Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation

By **Q. Wang** and **P. Moin**

1. Motivation and objectives

Adjoint equations of differential equations have seen widespread applications in optimization, inverse problems and uncertainty quantification. A major challenge in solving adjoint equations for time dependent systems has been the need to use the solution of the original system in the adjoint calculation and the associated memory requirement. In applications where storing the entire solution history is impractical, checkpointing methods have frequently been used. However, traditional checkpointing algorithms such as **revolve** require a priori knowledge of the number of time steps, making these methods incompatible with adaptive time stepping.

We propose a dynamic checkpointing algorithm applicable when the number of time steps is a priori unknown. Our algorithm maintains a specified number of checkpoints on the fly as time integration proceeds for an arbitrary number of time steps. The resulting checkpoints at any snapshot during the time integration have the optimal repetition number. The efficiency of our algorithm is demonstrated both analytically and experimentally in solving adjoint equations. This algorithm also has significant advantage in automatic differentiation when the length of execution is variable.

2. Background

In numerical simulation of dynamical systems, the adjoint equation is commonly used to obtain the derivative of a predefined objective function with respect to many independent variables that control the dynamical system. This approach, known as the adjoint method, has many uses in scientific and engineering simulations. In control theory-based optimization problems and inverse problems, the derivative obtained via the adjoint method is used to drive a gradient-based optimization iteration procedure Bewley *et al.* (2001) Jameson (1988); In posterior error estimation and uncertainty quantification, this derivative is used to analyze the sensitivity of an objective function to various uncertain parameters and conditions of the system Giles & Pierce (2002) Giles & Suli (2002). Efficient numerical solution of the adjoint equation is essential to these adjoint-based applications.

The main challenge in solving the adjoint equations for time dependent systems results from their time-reversal characteristic. Although a forward-time Monte Carlo algorithm Wang *et al.* (2008) has been proposed for solving the adjoint equations, checkpointing schemes remain the dominant method to address this challenge. For a general nonlinear dynamical system (referred to here as the original system as opposed to the adjoint system)

$$\dot{u} = G(u,t), \quad u(0) = u_0, \quad 0 \le t \le T,$$

its adjoint equation is a linear dynamical system with a structure similar to the original system, except that the adjoint equation is initialized at time T. It then evolves backward in time, such that

$$\dot{q} = A(u,t)q + b(u,t), \quad q(T) = q_0(u(T)), \quad 0 \le t \le T.$$

While the specific forms of q_0, A and b depend on the predefined objective function, the required procedure to solve the adjoint equation is the same: to initialize the adjoint equation, $u(T)$ must first be obtained; as the time integration proceeds backward in time, the solution of the original system u is needed from $t = T$ backward to $t = 0$. At each time step of the adjoint time integration, the solution of the original system at that time step must either be already stored in memory or recalculated from the solution at the last stored time step. If we have sufficient memory to store the original system at all the time steps, the adjoint equation can be solved without recalculating the original system. High fidelity scientific and engineering simulations, however, require both many time steps and large memory to store the solution at each time step, making the storage of the solution at all the time steps impractical. Checkpointing schemes, in which only a small number of time steps is stored, apply naturally in this situation. Surprisingly, checkpointing does not necessarily take more computing time than storing all time steps, since the modified memory hierarchy may compensate for the required recalculations. Kowarz & Walther (2006).

Checkpointing schemes can significantly reduce the memory requirement but may increase the computation time Charpentier (2000). In the first solution of the original system, the solutions at a small set of carefully chosen time steps called checkpoints are stored. During the following adjoint calculation, the discarded intermediate solutions are then recalculated by solving the original system restarting from the checkpoints. Old checkpoints are discarded once they are no longer useful, while new checkpoints are created in the recalculation process and replace the old ones. Griewank Griewank (1992) proposed a binomial checkpointing algorithm for certain values of the number of time steps. This recursive algorithm, known as **revolve**, has been proven to minimize the number of recalculations for any given number of allowed checkpoints and all possible number of time steps Griewank & Walther (2000). With δ number of allowed checkpoints and τ recalculations, $\binom{\delta+\tau}{\tau}$ time steps can be integrated. The **revolve** algorithm achieves logarithmic growth of spatial and temporal complexity with respect to the number of time steps, acceptable for the majority of scientific and engineering simulations.

Griewank's **revolve** algorithm assumes a priori knowledge of the number of time steps, which represents an inconvenience, perhaps even a major obstacle in certain applications. In solving hyperbolic partial differential equations, for example, the size of each time step Δt can be constrained by the maximum wavespeed in the solution field. As a result, the number of time steps taken for a specified period of time is not known until the time integration is completed. A simple, but inefficient workaround is to solve the original system once for the sole purpose of determining the number of time steps, before applying the **revolve** algorithm. Two algorithms have been developed to generate checkpoints with unknown number of time steps a priori. The first is **a-revolve** Hinze & Sternberg (2005). For a fixed number of allowed checkpoints, the **a-revolve** algorithm maintains a cost function that approximates the total computation time of recalculating all intermediate solutions based on current checkpoint allocation. As time integration proceeds, the heuristic cost function is minimized by re-allocating the checkpoints. Although a priori knowledge of the number of time steps is not required, the **a-revolve** algorithm is shown to be only slightly more costly than Griewank's **revolve** algorithm. However, it is difficult to prove a theoretical bound with this algorithm, making its superiority to the simple workaround questionable. The other algorithm is the **online checkpointing** Algorithm Heuveline & Walther (2006). This algorithm theoretically proves to be optimal; however, it explicitly assumes that the number of time steps is no more than $\binom{\delta+2}{\delta}$, and produces an error when this assumption is violated. This upper bound in the number of time steps is extended to $\binom{\delta+3}{\delta}$ in a recent work Stumm & Walther (2008). This limitation makes their algorithm useless when the number of time steps is large and the memory is limited.

We propose a dynamic checkpointing algorithm for solving adjoint equations. This algorithm, compared to previous ones, has three main advantages: First, unlike Griewank's **revolve**, it requires no a priori knowledge of the number of time steps, and is applicable to both static and adaptive time-stepping. Second, in contrast to Hinze and Sternberg's **a-revolve**, our algorithm is theoretically optimal in that it minimizes the repetition number τ, defined as the maximum number of times a specific time step is evaluated during the adjoint computation. As a result, the computational cost has a theoretical upper-bound. Third, our algorithm works for an arbitrary number of time steps, unlike previous online checkpointing algorithms which limit the length of the time integration. When the number of time steps is less than $\binom{\delta+2}{\delta}$, our algorithm produces the same set of checkpoints as Heuveline and Walther's result; as the number of time steps exceeds this limit, our algorithm continues to update the checkpoints and ensures their optimality in the repetition number τ. We prove that for arbitrarily large number of time steps, the maximum number of recalculations for a specific time step in our algorithm is as small as possible. This new checkpointing algorithm combines the advantages of previous algorithms, without having the drawbacks of them.

