mathcal{S}_1^{(d)}$.

Hence, the first W-type operation in $\mathcal{S}_1^{(d)}$ occurs before the first $\bar{F}$-type operation. Then there are two possibilities.

- The first operation in $\mathcal{S}_1^{(d)}$ is W_0^m. Because x_0 is stored in memory, it will not be read from the disk (otherwise $\mathcal{S}_\infty^{(d)}$ will not be optimal). Thus $\mathcal{S}_\infty^{(d)}$ is also a solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$ and

$$\text{Opt}_1^{(d)}(l, c_m, r_d) \geq \text{Opt}_0(l, c_m).$$

- Otherwise, the first operation in $\mathcal{S}_1^{(d)}$ is not W_0^m. Consider the first W-type operation. It occurs before the first $\bar{F}$-type operation and hence cannot be W_0^m (otherwise it would be the first operation in $\mathcal{S}_1^{(d)}$). Let W_j^m (there are no W^d-type operations) be the first W-type operation in $\mathcal{S}_1^{(d)}$, $j > 0$.
  ⋆ We proved that there are no $\bar{F}$-type operations before W_j^m in $\mathcal{S}_\infty^{(d)}$. By definition, there are no W-type operations before W_j^m in $\mathcal{S}_1^{(d)}$. Since the only value in one of the storage slots is x_0 (in the disk), the only possible R-type operation before W_j^s would be R_0^d. The only reason to execute R_0^d would be to perform F_0. However, F_0 can be executed at the beginning of $\mathcal{S}_1^{(d)}$ at no cost, since the value x_0 is already in the top buffer. Thus, there are only F-type operations before W_j^m in $\mathcal{S}_\infty^{(d)}$ (P6(i)).
  ⋆ According to (P6(iii)), after W_j^m, there are no operations involving values x_0 to x_{j-1} until after the operation $\bar{F}_j$.
  ⋆ According to (P6(ii)), there are no operations involving values x_j to x_l after the operation $\bar{F}_j$.
  ⋆ Since after the operation $\bar{F}_j$, the content of the top buffer is useless, the first operation after $\bar{F}_j$ has to be an R-type operation.
  ⋆ Moreover, since the only value from x_0 to x_{j-1} in one of the storage slots after $\bar{F}_j$ is x_0 (in the disk), it has to be R_0^d. Thus, based on all this considerations, $\mathcal{S}_1^{(d)}$ has the following shape:

$$\mathcal{S}_1^{(d)} = F_{0\to j} \cdot W_j^m \cdot \mathcal{S}_1 \cdot R_0^d \cdot \mathcal{S}_2$$

  where (i) no operations involve values x_0 to x_{j-1} in $\mathcal{S}_1$ and (ii) no operations involve values x_j to x_l in $\mathcal{S}_2$.
  Let $\mathcal{S}_1'$ br the sequence $W_j^m \cdot \mathcal{S}_1$, where every index of the operations is decreased by j (F_i becomes F_{i-j}, W_i^m becomes W_{i-j}^m,...). Then $\mathcal{S}_1'$ is a solution to $\text{PROB}(l-j, c_m)$, whose makespan is necessarily not smaller than $\text{Opt}_0(l-j, c_m, w_d, r_d)$.
  On the other hand, the sequence $\mathcal{S}_2$ executes all the $\bar{F}$-type operations from $\bar{F}_{j-1}$ down to $\bar{F}_0$ with no operations involving values x_j to x_l. Furthermore, $\mathcal{S}_2$ does not use disk slots, except the one already used by value x_0. Thus $\mathcal{S}_2$ is a solution to $\overline{\text{PROB}}_1^{(d)}(j-1, c_m, r_d)$, and its makespan is greater than $\text{Opt}_\infty^{(d)}(j-1, c_m, r_d)$. Finally, we have

$$\text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \geq ju_f + \text{Opt}_\infty(l-j, c_m, w_d, r_d) + r_d + \text{Opt}_1^{(d)}(j-1, c_m, r_d);$$

  in particular it is smaller than B.

