Numer. Math. Theor. Meth. Appl.
doi: 10.4208/nmtma.m12si07

Vol. **5**, No. 1, pp. 110-130
February 2012

A Geometric Space-Time Multigrid Algorithm for the Heat Equation

Tobias Weinzierl[1,*] and Tobias Köppl[2]

[1] *Institut für Informatik, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany.*
[2] *Institut für Mathematik, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany.*

Received 29 November 2010; Accepted (in revised version) 05 May 2011

Available 21 December 2011

Abstract. We study the time-dependent heat equation on its space-time domain that is discretised by a k-spacetree. k-spacetrees are a generalisation of the octree concept and are a discretisation paradigm yielding a multiscale representation of dynamically adaptive Cartesian grids with low memory footprint. The paper presents a full approximation storage geometric multigrid implementation for this setting that combines the smoothing properties of multigrid for the equation's elliptic operator with a multiscale solution propagation in time. While the runtime and memory overhead for tackling the all-in-one space-time problem is bounded, the holistic approach promises to exhibit a better parallel scalability than classical time stepping, adaptive dynamic refinement in space and time fall naturally into place, as well as the treatment of periodic boundary conditions of steady cycle systems, on-time computational steering is eased as the algorithm delivers guesses for the solution's long-term behaviour immediately, and, finally, backward problems arising from the adjoint equation benefit from the the solution being available for any point in space and time.

AMS subject classifications: 65M50, 65M55, 65N50, 65N55, 65M22

Key words: Adaptive Cartesian grids, geometric multiscale methods, heat equation, octree, spacetree, space-time discretisation.

1. Introduction

We study the heat equation

$$\partial_t u - \kappa \Delta u = f \quad \text{with} \quad u : \Omega \times (0,T) \mapsto \mathbb{R}, \tag{1.1a}$$

$$u(t=0) = u_0 \quad \text{or} \quad u(t=0) = u(T) \tag{1.1b}$$

$$u|_{\partial\Omega} = g, \kappa \in \mathbb{R}^+ \tag{1.1c}$$

*Corresponding author. *Email addresses:* weinzier@in.tum.de (T. Weinzierl), koeppl@ma.tum.de (T. Köppl)

on the unit square $\Omega = (0,1)^d \subset \mathbb{R}^d$ with $d \in \{1,2,3\}$ for a fixed time interval $(0,T)$. f, g, and u_0 are sufficiently smooth. The simple equation is a building block of many sophisticated applications in science and engineering—ranging from nano-scale computational fluid dynamics to chemical diffusion processes in turbulent flows. Software for this type of problems typically discretises the time with a standard ordinary differential equation integrator, which leads to a cascade of elliptic problems in space, and uses a sophisticated linear equation system solver such as a geometric multigrid algorithm for the latter.

While both time integration and multigrid solver are comprehensively studied, the problem nevertheless is far from an old-fashioned challenge. As more and more massively parallel computers are released, the question arises how such a cascade of elliptic problems scales. Often, the size of one single elliptic subproblem is too small to exploit the full power of all processing units. This holds in particular for coarse grid subproblems of multigrid algorithms. More and more cores then do not enable one to handle longer and longer simulated time intervals in a given period due to the sequential character of the underlying challenge [10, 11]. Parareal-type algorithms [5] solving several spatial subproblems in parallel promise to resolve this problem due to a good prediction for the solution's time behaviour. Other approaches deploy the first p subsequent time slices to the p different nodes of a parallel computer [9, 19] and implement a chain of water carriers where the first computing node takes over the work of the $p+1$th time step as soon as the initial time step's solution has converged. However, on massively parallel systems, some processing units here might solve rather irrelevant equation systems. As many experimental settings exhibit small regions of interest (boundary or interface subdomains, e.g.) dynamic adaptivity becomes one key ingredient of sophisticated solvers. The adaptivity in space implies multiscale behaviour in time. Regions with small scale spatial behaviour also require an accurate time representation. Local time stepping approaches promise to take this insight into account. However, local time stepping makes load balancing challenging on massively parallel computers. As more and more solvers change from stand-alone applications to applications embedded into steering environments, coupled software component ecosystems, and problem solving environments [1], today's solvers for parabolic equations not only have to deliver an accurate approximation of the real-world behaviour. Good guesses of the long term behaviour have to be delivered fast to enable users and other components to interact with the solver immediately. Smaller and smaller time step sizes required to get better approximations however force the environment to wait longer for results. Sequences of computations with finer and finer time step sizes promise to deliver coarse grain solutions on time. However, they often neglect fine scale effects stemming from the initial conditions. As more and more challenges change from forward problems to optimisation and calibration tasks, algorithms today often have to solve parabolic equations forward and backward in time simultaneously [2, 3]. For this, one needs a representation of the solution all along the time interval. Sophisticated checkpointing strategies promise to deliver such snapshots of the solution at any time for a fair trade-off between computing resources and storage requirements [2]. However, with less and less memory per core available, it is not clear how these approaches scale on supercomputers—in particular for $d = 3$, where a full, regularly resolved space-time grid already reveals the curse of dimension [8]. Placed

