WAVEFORM RELAXATION WITH ADAPTIVE PIPELINING (WRAP)[*]

FELIX KWOK [†] AND BENJAMIN W. ONG [‡]

Abstract. Schwarz waveform relaxation (SWR) methods have been developed to solve a wide range of diffusion-dominated and reaction-dominated equations. The appeal of these methods stem primarily from their ability to use non-conforming space-time discretizations; SWR are consequently well-adapted for coupling models with highly varying spatial and time scales. The efficacy of SWR methods are questionable however, since in each iteration, one propagates an error across the entire time interval. In this manuscript, we introduce an adaptive pipeline approach wherein one subdivides the computational domain into space-time blocks, and adaptively selects the waveform iterates which should be updated given a fixed number of computational workers. Our method is complementary to existing space and time parallel methods, and can be used to obtain additional speedup when the saturation point is reached for other types of parallelism. We analyze these waveform relaxation with adaptive pipelining (WRAP) methods to show convergence and the theoretical speedup that can be expected. Numerical experiments on solutions to the linear heat equation, the advection–diffusion equation, and a reaction–diffusion equation illustrate features and efficacy of WRAP methods for various transmission conditions.

Key words. Waveform Relaxation; Domain Decomposition; Adaptivity; Parallel Computing

AMS subject classifications. 65Y05, 65M20

1. Introduction.

The parallel numerical solution of time-dependent PDEs has long been the focus of the high performance computing community. The classical approach for leveraging high performance computing clusters is to apply a semi-discretization in time to the time-dependent PDE, and then apply domain decomposition (DD) in space, for which sophisticated and highly efficient methods exist [18]. For highly refined models however, accuracy or stability constraints often limit the size of the time step. The time stepping process, because of its sequential nature, consequently becomes the bottleneck. Hence, parallelization in the time direction has become an increasingly pressing issue, as attested to by the annual conference series in time-parallelization methods (sixth edition as of 2017, see http://parallelintime.org).

One approach for parallelization in time arises from a different way of using domain decomposition, the so-called waveform relaxation (WR) approach, see [6, 8, 1, 7, 10] and references therein. The WR idea is to decompose first in space to obtain a collection of (coupled) space-time subproblems, then iterate while exchanging interface information over the whole time window. In fact, one can formally create waveform relaxation variants out of any stationary iterative method based on DD. For example, the Neumann-Neumann and Dirichlet-Neumann DD methods can be adapted into a WR method [13, 15]. WR formulations provide flexibility for discretizing space and time, especially for problems in which the dynamics vary greatly across subdomains; see [10] for an application on ocean–atmospheric coupling. On the other hand, when the dynamics are uniform and DD is used purely for parallelization purposes, the convergence of WR methods is typically slower than their elliptic counterparts and deteriorates as the time window length T increases [8, 13]. Despite this apparent drawback, WR exposes additional opportunities for parallelization, particularly

[*]Submitted to the editors DATE.

[†]Dept. of Mathematics, Hong Kong Baptist University, Kowloon Tong, Hong Kong (felix_kwok@hkbu.edu.hk)

[‡] Dept. of Mathematical Sciences, Michigan Technological University, Houghton, MI, 49931, (ongbw@mtu.edu)

in the time direction. In [16], we presented the technique known as pipelining, in which different waveform iterations of the Schwarz WR method can be made to run simultaneously on different time steps, without affecting the mathematical properties of the algorithm. Pipeline parallelism is also possible for Neumann-Neumann and Dirichlet-Neumann WR relaxation methods [17]. In [5], the authors show that this can lead to a significant reduction in wall-clock time relative to a purely spatial DD implementation for the same total number of processors.

Another drawback of the basic WR method is the issue of *oversolving* in the initial time steps. Consider for example an initial value problem (P), posed for $t \in [0, T]$ and discretized using a uniform time step $\Delta t = T/N$. This contains as a subproblem the same PDE, but posed on the shorter time interval $t \in [0, T']$ with $T' = M\Delta t$, where $M < N$. Denoting this subproblem by (P$'$), we observe that any WR method for the problem (P) must require at least as many iterations to converge than the same WR method for (P$'$), at least if the stopping criterion is in terms of an L^p norm. This is because the iterates for (P$'$) are simply the restrictions of the iterates for (P) over a smaller time window, so convergence for (P) automatically implies convergence for (P$'$), but usually not the other way around. In practice, this means the error in the initial time steps is often several orders of magnitude smaller than the error at the final time, so the method is essentially using valuable computational cycles to oversolve the initial time steps relative to the overall tolerance.

In this paper, we address the oversolving problem by presenting a modified version of the pipelining algorithm in [16]; we call this new method Waveform Relaxation with Adaptive Pipelining (WRAP), because the time window on which the PDE is actively being integrated changes over the duration of the computation. Initially, the method uses a small time window, whose size is determined by the number of available processors. Once a solution in this time window is solved to sufficient accuracy, we accept the solution and stop iterating; instead, we expand the time horizon and reallocate the processor to solve for a solution at a later time window. We keep doing this until the final time horizon coincides with the original interval $[0, T]$. We describe this method in more detail in Section 2. Note that this method is mathematically different from the original WR method, because not every time step is iterated the same number of times starting from the same initial and interface conditions. Thus, to analyze the convergence of this method, we present a theoretical model that applies both to the classical Schwarz WR method and to the optimized SWR method with Robin conditions. This is done in Section 3, where we also prove an estimate on the theoretical speedup ratio as a function of the number of available processors P. We will see that the average number of iterations required per time step depends on P, but is independent of the time window size, unlike the original WR method. Finally, in Section 4 we present numerical results for a variety of diffusive problems and DD methods. The results confirm our theoretical analysis and show that it is possible for a WRAP method to obtain a speedup of at least 5–6 over a purely spatial DD method with sequential time-stepping.

2. Algorithms.

We start by considering an equivalent formulation of WR algorithms when the time horizon $[0, T]$ is subdivided into shorter intervals. Suppose that the space–time domain, $\Omega \times [0, T]$, is partitioned into space–time subdomains,

$$\{\Omega_1, \Omega_2, \dots, \Omega_J\} \otimes \{I_1, I_2, \dots, I_M\},$$

where the spatial partitioning $\{\Omega_1, \Omega_2, \dots, \Omega_J\}$ can be overlapping or non-overlapping, with the interfaces denoted by $\Gamma_j := \partial\Omega_j \setminus \partial\Omega$, and the temporal partitioning is

$I_m = [T_{m-1}, T_m], m = 1, \dots, M$. Let $u_{j,m}^{[k]}(x,t)$ denote the kth waveform iterate in $\Omega_j \times I_m$. Additionally, for ease of notation later, we denote the (spatially) distributed solution as $u_m^{[k]}(x,t)$, where

$$u_m^{[k]}(x,t) = \{u_{j,m}^{[k]}(x,t)\}_{j=1}^{J}.$$

Let `integrate` denote a subroutine that computes a numerical approximation to the spatially distributed solution $u_m^{[k]}(x,t)$. Specifically, the routine

$$[g_m^{[k]}, h_m^{[k]}] = \texttt{integrate}(I_m, f, g_{m-1}^{[k]}, h_m^{[k-1]}),$$

takes as its input:

- the interval of integration, $I_m = [T_{m-1}, T_m]$;
- boundary conditions for the PDE, f, on $\partial\Omega$;
- the (distributed) solution at the start of the time interval, $g_{m-1}^{[k]} = u_m^{[k]}(x, T_{m-1})$;
- the (time-dependent) coupling conditions, $h_m^{[k-1]}$;

and returns as its output:

- the (distributed) solution at the end of the time interval, $g_m^{[k]} = u_m^{[k]}(x, T_m)$;
- the updated (time-dependent) coupling conditions, $h_m^{[k]}$.

For example, in a classical Schwarz Waveform Relaxation (SWR) implementation, the coupling conditions, $h_m^{[k]} = \{h_{j,m}^{[k]}\}_{j=1}^{J}$ would be the set of Dirichlet interface conditions required to solve the PDE on $\{\Omega_j\} \times I_m$, i.e., $h_{j,m}^{[k]} = u_{j,m}^{[k]}|_{\Gamma_j \times I_m}$. The computation then proceeds as follows.

```
for m = 1:M
  specify $h_m^{[0]}(t)$ % guess initial coupling conditions
end
for k = 1:K
  Set $g_0^{[k]} = u_0(x)$ % initial condition
  for m = 1:M
     [$g_m^{[k]}, h_m^{[k]}$] = integrate($I_m, f, g_{m-1}^{[k]}, h_m^{[k-1]}$)
  end
end
```

Note that we have split the integration over $[0,T]$ into a sequence of shorter integration steps over I_m, $m = 1, \dots, M$. Pipeline parallelism is now possible [16], because multiple tasks (i.e., multiple `integrate` routine calls) can be launched once the `integrate` routine returns $g_m^{[k]}$ and $h_m^{[k]}$. For example, the completion of

$$[g_1^{[1]}, h_1^{[1]}] = \texttt{integrate}(I_1, f, g_0^{[1]}, h_1^{[0]}),$$

allows the spawning of two additional calls,

$$[g_2^{[1]}, h_2^{[1]}] = \texttt{integrate}(I_2, f, g_1^{[1]}, h_2^{[0]}),$$
$$[g_1^{[2]}, h_1^{[2]}] = \texttt{integrate}(I_1, f, g_0^{[2]}, h_1^{[1]}).$$

In general, a dependency graph can be generated to identify tasks that can be run in parallel. In Figure 1, the output of each `integrate` routine is shown in the purple boxes. Note that tasks belonging to the same column can all be run concurrently, provided enough processors are available. More precisely, if JP processors are available and each task requires J processors to complete (because we have J spatial subdomains), then the tasks belonging to the first P rows can be executed in parallel. Once all these tasks are completed, then the next group of P rows can be executed, in a pipeline fashion. This pipeline works best if the execution of each task (i.e. purple box) takes roughly the same wall time.

