Performance of Waveform Relaxation with Adaptive Pipelining on the Wave Equation

Laurence Halpern[0000−0002−7877−7130] and
Felix Kwok[0000−0002−8360−6328]

1 Introduction

The Waveform Relaxation with Adaptive Pipelining (WRAP) algorithm was introduced in [4] to parallelize Schwarz waveform relaxation (SWR) iterations across different time steps. In that paper, a recurrence involving error norms on the initial and transmission conditions allows one to bound the wall-clock time (excluding communication costs). The key assumption in this recurrence is that the error in the transmission condition decreases geometrically to zero for exact initial conditions. However, for wave-type problems, one often has a nilpotent iteration, i.e., the error exhibits no decrease until after $k > 1$ iterations, when it drops to zero. The goal of this paper is to understand the behaviour of WRAP for such iterations, as well as the effect of communication cost on overall performance. Here, we concentrate on initial value problems, although we get similar results when we use WRAP for gradient calculations arising from optimal control problems. Note that other time parallelization strategies exist for the wave equation, e.g. Parareal [5] and tent-pitching methods [3].

2 Convergence of SWR for the 1D wave equation

In [2], the authors introduced transparent transmission conditions for the 1D wave equation. For ease of presentation, we assume a constant wave speed $c = 1$ throughout the domain Ω, which is divided into non-overlapping spatial subdomains $(\Omega_i)_{i=1}^M$ of equal length H, arranged from left to right. Then for $k = 1, 2, \ldots$, one solves on each subdomain Ω_i

Laurence Halpern
LAGA, Université Sorbonne Paris Nord, France, e-mail: halpern@math.univ-paris13.fr

Felix Kwok
Université Laval, Canada, e-mail: felix.kwok@mat.ulaval.ca

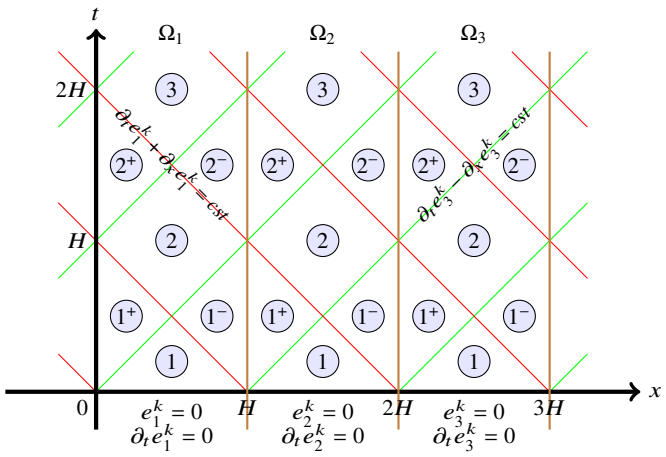

Fig. 1 Definition of regions used in the proof of Theorem 1. Regions (k), (k^+) and (k^-) correspond to places where e_i^k, $\mathcal{B}_+ e_i^k$ or $\mathcal{B}_- e_i^k$ first becomes zero at iteration k.

$$\partial_t^2 u_i^k = \partial_x^2 u_i^k + f \qquad \text{on } \Omega_i \times [0,T], \tag{1a}$$

$$\mathcal{B}_\pm u_i^k = \mathcal{B}_\pm u_{i\pm1}^{k-1} \qquad \text{on } (\partial\Omega_i \cap \overline{\Omega}_{i\pm1}) \times [0,T], \tag{1b}$$

$$u_i^k = u, \quad \partial_t u_i^k = \partial_t u \qquad \text{on } \Omega_i \times \{0\} \tag{1c}$$

where $\mathcal{B}_\pm = \partial_t \pm \partial_x$, with suitable adjustments for any Dirichlet or Neumann boundary conditions on the physical boundary. Periodic boundary conditions can also be handled by identifying Ω_0 with Ω_M and Ω_{M+1} with Ω_1. A slight modification of Theorem 3.4 in [2] allows us to prove the following.