in this arena of requirements and constraints, we integrate different ideas resulting from these different challenges into one realisation—a merge of well-known techniques rather than a new multigrid approach, but that merger proofs to be of value for some applications.

Few algorithms for parabolic equations discretise both time and space in the same manner. Instead, the time is typically meshed different from the space, and the design of effective solvers concentrates on the spatial components of the differential operator. We present a multigrid approach for the heat equation that discretises both space and time with an octree-like data structure in a Petrov-/Ritz-Galerkin way similar to [3, 8], i.e. it treats the time as an additional space dimension. From this point of view, we interpret the whole discretisation as one global load balancing challenge, we implement dynamic adaptivity in space and time (the latter mirroring adaptive time stepping techniques), we deliver rough approximations of the solution behaviour in time very fast on a coarse grid, and we always have the whole evolution of the solution at hand. Furthermore, the implementation of time-periodic initial-final conditions is straightforward [16]. The key enabler for this realisation is the fact that the dynamically adaptive multiscale spacetree grid comes along with a very low memory footprint [17, 18].

Our idea is advantageous if and only if it exhibits a spatial behaviour similar to multigrid algorithms for elliptic problems, and if and only if it can cope with standard time stepping schemes in terms of runtime. We hence implement a geometric multigrid solver for the spatial operator with a full approximation storage scheme on a hierarchical generating system similar to [10] where the implementation follows [6] and ideas of the fast adaptive composite grid methods [13]. The generating system yields a multiscale, multifrequency representation of the solution in space, and we use this multiscale representation to prolongate the solution behaviour in time. This approach mirrors the idea of Parareal schemes [5, 12] that switch to a spatial frequency representation explicitly to elaborate a first guess of what the solution looks like in the future. With a holistic approach that resolves the initial condition accurately and prolongates different resolutions of the solution to different subsequent time steps, we end up with a temporal full multigrid that exploits the multiscale behaviour of the solution—not the error which is well-known to be difficult [11]—in time and the multiscale behaviour of the error evolution in space. It applies standard coarsening in time and, consequently, scales in time. This paper emphasises the multigrid aspects and realisation.

The remainder is organised as follows: We first introduce our space-time spacetree yielding a multiscale representation of the space-time domain. Then, we define a generating system for finite elements on this space-time discretisation (Section 3). In Section 4, we first study the multiscale behaviour of the solution for standard time stepping schemes. This then leads to a space-time multiscale formulation of the global problem that treats all time snapshots simultaneously and adaptively. Some numerical results for simple settings, a short conclusion, and an outlook close the discussion.