Note that if there is no W-type operation, because we do not use any additional disk slot, $\mathcal{S}_\infty^{(d)}$ is also a solution to $\overline{\text{PROB}}_1^{(d)}(l, c_m, r_d)$ and $\text{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \geq \text{Opt}_1^{(d)}(l, c_m, r_d)$. This shows that $A \geq B$ and concludes the proof. □

THEOREM 3.16 (Simplification). *Let $l \in \mathbb{N}$, $c_m \in \mathbb{N}$, $w_d \in \mathbb{R}$ and $r_d \in \mathbb{R}$:*

$$Opt_\infty(l, c_m, w_d, r_d) = \min \begin{cases} Opt_0(l, c_m) \\ w_d + \min_{1 \leq j \leq l-1} \{ju_f + Opt_\infty(l-j, c_m, w_d, r_d) + r_d + Opt_1^{(d)}(j-1, c_m, r_d)\} \end{cases}$$

$$Opt_1^{(d)}(l, c_m, r_d) = \min \begin{cases} Opt_0(l, c_m) \\ \min_{1 \leq j \leq l-1} \{ju_f + Opt_0(l-j, c_m) + r_d + Opt_1^{(d)}(j-1, c_m, r_d)\} \end{cases}$$

Proof. Theorem 3.12 states that:

$$\mathrm{Opt}_\infty(l, c_m, w_d, r_d) = \min \begin{cases} \mathrm{Opt}_0(l, c_m) \\ w_d + \mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) \end{cases}$$

Let us show that if $\mathrm{Opt}_\infty(l, c_m, w_d, r_d) = w_d + \mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) < \mathrm{Opt}_0(l, c_m)$, then:

$$\mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) = \min_{1 \leq j \leq l-1} \{ju_f + \mathrm{Opt}_\infty(l-j, c_m, w_d, r_d) + r_d + \mathrm{Opt}_1^{(d)}(j-1, c_m, r_d)\}.$$

For convenience, let us note $A = \min_{1 \leq j \leq l-1} \{ju_f + \mathrm{Opt}_\infty(l-j, c_m, w_d, r_d) + r_d + \mathrm{Opt}_1^{(d)}(j-1, c_m, r_d)\}$. Assume by contradiction that $\mathrm{Opt}_\infty(l, c_m, w_d, r_d) = w_d + \mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) < \mathrm{Opt}_0(l, c_m)$ and $\mathrm{Opt}_\infty^{(d)}(l, c_m, w_d, r_d) = \mathrm{Opt}_1^{(d)}(l, c_m, r_d) < A$. If $\mathrm{Opt}_1^{(d)}(l, c_m, r_d) = \mathrm{Opt}_0(l, c_m)$, we would have $w_d + \mathrm{Opt}_0(l, c_m) < \mathrm{Opt}_0(l, c_m)$, which is absurd. So, according to Theorem 3.14, there exists a $j \in \{1, ..., l-1\}$, such that

$$\mathrm{Opt}_1^{(d)}(l, c_m, r_d) = ju_f + \mathrm{Opt}_0(l-j, c_m) + r_d + \mathrm{Opt}_1^{(d)}(j-1, c_m, r_d).$$

Since $\mathrm{Opt}_1^{(d)}(l, c_m, r_d) < A$, in particular,

$$\mathrm{Opt}_1^{(d)}(l, c_m, r_d) < ju_f + \mathrm{Opt}_\infty(l-j, c_m, w_d, r_d) + r_d + \mathrm{Opt}_1^{(d)}(j-1, c_m, r_d).$$

Thus, $\mathrm{Opt}_0(l-j, c_m) < \mathrm{Opt}_\infty(l-j, c_m, w_d, r_d)$, which is also absurd. Hence, if $\mathrm{Opt}_\infty(l, c_m, w_d, r_d) < \mathrm{Opt}_0(l, c_m)$, then $\mathrm{Opt}_\infty(l, c_m, w_d, r_d) = w_d + A$, which proves the first equation of the Theorem. The second equation is an immediate corollary of Theorem 3.15 , knowing that $\mathrm{Opt}_0(0, c_m) = u_b$. □