Theorem 1 *Let $u(x,t)$ be the exact solution of $\partial_t^2 u - \partial_x^2 u = f$ and u_i^k be defined by* (1a)–(1c). *Then for any $k \geq 0$ and any i, we have $u_i^{k+1}(x,t) = u(x,t)$ for any $x \in \Omega_i$ and $t \in [0, kH]$. In other words, optimized SWR with transparent transmission conditions converges to the exact solution in $k+1$ iterations, where $k = \lceil T/H \rceil$.*

Proof. By linearity, it suffices to consider the error $e_i^k = u_i^k - u|_{\Omega_i \times [0,T]}$, where e_i^k satisfies (1a)–(1c) with $f = 0$, $e_i^k(x,0) = 0$, $\partial_t e_i^k(x,0) = 0$, and homogeneous Dirichlet or Neumann conditions on the outer boundary. Consider the diagram in Figure 1, where the vertical edge separating the regions (j^+) and (j^-) corresponds to the time interval $[(j-1)H, jH]$. We now show by induction on k that

(i) $e_i^k = 0$ in any region (j) of Ω_i with $j \leq k$;
(ii) $\mathcal{B}_- e_i^k = 0$ in region (k^-) of Ω_i, and $e_i^k = 0$ in any region (j^-) of Ω_i with $j \leq k-1$;
(iii) $\mathcal{B}_+ e_i^k = 0$ in region (k^+) of Ω_i, and $e_i^k = 0$ in any region (j^+) of Ω_i with $j \leq k-1$.

Note that the above implies $e_i^k = 0$ for $0 \leq t \leq (k-1)H$, as required. We start by observing that the wave equation can be factorized as

$$(\partial_t + \partial_x)(\mathcal{B}_- e_i^k) = (\partial_t - \partial_x)(\mathcal{B}_+ e_i^k) = 0,$$

which implies that $\mathcal{B}_- e_i^k$ is constant along characteristics with slope +1. Thus, at iteration $k = 1$, the initial conditions $e_i^1(x,0) = \partial_t e_i^1(x,0) = 0$ imply $\mathcal{B}_- e_i^1 = 0$ within the regions (1) and (1^-). Similarly, since $\mathcal{B}_+ e_i^1$ is constant along characteristics with slope -1, the initial conditions imply that $\mathcal{B}_+ e_i^1 = 0$ within the regions (1) and (1^+). Since $\mathcal{B}_- e_i^1 = \mathcal{B}_+ e_i^1 = 0$ in region (1), we conclude that $e_i^1 = 0$ there. We have therefore proved (i)–(iii) for $k = 1$.

We assume inductively that (i)–(iii) hold up to $k-1$. Let $j \le k-1$ and $\Gamma_L(i, j^+)$ be the portion of the left boundary of Ω_i adjacent to the region (j^+). If this is an interface with Ω_{i-1}, then it coincides with $\Gamma_R(i-1, j^-)$, the portion of the right boundary of Ω_{i-1} adjacent to its own region (j^-). Then from the transmission condition (1b), we have

$$(\mathcal{B}_- e_i^k)|_{\Gamma_L(i,j^+)} = (\mathcal{B}_- e_{i-1}^{k-1})|_{\Gamma_R(i-1,j^-)} = 0 \qquad \text{by (ii).}$$