As with **revolve**, our dynamic checkpointing algorithm is applicable both in solving adjoint equations and in automatic differentiation (AD). **revolve** and other static checkpointing algorithms can be inefficient in differentiating certain types of source code whose length of execution is a priori uncertain. Examples include "while" statements where the number of loops to be executed is only determined during execution, and procedures with large chunks of code in "if" statements. These cases may seriously degrade the performance of static checkpointing AD schemes. Our dynamic checkpointing algorithm can solve this issue by achieving close to optimal efficiency regardless of the length of execution. Since most source codes to which automatic differentiation is applied are complex, our algorithm can significantly increase the overall performance of automatic differentiation software.

This paper is organized as follows. In Sec. 3 we first demonstrate our checkpoint-generation algorithm. We prove the optimality of the algorithm, based on an assumption about the adjoint calculation procedure. Section 4 proves this assumption by introducing and analyzing our full algorithm for solving adjoint equations, including checkpoint-generation and checkpoint-based recalculations in backward adjoint calculation. The performance of the algorithm is theoretically analyzed and experimentally demonstrated in Sec. 5. Conclusions are provided in Sec. 6.

3. Dynamic checkpointing algorithm

The key concepts of **time step index**, **level** and **dispensability** of a checkpoint must be defined before introducing our algorithm. The time step index is used to label the intermediate solutions of both the original equation and the adjoint equation at each time step. It is 0 for the initial condition of the original equation, 1 for the solution after the first time step, and increments as the original equation proceeds forward. For the adjoint solution the time step index decrements and reaches 0 for the last time step of the adjoint time integration. The checkpoint with time step index i, or equivalently, the checkpoint at time step i, is defined as the checkpoint that stores the intermediate solution with time step index i.

We define the concept of **level** and **dispensability** of checkpoints in our dynamic checkpointing algorithm. As each checkpoint is created, a fixed level is assigned to it. We will prove in Sec. 4 that this level indicates how many recalculations are needed to reconstruct all intermediate solutions between the checkpoint and the previous checkpoint of the same level or higher. We define a checkpoint as **dispensable** if its time step index is smaller than another checkpoint of a higher level. When a checkpoint is created, it has the highest time step index, and is therefore not dispensable. As new checkpoints are added, existing checkpoints become dispensable if a new checkpoint has a higher level. Our algorithm allocates and maintains δ checkpoints and one placeholder based on the level of the checkpoints and whether they are dispensable. During each time step, a new checkpoint is allocated. If the number of checkpoints exceeds δ, an existing dispensable checkpoint is removed to maintain the specified number of checkpoints.

Algorithm 1 Dynamic allocation of checkpoints

```
Require: $\delta > 0$ given.
  Save time step 0 as a checkpoint of level $\infty$;
  for $i = 0, 1, \ldots$ do
    if the number of checkpoints <= $\delta$ then
      Save time step $i + 1$ as a checkpoint of level 0;
    else if At least one checkpoint is dispensable then
      Remove the dispensable checkpoint with the largest time step index;
      Save time step $i + 1$ as a checkpoint of level 0;
    else
      $l \Leftarrow$ the level of checkpoint at time step $i$;
      Remove the checkpoint at time step $i$;
      Save time step $i + 1$ as a checkpoint of level $l + 1$;
    end if
    Calculate time step $i + 1$ of the original system;
  end for.
```

Two simplifications have been made in Algorithm 1. First, the algorithm maintains $\delta + 1$ checkpoints, one more than what is specified. This is because the last checkpoint is always at time step $i + 1$, where the solution has yet to be calculated. As a result, the last checkpoint stores no solution and takes little memory space (referred to here as a placeholder checkpoint), making the number of real checkpoints δ. The other simplification is the absence of the adjoint calculation procedures, which is addressed in Sec. 4.

In the remainder of this section, we calculate the growth rate of the checkpoint levels as the time step i increases. This analysis is important because the maximum checkpoint level determines the maximum number of times of recalculation in the adjoint calculation τ, as proven in Sec. 4. Through this analysis, we show that our algorithm achieves the same number of time steps as Griewank's **revolve** for any given δ and τ. Since **revolve** is known to maximize the number of time steps for a fixed number of checkpoints δ and times of calculations τ, our algorithm is thus optimal in the same sense.

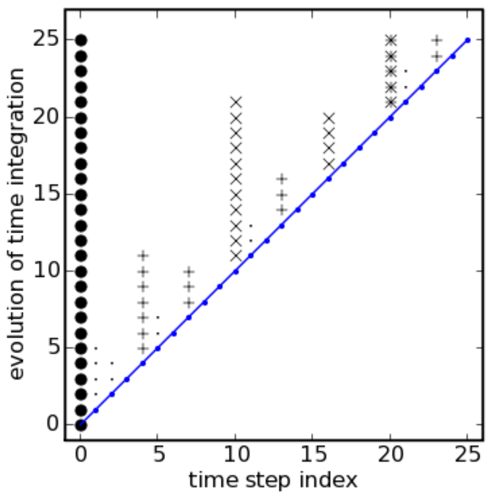

FIGURE 1. Dynamic allocation of checkpoints for $\delta = 3$. The plot shows the checkpoints distribution during 25 time steps of time integration. Each horizontal cross-section on the plot represents a snapshot of the time integration history, from time step 0 to time step 25, indicated by the y axis. Different symbols represent different levels of checkpoints: Circles are level ∞ checkpoint at time step 0. Thin dots, "+", ''X" and star symbols correspond to level 0, 1, 2 and 3 checkpoints respectively. The thick dots connected by a line indicate the time step index of the current solution, which is also the position of the placeholder checkpoint.

Algorithm 1 starts by saving time step 0 as a level infinity checkpoint, and time steps 1 through time step δ as level 0 checkpoints. When $i = \delta$, there are already $\delta + 1$ checkpoints, none of which are dispensable. The algorithm enters the "else" clause with $l = 0$. In this clause, the checkpoint at time step δ is removed, and time step $\delta + 1$ is saved as a level 1 checkpoint, making all checkpoints except for the first and last ones dispensable. As the time integration continues, these $\delta - 1$ dispensable checkpoints are then recycled, while time steps $\delta + 2$ to 2δ take their place as level 0 checkpoints. A third checkpoint of level 1 is created for time step $2\delta + 1$, while the remaining $\delta - 2$ level 0 checkpoints become dispensable. Each time a level 1 checkpoint is made, one less level 0 checkpoint is dispensable, resulting in one less space between the current and the next level 1 checkpoints. The $\delta + 1$st level 1 checkpoint is created for time step $(\delta + 1) + \delta + (\delta - 1) + \ldots + 2 = \binom{\delta+2}{2} - 1$. At this point, all $\delta + 1$ checkpoints are level 1, while the space between them is an arithmetic sequence. All checkpoints created thus far are exactly the same as the online checkpointing algorithm Heuveline & Walther (2006), although their algorithm breaks down and produces an error at the very next time step.