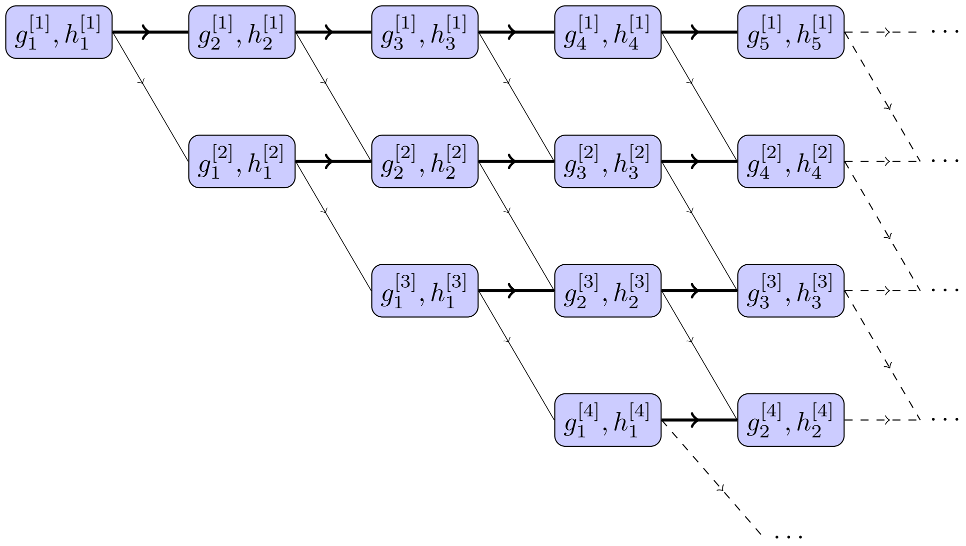

Fig. 1: Dependency graph for classical Schwarz WR or Neumman–Neumann WR. The variables within the purple boxes denote the outputs of the `integrate` routine. The width of the arrows reflect the size of information that needs to be passed to the newly spawned tasks. If the execution of each task (purple box) takes roughly the same wall time, each column of tasks can be simultaneously computed if sufficient processors are available.

One way to save computation is to *prune* the dependency graph and remove tasks that are either unnecessary or ineffective in reducing the error in the solution. To accomplish this pruning, we propose an adaptive framework that utilizes two key ideas. Firstly, suppose for example, that the error associated with computing $u_1^{[2]}$ satisfies some user prescribed tolerance. Then, one can stop iterating on time interval I_1 and use the converged solution at the end of this interval to spawn any future task involving interval I_2, thereby reducing the total number of tasks within each column. An example of this modified dependency graph is shown in Figure 2. More generally, one can utilize the `integrate` routine to return $(g_m^{[k]}, h_m^{[k]})$ given $(g_{m-1}^{[j]}, h_m^{[k-1]})$, where $j \leq k$ i.e.,

$$[g_m^{[k]}, h_m^{[k]}] = \texttt{integrate}(I_m, f, g_{m-1}^{[j]}, h_m^{[k-1]}), \text{ where } j \leq k.$$

Secondly, if we suspect that a certain $g_{m-1}^{[k]}$ is so inaccurate that further iteration in $I_m, I_{m+1}, \ldots$ would not lead to a significant reduction in error, then it is advantageous to wait until a more accurate solution $g_{m-1}^{[j]}$, $j > k$ becomes available, and use that as the initial conditions for further integration. For example, $g_1^{[2]}$ can be used instead of $g_1^{[1]}$ when solving for $g_2^{[1]}$, as shown in Figure 3. In other words, we have shifted everything to the right of $(g_1^{[k]}, h_1^{[k]})$ downward and to the right and changed the dependencies, as shown in red in Figure 3. More generally, one can utilize the

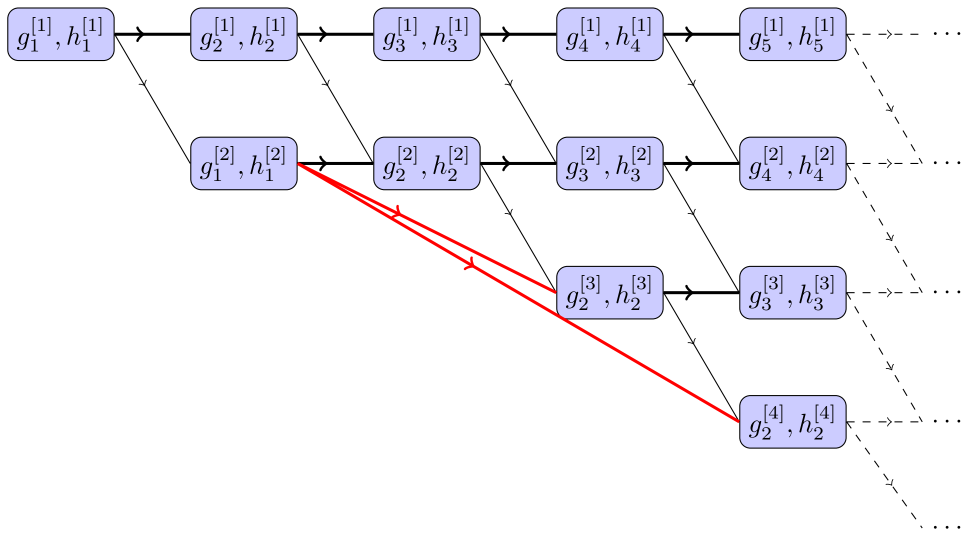

Fig. 2: Dependency graph for classical Schwarz WR or Neumman–Neumann WR if the error associated with $u_1^{[2]}$ satisfies some user prescribed tolerance. The new dependencies are shown in red.

`integrate` routine to return $(g_m^{[k]}, h_m^{[k]})$ given $(g_{m-1}^{[j]}, h_m^{[k-1]})$, where $j \geq k$, i.e.,

$$[g_m^{[k]}, h_m^{[k]}] = \texttt{integrate}(I_m, f, g_{m-1}^{[j]}, h_m^{[k-1]}), \text{ where } j \geq k.$$

This transformation changes the mathematical properties of the WR algorithm, and new convergence estimates must be proved, which we will do in Section 3.

We are now ready to present the Waveform Relaxation with Adaptive Pipelining (WRAP) method. To begin, let `tasklist` be a list[1] of tuples (k, m), corresponding to the solution values $(g_m^{[k]}, h_m^{[k]})$ that can presently computed because the dependencies are satisfied. For example, consider the dependency graph for classical Schwarz WR, Figure 1. The initial `tasklist` consists of the entry $(1, 1)$, since only one iteration in interval I_1 can be computed given the initial condition at time t_0. After $(g_1^{[1]}, h_1^{[1]})$ is computed, the two tasks

$$\texttt{[}g_2^{[1]}, h_2^{[1]}\texttt{] = integrate(}I_2, f, g_1^{[1]}, h_2^{[0]}\texttt{),}$$
$$\texttt{[}g_1^{[2]}, h_1^{[2]}\texttt{] = integrate(}I_1, f, g_0^{[2]}, h_1^{[1]}\texttt{).}$$

can be spawned if they are necessary, i.e., if the interval I_2 exists, and that $(g_1^{[1]}, h_1^{[1]})$ has not already converged to sufficient accuracy. In this case, we remove the entry $(1, 1)$ from `tasklist`, and add to it the two new entries $(1, 2)$ and $(2, 1)$. In general, the following algorithm can be used to update `tasklist` adaptively:

```
# Suppose task = tasklist[i] has been completed
# the task list can then be updated as follows.
if (task.k == 1) && (task.m < Nt)
  tasklist.append(task.k,task.m+1)
end
```

[1]which will be implemented as a hash map for efficiency.

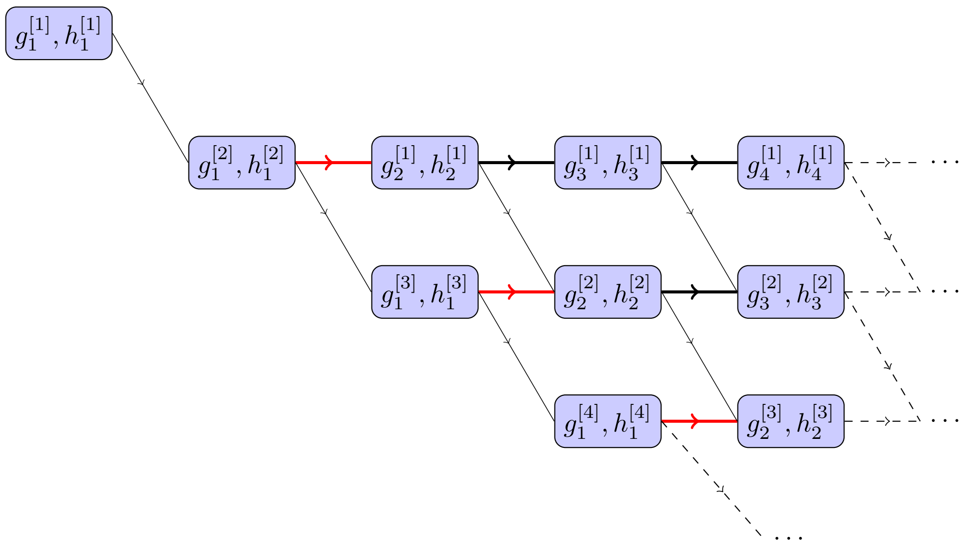

Fig. 3: Dependency graph for classical Schwarz WR or Neumman–Neumann WR if the solution in I_1 is iterated twice before the pipeline computations are initiated. The new dependencies are shown in red.

```
if error_estimate(g(task.k,task.m),h(task.k,task.m)) > TOL
  tasklist.append(task.k+1,task.m)
end
tasklist.remove(task)
```

Now suppose *ntasks* $< \infty$ is the maximum number of tasks that can be executed simultaneously by our machine[2], and there are more than *ntasks* elements in `tasklist`. To choose which tasks in `tasklist` to execute, we use the heuristic that *more accurate initial conditions always leads to faster error reduction*: we select from the list *ntasks* elements with the smallest m, i.e., corresponding to the earliest time intervals. Thus, tasks with larger m will be *delayed* until the solution at earlier time intervals has converged.

There are two limiting cases of interest. If *ntasks* $= 1$ and time window $I_m = [T_{m-1}, T_m]$ consists of a single time step, Δt, then the WRAP framework simplifies to a classical domain decomposition method, where $g_m^{[k]}$ is iterated to convergence before computing $g_{m+1}^{[1]}$. The second limiting case is when *all* the tasks in `tasklist` are simultaneously computed before a new task list is generated based on the recently completed tasks. We shall denote this as *ntasks* $= \infty$, with the understanding that the maximum number of simultaneous tasks that can be computed is limited by the number of time steps used in the discretization. In this case, WRAP produces iterates that are *the same to those of classical WR* up to the preset tolerance `TOL`, since the dependency graph is identical.