If $\Gamma_L(i, j^+)$ is an outer boundary, conditions of the Dirichlet or Neumann type will lead to either $\partial_t e_i^k = 0$ or $\partial_x e_i^k = 0$, which can be combined with $\mathcal{B}_+ e_i^k = 0$ (by (iii)) to deduce that $\mathcal{B}_- e_i^k = 0$ on $\Gamma_L(i, j^+)$. Propagating this information within Ω_i along characteristics with slope +1 leads to $\mathcal{B}_- e_i^k = 0$ in regions (j^+), $(j+1)$ and $((j+1)^-)$. The same argument on the right boundary $\Gamma_R(i, j^-)$ shows that we have $\mathcal{B}_+ e_i^k = 0$ in regions (j^-), $(j+1)$ and $((j+1)^+)$. In summary, we have $\mathcal{B}_- e_i^k = \mathcal{B}_+ e_i^k$ in regions (1), (2), …, (k), as well as in regions (1^-), …, $((k-1)^-)$ and (1^+), …, $((k-1)^+)$, so $e_i^k = 0$ in all these regions. Furthermore, we have $\mathcal{B}_- e_i^k = 0$ in region (k^-) and $\mathcal{B}_+ e_i^k = 0$ in region (k^+), so (i)–(iii) is verified up to k, and the induction is complete. □

Remark 1 The above proof shows that if $t > jH$ for some integer j, then the errors $e_i^1(\cdot,t), e_i^2(\cdot,t), \ldots, e_i^j(\cdot,t)$ have no influence on the subsequent convergence behaviour of the algorithm, since the solution at time t will be overwritten by incoming data anyway at a later iteration. This behaviour is typical of SWR methods for hyperbolic problems and for problems where information propagates with finite speed, such as explicit time discretizations. This observation will be important for understanding the convergence of the adaptive pipeline in the next section.

3 Convergence of the adaptive pipeline

We now apply the previous convergence analysis to study the convergence of the adaptive pipeline, see [4] for the full description of the algorithm. In essence, WRAP subdivides the SWR iteration into "tasks" $\mathcal{T}_{i,n}^k$, indexed by their associated subdomain Ω_i, time interval (or time block) $[T_{n-1}, T_n]$ and iteration number k. The WRAP algorithm then assigns P processors per spatial subdomain to perform tasks that can be processed in parallel; see Figure 2 for an illustration for $P = 3$ processors, where tasks that are connected by a line are performed in parallel. When there are more outstanding tasks than processors available, earlier time blocks are prioritized,

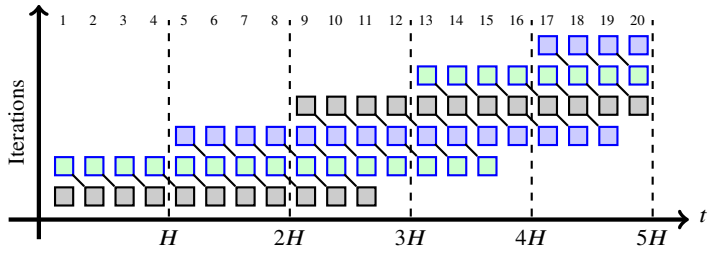

Fig. 2 Tasks executed by the adaptive pipeline running $P = 3$ processors per subdomain in parallel. Tasks that are connected are run in parallel, with a different colour identifying each processor. Each task integrates over a time block of size $H/4$. Some tasks are delayed initially due to the lack of available processors. In this example, we have $D_{12..15} = 1$, $D_{16..19} = 2$ and $D_{20} = 3$. The pipeline converges after $k = 6$ SWR iterations, after executing $k + N - 1 = 25$ non-concurrent tasks, for a theoretical speedup of $2N/(k + N - 1) = 1.6$ over sequential integration.

meaning that other tasks are delayed due to a lack of processors; this is the case for time block 12 in Figure 2, which cannot be processed until iteration 2. We define D_n to be the delay in starting the nth time block: here, we have $D_{12} = 1$. Time block n eventually converges after iteration E_n (e.g., $E_{12} = 4$ in Figure 2), which frees up a processor for computation in subsequent time blocks. The iteration terminates after $k = E_N$ iterations, where N is the total number of time blocks.