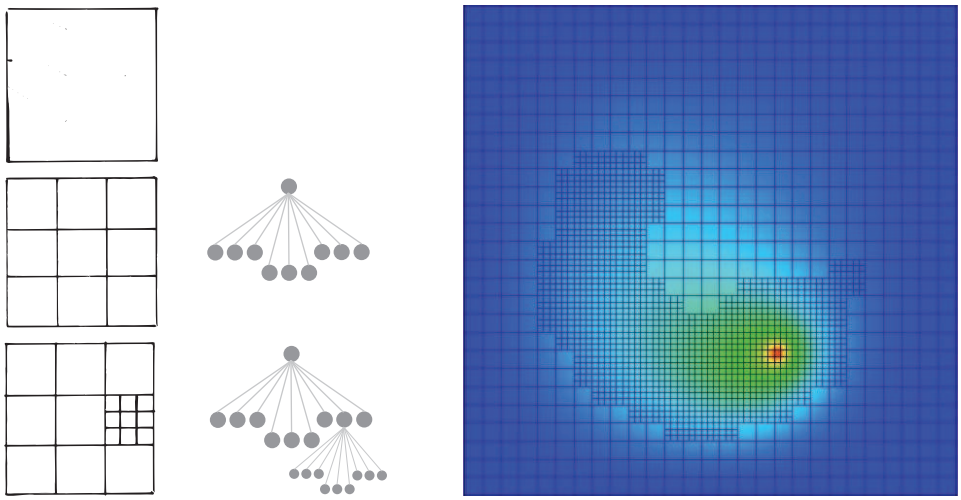

Figure 1: First three steps of the construction of a spacetree (left); snapshot of an adaptive two-dimensional spacetree grid resolving a rotating heat source (right).

2. The spacetree grid

We embed the computational domain $\Omega \times (0,T)$ from (1.1) into an axis-aligned bounding box. In our case, it is a $d+1$-dimensional hypercube. Then, we cut this hypercube into k equally sized parts along each coordinate axis. The cuts yield a grid that is refined into k^d parts spatially, and it represents k time steps. Finally, we continue to cut these k^{d+1} hypercubes recursively, while we decide for each hypercube independently of all other cubes whether to refine or not.

Such a scheme ends up with an adaptive Cartesian grid both in space and time. The face with a normal along axis $d+1$ that runs through the coordinate system's origin represents the initial condition. The opposite face of the bounding hypercube stands for the final condition at $t = T$. The adaptive Cartesian grid is spanned by a tree data structure where each tree node has either none or k^{d+1} children, i.e. each node represents either a cell refined into k^{d+1} subcells or an unrefined cell. Let a tree node have a level that is equal to the path length from the tree's root to this node. Each level of the k-spacetree represents a regular Cartesian grid, and, thus, the k-spacetree gives us a sequence of finer and finer grids. Finer and finer implies both refinement in space and time.

All reasoning here holds for arbitrary $k \geq 2$. However, we present results solely for $k = 3$, as we have a PDE solver framework for $k = 3$ at hand [17, 18]. In accordance with these publications, we work with the tree instead of the adaptive Cartesian grids as the tree gives us a multiscale representation of the space-time discretisation (Fig. 1, left side). Dynamic adaptive refinement and coarsening fit naturally into this concept, as they correspond to a further extension of leaf tree nodes, i.e. we apply the recursive construction scheme to unrefined nodes, or to removing subtrees respectively. In return, there might be several vertices $v = (x,\ell)_v, x \in \Omega \times (0,T), \ell \in \mathbb{N}_0^+$ at the same location in space-time,

however, each of them belongs to one specific grid level ℓ.

In accordance with [17, 18], the adaptive Cartesian grid can be stored with one byte per grid cell. As a result, it is possible to store huge grids on a single core of a supercomputer, and it is also possible to store big $(d+1)$–dimensional space-time grids. In comparison with alternative approaches to handle $d+1$–dimensional grids such as sparse grids [8], the spacetree provides a simpler algorithmic environment due to its plain recursive definition. This is particularly advantageous for dynamically adaptive grids and in-situ mesh generation. However, the recursive construction yields a standard coarsening in time. From a storage complexity point of view, this is a pro. From a solver point of view, the standard coarsening in time has to be studied carefully.