3. Convergence Analysis.

To understand the convergence properties of the WRAP method, we first introduce a computational model that is valid for both clas-

[2]If a hybrid MPI-OpenMP framework is used to implement the adaptive WR methods, *ntasks* can be initialized to the number of processing cores available on each socket.

sical and adaptive WR methods. Consider again the dependency graph for classical WR, shown in Figure 1. Let $G(m,k)$ and $H(m,k)$ be some error measures related to the iterates $g_m^{[k]}$ and $h_m^{[k]}$, which must be suitably defined according to the problem and method chosen. If the error measures satisfy the coupled recurrence

$$\text{(1)}\qquad \begin{cases} G(m,k) \le \alpha G(m-1,k) + H(m,k), \\ H(m,k+1) \le G(m-1,k) + \beta H(m,k), \end{cases}$$

then the method converges if $G(m,k)$ and $H(m,k)$ tend to zero as $k \to \infty$ for all $1 \le m \le M$. In the next two theorems, we show that for the linear heat equation, this computational model is valid for both classical SWR and optimized SWR with Robin transmission conditions, provided we choose the error measures correctly. Since the heat equation is linear, it suffices to consider the homogeneous problem with an arbitrary initial guess along the artificial interfaces.

Theorem 3.1. *Consider the classical Schwarz WR applied to the homogeneous heat equation,*

$$\partial_t u_j^{[k]} - \Delta u_j^{[k]} = 0, \quad u_j^{[k]}\Big|_{t=T_0} = 0,$$

with initial guesses on the artificial interfaces $\partial\Omega_j \setminus \partial\Omega$, $j = 1,\dots,J$. Denote the time sub-intervals by $I_1,\dots,I_M$, where $I_m = [T_{m-1}, T_m]$, $m = 1,2,\dots,M$. If

$$G(m,k) = \max_j \|u_j^{[k]}(\cdot,T_m)\|_{L^\infty(\Omega_j)},$$
$$H(m,k) = \max_j \left(\sup_{t\in I_m} \|u_j^{[k]}(\cdot,t)\|_{L^\infty(\partial\Omega_j)} \right),$$

then $\{G(m,k)\}_{k,m\ge1}$ and $\{H(m,k)\}_{k,m\ge1}$ satisfy the recurrence (1) for some $0 < \alpha < 1$ and $0 < \beta < 1$.

Proof. We consider the solution at the kth iteration inside the patch $(x,t) \in \Omega_j \times I_m$. The solution satisfies $\partial_t u_j^{[k]} - \Delta u_j^{[k]} = 0$ with initial and boundary conditions

$$\|u_j^{[k]}(\cdot,T_{m-1})\|_{L^\infty(\Omega_j)} \le G(m,k), \quad \|u_j^{[k]}(\cdot,t)\|_{L^\infty(\partial\Omega_j)} \le H(m,k) \quad \forall t \in I_m.$$

Since the PDE is linear, it suffices to estimate $G(m,k)$ by first setting $H(m,k) = 0$, then estimating $G(m,k)$ by setting $G(m-1,k) = 0$, and finally adding the two estimates together. The same procedure can be applied to estimate $H(m,k+1)$. Thus, we first consider the subdomain problem with zero interface conditions

$$\partial_t u_j^{[k]} - \Delta u_j^{[k]} = 0, \quad u_j^{[k]}\Big|_{t=T_{m-1}} = 1, \quad u_j^{[k]}\Big|_{\partial\Omega_j} = 0.$$

By the maximum principle, we have

$$0 \le u_j^{[k]}(x,t) \le \alpha_j(t) < 1, \quad \forall (x,t) \in \Omega_j \times I_m$$

In anticipation of showing convergence of the solution at the final time T_m, $u_j^{[k]}(x,T_m)$, we define

$$\alpha_j := \alpha_j(T_m), \quad \alpha := \max_j \alpha_j < 1.$$

Note that although α depends on the length of the time interval I_m and on the diameter of the subdomains, such an α always exists.

Next, if we consider the subdomain problem with zero initial conditions,

$$\partial_t u_j^{[k]} - \Delta u_j^{[k]} = 0, \quad u_j^{[k]}\Big|_{t=T_{m-1}} = 0, \quad u_j^{[k]}\Big|_{\partial\Omega_j} = 1,$$

we get trivially that

$$0 \le u_j^{[k]}(x,t) \le 1, \quad \forall (x,t) \in \Omega_j \times I_m.$$

However, on a set $\Gamma \subset \Omega_j$ that is at a distance of at least δ away from $\partial\Omega_j$, we in fact have [7, Lemma 3.1]

$$\|u_j^{[k]}(\cdot,t)\|_{L^\infty(\Gamma)} \le \beta < 1,$$

where β depends on the distance δ.

Thus, for the general problem $\partial_t u_j^{[k]} - \Delta u_j^{[k]} = 0$ with

$$|u_j^{[k]}(x,T_{m-1})| \le G(m,k), \quad \forall x \in \Omega_j, \quad |u_j^{[k]}(x,t)| \le H(m,k), \quad \forall (x,t) \in \partial\Omega_j \times I_m,$$

we have $|u_j^{[k]}(\cdot,t)| \le \alpha_j(t) G(m-1,k) + H(m,k)$, which leads to

$$|u_j^{[k]}(\cdot,T_m)| \le \alpha G(m-1,k) + H(m,k). \tag{2}$$

However, the Dirichlet values transmitted to the neighbours of Ω_j lie in a set Γ at least δ away from $\partial\Omega_j$, so we have the estimate

□

$$\|u_j^{[k]}(\cdot,t)\|_{L^\infty(\Gamma)} \le G(m-1,k) + \beta H(m,k), \quad \forall t \in I_m.$$

For OSWR, we have the following result if we use P^1 finite elements for the spatial discretization and the Theta method [12] with $\frac{1}{2} \le \theta \le 1$ for discretization in time[3]. For simplicity, we assume that each time block consists of a single time step, and that the spatial decomposition is non-overlapping with no cross points. We denote by $\Gamma_{ij} = \partial\Omega_i \cap \partial\Omega_j$ the interface bewteen Ω_i and Ω_j.

Theorem 3.2. *Consider the optimized Schwarz WR applied to the homogeneous heat equation discretized with the Theta method in time and P^1 finite elements in space over a shape regular, quasi-uniform triangulation $\mathcal{T}_h$. More precisely, let $u_{jm}^{[k]} \approx u_j^{[k]}(\cdot,T_m)$ satisfy*

$$\int_{\Omega_j} v \left(\frac{u_{jm}^{[k]} - u_{j,m-1}^{[k]}}{\Delta t_m} \right) + \int_{\Omega_j} \nabla \bar{w}_{jm}^{[k]} \cdot \nabla v + \int_{\partial\Omega_j \setminus \partial\Omega} p \bar{w}_{jm}^{[k]} v = \int_{\partial\Omega_j \setminus \partial\Omega} R_{jm}^{[k]} v, \quad \forall v \in V_j^h, \tag{3}$$

$$R_{jm}^{[k+1]}|_{\Gamma_{ij}} = (2p\bar{w}_{im}^{[k]} - R_{im}^{[k]})|_{\Gamma_{ij}}, \tag{4}$$

where $\Delta t_m = T_m - T_{m-1}$, $\bar{w}_{jm}^{[k]} = (1-\theta)u_{j,m-1}^{[k]} + \theta u_{jm}^{[k]}$ with $\frac{1}{2} \le \theta \le 1$, and the initial Robin traces $R_{jm}^{[1]}$ are posed on the artificial interfaces $\partial\Omega_j \setminus \partial\Omega$, $j = 1, \dots, J$, cf. [3]. If

$$G(m,k) = \left(\frac{1}{2} \sum_j \|u_{jm}^{[k]}\|^2_{L^2(\Omega_j)} \right)^{1/2}, \quad H(m,k) = \left(\Delta t_m \sum_j \|R_{jm}^{[k]}\|^2_{L^2(\partial\Omega_j \setminus \partial\Omega)} \right)^{1/2},$$

[3]This method is also known as the θ scheme in [9], or the generalized trapezoidal rule in [11].

then $\{G(m,k)\}_{k,m\geq 1}$ and $\{H(m,k)\}_{k,m\geq 1}$ satisfy the recurrence (1) for $\alpha = 1$ and some $0 < \beta < 1$, where β depends on the length of the time step size Δt_m.

Proof. Let $v = \bar{w}_{jm}^{[k]}$ in equation (3) and calculate

$$\begin{aligned} (5)\quad & \frac{1}{2\Delta t_m}\int_{\Omega_j}\left[(u_{jm}^{[k]})^2 - (u_{j,m-1}^{[k]})^2 + (2\theta-1)(u_{jm}^{[k]} - u_{j,m-1}^{[k]})^2\right] + \int_{\Omega_j}|\nabla \bar{w}_{jm}^{[k]}|^2 \\ &= \int_{\partial\Omega_j\setminus\partial\Omega}(R_{jm}^{[k]} - p\bar{w}_{jm}^{[k]})\bar{w}_{jm}^{[k]} = \int_{\partial\Omega_j\setminus\partial\Omega}\left[(R_{jm}^{[k]})^2 - (2p\bar{w}_{jm}^{[k]} - R_{jm}^{[k]})^2\right]. \end{aligned}$$

Using the update formula (4) and the fact that $2\theta - 1 \geq 0$, we obtain, after summing over all subdomains Ω_j, that

$$\begin{aligned} &\frac{1}{2}\sum_j \|u_{jm}^{[k]}\|_{L^2(\Omega_j)}^2 + \Delta t_m \sum_j \|R_{jm}^{[k+1]}\|_{L^2(\partial\Omega_j\setminus\partial\Omega)}^2 \\ &\qquad\qquad \leq \frac{1}{2}\sum_j \|u_{j,m-1}^{[k]}\|_{L^2(\Omega_j)}^2 + \Delta t_m \sum_j \|R_j^{[k]}\|_{L^2(\partial\Omega_j\setminus\partial\Omega)}^2. \end{aligned}$$

In other words, we have

$$G(m,k)^2 + H(m,k+1)^2 \leq G(m-1,k)^2 + H(m,k)^2,$$

which immediately implies the recurrence relation (1) with $\alpha = \beta = 1$. To see that β can in fact be chosen to be less than 1, it suffices by linearity to consider the case where $u_{j,m-1}^{[k]} = 0$ for all j and show that $H(m,k+1) \leq \beta H(m,k)$ for some $\beta < 1$.