3.3. Optimal algorithms. Based on the dynamic programs presented in Theorem 3.16, we can design two polynomial algorithms to compute an optimal solution to $\textsc{Prob}_1^{(d)}(l, c_m, w_d, r_d)$ and $\textsc{Prob}_\infty(l, c_m, w_d, r_d)$. These algorithms use the binomial checkpointing algorithm Revolve, designed by Griewank and Walther [14], that returns an optimal solution to $\textsc{Prob}(l, c_m)$. We also define Shift, the routine that takes a sequence S and an index `ind` and returns S shifted by `ind` (meaning for all $i \leq l$, $s \in \{m, d\}$, W_i^s are replaced by $W_{i+\mathrm{ind}}^s$, R_i^s are replaced by $R_{i+\mathrm{ind}}^s$, F_i by $F_{i+\mathrm{ind}}$, and $\bar{F}_i$ by $\bar{F}_{i+\mathrm{ind}}$). Note that sequence $\textsc{Shift}(S, \texttt{ind})$ has the same execution time as sequence S.

We design the polynomial algorithm 1D-Revolve (Algorithm 1) that, given the auto-adjoint graph size $l \in \mathbb{N}$, $c_m \in \mathbb{N}$ memory slots and a cost $r_d \geq 0$ to read from disk, returns 1D-$\textsc{Revolve}(l, c_m, r_d)$ an optimal schedule for $\overline{\textsc{Prob}}_1^{(d)}(l, c_m, r_d)$. We also design the polynomial algorithm Disk-Revolve (Algorithm 2) that, given the

values l, c_m, w_d and r_d, returns DISK-REVOLVE(l, c_m, w_d, r_d) an optimal schedule for PROB$_\infty(l, c_m, w_d, r_d)$.

These algorithms need the values $\mathrm{Opt}_0(x, c_m)$, $\mathrm{Opt}_1^{(d)}(x, c_m, w_d, r_d)$ and $\mathrm{Opt}_\infty(x, c_m, w_d, r_d)$ for every $x \in \{1, ..., l\}$. They have the same complexity as the dynamic programs presented in Section 3.2, namely $O(l^2)$. Note that, for convenience, we assumed that the values of $\mathrm{Opt}_0(x, c_m)$, $\mathrm{Opt}_1^{(d)}(x, c_m, w_d, r_d)$ and $\mathrm{Opt}_\infty(x, c_m, w_d, r_d)$ were precomputed before the execution, but these algorithms can be easily modified to compute these values on the fly.

Algorithm 1 1D-REVOLVE

procedure 1D-REVOLVE(l, c_m, r_d)
  $\mathcal{S} \leftarrow \emptyset$
  if $\mathrm{Opt}_1^{(d)}(l, c_m, r_d) = \mathrm{Opt}_0(l, c_m)$ **then**
    $\mathcal{S} \leftarrow$ REVOLVE(l, c_m)
  else
    Let j such that

$$\mathrm{Opt}_1^{(d)}(l, c_m, r_d) = ju_f + \mathrm{Opt}_0(l - j, c_m) + r_d + \mathrm{Opt}_1^{(d)}(j - 1, c_m, r_d)$$

    $\mathcal{S} \leftarrow F_{0 \to (j-1)}$
    $\mathcal{S} \leftarrow \mathcal{S} \cdot$ SHIFT (REVOLVE$(l - j, c_m), j)$
    $\mathcal{S} \leftarrow \mathcal{S} \cdot R^d_{\mathrm{ind}} \cdot$ 1D-REVOLVE $(j - 1, c_m, r_d)$
  end if
  return $\mathcal{S}$
end procedure

Algorithm 2 DISK-REVOLVE

procedure DISK-REVOLVE(l, c_m, w_d, r_d)
  $\mathcal{S} \leftarrow \emptyset$
  if $\mathrm{Opt}_\infty(l, c_m, w_d, r_d) = \mathrm{Opt}_0(l, c_m)$ **then**
    $\mathcal{S} \leftarrow$ REVOLVE(l, c_m)
  else
    Let j such that

$$\mathrm{Opt}_\infty(l, c_m, w_d, r_d) = w_d + ju_f + \mathrm{Opt}_\infty(l - j, c_m, w_d, r_d) + r_d + \mathrm{Opt}_1^{(d)}(j - 1, c_m, r_d)$$