Our algorithm continues by allocating a level 2 checkpoint for time step $\binom{\delta+2}{2}$. At the same time, the level 1 checkpoint at time step $\binom{\delta+2}{2} - 1$ is deleted, and all other level 1 checkpoints become dispensable. A similar process of creating level 2 checkpoints ensues. The third level 2 checkpoint is created for time step $\binom{\delta+2}{2} + \binom{\delta+1}{2}$, after the same evolution described in the previous paragraph with only $\delta - 1$ free checkpoints. The creation of level 2 checkpoints continues until the $\delta + 1$st level 2 checkpoint is created with time step index $\binom{\delta+2}{2} + \binom{\delta+1}{2} + \ldots + \binom{3}{2} = \binom{\delta+3}{3} - 1$. The creation of the first level 3 checkpoint follows at time step $\binom{\delta+3}{3}$. Figure 3 illustrates an example of this process where $\delta = 3$. Until now, we found that the time steps of the first level 0, 1, 2 and 3 checkpoints are respectively, 1, $\delta+1$, $\binom{\delta+2}{2}$, and $\binom{\delta+3}{3}$. This interesting pattern leads to our first proposition.

PROPOSITION 1. *In Algorithm 1, the first checkpoint of level τ is always created for time step $\binom{\delta+\tau}{\tau}$.*

To prove this proposition, we note that it is a special case of the following lemma when $i=0$, making it only necessary to prove the lemma.

LEMMA 1. *In Algorithm 1, let i be the time step of a level $\tau-1$ or higher checkpoint. The next checkpoint with level τ or higher is at time step $i+\binom{\delta-n_i+\tau}{\tau}$ and is level τ, where n_i is the number of indispensable checkpoints allocated before time step i.*

Proof. We use induction here. When $\tau=0$, $\binom{\delta-n_i+\tau}{\tau}=1$. The next checkpoint is allocated at time step $i+1$, and its level is non-negative; therefore, it is level 0 or higher. If there is a dispensable checkpoint at time step i, the new checkpoint is level 0; otherwise, it is level 1.

Assuming that the lemma holds true for any $0\le\tau<\tau_0$, we now prove it for $\tau=\tau_0$. First, if $n_i=\delta$, $\binom{\delta-n_i+\tau}{\tau}=1$. No dispensable checkpoint exists in this case; thus, the "else" clause is executed to create a higher level checkpoint at time step $i+1$. If $n_i<\delta$, we use the induction hypothesis. The next level $\tau-1$ checkpoint is created at time step

$$i_1 = i+\binom{\tau-n_i+\tau-1}{\tau-1},$$

incrementing the number of indispensable checkpoints

$$n_{i_1}=n_i+1.$$

As a result, the following level $\tau-1$ checkpoints are created at time steps

$$i_2 = i_1+\binom{\delta-n_{i_1}+\tau-1}{\tau-1},$$

$$i_3 = i_2+\binom{\delta-n_{i_2}+\tau-1}{\tau-1},$$

$$\cdots$$

$$i_{k+1} = i_k+\binom{\delta-n_{i_k}+\tau-1}{\tau-1},$$

$$\cdots$$

This creation of level $\tau-1$ checkpoints continues until $n_{i_k}=n_i+k=\delta+1$. At this point, all existing checkpoints are level $\tau-1$. Consequently, the "else" clause is executed with $l=\tau-1$ and creates the first level τ checkpoint at time step

$$i_{\delta-n_i+1} = i+\binom{\delta-n_i+\tau-1}{\tau-1}+\binom{\delta-n_i-1+\tau-1}{\tau-1}+\ldots+\binom{\tau-1}{\tau-1}.$$

Using Pascal's rule

$$\binom{m}{k}=\binom{m-1}{k}+\binom{m-1}{k-1}=\sum_{i=0}^{k}\binom{m-1-i}{k-i}$$

with $m = \delta - n_i + \tau$ and $k = \delta - n_i$, this equation simplifies to

$$i_{\delta - n_i + 1} = i + \binom{\delta - n_i + \tau}{\tau},$$

which completes the induction. □

Note that the checkpoints produced by our algorithm and the **revolve** algorithm are identical when the number of time steps equals to $\binom{\tau+\delta}{\tau} - 1$ for any τ. In this case, our algorithm generates δ level τ checkpoints, whose time step indices are separated with distances $\binom{\delta+\tau-i}{\delta}, i = 0, 1, \ldots, \delta - 1$, respectively. On the other hand, the checkpoints created by the **revolve** algorithm have the same distance of separation. Although this looks like a coincidence, it actually indicates the optimality of both algorithms in this case †.

Theorem 1 leads us to the key conclusion of our theoretical analysis.

Corollary 1. *For any $\tau > 0$ and $\delta > 0$, η time steps can be taken in Algorithm 1 without any checkpoint being raised to level $\tau + 1$, if and only if*

$$\eta \le \binom{\delta + \tau + 1}{\tau + 1}.$$

Combined with Theorem 6.1 of Griewank (1992), Corollary 1 indicates that the checkpoints produced by Algorithm 1 have the optimal repetition number, determined by

$$\binom{\delta + \tau}{\tau} < \eta \le \binom{\delta + \tau + 1}{\tau + 1}$$

This conclusion is based on the fact that

$$\textbf{maximum checkpoint level} \ge \textbf{repetition number}. \tag{3.1}$$

The repetition number here does not count the first time the original system is solved. This equality will be proved in the next section by analysis of the adjoint calculation algorithm.

4. Adjoint calculation

In this section, we fully describe our algorithm of solving adjoint equations using dynamic checkpointing. The algorithm consists of a forward sweep and a backward sweep. The forward sweep solves the original system and stores intermediate solutions at checkpoints. The adjoint system is then solved in the backward sweep, using the stored checkpoints to initialize recalculations. Algorithm 2 describes this high level scheme; the details of checkpoint manipulations and recalculations are given in Algorithms 3 and 4.

The algorithm for advancing the original system is essentially identical to Algorithm 1. In the first forward sweep, checkpoints generated by repeatedly calling Algorithm 3 satisfy Lemma 1, Theorem 1, and hence Corollary 1. Moreover, a recalculation sweep consisting of a series of calls to Algorithm 3 also satisfies Lemma 1. In each time step, the solution at time step i, instead of time step $i + 1$, is stored. This strategy ensures that the last checkpoint at time step $i + 1$ is always a placeholder checkpoint. Although our algorithm updates and maintains $\delta + 1$ checkpoints, only δ of them store solutions.

† Through some analysis, one can show that this checkpoint pattern is the only optimal one for this number of time steps.