Note that $u_{i,n}^k$ does not necessarily coincide with the restriction of the SWR iterate u_i^k onto $t \in [T_{n-1}, T_n]$, since no updates can be performed in time block n during the first D_n iterations. Nonetheless, the following theorem characterizes E_n for all n:

Theorem 2 *Let $0 = T_0 < T_1 < \cdots < T_N = T$ be a partition of $[0, T]$. For any $k \geq 1$, $1 \leq i \leq M$ and $1 \leq n \leq N$, let $\mathcal{T}_{i,n}^k$ be the task that computes $u_{i,n}^k$, the approximate solution in the time block $[T_{n-1}, T_n]$, within an adaptive pipeline running $P \geq 2$ tasks in parallel. If (k, n) satisfies $(k-1)H < T_n \leq kH$, then for all $1 \leq i \leq M$, $\mathcal{T}_{i,n}^{k+1}$ produces the exact solution on Ω_i, i.e., $u_{i,n}^{k+1}(x,t) = u(x,t)$ for all $x \in \Omega_i$, $t \in [T_{n-1}, T_n]$. In other words, we have $E_n = \lceil T_n/H \rceil + 1$.*

Proof. We proceed by induction on n. For the base case, let us consider $1 \leq n \leq P$. Then $D_n = 0$, since there are enough processors to start the nth task $\mathcal{T}_{i,n}^1$ without any delay. Therefore, $u_{i,n}^{k+1}(x,t)$ is identical to the SWR iterate $u_i^{k+1}(x,t)$ for any $t \in [T_{n-1}, T_n]$, so by Theorem 1, we have $u_{i,n}^{k+1} = u_i^{k+1} = u$ on $\Omega_i \times [T_{n-1}, T_n]$ whenever $T_n \leq kH$. Now suppose the theorem statement is true up to $n-1$. We need to prove that it is true for n. As noted in Remark 1, the value of the initial iterates $u_i^j(x,t)$ has no effect on convergence if $t > jH$. Therefore, if $(\ell-1)H < T_{n-1} \leq \ell H$, then the values of $u_{i,n}^j$ has no effect as long as $j \leq \ell - 1$; thus, as long as the delay of the nth task satisfies $D_n \leq \ell - 1$, we still have $u_{i,n}^j = u_i^j$ on $\Omega_i \times [T_{n-1}, T_n]$ for any $j \geq \ell$. But $D_n = E_{n-P+1} - P = \lceil T_{n-P+1}/H \rceil + 1 - P \leq \ell + 1 - P \leq \ell - 1$, since $P \geq 2$. Thus, $u_{i,n}^{k+1} = u_i^{k+1} = u$ on $\Omega_i \times [T_{n-1}, T_n]$ whenever $T_n \leq kH$, as required. □

The above theorem implies that for the 1D problem, as long as we use $P \geq 2$ processors, the pipeline will converge in a fixed number of iterations proportional to T. Since this iteration count is the same for any $P \geq 2$, no additional speedup will be gained by using more than two processors.

4 Choosing the time block size to maximize speedup

Theorem 2 shows that for a fixed time horizon T, the number of iterations required for convergence is independent of the number of time blocks N used, and the theoretical speedup increases as N becomes large. However, this will change when communication costs are non-negligible, since more frequent communication will be required if we use a large number of time blocks. We therefore need a computational model that takes communication into account.

To fix ideas, we consider spatial subdomains of width H and apply the discretization in [2], which is second order in time and space and equivalent to the leap-frog scheme for constant wave speeds. We then choose the time step size Δt to be equal to the spatial mesh size h, which makes the numerical scheme exact. We also assume from now on that all the time blocks have the same length ΔT, such that $N = T/\Delta T$. We consider the following costs incurred by the algorithm:

0. Costs that are independent of the size of the problem (i.e., setup costs);

and for each of the $N + K$ non-concurrent tasks, where K is the number of iterations:

1. Costs that are proportional to $(\Delta T/\Delta t) \cdot (H/h)^d$, where d is the number of spatial dimensions[1] (e.g. time integration, exchange of interface traces);
2. Costs that are constant per task (e.g., communication latency, task management)