    $\mathcal{S} \leftarrow W_0^d \cdot F_{0 \to (j-1)}$
    $\mathcal{S} \leftarrow \mathcal{S} \cdot$ SHIFT (DISK-REVOLVE$(l - j, c_m, w_d, r_d), j)$
    $\mathcal{S} \leftarrow \mathcal{S} \cdot R^d_{\mathrm{ind}} \cdot$ 1D-REVOLVE $(j - 1, c_m, r_d)$
  end if
  return $\mathcal{S}$
end procedure

4. Simulations. In this section we compare our optimal algorithm with the only (to the best of our knowledge) algorithm for multilevel checkpointing, introduced by Stumm and Walther [13].

4.1. Stumm and Walther's algorithm (SWA$^\star(l, c_m, w_d, r_d)$). Stumm and Walther [13] solve Problem 2 using a variant of REVOLVE [14]. REVOLVE takes l the size of the AC graph and s the number of storage slots as argument and returns an optimal solution for Problem 1. Stumm and Walther show that in REVOLVE(l, s),

some storage slots are less used than others. They design the SWA algorithm that takes l the size of the AC graph, c_m the number of memory slots, and c_d the number of disk slots as argument and returns the solution $\textsc{Revolve}(l, c_d + c_m)$ where the c_d storage slots that are the least used are considered as disk slots and all the others are considered as memory slots.

To solve Problem 2, SWA* returns the best solution among the solutions returned by $\mathrm{SWA}(l, c_m, c_d, w_d, r_d)$, that is,

$$\mathrm{SWA}^\star(l, c_m, w_d, r_d) = \min_{c_d=0\ldots l-c_m} \mathrm{SWA}(l, c_m, c_d, w_d, r_d)$$

(having more than l storage slots is useless).

4.2. Simulation setup. For the simulations we have tested our algorithm and Stumm and Walther's algorithm on AC graphs of size up to 20,000 with different numbers of memory checkpoints. In global ocean circulation modeling [7], a graph of size 8,640 represents one year of results with an hourly timestep.

In the experiments, we normalize all time values by setting u_f to 1. We take $u_b = 2.5$ as a representative value [7]. Here we present results for $c_m \in \{2, 5, 10, 25\}$ and $w_d = r_d \in \{1, 2, 5, 10\}$.

In Figure 5 we reproduce Stumm and Walther's results in order to study the behavior of SWA. We plot the execution time of SWA as a function of c_d for a fixed graph size (note that we used a logarithmic scale for the horizontal axis for better readability). We compare it with the optimal solution $\mathrm{Opt}_\infty(l, c_m, w_d, r_d)$.

In Figures 6 and 7 we plot the ratio between SWA* and $\mathrm{Opt}_\infty(l, c_m, w_d, r_d)$ as a function of the size of the AC graph with different values of c_m, w_d and r_d.

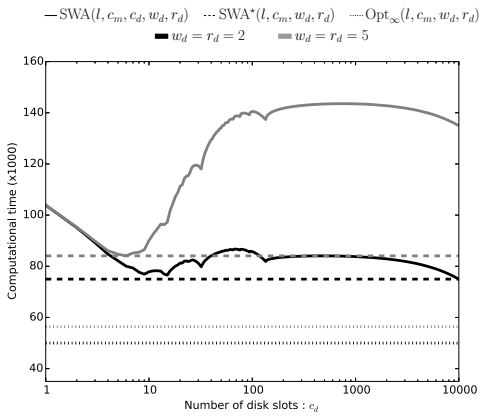


Figure 5: Makespan of SWA on an AC graph of size 10,000 as a function of c_d for $c_m = 5$. SWA* and $\mathrm{Opt}_\infty(l, c_m, w_d, r_d)$ are also plotted for comparison.