Algorithm 2 High level scheme to solve the adjoint equation

```
Initialize the original system;
$i \Leftarrow 0$;
**Save** time step 0 as a placeholder checkpoint of level $\infty$;
**while** the termination criteria of the original system is not met **do**
  Solve the original system from time step $i$ to $i+1$ using Algorithm 3;
  $i \Leftarrow i+1$;
**end while**
Initialize the adjoint system;
**while** $i >= 0$ **do**
  $i \Leftarrow i-1$;
  Solve the adjoint system from time step $i+1$ to $i$ using Algorithm 4;
**end while**.
```

Algorithm 3 Solving the original system from time step i to $i+1$

```
**Require:** $\delta > 0$ given; solution at time step $i$ has been calculated.
  **if** the number of checkpoints $<= \delta$ **then**
    **Save** time step $i+1$ as a checkpoint of level 0;
  **else if** At least one checkpoint is dispensable **then**
    **Remove** the dispensable checkpoint with the largest time step index;
    **Save** time step $i+1$ as a checkpoint of level 0;
  **else**
    $l \Leftarrow$ the level of checkpoint at time step $i$;
    **Remove** the checkpoint at time step $i$;
    **Save** time step $i+1$ as a checkpoint of level $l+1$;
  **end if**
  **if** time step $i$ is in the current set of checkpoints **then**
    **Store** the solution at time step $i$ to the checkpoint;
  **end if**
  **Calculate** time step $i+1$ of the original system.
```

Compared to checkpoint allocation, retrograding the adjoint system is relatively simple. Solving the adjoint solution at time step i requires both the solution to the adjoint system at time step $i+1$ and the solution to the original system at time step i. The latter can be directly retrieved if there is a checkpoint for time step i; otherwise, it must be recalculated from the last saved checkpoint. Note that this algorithm calls Algorithm 3 to recalculate the solution of the original system at time step i from the last saved checkpoint, during which more checkpoints are created between the last saved checkpoint and time step i. These new checkpoints reduce the number of recalculations using memory space freed by removing checkpoints after time step i.

Figure 2 and 3 shows examples of the entire process of Algorithm 2 with four different values of δ. As can be seen, the less the specified number of checkpoints δ is, the more recalculations of the original equation are performed, and the longer it takes to solve the adjoint equation. When $\delta \geq 25$, there is enough memory to store every time step, so no recalculation is done. The maximum finite checkpoint level is 0 in this case. When $\delta = 6$, the maximum finite checkpoint level becomes 1, and at most 1 recalculation is done for each time step. When δ decreases to 5 and 3, the maximum finite checkpoint level increases to 2 and 3 respectively, and the maximum number of recalculations also

Algorithm 4 Solving the adjoint system from time step $i+1$ to i

```
Require: $\delta > 0$ given; adjoint solution at time step $i+1$ has been calculated.
  Remove the placeholder checkpoint at time step $i+1$;
  if the last checkpoint is at time step $i$ then
    Retrieve the solution at time step $i$, making it a placeholder checkpoint;
  else
    Retrieve the solution at the last checkpoint, making it a placeholder checkpoint;
    Initialize the original system with the retrieved solution;
    Solve the original system to time step $i$ by calling Algorithm 3;
  end if
  Calculate time step $i$ of the adjoint system.
```

increases to 2 and 3 respectively. From these examples, we see that the number of recalculations at each time step is bounded by the level of the checkpoints after that time step. In the remaining part of this section, we focus on proving this fact, starting with Lemma 2.

LEMMA 2. *Denote τ_i as the highest level of all checkpoints whose time steps are greater than i. For any i, τ_i does not increase for each adjoint step. Furthermore, if i is between the time steps of the last two indispensable checkpoints before an adjoint step, τ_i decreases after this adjoint step.*

Proof. Fix i, consider an adjoint step from time step j to $j-1$, where $j > i$. Note that before this adjoint step, time step j is a placeholder checkpoint. Denote the level of this checkpoint as l_j. If time step $j-1$ is stored in a checkpoint before this adjoint step, then checkpoint j is removed and all other checkpoints remain, making Lemma 2 trivially true. If time step $j-1$ is not stored before this adjoint step, the original system is recalculated from the last checkpoint to time step $j-1$. We now prove that this recalculation sweep does not produce any checkpoints with level higher or equal to l_j. As a result, τ_i does not increase; furthermore, if no other level τ_i checkpoint has a time step greater than i, τ_i decreases.

Denote the time step of the last checkpoint as k, and its level as l_k. We use capital letter B to denote the recalculation sweep in the current adjoint step. Note that sweep B starts from time step k and ends at time step $j-1$. On the other hand, the checkpoint at time step j is created either during the first forward sweep, or during a recalculation sweep in a previous adjoint step. We use capital letter A to denote a part of this sweep from time step k to when the checkpoint at time step j is created. To compare the sweeps A and B, note the following two facts: First, the checkpoint at time step k exists before sweep A. This is because sweep A created the checkpoint at time step j, making all subsequent recalculations before sweep B start from either this checkpoint or a checkpoint whose time step is greater than j. Consequently, the checkpoint at time step k is not created after sweep A, it is created either by sweep A or exists before sweep A. Secondly, because any sweep between A and B starts at time step j or greater, it does not create any checkpoint with a time step index smaller than k. On the other hand, any checkpoint removed during or after sweep A and before B is always the lowest level at that time, and thus, does not cause the increase of the number of dispensable checkpoints. As a result, the number of dispensable checkpoints with time step indices smaller than k is no less at the beginning of sweep A than at the beginning of sweep B. This is identical to stating

$$n_k^A \geq n_k^B,$$

where n_k is the number of indispensable checkpoints with time step indices less than k; the superscript identifies the sweep in which the indispensable checkpoints are counted.

Now, we complete the proof by comparing sweep A and B in the following two cases: If $l_k < l_j$, we assert that sweep A does not create any higher level checkpoint than l_k at time steps between k and j. Suppose the contrary is true, that sweep A has created an at least level $l_k + 1$ checkpoint between the two time steps. Because no checkpoint is between time steps k and j when sweep B starts, the supposed checkpoint must have been removed before sweep B happens. But in that case, the checkpoint at time step k must be also removed, because its level is lower. This cannot happen because sweep B starts at this checkpoint. This contradiction proves our assertion. Because time step j is the first higher level checkpoint than l_k created by sweep A with larger time step index than k, its time step index, based on Lemma 1, is

$$j = k + \binom{\delta - n_k^A + l_k + 1}{l_k + 1}.$$

Since $n_k^A \geq n_k^B$, we further get

$$j \leq k + \binom{\delta - n_k^B + l_k + 1}{l_k + 1}.$$

Using Lemma 1 again, we conclude that the first checkpoint with level higher than l_k is at a time step index greater or equal to j. But time step $j - 1$ is the last step of sweep B; therefore, sweep B does not create any checkpoint with a level higher than l_k. Since $l_k < l_j$, no level l_j or higher checkpoint is created by sweep B. This completes the proof in the first case.