We proceed by substituting $u_{j,m-1}^{[k]} = 0$ into equation (3), so that $\bar{w}_{jm}^{[k]} = \theta u_{jm}^{[k]}$:

$$(6)\qquad \int_{\Omega_j}\frac{u_{jm}^{[k]}v}{\Delta t_m} + \theta\left(\int_{\Omega_j}\nabla u_{jm}^{[k]}\cdot\nabla v + \int_{\partial\Omega_j\setminus\partial\Omega} p u_{jm}^{[k]} v\right) = \int_{\partial\Omega_j\setminus\partial\Omega} R_{jm}^{[k]} v.$$

By Lemma 4.10 in [18] and Theorem 4.5.11 in [2], there exists a discrete harmonic extension $v \in V_j^h$ of $R_{jm}^{[k]}$, such that $v|_{\partial\Omega_j\setminus\partial\Omega} = R_{jm}^{[k]}$ and

$$\|v\|_{H^1(\Omega_j)} \leq C\|R_{jm}^{[k]}\|_{H^{1/2}(\partial\Omega_j\setminus\partial\Omega)} \leq Ch^{-1/2}\|R_{jm}^{[k]}\|_{L^2(\partial\Omega_j\setminus\partial\Omega)}.$$

Substituting this v into equation (6) and using the Cauchy-Schwarz inequality on the left, we obtain

$$\begin{aligned} \int_{\partial\Omega_j\setminus\partial\Omega}(R_{jm}^{[k]})^2 &\leq \frac{1}{\Delta t_m}\|u_{jm}^{[k]}\|_{L^2(\Omega_j)}\,\|v\|_{L^2(\Omega_j)} + \theta|u_{jm}^{[k]}|_{H^1(\Omega_j)}\,|v|_{H^1(\Omega_j)} \\ &\quad + \theta p\|u_{jm}^{[k]}\|_{L^2(\partial\Omega_j\setminus\partial\Omega)}\,\|R_{jm}^{[k]}\|_{L^2(\partial\Omega_j\setminus\partial\Omega)} \\ &\leq \left(\frac{\theta}{\Delta t_m}\|u_{jm}^{[k]}\|_{L^2(\Omega_j)}^2 + \theta^2|u_{jm}^{[k]}|_{H^1(\Omega_j)}^2\right)^{1/2}\left(\frac{1}{\theta\Delta t_m}\|v\|_{L^2(\Omega_j)}^2 + |v|_{H^1(\Omega_j)}^2\right)^{1/2} \\ &\quad + \theta p\|u_{jm}^{[k]}\|_{L^2(\partial\Omega_j\setminus\partial\Omega)}\,\|R_{jm}^{[k]}\|_{L^2(\partial\Omega_j\setminus\partial\Omega)} \\ &\leq \left(\frac{\theta}{\Delta t_m}\|u_{jm}^{[k]}\|_{L^2(\Omega_j)}^2 + \theta^2|u_{jm}^{[k]}|_{H^1(\Omega_j)}^2\right)^{1/2}\left(\frac{C_1}{\sqrt{\theta h\Delta t_m}} + C_2 p\right)\|R_{jm}^{[k]}\|_{L^2(\partial\Omega_j\setminus\partial\Omega)}. \blacksquare \end{aligned}$$

Dividing both sides by $\|R_{jm}^{[k]}\|_{L^2(\partial\Omega_j\setminus\Omega)}$, we see that

$$\int_{\partial\Omega_j\setminus\partial\Omega}(R_{jm}^{[k]})^2 \le \bar{C}\left(\frac{\theta}{\Delta t_m}\|u_{jm}^{[k]}\|^2_{L^2(\Omega_j)} + \theta^2|u_{jm}^{[k]}|^2_{H^1(\Omega_j)}\right),$$

where $\bar{C} > 1$ depends on Δt_m, h and p. Substituting into equation (5), and keeping in mind the assumption that $u_{j,m-1}^{[k]} = 0$, we deduce that

$$\begin{aligned}\int_{\partial\Omega_j\setminus\partial\Omega}\left[(R_{jm}^{[k]})^2 - (2p\bar{w}_{jm}^{[k]} - R_{jm}^{[k]})^2\right] &= \frac{\theta}{\Delta t_m}\int_{\Omega_j}(u_{jm}^{[k]})^2 + \theta^2\int_{\Omega_j}|\nabla u_{jm}^{[k]}|^2\\ &\ge \bar{C}^{-1}\int_{\partial\Omega_j\setminus\partial\Omega}(R_{jm}^{[k]})^2.\end{aligned}$$

We conclude that

$$\int_{\partial\Omega_j\setminus\partial\Omega}(2p\bar{w}_{jm}^{[k]} - R_{jm}^{[k]})^2 \le (1-\bar{C}^{-1})\int_{\partial\Omega_j\setminus\partial\Omega}(R_{jm}^{[k]})^2,$$

so summing over all j shows that $H(m, k+1) \le \beta H(m, k)$ with $\beta = 1 - \bar{C}^{-1} < 1$, as required. ∎

3.1. The non-adaptive case. We use the above computational model to deduce an error estimate for classical Schwarz WR. The case of optimized SWR can be derived similarly. Note that this is only a linear estimate and is less sharp than the estimate in [7], but the linear estimate is much more amenable to our later analysis, when the dependency graph no longer resembles Figure 1.

Lemma 3.3. *Consider the classical Schwarz WR with*

$$u_j^{[k]}(x, T_0) = 0 \quad \text{and} \quad \|u_j^{[1]}(\cdot, t)\|_{L^\infty(\partial\Omega_j)} \le 1$$

for all j. Let $\xi \ge 1$ and $\eta > \beta > 0$ be constants that satisfy $(\xi - \alpha)(\eta - \beta) = 1$. Then

$$|u_j^{[k]}(x,t)| \le G(m-1,k) + H(m,k) \quad \text{on} \quad \Omega_j \times [T_{m-1}, T_m],$$

where

$$H(m,k) \le \xi^{m-1}\eta^{k-1}, \tag{7}$$

$$G(m,k) \le (\eta-\beta)\xi^m\eta^{k-1}. \tag{8}$$

Proof. Since $\xi \ge 1$ and $H(m, 1) \le 1$ by definition, we see that equation (7) holds for $k = 1$. Moreover, since $(\eta - \beta)\xi = 1 + \alpha(\eta - \beta) > 1$, equation (2) implies

$$G(1,k) \le H(1,k) \le \eta^{k-1} \le (\eta-\beta)\xi\eta^{k-1},$$

which proves equation (8) for $m = 1$. We now prove equations (7) and (8) by induction on m and k using the recurrence (1). Indeed, we have

$$H(m,k+1) \le G(m-1,k) + \beta H(m,k) \le (\eta-\beta)\xi^{m-1}\eta^{k-1} + \beta\xi^{m-1}\eta^{k-1} = \xi^{m-1}\eta^k.$$

Moreover,

$$G(m,k) \le \alpha G(m-1,k) + H(m,k) \le (\alpha(\eta-\beta)+1)\xi^{m-1}\eta^{k-1} = (\eta-\beta)\xi^m\eta^{k-1},$$

since $1 = (\xi - \alpha)(\eta - \beta)$. We have thus proved equations (7) and (8) inductively, as required. ∎

Note that there is some flexibility in choosing ξ and η, as long as the constraint $(\xi-\alpha)(\eta-\beta)=1$ is satisfied. One example is

$$\xi = \frac{1+\alpha}{1-\beta} > 1, \quad \eta = \frac{1+\alpha\beta^2}{1+\alpha\beta} < 1.$$

We see from Lemma 3.3 that $H(m,k)$ converges to zero as $k\to\infty$ for fixed m, but the constant increases with m. One can choose an η arbitrarily close to, but larger than, β, but one must then live with the growth in m that comes from a large ξ.

3.2. Adaptive Case. To analyze the adaptive case, we start by referring to the dependency graph in Figure 3. To facilitate the analysis, it is more convenient to label each task in a row with the same iteration number k; thus, from now on we redefine the iteration number k as in Figure 4. The old and new labels are related by

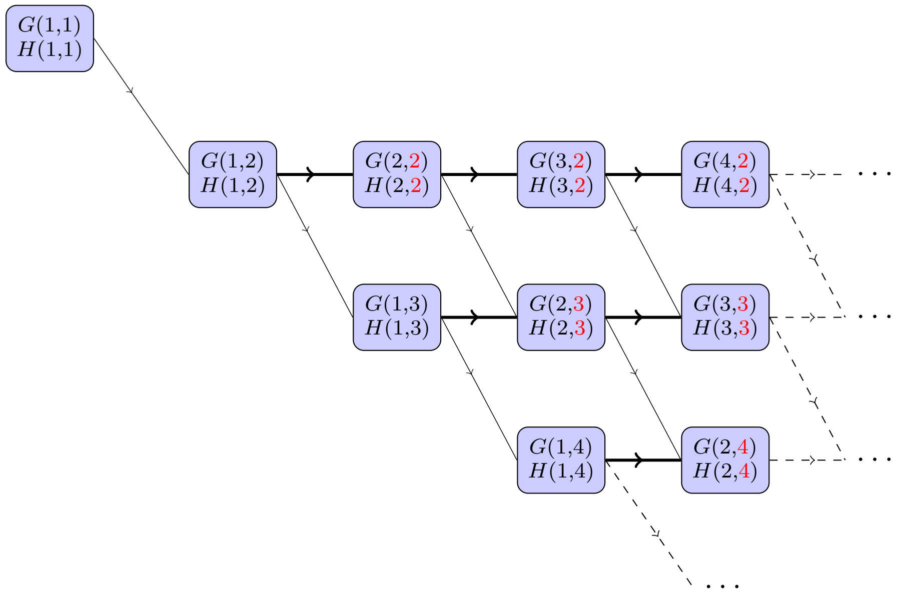

Fig. 4: This figure gives the dependency graph for the same iterative process previously shown in Figure 3, but with new labels for k, and where $\{G(m,k), H(m,k)\}$ in each task denotes the error measures related to the iterates $g_m^{[k]}$ and $h_m^{[k]}$ respectively. The new labels, k, are related to the old labels, $\tilde{k}$, by the relation $k=\tilde{k}-D_m$, where D_m is the delay in starting the method for the mth time interval because processors are not available to complete this task. In this example, the solution in I_1 is iterated twice before the pipeline computations are initiated. Hence, we have $D_2=D_3=D_4=1$. The new labels are shown in red.

D_m, the *delay* in starting the method for the mth time interval because processors are not available to compute the mth interval. This delay does not include the "burn-in" time, i.e., the amount of time waiting for appropriate initial or boundary conditions to begin the computation on the mth time interval. For the adaptive SWR for instance, we have

$$G(m,k+D_m) = \max_j \|g_{j,m}^{[k]}\|_{L^\infty(\Omega_j)}.$$

The delay D_m has the following properties:

- $D_m \leq D_{m+1}$ for all m;
- If the machine can run P tasks simultaneously, then $D_1 = \cdots = D_P = 0$. This is because the first P time intervals always have priority over later times in the task list.

With the new numbering, our computational model becomes

$$G(m,k) \leq \alpha G(m-1,k) + H(m,k), \qquad k > D_m, \tag{9}$$
$$H(m,k+1) \leq G(m-1,k) + \beta H(m,k), \qquad k > D_m, \tag{10}$$
$$H(m,k) \leq 1, \qquad k \leq D_m. \tag{11}$$

The last condition simply indicates that there can be no reduction of error in the interface conditions until the method starts iterating on the interval I_m. To solve equations (9)–(11), we need the following lemma, whose proof is identical to that of Lemma 3.3.