4.3. Simulation results. First we observe the behavior of SWA given the amount of available disk slots for different disk access costs. The two plots in Figure 5 are representative of the two behaviors we observed for SWA during our experiments. We can see that the makespan of $\mathrm{SWA}(l, c_m, c_d, w_d, r_d)$ is always strictly higher than the optimal one for an infinite number of disk slots $\mathrm{Opt}_\infty(l, c_m, w_d, r_d)$ (see the dotted line). For all values studied, the evolution of the execution time of SWA when the

amount of storage increases follows a specific pattern that can be divided into three phases.

1. A very fast decrease with the first additional disk slots.
2. A succession of small increases and decreases.
3. A slow but steady decrease until all steps are stored (remember that the horizontal axis has a logarithmic scale for better readability).

In all our experiments, the minimum for SWA is reached either at the end of step 1 with a very low number of disk checkpoints ($c_d = r_d = 5$ in Figure 5) or at the end of step 3 with a very high number of disk checkpoints ($c_d = r_d = 2$ in Figure 5), depending on the disk access costs and the number of memory slots. Eventually, given the general shape of the SWA performances, we assume that when the size of the graph increases enough, the minimum value is always reached at the end of step 3, when every output of the AC graph is stored in one of the storage slots.

Note that Stumm and Walther observed a fourth phase [13] where the computational time increases again when the number of disk checkpoints gets closer to the total number of steps. They explained it by saying that when the volume of data stored on the disk reaches a threshold, the cost of a disk access increases, which in turn increases the computational time. We do not observe such a fourth step because we plot the computational time obtained when giving the model parameters as input to SWA (and the cost of disk access remains constant).

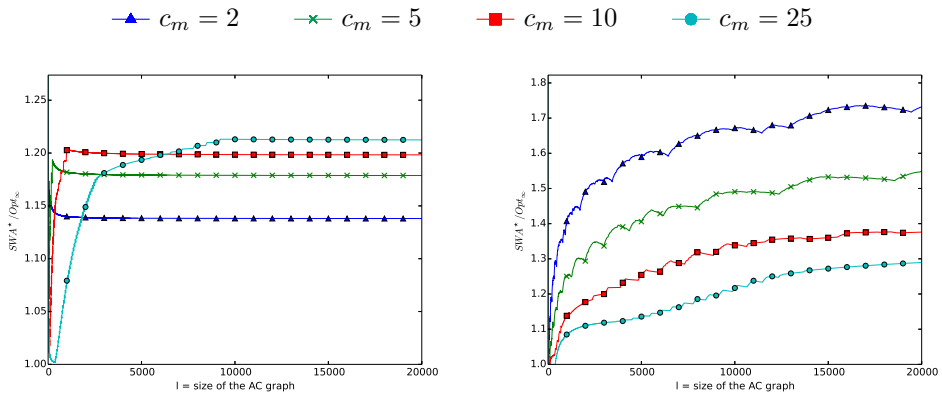
In the following, the time complexity of SWA* does not allow us to run large instances of l. To be able to plot SWA* for large AC graphs, we plot a faster version of SWA* that takes the previous remarks into account, namely, that assume that the minimum is either reached at the end of phase 1 (for small values of c_d) or at the end of phase 3. More precisely, we consider that the end of phase 1 is reached before a number of disk size equal to 200 (for the problem sizes considered in this paper), and we plot a faster version of SWA$^\star(l, c_m, w_d, r_d)$:

$$\text{SWA}^\star(l, c_m, w_d, r_d) = \min\left(\min_{c_d=0\ldots200} \text{SWA}(l, c_m, c_d, w_d, r_d), \text{SWA}(l, c_m, l-c_m, w_d, r_d)\right).$$

This assumption allows us to compute SWA* for large values of l and to compare it with the optimal computational time for an infinite number of disk slots.