In the second case, $l_k \geq l_j$, we assert that the checkpoint at time step j is the first one created by sweep A with level higher or equal to l_j. Suppose the contrary is true, that sweep A has created an at least level l_j checkpoint between the two time steps. Because no checkpoint is between time steps k and j when sweep B starts, the supposed checkpoint must have been removed before sweep B happens. But in that case, the checkpoint at time step j must be also removed, since it has the same level and is at a larger time step index. This is contradictory because sweep B starts at this checkpoint. Therefore, our assertion is true, and from Lemma 1 we get

$$j \leq k + \binom{\delta - n_k^A + l_j}{l_j}.$$

Since $n_k^A \geq n_k^B$, we further get

$$j \leq k + \binom{\delta - n_k^B + l_j}{l_j}.$$

Using Lemma 1 again, we conclude that the first checkpoint with level higher or equal to l_j is at a time step index greater or equal to j. But time step $j - 1$ is the last step of sweep B; therefore, sweep B does not create any checkpoint with a level higher or equal to l_j. This completes the proof in the second case. □

Lemma 2 indicates that the highest level of all checkpoints whose time step index is greater than a particular value is monotone decreasing during the adjoint time integration. Equipped with this result, we prove the main proposition in this section.

PROPOSITION 2. *Let $i \geq 0$. Time integrating the adjoint system from any time step $i' > i$ to time step i requires calling Algorithm 3 at most τ_i times for advancing from time step i to time step $i+1$, where τ_i is the highest level of all the checkpoints with time step indices greater than i before the adjoint time integration.*

Proof. To prove this proposition, we use induction again. If $\tau_i = 0$, then the "else" clause in Algorithm 3 is never executed after time step i. All checkpoints created after time step are i neither dispensable nor removed, including the $i+1$st time step. No recalculation from time step i to $i+1$ is necessary to obtain the solution at time step $i+1$, in which case the proposition holds.

Assuming that the proposition holds true for all $\tau_i \leq \mathcal{T}$, we prove it for $\tau_i = \mathcal{T} + 1$. Among all level τ_i checkpoints with a time step index larger than i, let j be the smallest time step index of such checkpoints. Because at the beginning of the adjoint time integration, no checkpoint with a time step index greater than i has a higher level than τ_i, by Lemma 2, the entire adjoint time integration from time step i' to i does not produce any checkpoint with a higher level than τ_i and time step index greater than i. As a result, the level τ_i checkpoint at time step j is not removed by any recalculation sweep until the adjoint time integration reaches time step j. Any recalculation sweep before the adjoint time integration reaches time step j starts either at time step j or from a checkpoint with greater time step than j. Therefore, recalculation at time step i is only possible after the adjoint time integration reaches time step j. We focus on this part of the adjoint calculation during the rest of the proof.

If $i = k - 1$, because the solution at $i + 1 = k$ is no longer needed, no recalculation from i to $i+1$ is done. Otherwise, $j-1$ is not a checkpoint, and a recalculation sweep is performed from the last checkpoint to time step $j-1$. This sweep includes zero or one recalculation from time step i to time step $i+1$, depending on where it is initialized. Because k is the smallest time step index of level τ_i checkpoints with time step greater than i, time step i is between the last two indispensable checkpoints before the adjoint step from time step j to $j-1$. This enables us to use the "furthermore" part of Lemma 2 and conclude that τ_i decreases to $\mathcal{T}$ or lower after this adjoint step. Therefore, from our induction hypothesis, the number of recalculations at time step i after this adjoint step is at most $\tau_i - 1$. Combining this number with the possible 1 recalculation during the adjoint step from j to $j-1$, the total recalculations from time step i to time step $i+1$ during all the adjoint steps from time step i' to i is at most τ_i, completing our induction. □

As a special case of the proposition, consider time step i' to be the last time step in solving the original system, and $i = 0$. Let $\tau = \tau_0$ be the highest finite level of all checkpoints when the first forward sweep is completed. According to the proposition, $\tau_i \leq \tau$ for all i, resulting in the following corollary.

COROLLARY 2. *In Algorithm 3, let τ be the maximum finite checkpoint level after the original system is solved for the first time. Algorithm 4 is called at most τ times for each time step during the backward sweep in solving the adjoint equation.*

Corollary 2 effectively states Eq. (3.1). Combined with Corollary 1, it proves that our dynamic checkpointing algorithm achieves a repetition number of τ for $\binom{\delta+\tau}{\tau}$ time steps.

This, as proven by Griewank (1992), is the optimal performance for any checkpointing algorithm.

5. Algorithm efficiency

The previous two sections presented our dynamic checkpointing algorithm and its optimality in the repetition number. Here, we discuss the implication of this optimality on the performance of this algorithm, and demonstrate its efficiency using numerical experiments. We begin by providing a theoretical upper bound on the total number of time step recalculations in our adjoint.

PROPOSITION 3. *The overall number of forward time step recalculations in the adjoint calculation n_r is bounded by*

$$n_r < \tau\eta - \binom{\delta+\tau}{\tau-1}, \tag{5.1}$$

where η is the number of time steps, τ is the repetition number determined by

$$\binom{\delta+\tau}{\tau} < \eta \le \binom{\delta+\tau+1}{\tau+1},$$

and δ is the number of allowed checkpoints.

Proof. Since the repetition number is τ, there is at least one checkpoint of level τ, and no checkpoint of a higher level (Corollary 1). The first level τ checkpoint is created at time step index $\binom{\delta+\tau}{\tau}$, according to Proposition 1. Since no checkpoint of a higher level is created, this first level τ checkpoint can not removed by Algorithm 1. We split the adjoint calculation at the first level τ checkpoint at time step index $\binom{\delta+\tau}{\tau}$. First, in calculating the adjoint steps of index from η to $\binom{\delta+\tau}{\tau}$, every forward step is recalculated at most τ times by definition of the repetition number τ. The total number of forward time step recalculations in this part is less or equal to

$$\tau\left(\eta - \binom{\delta+\tau}{\tau}\right).$$

In fact it is always less than, since the very last time step is never recalculated. Second, in calculating the adjoint steps of index from $\binom{\delta+\tau}{\tau} - 1$ to 0, if no checkpoint exists in this part, the total number of forward time step recalculations is

$$\tau\binom{\delta+\tau}{\tau} - \binom{\delta+\tau}{\tau-1}$$

Griewank & Walther (2000). The total number of recalculations in this part is less than this number if there is one or more checkpoints between time step 0 and $\binom{\delta+\tau}{\tau}$. Therefore, the total number of forward time step recalculations in the entire adjoint calculation, which is the sum of the number of recalculations in the two parts, is less than

$$\tau\eta - \binom{\delta+\tau}{\tau-1}.$$

□

Having this upper bound, we can the total number of recalculations of our algorithm with the optimal static checkpointing scheme. The minimum number of total forward time step calculations for an adjoint calculation of length η is

$$(\tau+1)\,\eta - \binom{\delta+\tau+1}{\tau}$$

Griewank & Walther (2000), including the $\eta - 1$ forward time step calculations before the adjoint calculation begins. Therefore, the minimum number of total recalculations is

$$n_r \geq n_{r\,opt} = \tau\eta - \binom{\delta+\tau+1}{\tau} + 1. \tag{5.2}$$

Eqs. (5.1) and (5.2) bound the total number of forward time step recalculations of our dynamic checkpointing scheme. They also bound the deviation from optimality in terms of total recalculations.