Lemma 3.4. *Let $\xi \geq 1$ and $\eta > \beta > 0$ be constants that satisfy $(\xi-\alpha)(\eta-\beta) = 1$. Let $A_r = A_r(\xi,\eta)$ be any non-negative function of ξ and η such that*

$$\xi^{m-1}\eta^{D_m}\sum_{r=1}^{m} A_r(\xi,\eta) \geq 1. \tag{12}$$

If $G(m,k)$ and $H(m,k)$ satisfy equations (9)–(11) for all $m,k \geq 1$, then

$$H(m,k) \leq \xi^{m-1}\eta^{k-1}\sum_{r=1}^{m} A_r, \quad G(m,k) \leq (\eta-\beta)\xi^{m}\eta^{k-1}\sum_{r=1}^{m} A_r.$$

We are now going to choose the A_r so that condition (12) is satisfied.

Lemma 3.5. *Suppose the hypotheses of Lemma 3.4 hold. For each $m \geq 1$, define*

$$A_m = \xi^{1-m}\eta^{-D_m}\max\left(0, 1 - \textstyle\sum_{r=1}^{m-1} A_r\xi^{m-1}\eta^{D_m}\right).$$

Then

$$\xi^{m-1}\eta^{k-1}\sum_{r=1}^{m} A_r = \max_{1\leq j\leq m} \xi^{m-j}\eta^{k-1-D_j}, \tag{13}$$

so that

$$H(m,k) \leq \max_{1\leq j\leq m} \xi^{m-j}\eta^{k-1-D_j}, \quad G(m,k) \leq (\eta-\beta)\max_{1\leq j\leq m} \xi^{m-j+1}\eta^{k-1-D_j}.$$

Proof. By induction on m. The base case $m = 1$ reads

$$\eta^{k-1}A_1 = \eta^{k-1}\eta^{-D_m} = \max_{1\leq j\leq m} \eta^{k-1-D_j}.$$

Assume inductively that equation (13) holds for m. Then for $m+1$, we have

$$\begin{aligned}
\xi^{m}\eta^{k-1}\sum_{r=1}^{m+1} A_r &= \xi\max_{1\leq j\leq m} \xi^{m-j}\eta^{k-1-D_j} + \xi^{m}\eta^{k-1}A_{m+1}\\
&= \max_{1\leq j\leq m} \xi^{m+1-j}\eta^{k-1-D_j} + \max\left(0, \eta^{k-1-D_{m+1}} - \textstyle\sum_{r=1}^{m} A_r\xi^{m}\eta^{k-1}\right)\\
&= \max_{1\leq j\leq m} \xi^{m+1-j}\eta^{k-1-D_j} + \max\left(0, \eta^{k-D_{m+1}} - \xi\max_{1\leq j\leq m} \xi^{m-j}\eta^{k-1-D_j}\right)
\end{aligned}$$

Thus,

$$\xi^m \eta^{k-1} \sum_{r=1}^{m+1} A_r = \begin{cases} \eta^{k-1-D_{m+1}}, & \text{if } \eta^{k-1-D_{m+1}} \geq \max\limits_{1\leq j\leq m} \xi^{m+1-j}\eta^{k-1-D_j}, \\ \max\limits_{1\leq j\leq m} \xi^{m+1-j}\eta^{k-1-D_j}, & \text{otherwise.} \end{cases}$$

It follows that

$$\xi^m \eta^{k-1} \sum_{r=1}^{m+1} A_r = \max_{1\leq j\leq m+1} \xi^{m+1-j}\eta^{k-1-D_j},$$

which completes the induction. □

3.3. Theoretical Speedup. We are now ready to estimate the theoretical speedup of WRAP when a only a finite number of tasks can be executed simultaneously. Let $P = \mathit{ntasks}$ be this number. Then one cannot start iterating on the time interval I_m until the iteration on I_{m-P} has converged. Define E_m to be the *ending time* for the mth time interval, i.e., the smallest k such that $H(m,k) \leq \epsilon$, where ϵ is some predefined tolerance. Then by definition, we have $E_m = k$, where

$$H(m, k+1) \leq \epsilon \leq H(m,k) \leq \max_{1\leq j\leq m} \xi^{m-j}\eta^{k-1-D_j}.$$

Suppose the maximum on the right hand side of the above equation is achieved for $j = j^*$. Then taking logarithms yields

$$(m - j^*)\log\xi - (E_m - D_{j^*} - 1)|\log\eta| \geq -|\log\epsilon|,$$

or

$$E_m \leq 1 + D_{j^*} + \frac{|\log\epsilon|}{|\log\eta|} + (m - j^*)\frac{\log\xi}{|\log\eta|}. \tag{14}$$

Moreover, since j^* maximizes $\xi^{m-j}\eta^{k-1-D_j}$, we see that

$$(m - j^*)\log\xi - (k - 1 - D_{j^*})|\log\eta| \geq (m - j)\log\xi - (k - 1 - D_j)|\log\eta|,$$

for all $1 \leq j \leq m$. In other words, we have

$$D_{j^*} - j^*\frac{\log\xi}{|\log\eta|} \geq D_j - j\frac{\log\xi}{|\log\eta|}, \quad j = 1, \dots, m.$$

This function will be important later, so let us define

$$F_m := D_m - m(\log\xi/|\log\eta|). \tag{15}$$

We can then rewrite equation (14) as

$$E_m - D_m \leq 1 + \frac{|\log\epsilon|}{|\log\eta|} + \max_{1\leq j\leq m} F_j - F_m. \tag{16}$$

Note that the left hand side is the number of iterations required for convergence in the mth time window.

The term $\left(1 + \frac{\log\epsilon}{\log\eta}\right)$, on the right hand side of equation (16), is comparable to the iteration count for a classical (non WR) method on the corresponding elliptic problem, which is bounded by $\left(1 + \frac{\log\epsilon}{\log\beta}\right)$. The remaining terms measure the additional iterations required because of the adaptive waveform relaxation. If $\max_{1\le j\le m} F_j - F_m$ were bounded by a constant, then we will have proven that the iteration count is independent of the time horizon. This is a difficult task in general, because the error estimates in our computational model is only an upper bound; however, we will be able to bound E_m as a constant times m, which means the iteration count per time interval is bounded by a constant *in an amortized sense.*

Bounding E_m when $m \le P$ is trivial. Recall that $D_m = 0$ for $m = 1, \dots, P$, because the first P time intervals have priority over later time intervals. Equation (15) simplifies to

$$F_m = -m\frac{\log \xi}{|\log \eta|}, \quad m = 1, \dots, P. \tag{17}$$

Hence, equation (16) for $m = 1, 2, \dots, P$ gives

$$E_m \le 1 + \frac{|\log \epsilon|}{|\log \eta|} + \max_{1\le j\le m} F_j - F_m = 1 + \frac{|\log \epsilon|}{|\log \eta|},$$

which is close to the iteration count for a classical WR method on these time blocks when $\eta \approx \beta$. If $m > P$, D_m is no longer zero, and we need to resort to the following recurrence relation to derive an equation for the delay,

$$D_{m+P} = E_m - P, \quad m = 1, 2, \dots. \tag{18}$$

From equation (15), we have

$$\begin{aligned} F_{m+P} &= D_{m+P} - (m+P)\frac{\log \xi}{|\log \eta|} \\ &= (E_m - P) - (m+P)\frac{\log \xi}{|\log \eta|} \\ &\le 1 + \frac{|\log \epsilon|}{|\log \eta|} + \max_{1\le j\le m} F_j - D_m - P\frac{\log \xi}{|\log \eta|} - P, \end{aligned}$$

or equivalently,

$$F_m \le 1 + \frac{|\log \epsilon|}{|\log \eta|} + \max_{1\le j\le (m-P)} F_j - D_{m-P} - P\frac{\log \xi}{|\log \eta|} - P. \tag{19}$$

Since $D_m = 0$ for $m = 1, \dots, P$, it will be convenient to simplify $\max_{1\le j\le m} F_m$ iteratively for $\ell P < m \le (\ell+1)P$. Consider the case $\ell = 1$, i.e., $P < m \le 2P$. Using equation (17), equation (19) simplifies to

$$F_m \le 1 + \frac{|\log \epsilon|}{|\log \eta|} - (P+1)\frac{\log \xi}{|\log \eta|} - P.$$

If

$$\Delta := 1 + \frac{|\log \epsilon|}{|\log \eta|} - P\left(1 + \frac{\log \xi}{|\log \eta|}\right)$$

is positive, then

$$\max_{1\le m\le 2P} F_m \le -\frac{\log \xi}{|\log \eta|} + \Delta,$$

otherwise it is just bounded by $-\log\xi/|\log\eta|$. Repeating this argument for $\ell = 2, 3, \ldots$, we see that for $\ell P < m \le (\ell+1)P$,

$$F_m \le \begin{cases} -\dfrac{\log\xi}{|\log\eta|} + \ell\Delta, & \Delta > 0, \\[2ex] -\dfrac{\log\xi}{|\log\eta|} + \Delta, & \Delta \le 0. \end{cases}$$

By substituting the above into equation (19), we obtain the following theorem.