These costs are represented by the proportionality constants c_0, c_1 and c_2 in the following model for R, the total running time:

$$R = c_0 + (N + K)\left(c_1 \frac{\Delta T}{\Delta t} \cdot \frac{H^d}{h^d} + c_2\right) \tag{2}$$

Substituting $d = 1$, $\Delta T = T/N$, $K = T/H$ and $h = \Delta t$ (since CFL=1), we obtain

$$R(N) = c_0 + c_1 \frac{T}{h^2}\left(H + \frac{T}{N}\right) + c_2\left(N + \frac{T}{H}\right), \tag{3}$$

which is minimized for the optimal value of N

$$N^* = \frac{T}{h}\sqrt{\frac{c_1}{c_2}}, \qquad \text{with} \qquad R(N^*) = c_0 + \frac{T}{H}\left(\frac{H}{h}\sqrt{c_1} + \sqrt{c_2}\right)^2. \tag{4}$$

[1] We keep the spatial dimension d in the computational costs below, since they are also valid for higher dimensions; only K, the number of iterations required for convergence, behaves differently when $d > 1$.

N	Block size	P	Time (s)	Speedup
800	2	1	0.2269	1.37
		2	0.1660	
400	4	1	0.1996	1.47
		2	0.1357	
200	8	1	0.1862	1.53
		2	0.1214	
160	10	1	0.1804	1.50
		2	0.1204	
100	16	1	0.1763	1.47
		2	0.1200	
50	32	1	0.1704	1.30
		2	0.1309	
25	64	1	0.1683	1.04
		2	0.1617	

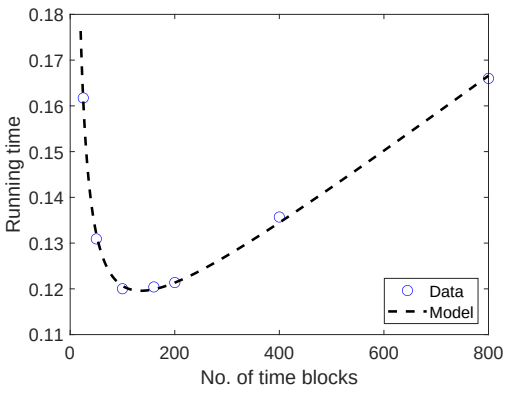

Fig. 3 Left: running time of the adaptive pipeline as a function of the number of time blocks. Right: the regression curve for the model in Equation (3) using data for $P = 2$ processors.

Observe that $T/N^* = \sqrt{c_2/c_1}h = \sqrt{c_2/c_1}\Delta t$, meaning that the optimal time block size is a constant number of time steps, irrespective of the total time horizon T. Specifically, the optimal number of time steps per block is $\sqrt{c_2/c_1}$, i.e., the square root of the ratio between the constant costs (latency and task management) and the cost per degree of freedom arising from time integration.

We implemented the adaptive pipeline using the `spmd` construct in the Matlab Parallel Computing Toolbox (version 2024a) and tested it on a 1D problem with $T = 8$ and the spatial subdomain $\Omega = (0, 2)$ subdivided into 5 equal non-overlapping subdomains, and $\Delta t = h = 0.005$. No spatial parallelism is implemented, i.e., the subdomain problems within each time block are solved sequentially by a single processor, while different time blocks are handled by different processors in parallel. We run the algorithm on an Intel Core i9-14900K machine with 128Gb of RAM and 24 cores running at 2.4 GHz. For the 1D test, we only use $P = 1$ or 2 processors per time block, since no reduction in the number of iterations will result from using more processors. The table on the left of Figure 3 shows the running time, averaged over 10 runs, as a function of the number of time blocks or, equivalently, the size of each time block as a multiple of Δt. Using linear regression on the data for $P = 2$ (see the right panel of Figure 3), we find that $c_0 = 0.017\,\text{s}$, $c_2 = 0.085\,\text{ms}$, and $c_1 = 0.61\ \mu\text{s/dof}$. For these values, the optimal number of time blocks predicted by our model is $N^* \approx 135.5$; the nearest feasible values are $N = 100$ and 160, i.e., for block sizes 16 and 10. As predicted, these choices yield the fastest running times.