$$n_r - n_{r\,opt} < \binom{\delta+\tau+1}{\tau} - \binom{\delta+\tau}{\tau-1} - 1 = \binom{\delta+\tau}{\tau} - 1$$

This bound naturally leads to the following corollary:

Corollary 3. *Using our dynamic checkpointing scheme takes less total recalculations than running the simulation forward, determining the number of time steps, and then (knowing the time step count) using the proven optimal* **revolve** *algorithm.*

Proof. Running the simulation forward to determine the number of time steps, then use **revolve** takes a total number of $n'_r = \eta + n_{r\,opt}$ recalculations. Since $\eta > \binom{\delta+\tau}{\tau}$, we have $n'_r > n_{r\,opt} + \binom{\delta+\tau}{\tau} > n_r$. □

With these theoretical bounds, we next study experimentally the actual number of recalculations of our dynamic checkpointing algorithm. Figures 5 plots the actual number of forward time step recalculations together with the upper and lower bound defined by Eqs. (5.1) and (5.2). The total number of recalculations is divided by the number of time steps, and the resulting average number of recalculations for each time step is plotted. As can be seen, the actual number lies between the lower and upper bound, as predicted by the theory. Also, more points tend to be closer to the lower bound than to the upper bound, and some points lie exactly on the lower bound. This means that our algorithm in most cases outperforms what Corollary 3 guarantees.

While the total number of recalculations of our dynamic checkpointing algorithm is not necessarily as small as static checkpointing schemes, the repetition number is provably optimal in any situation. Table 5 shows the maximum number of time steps our algorithm can proceed for a given number of checkpoints and number of recalculations (repetition number). The range of checkpoints are typical for most of today's unsteady simulations in fluid mechanics, which range from 10 checkpoints in high-fidelity multi-physics simulations where limited memory is a serious issue, to 100 checkpoints in calculations where memory requirements are less stringent. The maximum number of time steps is calculated by the formula $\binom{\delta+\tau}{\tau}$, where δ is the number of checkpoints, and τ the number of recalculations plus 1 (to include the first calculation of the original system.) As can be seen, the number of time steps grows very rapidly as either the checkpoints or the recalculations increase. With only 10 checkpoints, 5 to 7 recalculations should be sufficient for the majority of today's unsteady flow simulations. With 100 checkpoints, only one or two recalculations are needed to achieve the same number of time steps. With this many

	$\tau = 1$	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 10$
10 checkpoints	66	286	1001	3003	1.85×10^5
25 checkpoints	351	3276	23751	1.43×10^5	1.84×10^8
50 checkpoints	1326	23426	3.16×10^5	3.48×10^6	7.54×10^{10}
100 checkpoints	5151	1.77×10^5	4.60×10^6	9.66×10^7	4.69×10^{13}

TABLE 1. Maximum number of time steps for a fixed number of checkpoints and repetition number τ
.

checkpoints, our algorithm requires only three recalculations for 4.6×10^6 time steps, much more than current calculations use.

To end this section, we use a numerical experiment to demonstrate that our algorithm is not only theoretically advantageous, but also highly efficient in practice. In this experiment, the original system is the inviscid Burgers' equation,

$$u_t + \frac{1}{2}\left(u^2\right)_x = 0, \quad x \in [0,1], \quad t \in [0,1];$$
$$u|_{t=0} = \sin 2\pi x, \quad u|_{x=0,1} = 0.$$

We discretize this partial differential equation with a first-order up-winding finite-volume scheme with 250 mesh volumes, and use forward Euler time integration with a fixed CFL number $|u_{\max}|\frac{\Delta t}{\Delta x}$. As the time integration proceeds, a shock wave forms at $x = 0.5$. The shock wave dissipates the solution, decreases the maximum wavespeed $|u|_{\max}$, and increases the size of each time step due to the fixed CFL number. As a result, the number of time steps needed to proceed to $t = 1$ is not known a priori. To vary the number of time steps, we chose five different CFL numbers ranging from 1.0 to 0.002. The discrete adjoint equation of the original system is solved with initial and boundary conditions,

$$\phi|_{t=1} = \sin 2\pi x, \quad \phi|_{x=0,1} = 0.$$

Four different numbers of checkpoints, $\delta = 10, 25, 50$ and 100, were specified. For each δ, we ran five calculations with different CFL numbers. We recorded the ratio of computation time between the backward sweep of solving adjoint system and the forward sweep of solving the original system. Because the computation time of the forward sweep is the cost of solving the original system alone, this ratio reflects the additional cost of solving the adjoint equation. We compare the ratio with its theoretical bound defined by Eqs. (5.2) and (5.1) and by assuming that solving an adjoint step costs the same amount of computation time as a forward step.

Figure 5 plots this experimental time ratio with the theoretical bounds. The four subplots correspond to the different number of checkpoints used. Each subplot shows the resulting number of time steps and ratio of computation time for five different CFL numbers. As can be seen, most of the experimental time ratios are within or close to the theoretical bounds, indicating that our algorithm works in practice as efficiently as theoretically proven. In this experiment, the computational cost of calculating a linear adjoint step may be smaller than solving a non-linear Burgers' step, which can explain why some points lie below the theoretical lower bound.

6. Conclusion and discussion

We propose a checkpointing adjoint solver, including an algorithm for dynamically allocating checkpoints during the initial calculation of the original system and subsequent recalculations. Its three main advantages over previous algorithms are: the number of time steps does not need to be known beforehand; the number of recalculations is minimized; an arbitrary number of time steps can be integrated. For an original system with no more than $\binom{\delta+\tau}{\tau}$ time steps, each time step is calculated at most τ times, as has been proven optimal in the previous literature on checkpointing schemes Griewank (1992). Despite the lengthy proof of this optimality, the algorithm itself is conceptually simple to implement and has widespread applications in scientific and engineering simulations of complex systems, where adaptive time-stepping is often desirable, if not necessary.

Although this paper is biased towards solving adjoint equations of time dependent differential equations, a more compelling application of our algorithm is in reverse mode automatic differentiation. Most scientific computation code contains "if" and "while" statements, making their length of execution uncertain a priori. Therefore, our dynamic checkpointing algorithm can be more suitable than static optimal checkpointing algorithms in these cases.