THEOREM 3.6. *Let E_m be the time to convergence for the mth time window, and let ℓ be an integer such that $\ell P < m \le (\ell+1)P$. Then*

$$E_m \le 1 + \frac{|\log\epsilon|}{|\log\eta|} + (m-1)\frac{\log\xi}{|\log\eta|} + \ell\cdot\max\left\{0, 1 + \frac{|\log\epsilon|}{|\log\eta|} - P\left(1 + \frac{\log\xi}{|\log\eta|}\right)\right\}.$$

To estimate the wall time needed to complete the integration, we introduce the concept of *effective parallel linear solves* (EPLS), which is defined as the number of columns in the dependency graph, assuming that all tasks in a column are simultaneously computed. For the standard time-stepping algorithm, the number of EPLS is estimated by

$$M\left(1 + \frac{|\log\epsilon|}{|\log\beta|}\right) =: k_{\text{std}},$$

where $\beta < \eta$ is the actual contraction rate when we have exact initial conditions, and $\left(1 + \frac{|\log\epsilon|}{|\log\beta|}\right)$ is the number of iterations required for convergence on a single time interval. For the WRAP algorithm, the EPLS is given by $E_M + (M-1)$, where the extra $M-1$ solves arise because the task involving time interval I_M can only appear in the task list after $M-1$ updates, even if there is no delay in execution. Letting $M - 1 = \ell P + r$, where $0 \le r < P$, we have

$$\text{EPLS} \le \begin{cases} (\ell+1)\left(1 + \frac{|\log\epsilon|}{|\log\eta|}\right) + r\left(1 + \frac{\log\xi}{|\log\eta|}\right), & P < \left(1 + \frac{|\log\epsilon|}{|\log\eta|}\right) / \left(1 + \frac{\log\xi}{|\log\eta|}\right), \\ 1 + \frac{|\log\epsilon|}{|\log\eta|} + (M-1)\left(1 + \frac{\log\xi}{|\log\eta|}\right), & \text{otherwise.} \end{cases}$$

We see that the ratio,

$$P^* = \left(1 + \frac{|\log\epsilon|}{|\log\eta|}\right) / \left(1 + \frac{\log\xi}{|\log\eta|}\right),$$

determines the optimal number of processors per subdomain. In fact, if $P < P^*$, then we have

$$\text{EPLS} \le \left(1 + \frac{|\log\epsilon|}{|\log\eta|}\right)(\ell + 1 + r/P^*) \le \frac{M + P - 1}{P}\left(1 + \frac{|\log\epsilon|}{|\log\eta|}\right).$$

If we have $\eta \approx \beta$, then the theoretical speedup becomes

$$\text{Speedup} = \frac{k_{\text{std}}}{\text{EPLS}} \gtrapprox P\left(1 + \frac{P-1}{M}\right)^{-1},$$

meaning the speedup approaches P as the number of time intervals becomes large. Thus, we get perfect speedup in the limit. On the other hand, if $P \ge P^*$, then

$$\text{EPLS} \gtrapprox (M + P^* - 1)\left(1 + \frac{\log\xi}{|\log\eta|}\right),$$

so the speedup is bounded above by

$$\text{Speedup} \le \frac{MP^*}{M+P^*-1} \to P^* \quad \text{as } M \to \infty.$$

Remark. If we assume (16) is a reasonable approximation of the actual iteration count, i.e., if

$$k_m \approx 1 + \frac{|\log \epsilon|}{|\log \eta|} + \max_{1\le j\le m} F_j - F_m,$$

then a straightforward substitution yields

$$k_m \approx \begin{cases} \frac{|\log \epsilon|}{|\log \eta|} + (m-1)\frac{\log \xi}{|\log \eta|}, & 1 \le m \le P, \\ \max\left(P(1+\frac{\log\xi}{|\log\eta|}), \frac{|\log\epsilon|}{|\log\eta|}\right), & m > P. \end{cases}$$

Thus, for the initial time intervals, we need to take additional iterations to offset the growth of the error as m increases. The same thing happens with the non-adaptive WR method. Beyond the first P intervals, however, the number of iterations is essentially constant, but the constant depends on the number of processors P. For small P, we take the same number of iterations as the sequential method, but for large P, the constant is proportional to P. This is in agreement with our numerical experiments, see Section 4.

Remark. The maximum possible speadup when $P = M$ has been previously studied for the non-adaptive WR method [16]. Specifically, EPLS $= M + K$, where K is the number of waveform iterations computed for the non-adaptive WR method. To compute the maximum possible speedup when $P = M$ for the adaptive WR method, we first let k_m be the number of iterations required by a Schwarz iteration in time block I_m (i.e., the adaptive WR method with $P = 1$). Denote $k_{\text{tot}} = \sum_{m=1}^{M} k_m$. Let $\tilde{k}_m$ be the number of iterations required in time block I_m for the adaptive WR method with $P = M$, and denote $\tilde{k}_{\max} = \max_{1\le m\le M} \tilde{k}_m$. Then the maximum possible speedup for the adaptive WR method with $P = M$ is

$$\frac{k_{\text{tot}}}{M + \tilde{k}_{\max}}. \tag{20}$$

This speedup can be estimated by realizing that $k_{\text{tot}} = Mk_{\text{avg}}$, and the ratio $\frac{M}{M+\tilde{k}_{\max}}$ is bounded above by one. Hence, the maximum possible speedup is bounded by k_{avg}.

4. Numerical Experiments.

In this section, we perform five different experiments that illustrate the behaviour of the WRAP framework applied to different DD methods and problems. In the first three experiments, we solve the heat equation using three DD methods, namely, the classical and optimized Schwarz WR methods, as well as the Neumann-Neumann WR method. In the fourth experiment, we consider an advection-diffusion equation that is advection dominated; this is an interesting case because the performance of other time-parallel methods such as parareal [14], deteriorates as the equation becomes more and more dominated by advection. Finally, we present a nonlinear PDE system that models an idealized autocatalytic reaction.

In the first experiment, the adaptive classical Schwarz waveform relaxation approach is used to solve the linear heat equation in $\mathbb{R}^1$,

$$u_t = u_{xx}, \quad x \in [0,1], \quad t \in [0,1],$$
$$u(0,x) = \sin(\pi x),$$

We discretize the system using backward Euler in time and central differences in space, with $\Delta x = 1/1024$ and $\Delta t = 0.01$. The spatial domain is subdivided into four overlapping subdomains; the width of the overlap region is chosen to be $\frac{1}{16}$th of the subdomain width, requiring the classical Schwarz WR method to take *many* iterations to converge to the mono-domain solution. One hundred time blocks, each consisting of one time step, are used. For a tolerance of 10^{-6}, the number of waveform iterates required at each time step for various *ntasks* values are shown in Figure 5.

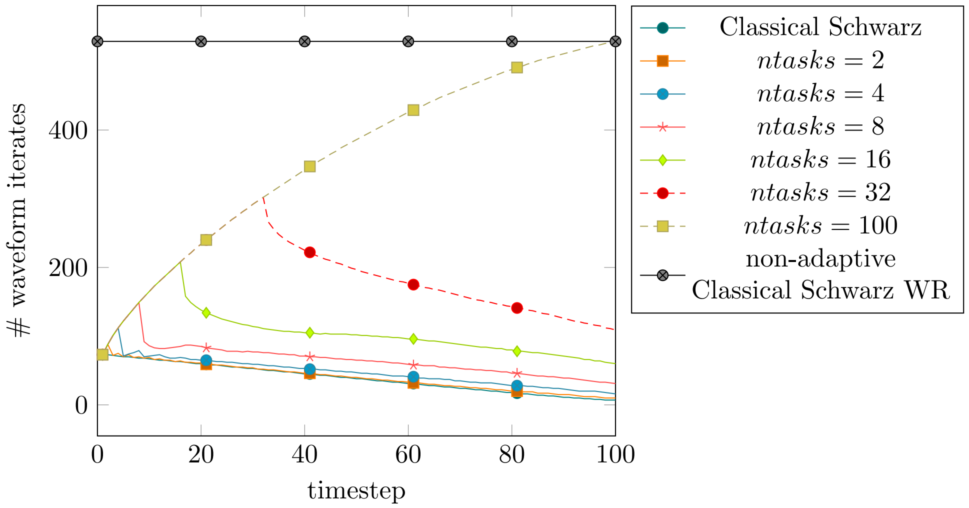

Fig. 5: Classical Schwarz coupling conditions: Number of waveform iterates at each time step required to reach the same final tolerance for varying numbers of simultaneous tasks.

From Figure 5, several observations should be made. First, consider the total number of iterations (tasks) required for each implementation with *ntasks*, i.e., the area under each curve in Figure 5. The implementation requiring the fewest total number of iterations is $ntask = 1$, corresponding to the classical Schwarz DD method. This is unsurprising since we are iterating each time step until convergence, before propagating the resulting small error. Second, the total number of waveform iterates for the adaptive WR approach is significantly lower than for the non-adaptive classical Schwarz WR approach. Lastly, as *ntasks* is increased, the total number of waveform iterates required increases.

Figure 5 does not address the speedup that is possible using adaptive pipelining, however. In Figure 6, we depict the computation of the waveform iterates for each time step (x-axis) relative to when they are computed in the simulation (y-axis) for the case $ntasks = 8$. (Figure 6 can be viewed as the silhouette of the dependency graph, rotated by 90 degrees.) Observe that the height of the bar corresponds to the number of iterations required at each time step. The WRAP algorithm does more iterations initially, consistent with the analysis. Also observe that each horizontal slice of the plot in Figure 6 will have at most eight markers because the maximum number of tasks that are simultaneously computed in this example is $ntasks = 8$. Finally, we see that the WRAP algorithm has a preference for iterating earlier time steps to convergence; later time steps are not started until the earlier time steps are

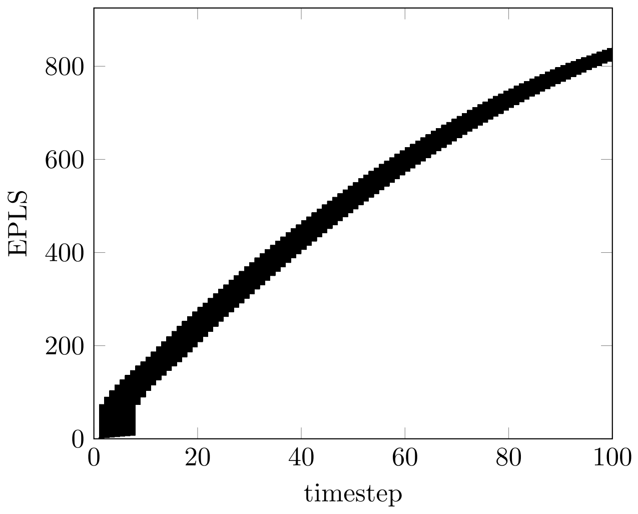

Fig. 6: Classical Schwarz coupling conditions: bars denote computation of the waveform iterates for each time step (x-axis) relative to when they are computed in the simulation (y-axis) for the case $ntasks = 8$. Here, the walltime unit is the amount of time it would take to compute one parallel solve. This WRAP method using $ntasks = 8$ requires 841 effective parallel linear solves.

iterated to convergence.

Table 1 shows the speedup that can be expected using the adaptive pipeline WR approach with classical Schwarz transmission conditions. The theoretical speedup is computed by taking the ratio of the the number of effective parallel linear solves using the adaptive pipeline WR framework against that of the classical Schwarz domain decomposition method ($ntasks = 1$). The speedup increases monotonically with $ntasks$, but saturates at approximately 6, even when $ntasks = M$. The observed saturated speedup is in agreement with equation (20). Specifically, $\tilde{k}_{\max} = 529$ for the adaptive WR method with $ntasks = 100$. Since $M = 100$ and $k_{\text{tot}} = 3841$ (note: $k_{\text{tot}} =$ EPLS for $ntasks = 1$), equation (20) gives a theoretical maximum speedup of 6.1.

$ntasks$	# procs	EPLS	speedup
1	4	3841	–
2	8	2017	1.90
4	16	1195	3.21
8	32	841	4.57
16	64	687	5.59
32	128	629	6.11
100	400	628	6.12

Table 1: Classical Schwarz coupling conditions: Theoretical speedup using the adaptive pipeline WR approaches for various $ntasks$, with $M = 100$ time blocks. The effective number of parallel linear solve (EPLS) is defined in Section 3.3.