5 A two-dimensional problem

We now study the performance of WRAP for the 2D wave equation via an `spmd` implementation in Matlab. For simplicity, we consider a rectangular domain Ω with

Dirichlet boundary conditions. We subdivide Ω into overlapping strips and discretize using the second order leap-frog scheme. We then apply the orthogonal variant of the absorbing transmission conditions in [1] between subdomains, since they are easy to implement and effective in reducing errors carried by propagating waves. By [1, Theorem 5.5], the interface error e_Γ^n after n iterations of SWR behaves as follows:

- If $T \leq n\delta$, then $e_\Gamma^n = 0$.
- If $T > n\delta$, then $\|e_\Gamma^n\| \leq \left(\frac{1-n\delta/T}{1+n\delta/T}\right)^n \|e_\Gamma^0\|$.

We thus have a combination of the nilpotent behaviour in 1D and the geometric convergence described in [4] for parabolic problems. We therefore expect P, the number of processors per subdomain, to play a bigger role on convergence than in the 1D case. To illustrate this, we run WRAP on the wave equation on $\Omega = (0, 2) \times (0, 1)$ for $0 \leq t \leq T = 2$. The problem is discretized on a mesh with $(\Delta x, \Delta y, \Delta t) = (1/384, 1/192, 1/440)$, so that the CFL number $\Delta t\sqrt{\Delta x^{-2} + \Delta y^{-2}} \approx 0.9757$ is close to 1. The domain is subdivided in the x direction into 8 subdomains of width $\frac{1}{4} + \delta$, where $\delta = 2\Delta x$ is the overlap (except for the first and last subdomains). The initial conditions are $u(x, y, 0) = 2e^{-100((x-1/2)^2+(y-1/2)^2)} + 2e^{-100((x-3/4)^2+(y-1/2)^2)}$, $u_t(x, y, 0) = 0$. Table 1 shows the convergence behaviour for different numbers of processors and time block sizes; we also report the median running times over 5 runs for a Matlab `spmd` implementation. Since we are now varying the number of processors P, we must modify our model (2) to include a dependence on P. Indeed, note that the head node needs to figure out which of the P tasks should be assigned to which of the P processors, and then communicate this assignment to each processor. Thus, the setup and tasks management costs are proportional to P, leading to the revised model

$$R = c_0 P + (N + K)\left(c_1 \frac{\Delta T}{\Delta t} \cdot \frac{H^d}{h^d} + c_2 P\right). \tag{5}$$

We use the running times from Table 1 to get a least-squares fit for c_0, c_1 and c_2 and use them to compute a predicted running time, which we show in Table 2 together with the error relative to the actual running times. We see that the model errors are mostly within 10%, and never over 20%. Note that in this example, the communication overhead dominates the running times. Since we cannot control the communication time in Matlab, we simulate a machine with lower communication cost by changing the cost ratios: we modify our program to repeat 10 times each line that contains actual computation of the solution, without modifying any lines involving communication or task management. This corresponds to multiplying the constant c_1 by 10 in the model (5) without changing the other parameters. The results, shown in Table 3, now show that $P = 4$ is the fastest. Thus, when computation costs dominate over communication and management costs, pipelining has the potential to reduce running times, assuming extra processors are available.

Finally, we applied WRAP to the solution of optimal control problems, where each step of the conjugate gradient method requires solving the wave equation forward and backward in time. Our results in terms of convergence and running times led to conclusions similar to those for the initial value problems above.

Table 1 Number of iterations ($N + K$) and running times (in seconds) for WRAP applied to the 2D problem.