Although we proved that our dynamic checkpointing algorithm has the optimal repetition number, it is not always optimal in terms of the total number of time step recalculations. When the number of time step is between $\binom{\delta+2}{2}$ and $\binom{\delta+3}{3}$, the improved online checkpointing scheme Stumm & Walther (2008) may outperform our algorithm. Therefore, our dynamic checkpointing algorithm can still be improved in terms of the total number of recalculations by placing and replacing low level checkpoints in a more planned manner. One may also combine our algorithm with the improved **online checkpointing** scheme by using **online checkpointing** until the number of time steps reaches $\binom{\delta+3}{3}$, and switch to our algorithm to proceed.

Our dynamic checkpointing algorithm aims only to optimize the repetition number and reduce the number of time step calculations, ignoring the computational cost of writing and reading checkpoints. In many applications, such as solving incompressible Navier-Stokes equations where each time step involves solving a full Poisson equation, the cost of reading and writing checkpoints is negligible because calculating each time step takes much more computation time. In other cases, especially when the checkpoints are written in and read from a hard disk instead of being kept in RAM, the I/O time associated with checkpoint writing and reading cannot be ignored. In such cases, our dynamic checkpointing algorithm may not be the best choice, since it requires significantly more checkpoint writing than **revolve** does.

An implicit assumption of our algorithm is the uniform cost of calculating each time step of the original system. Although this is true for the majority of simple partial differential equations, it is not true for some multi-physics simulations. Moreover, this assumption is false in some automatic differentiation applications. While extension of this algorithm to account for the non-uniform cost of each time step should not be very difficult, maintaining provable performance bound in the extension is subject to further investigation.

Another assumption on which we base our algorithm is that calculating the adjoint solution at time step i from time step $i+1$ only requires the solution to the original system at time step i. In practice, especially if advanced time integration methods are used in solving the original equation, solving the discrete adjoint equation at time step i may require more than one time step of the original system. Future research plans includes investigating ways to adjust our algorithm for this case.

Acknowledgments

This work was funded by the United States Department of Energy's PSAAP at Stanford University.

REFERENCES

Bewley, T., Moin, P. & Temam, R. 2001 DNS-based predictive control of turbulence: an optimal target for feedback algorithms. *J. Fluid Mech.* **447**, 179–225.

Charpentier, I. 2000 Checkpointing schemes for adjoint codes: Application to the meteorological model meso-nh. *SIAM J. Sci. Comput.* **22** (6), 2135–2151.

Giles, M. B. & Pierce, N. 2002 *Adjoint Error Correction for Integral Outputs*, *Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics* . Heidelberg: Springer-Verlag. pp. 47–96.

Giles, M. B. & Suli, E. 2002 Adjoint methods for pdes: a posteriori error analysis and postprocessing by duality. *Acta Numer.* **11**, 145–236.

Griewank, A. 1992 Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software* **1**, 35–54.

Griewank, A. & Walther, A. 2000 Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.* **26** (1), 19–45.

Heuveline, V. & Walther, A. 2006 Online checkpointing for parallel adjoint computation in pdes: Application to goal oriented adaptivity and flow control. In *Proceedings of Euro-Par 2006 Parallel Processing* (ed. W. Nagel. et al.), pp. 689–699.

Hinze, M. & Sternberg, J. 2005 A-revolve: An adaptive memory- and run-time-reduced procedure for calculating adjoints; with an application to the instationary navier-stokes system. *Optimization Methods and Software* **20**, 645–663.

Jameson, A. 1988 Aerodynamic design via control theory. *J. of Scientific Computing* **3**, 233–260.

Kowarz, A. & Walther, A. 2006 Optimal checkpointing for time-stepping procedures. In *Proceedings of ICCS 2006, LNCS 3994* (ed. V. Alexandrov. et al.), pp. 541–549.

Stumm, P. & Walther, A. 2008 Towards the economical computation of adjoints in pdes using optimal online checkpointing. Deutsche Forschungsgemeinschaft Schwerpunktprogramm 1253, Tech. Rep. DFG-SPP 1253-15-04.

Wang, Q., Gleich, D., Saberi, A., Etemadi, N. & Moin, P. 2008 A Monte Carlo method for solving unsteady adjoint equations. *Journal of Computational Physics* **227** (12), 6184–6205.

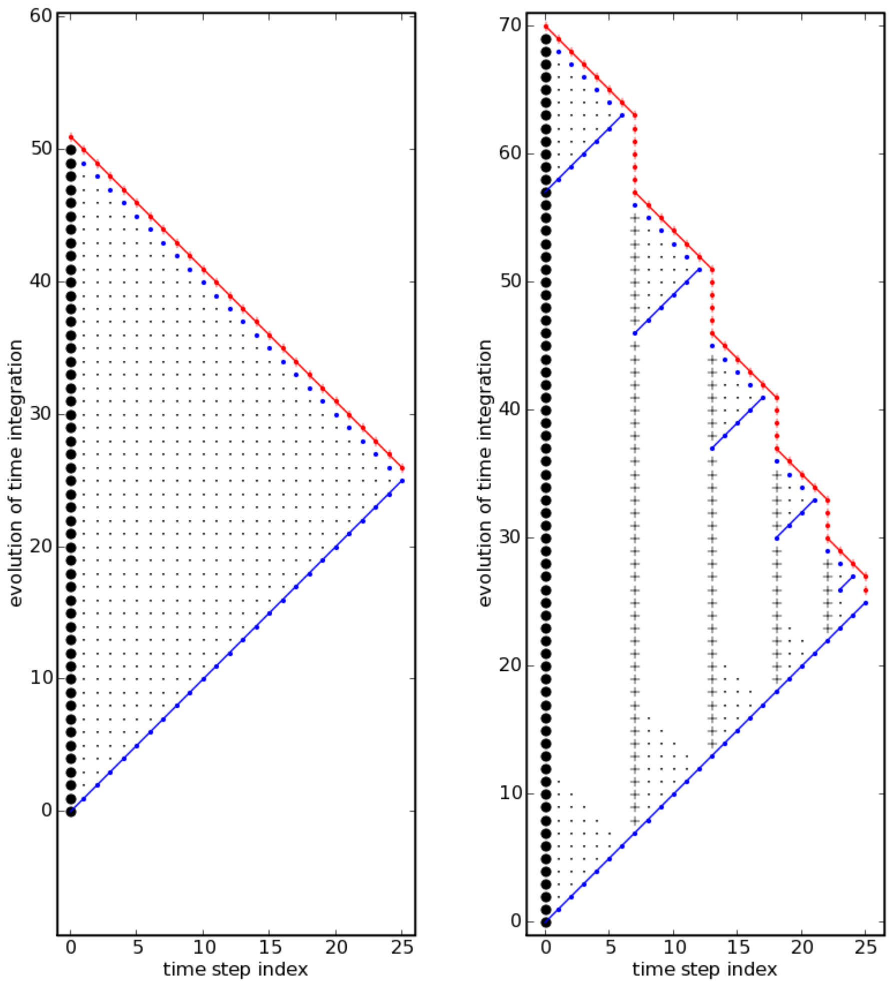

FIGURE 2. Distribution of checkpoints during the process of Algorithm 2. The left plot is for $\delta \geq 25$, the right plot is for $\delta = 6$. Each horizontal cross-section on the plot represents a snapshot of the time integration history, from the beginning of the forward sweep to the end of the adjoint sweep, indicated by the y-axis. Different symbols represent different levels of checkpoints: Circles are level ∞ checkpoint at time step 0. Thin dots, "+", ''X" and star symbols corresponds to level 0, 1, 2 and 3 checkpoints, respectively. The round, thick dots indicate the time step index of the current original solution, which is also the position of the placeholder checkpoint; the lines connecting these round dots indicate where and when the original equation is solved. The thick dots with a small vertical line indicate the time step index of the current adjoint solution, while the lines connecting them indicate where and when the adjoint equation is solved.