Figures 6 and 7 depict the overhead of using SWA* compared with the optimal solution $\text{Opt}_\infty(l, c_m, w_d, r_d)$ that we designed in § 3.2. For unlimited disk slots, SWA* returns the best solution among the solutions returned by SWA, and this solution is always greater than $\text{Opt}_\infty(l, c_m, w_d, r_d)$. We observe that the ratio increases until the size of the graph reaches a threshold where the ratio becomes constant. This is particularly visible in Figure 6a where we can see that the value of this threshold increases with the number of memory slots c_m. In practice, the threshold delimits the moment when the number of disk slots used by SWA* goes from a relatively small number (when the minimum for $\text{SWA}(l, c_m, c_d, w_d, r_d)$ is reached at the end of phase 1) to $c_d = l - c_m$ (when the minimum for $\text{SWA}(l, c_m, c_d, w_d, r_d)$ is reached when all forward steps are checkpointed, at the end of phase 3).

We are interested in the limit ratio reached after the threshold because we are considering the problem for very large graphs. In Figures 6a and 7 we can see that this ratio increases when c_m or w_d and r_d increase. When $r_d = w_d = 1$, the ratio limit for $c_m = 2$ is approximately 1.14, which means that SWA* is 14% slower than the optimal algorithm we designed in §3.2. For a memory of size $c_m = 10$, this overhead increases to 20% for large AC graphs. When $r_d = w_d = 5$, the ratio limit is not



(a) $w_d = r_d = 1$ (b) $w_d = r_d = 5$

Figure 6: Ratio $\text{SWA}^\star(l, c_m, w_d, r_d)/\text{Opt}_\infty(l, c_m, w_d, r_d)$ as a function of l.

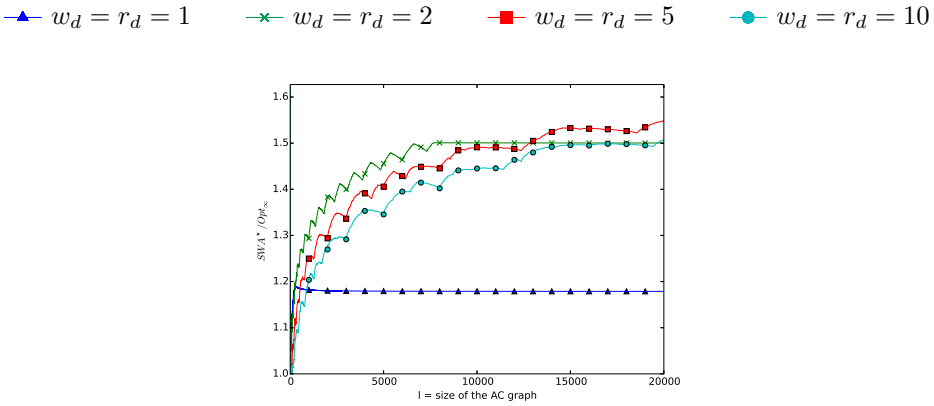


Figure 7: Ratio $\text{SWA}^\star(l, c_m, w_d, r_d)/\text{Opt}_\infty(l, c_m, w_d, r_d)$ as a function of l, for $c_m = 5$.

reached for AC graphs of size inferior to 20,000. But since the ratio for $c_m = 2$ will be higher than 1.6, we can state that SWA* will perform at least 60% slower than the optimal algorithm on large AC graphs for any memory sizes.

5. Related work. On the more theoretical side, this work is related to the many papers that have addressed the pebble game and its variants. In the pioneering work of Sethi and Ullman [12], the objective was to minimize the number of registers that must be used while computing an arithmetic expression. The problem of determining whether a general DAG can be executed with a given number of pebbles has been shown NP-hard by Sethi [11] if no vertex is pebbled more than once. The general problem allowing recomputation, that is, repebbling a vertex that has been pebbled before, has been proven PSPACE complete by Gilbert, Lengauer, and Tarjan [2]. However, this problem has a polynomial complexity for tree-shaped graphs [12]. Problem 1 can be seen as a game with c_m pebbles for the (very specific) AC dependence graph of Figure 1.