		$P = 1$		$P = 2$		$P = 3$		$P = 4$		$P = 6$	
N	Block size	Iters	Runtime	Iters	Runtime	Iters	Runtime	Iters	Runtime	Iters	Runtime
440	2	880	2.0636	771	2.8561	688	2.9775	637	3.1309	575	3.7323
220	4	660	3.0533	498	3.3148	424	3.2049	385	3.2229	331	3.6483
110	8	550	4.9897	340	4.2345	279	3.8238	241	3.5016	202	3.5515
80	11	559	6.8725	296	4.8384	228	4.1579	195	3.8121	162	3.7557
55	16	440	7.9011	242	5.7644	184	4.6866	155	4.2169	129	4.0928
40	22	399	9.6606	203	6.3614	149	5.0406	126	4.6063	108	4.4163

Table 2 Running times (in seconds) predicted by the model (5) and the model error relative to the true running times for the 2D problem.

		$P = 1$		$P = 2$		$P = 3$		$P = 4$		$P = 6$	
N	Block size	Model	Error	Model	Error	Model	Error	Model	Error	Model	Error
440	2	2.4347	+18%	2.6350	-7.7%	2.8670	-3.7%	3.1629	+1.0%	3.8311	+2.6%
220	4	3.3696	+10%	3.0324	-8.5%	3.0744	-4.1%	3.2702	+1.5%	3.7514	+2.8%
110	8	5.3010	+6.2%	3.7809	-10%	3.5878	-6.2%	3.5932	+2.6%	3.9311	+11%
80	11	7.2424	+5.4%	4.3590	-9.9%	3.8654	-7.0%	3.7996	-0.3%	4.0705	+8.4%
55	16	8.2084	+3.9%	5.0220	-13%	4.3244	-7.7%	4.1413	-1.8%	4.3464	+6.2%
40	22	10.1283	+4.8%	5.6663	-11%	4.6753	-7.3%	4.4436	-3.5%	4.6689	+5.7%

Table 3 Running times (in seconds) of a modified algorithm where computation is 10 times more expensive than in Table 1.

N	Block size	$P = 1$	$P = 2$	$P = 3$	$P = 4$	$P = 6$
440	2	12.2321	12.4663	11.9998	11.3855	11.5271
220	4	18.3951	15.5541	13.9407	12.9941	12.1711
110	8	30.1283	20.6137	17.9036	15.4724	14.1197
80	11	42.1241	24.4329	19.9733	17.1410	15.2318
55	16	48.7582	28.9775	23.2071	19.6326	17.4053
40	22	60.5008	33.2461	25.8180	21.9029	19.7840

Acknowledgements F. Kwok acknowledges support from the National Science and Engineering Research Council of Canada (RGPIN- 2021-02595). The work described in this paper is partially supported by a grant from the ANR/RGC joint research scheme sponsored by the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. A-CityU203/19) and the French National Research Agency (Project ALLOWAPP, grant ANR-19-CE46-0013-01).

References

1. Gander, M.J., Halpern, L.: Absorbing boundary conditions for the wave equation and parallel computing. Math. Comp. **74**(249), 153–176 (2005)
2. Gander, M.J., Halpern, L., Nataf, F.: Optimal Schwarz waveform relaxation for the one dimensional wave equation. SIAM J. Numer. Anal. **41**(5), 1643–1681 (2003)
3. Gopalakrishnan, J., Schöberl, J., Wintersteiger, C.: Mapped tent pitching schemes for hyperbolic systems. SIAM J. Sci. Comput. **39**(6), B1043–B1063 (2017)
4. Kwok, F., Ong, B.W.: Schwarz waveform relaxation with adaptive pipelining. SIAM J. Sci. Comput. **41**(1), A339–A364 (2019)
5. Lions, J.L., Maday, Y., Turinici, G.: Résolution d'EDP par un schéma en temps «pararéel». C. R. Acad. Sci., Ser. I: Math. **332**(7), 661–668 (2001)