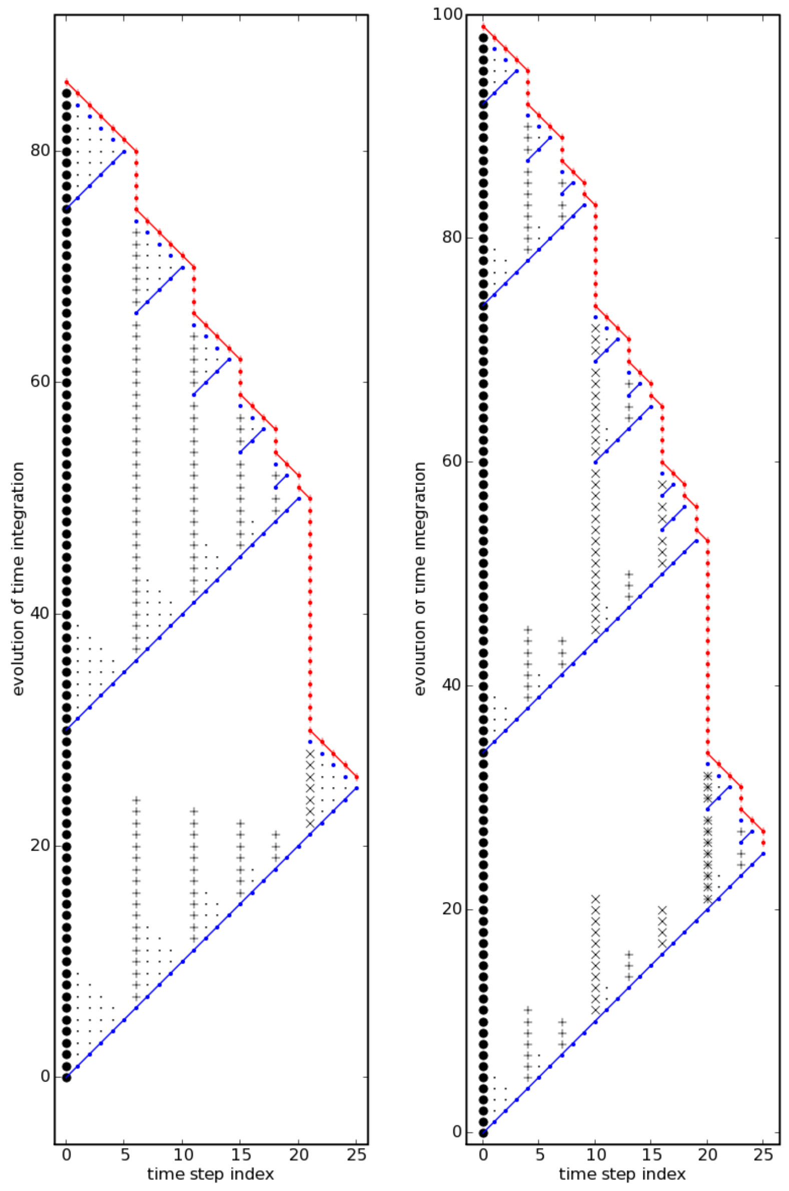

FIGURE 3. Distribution of checkpoints during Algorithm 2. $\delta = 5$ and 3 for left and right plot, respectively.

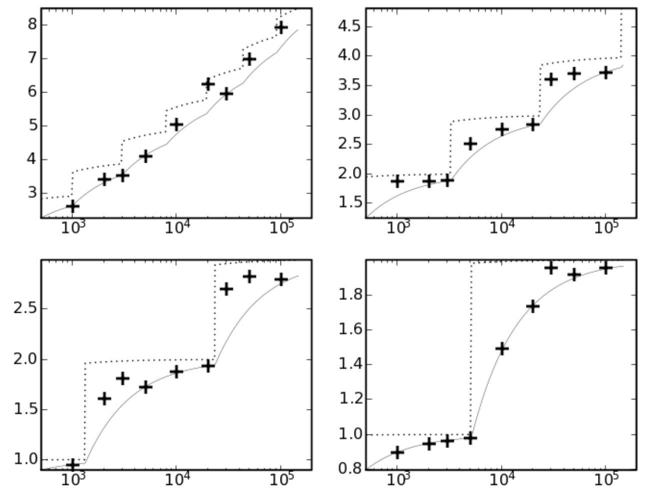
FIGURE 4. The x-axis is the number of time steps, and the y-axis is the average number of recalculations for each time step, defined as the total number of recalculations divided by the number of time steps. Plus signs are the actual average number of recalculations of the dynamic checkpointing scheme; the solid line and the dotted line are the lower and upper bound defined by Eqs. (5.1) and (5.2). The upperleft, upperright, lowerleft and lowerright plots correspond to 10, 25, 50 and 100 allowed checkpoints, respectively.

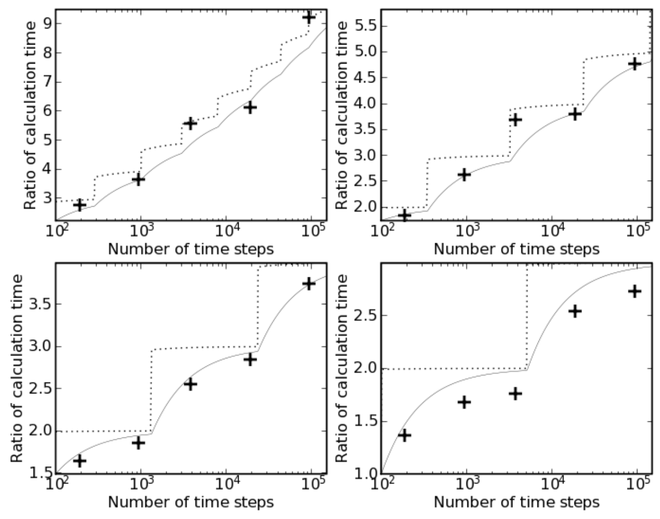

FIGURE 5. Comparison of theoretical bounds (dotted and solid lines) and experimental performance (plus markers) of our dynamic checkpointing adjoint solver. The top-left, top-right, bottom-left and bottom-right plots correspond to $\delta = 10, 25, 50$ and 100, respectively.