Speedup can be potentially improved when more time blocks are used, since the processors can then march in a pipe for a larger number of tasks. In Table 2, we repeat the previous numerical experiment with $\Delta t = 0.001$, so that up to 1000 tasks can be launched. We observe that the speedup now saturates at around 7, which is better than before, but only marginally. The reason is that the problem has become easier as Δt becomes smaller: for $ntasks = 1$, i.e. the standard time stepping method only requires an average of 9.9 EPLS per time step, instead of 38 EPLS per time step when $\Delta t = 0.01$. Also note that WRAP now only takes 1388 effective solves (with $ntasks = 100$) to complete a 1000-step integration, i.e., about 1.4 EPLS per step. With such a low EPLS per step, it is unlikely that further speedup can be obtained by adding processors in the time direction. Nevertheless, this speedup comes *on top of any spatial parallelism*, so an extra multiplicative factor of 5 to 7 in the speedup should be considered significant.

ntasks	# procs	EPLS	speedup
1	4	9902	–
2	8	5182	1.91
4	16	3101	3.19
8	32	2183	4.54
16	64	1748	5.66
32	128	1537	6.44
100	400	1388	7.13

Table 2: Classical Schwarz coupling conditions: Theoretical speedup using the adaptive pipeline WR approaches for various *ntasks*, with $M = 1000$ time blocks.

In the second experiment, we again solve the linear heat equation using the adaptive waveform relaxation framework, this time with optimized transmission conditions. Numerical results are presented for four non-overlapping domains, optimized parameter, $p = \frac{1}{\sqrt{\Delta t}}$, 100 time blocks, each time block consisting of a single time step. For a tolerance of 10^{-12}, the number of waveform iterates required at each time step for various *ntasks* values are shown in Figure 7. Similar to the previous experiment, Table 3 shows the theoretical speedup that can be expected using the WRAP approach with optimized transmission conditions. The theoretical speedup is computed by comparing the number of effective parallel linear solves using the adaptive pipeline WR framework compared against the optimized Schwarz WR approach ($ntasks = 1$). Similar observations to the previous numerical experiment, consistent with the analysis can be made. Again, the modest parallel speedup numbers can be explained by the low number of EPLS per time step, which went from 8.65 for $ntasks = 1$ to 1.65 for $ntasks \geq 16$.

Recently, a pipeline parallel implementation for Neumann–Neumann waveform relaxation (NNWR) methods was explored [17]. The NNWR method performs a two-step iteration consisting of first solving a “Dirichlet” sub-problem on each space–time domain, followed by solving an auxiliary “Neumann” sub-problem. Although no analysis is provided for the adaptive pipeline framework applied to the Neumman–Neumann waveform relaxation method, we show in this third numerical experiment that similar behavior to previous numerical experiments can be observed. The linear heat equation in $\mathbb{R}^1$ is solved with the spatial domain divided into four non-overlapping subdomains. For a tolerance of 10^{-12}, the number of distributed linear solves required

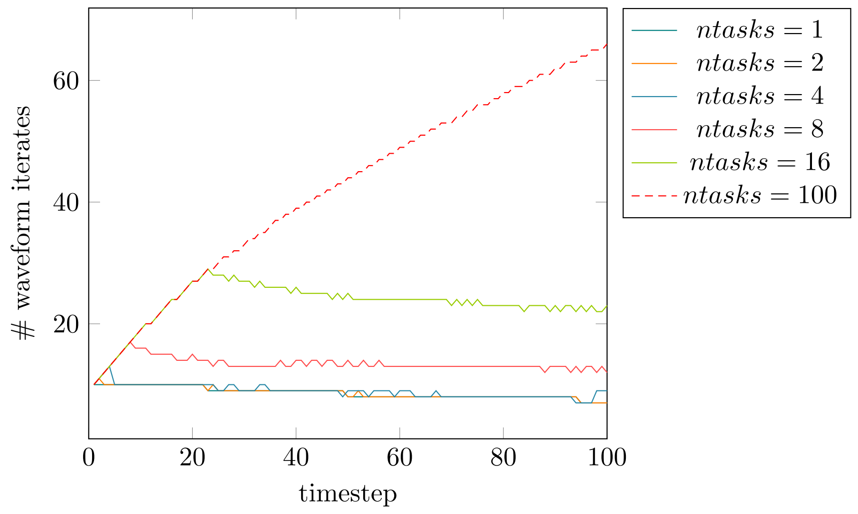

Fig. 7: WRAP with optimized transmission conditions: plot shows the number of waveform iterates at each time step required to reach the same final tolerance for varying numbers of simultaneous tasks.

ntasks	# procs	EPLS	speedup
1	4	865	–
2	8	436	1.98
4	16	228	3.79
8	32	176	4.91
16	64	165	5.24
100	400	165	5.24

Table 3: Theoretical speedup using the WRAP method with optimized transmission conditions for $M = 100$ time blocks and various *ntasks*.

at each time step for various *ntasks* values are shown in Figure 8. Here, each Dirichlet update or Neumann update requires a distributed linear solve. Table 4 shows the theoretical speedup that can be expected using the adaptive pipeline NNWR approach compared with a classical Neumann-Neumann iteration. The theoretical speedup is computed by comparing the number of effective parallel linear solves using the adaptive pipeline WR framework compared against the Neumann–Neumann domain decomposition method ($ntasks = 1$).

For the fourth experiment, we solve the advection-diffusion equation

$$u_t = \nu u_{xx} + u_x, \quad x \in [0, 2], \quad t \in [0, 4],$$

with periodic boundary conditions,

$$u(0, t) = u(2, t), \quad u_x(0, t) = u_x(2, t), \qquad t \in (0, T),$$

and with initial conditions $u(x, 0) = e^{-20(x-1)^2}$ for $x \in (0, 2)$. We discretize the system using backward Euler in time and first order upwind in space, with $\Delta x = 1/512$ and

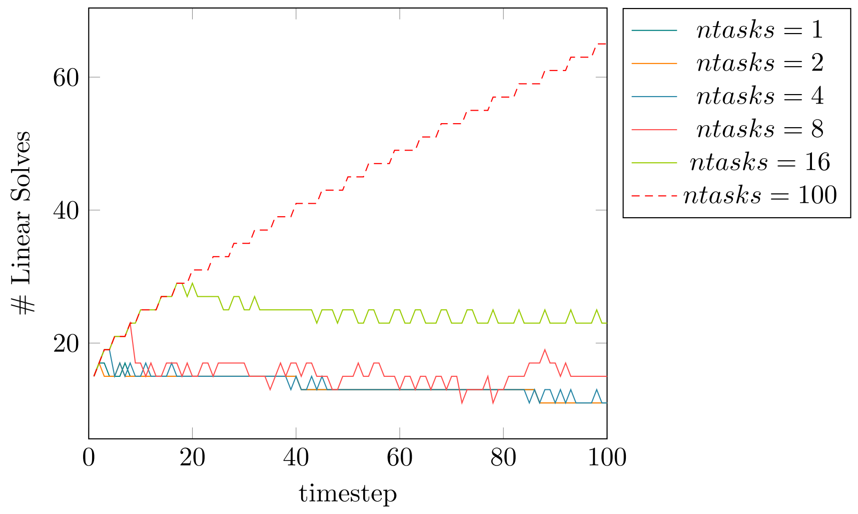

Fig. 8: WRAP framework applied to NNWR methods: plot shows the number of waveform iterates at each time step required to reach the same final tolerance for varying numbers of simultaneous tasks.

ntasks	# procs	EPLS	speedup
1	4	1358	–
2	8	681	1.99
4	16	350	3.88
8	32	206	6.59
16	64	170	7.99
100	400	164	8.28

Table 4: Theoretical speedup using the WRAP framework applied to NNWR method for $M = 100$ time blocks and various *ntasks*.

$\Delta t = 0.01$. As $\nu \to 0$, the problem becomes more and more advection dominated. It has been shown in [4] that the convergence of the Parareal method deteriorates for small ν, and speedup suffers as a result. We show our results for $\nu = 0.05$ and $\nu = 0.005$ in Tables 5 and 6 respectively. Four overlapping subdomains and Dirichlet transmission conditions are used in both cases. We see that our speedup remains reasonable even for these highly advection-dominated cases. In fact, the less favorable speedup for $\nu = 0.005$ is due to the problem being *easier*: serial time-stepping only requires 2400 EPLS, or 6 EPLS per time step, instead of 4589 EPLS (or 11.5 EPLS per time step) in the more diffusive case.

In the last experiment, we consider an idealized autocatalytic reaction, which can be modelled by the following reaction–diffusion system,

$$u_t = A + u^2 v - (B+1)u + \alpha\, u_{xx},$$
$$v_t = B\,u - u^2 v + \alpha\, v_{xx}.$$

Here, $A = 1$ and $B = 3$ are rate constants, and $\alpha = \frac{1}{50}$ is the diffusion constant. The

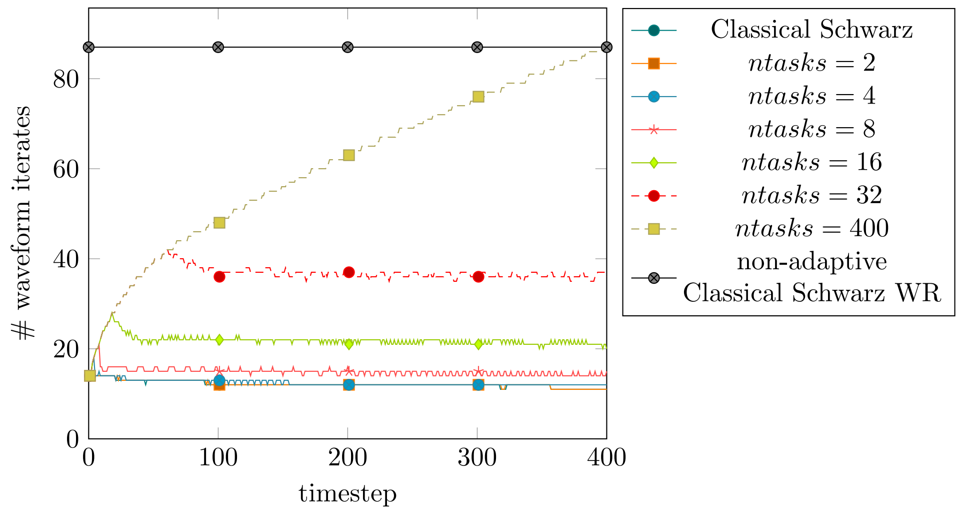

Fig. 9: WRAP for advection-diffusion equation with $\nu = 0.05$: plot shows the number of waveform iterates at each time step required to reach the same final tolerance for varying numbers of simultaneous tasks.