A variant of the game with two levels of storage was introduced by Hong and Kung [8] under the name of *I/O pebble game*, which was used to derive lower bounds on I/O operations and study the tradeoff between I/O operation and main memory size for particular graphs. A comprehensive summary of results for pebble games can be found in the book by Savage [10]. Problem 2 can be seen as an I/O pebble game with the (very specific) AC dependence graph of Figure 1.

6. Conclusion and future work. In this paper we have provided optimal algorithms for the adjoint checkpointing problem with two storage locations: a bounded number of memory slots with zero access cost and an infinite number of disk slots with a given write and read costs. We have compared our optimal solution with existing work, showing that our solution gives significantly better execution time.

We have identified applications in computational fluid dynamics and earth systems modeling that could benefit from our approach. We will examine whether the theoretical benefits of the optimal multistage schedule can be realized in practice. Future theoretical directions include the solution to the online AC problem (where the size l of the AC graph is not known before execution), within the same framework. Another possible extension could be to solve the same problem as in this paper but with a limited number of disk checkpoints. Large-scale platforms are failure-prone, and checkpointing for resilience in addition to checkpointing for performance will lead to challenging algorithmic problems. As an intermediate step, we will examine the problem of maximizing progress during a fixed time period. This situation arises in practice when the job scheduler limits the maximum duration of jobs (limits such as 12 hours are common).

Acknowledgments. We thank Sri Hari Krishna Narayanan and Vladimir Ufimtsev for constructive conversations. This material is based upon work supported by the French Research Agency (ANR) through the Rescue project and the Joint Laboratory for Extreme-Scale Computing and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program, under contract number DE-AC02-06CH11357. Yves Robert is with Institut Universitaire de France.

REFERENCES

[1] R. Giering and T. Kaminski, *Recomputations in reverse mode AD*, in Automatic Differentiation: From Simulation to Optimization, G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, eds., Computer and Information Science, Springer, New York, 2002, ch. 33, pp. 283–291.

[2] J. R. Gilbert, T. Lengauer, and R. E. Tarjan, *The pebbling problem is complete in polynomial space*, SIAM J. Comput., 9 (1980), pp. 513–524.

[3] M. B. Giles and N. A. Pierce, *An introduction to the adjoint approach to design*, Flow, Turbulence and Combustion, 65 (2000), pp. 393–415.

[4] A. Griewank, *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*, Optimization Methods and software, 1 (1992), pp. 35–54.

[5] A. Griewank and A. Walther, *Algorithm 799: Revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation*, ACM Transactions on Mathematical Software (TOMS), 26 (2000), pp. 19–45.

[6] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, no. 105 in Other Titles in Applied Mathematics, SIAM, Philadelphia, PA, 2nd ed., 2008.

[7] P. Heimbach, C. Hill, and R. Giering, *An efficient exact adjoint of the parallel MIT general circulation model, generated via automatic differentiation*, Future Generation Computer Systems, 21 (2005), pp. 1356–1371.

[8] J.-W. Hong and H. Kung, *I/O complexity: The red-blue pebble game*, in STOC'81: Proceedings of the 13th ACM symposium on Theory of Computing, ACM Press, 1981, pp. 326–333.
[9] A. Jameson, *Aerodynamic shape optimization using the adjoint method*, Lectures at the Von Karman Institute, Brussels, (2003).
[10] J. E. Savage, *Models of Computation: Exploring the Power of Computing*, Addison-Wesley, 1997.
[11] R. Sethi, *Complete register allocation problems*, in STOC'73: Proceedings of the fifth annual ACM symposium on Theory of computing, ACM Press, 1973, pp. 182–195.
[12] R. Sethi and J. Ullman, *The generation of optimal code for arithmetic expressions*, J. ACM, 17 (1970), pp. 715–728.
[13] P. Stumm and A. Walther, *Multistage approaches for optimal offline checkpointing*, SIAM Journal on Scientific Computing, 31 (2009), pp. 1946–1967.
[14] A. Walther, *Program reversal schedules for single-and multi-processor machines*, PhD thesis, PhD thesis, Institute of Scientific Computing, Technical University Dresden, Germany, 1999.