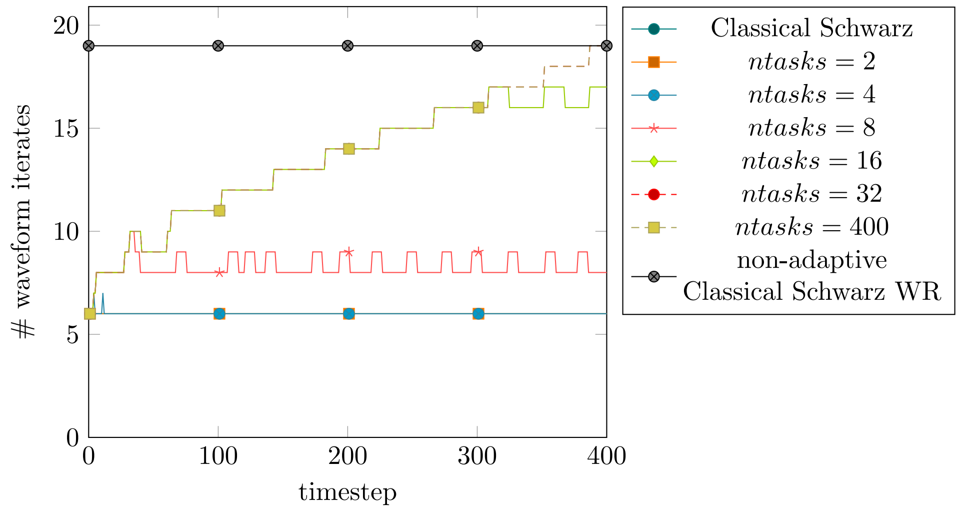

Fig. 10: WRAP for advection-diffusion equation with $\nu = 0.005$: plot shows the number of waveform iterates at each time step required to reach the same final tolerance for varying numbers of simultaneous tasks.

initial and boundary conditions are:

$$u(0,t) = u(1,t) = 1, \qquad v(0,t) = v(1,t) = 3,$$
$$u(x,0) = 1 + \sin 2\pi\, x, \qquad v(x,0) = 0.$$

This reaction system is nonlinear, and stiff due to the diffusion. We discretize the system using an IMEX scheme: the reaction term is handled explicitly using the

ntasks	# procs	EPLS	speedup
1	4	4859	–
2	8	2437	1.99
4	16	1250	3.89
8	32	754	6.44
16	64	562	8.65
32	128	489	9.94
100	400	487	10.00

Table 5: Theoretical speedup using the WRAP framework applied to the advection-diffusion problem with $\nu = 0.05$ and $M = 400$ time blocks.

ntasks	# procs	EPLS	speedup
1	4	2400	–
2	8	1201	2.00
4	16	604	3.97
8	32	422	5.69
16	64	418	5.74
32	128	418	5.74
100	400	418	5.74

Table 6: Theoretical speedup using the WRAP framework applied to the advection-diffusion problem with $\nu = 0.005$ and $M = 400$ time blocks.

explicit Euler integrator, and the diffusion term is handled implicitly using the implicit Euler integrator. A centered finite difference approximation is used to approximate the diffusion term. The spatial domain is subdivided into four overlapping subdomains; the width of the overlap region is again chosen to be $\frac{1}{16}$th of the subdomain width. One hundred time blocks, each consisting of one time step, are used. Similar observations to the first numerical experiment can be made. For a tolerance of 10^{-6}, the number of waveform iterates required at each time step for various *ntasks* values are shown in Figure 11. The theoretical speedup is summarized in Table 7.

ntasks	# procs	EPLS	speedup
1	4	975	–
2	8	495	1.97
4	16	270	3.61
8	32	184	5.30
16	64	154	6.33
100	400	149	6.54

Table 7: Theoretical speedup for solving the Brusselator system using the WRAP framework with Dirichlet transmission condition and $M = 100$ time blocks.

5. Conclusions.

Adaptive pipelining is introduced to efficiently utilize a fixed number of computational workers for waveform relaxation methods. In this method,

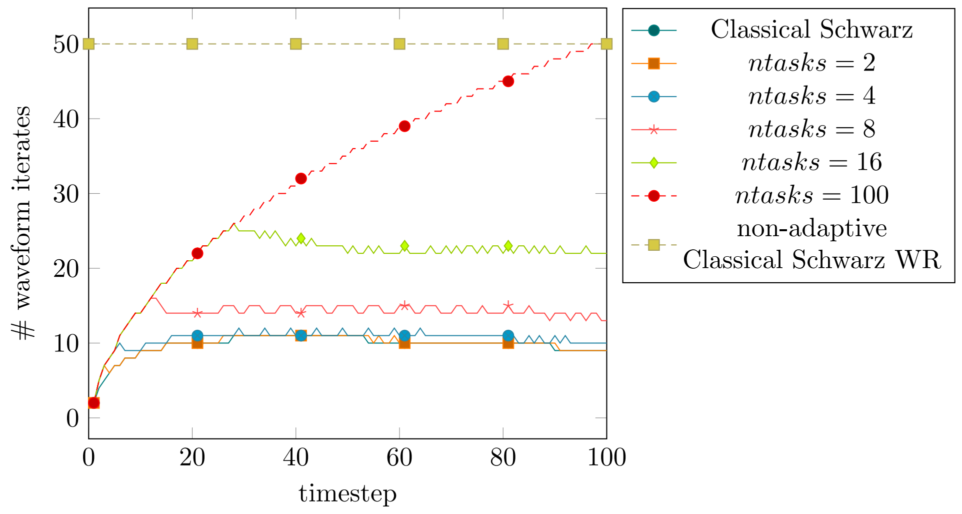

Fig. 11: Solving the Brusselator equation using the WRAP framework, Dirichlet transmission conditions. Here, we plot the number of waveform iterates at each time step required to reach the same final tolerance for varying numbers of simultaneous tasks.

we address two main issues of WR methods, namely convergence degradation for long-time integration, and oversolving in the initial time steps. We do so by keeping the effective window of integration small, and reassigning workers from converged time steps in order to grow the time horizon. The new WRAP methods are analyzed to show the theoretical speedup that can be expected. The WRAP framework has several desirable properties. First, one limiting case recovers Schwarz DD methods, allowing a direct comparison with classical DD methods. Another limiting case recovers classical WR methods. The numerical experiments show that parallel speedup with moderate efficiency over classical DD methods can be expected with the WRAP framework. Secondly, although the parallel speedup saturates as the number of tasks (i.e. number of waveform iterates computed in parallel) increases, the speedup appears as a multiplicative factor when used in combination with other temporal or spatial parallelism. In fact, this method can be used within parareal itself in order to accelerate the fine integration steps. Thus, our method is complementary to existing space and time parallel methods, and can be used to speed up computation significantly when the saturation point is reached for other types of parallelism.

REFERENCES

[1] D. Bennequin, M. J. Gander, and L. Halpern, *A Homographic Best Approximation Problem with Application to Optimized Schwarz Waveform Relaxation*, Math. of Comp., (2009), pp. 185–223.

[2] S. Brenner and R. Scott, *The Mathematical Theory of Finite Element Methods*, vol. 15 of Texts in Applied Mathematics, Springer Science & Business Media, 2007.

[3] V. Dolean, P. Jolivet, and F. Nataf, *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*, SIAM, 2015.

[4] M. J. Gander, *Five decades of time parallel time integration, and a note on the degradation of the performance of the parareal algorithm as a function of the reynolds number*, in

Oberwolfach Report, 2017.

[5] M. J. Gander, F. Kwok, and B. C. Mandal, *Dirichlet-neumann and neumann-neumann waveform relaxation algorithms for parabolic problems*, Elect. Trans. Numer. Anal., 45 (2016), pp. 424–456.

[6] M. J. Gander and A. M. Stuart, *Space-time continuous analysis of waveform relaxation for the heat equation*, SIAM J. Sci. Comput., 19 (1998), pp. 2014–2031.

[7] M. J. Gander and H. Zhao, *Overlapping Schwarz waveform relaxation for the heat equation in n dimensions*, BIT Numerical Mathematics, 42 (2002), pp. 779–795.

[8] E. Giladi and H. B. Keller, *Space-time domain decomposition for parabolic problems*, Numerische Mathematik, 93 (2002), pp. 279–313.

[9] B. Gustafsson, H.-O. Kreiss, and J. Oliger, *Time dependent problems and difference methods*, vol. 24, John Wiley & Sons, 1995.

[10] L. Halpern, C. Japhet, and J. Szeftel, *Optimized Schwarz waveform relaxation and discontinuous galerkin time stepping for heterogeneous problems*, SIAM J. Numer. Anal., 50 (2012), pp. 2588–2611.

[11] T. J. Hughes, *The finite element method: linear static and dynamic finite element analysis*, Courier Corporation, 2012.

[12] A. Iserles, *A first course in the numerical analysis of differential equations*, no. 44, Cambridge University Press, 2009.

[13] F. Kwok, *Neumann-Neumann Waveform Relaxation for the Time-Dependent Heat Equation*, in Domain Decomposition in Science and Engineering XXI, J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. B. Widlund, eds., vol. 98, Springer-Verlag, 2014, pp. 189–198.

[14] J.-L. Lions, Y. Maday, and G. Turinici, *A parareal in time discretization of PDEs*, C.R. Acad. Sci. Paris, Serie I, 332 (2001), pp. 661–668.

[15] B. C. Mandal, *A Time-Dependent Dirichlet-Neumann Method for the Heat Equation*, in Domain Decomposition in Science and Engineering XXI, J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. B. Widlund, eds., vol. 98, Springer-Verlag, 2014, pp. 467–475.

[16] B. W. Ong, S. High, and F. Kwok, *Pipeline schwarz waveform relaxation*, in Domain Decomposition Methods in Science and Engineering XXII, T. Dickopf, M. J. Gander, L. Halpern, R. Krause, and L. Pavarino, eds., Lecture Notes in Computational Science and Engineering, Springer International Publishing, 2016, pp. 179–187, https://doi.org/10.1007/978-3-319-18827-0_36, http://dx.doi.org/10.1007/978-3-319-18827-0_36.

[17] B. W. Ong and B. C. Mandal, *Pipeline implementations of Neumann–Neumann and Dirichlet–Neumann waveform relaxation methods*, Numerical Algorithms, (2017), https://doi.org/10.1007/s11075-017-0364-3, https://doi.org/10.1007/s11075-017-0364-3.

[18] A. Toselli and O. B. Widlund, *Domain Decomposition Methods: Algorithms and Theory*, vol. 34 of Springer Series in Computational Mathematics, Springer, 2005.