3. A finite element space for the k-spacetree grid

We solve the heat equation (1.1) in its weak formulation, i.e.

$$\int_{\Omega\times(0,T)} \big(\partial_t u - \kappa\Delta u, \varphi\big)\ d(x,t) = \int_{\Omega\times(0,T)} \big(f,\varphi\big)\ d(x,t), \qquad \forall\varphi, \tag{3.1}$$

with $\varphi : \Omega\times(0,T) \mapsto \mathbb{R}$, u and φ sufficiently smooth, and derivatives that exist in the corresponding weak sense. The spacetree defines cells, vertices with 2^{d+1} adjacent cells on the same spacetree level, and vertices with less than 2^{d+1} adjacent cells on the same level. Latter vertices are *hanging nodes* in space-time. Let each non-hanging vertex correspond to one shape function $u_v(x,t) = \phi_v(x)\varrho_v(t)$ and one test function $\varphi_v(x,t) = \phi_v(x)\tilde{\varrho}_v(t)$. Both exhibit tensor product structure with $\phi_v(x)$ as d-linear hat function in space.

The linear combination of all these u_v is a generating system [7]. It is not a basis since multiple vertices might exist at the same position in space. Let $\mathbb{U}$ be all the shape functions induced by the k-spacetree that belong to a vertex that is neither hanging nor is there a non-hanging vertex on a finer spacetree level, i.e. shape functions belonging to vertices v_1 with

$$u_{v_1=(x,\ell)_1} \in \mathbb{U} \Rightarrow \nexists v_2 = (x,\ell)_2 : x_2 = x_1 \wedge \ell_2 > \ell_1.$$

v_1 and v_2 are not hanging. $\mathbb{U}$ induces a fine grid ansatz space and we approximate the solution u of (3.1) with a linear combination of shape functions from this $\mathbb{U}$. $\mathbb{V}$ is the corresponding test space. The leaves of the k-spacetree induce a non-conform tessellation $\mathscr{C}$ whose cells cover the whole discretised computational domain.

With a grid, shape functions, and test functions at hand, we reformulate the integral in (3.1) as a sum over integrals on k-spacetree leaf nodes. Depending on the choice of ϱ and $\tilde{\varrho}$, we end up with different equation systems. One of the simplest choices is to make ϱ also piecewise linear and $\tilde{\varrho}$ piece-wise constant and running backward in time, i.e. the

$\tilde{\varrho}_v$ support covers the 2^d cells adjacent to v that run backwards in time. The integral

$$\begin{aligned}
&\sum_{c\in\mathscr{C}} \int_c \left(\partial_t u, \varphi\right)\ d(x,t) - \kappa \int_c \left(\Delta u, \varphi\right)\ d(x,t) - \int_c \left(f, \varphi\right)\ d(x,t) \\
=&\sum_{c\in\mathscr{C}} \int_c \left(\partial_t u, \varphi\right)\ d(x,t) + \kappa \int_{(t_0,t_1)} \int_{(x_0,x_1)} \left(\nabla u, \nabla \varphi\right)\ dx\ dt \\
&- \int_c \left(f, \varphi\right)\ d(x,t), \quad u \in \mathbb{U}, v \in \mathbb{V},
\end{aligned}$$

integrated by parts in terms of the space derivatives then yields a Crank-Nicholson-type equation system sometimes referred to as cG(1)CN. Different choices of ϱ and $\tilde{\varrho}$ give us other standard time-stepping schemes such as an implicit Euler [3,8] (cG(1)dG(0)). Both the Crank-Nicolson-like scheme and the implicit Euler-like scheme are subject of further discussion. The "like" emphasises that these operators do not result from a time stepping scheme but from a Petrov-Galerkin or Ritz-Galerkin, respectively, formulation on the space-time domain. This gives us equations mirroring adaptive time-stepping: if the k-spacetree is not perfectly balanced, $\mathscr{C}$ is an adaptive mesh in both time and space. In regions with smaller $d+1$-dimensional hypercubes, more time steps are enforced than in regions with coarse spacetree cells.

An interesting property of both schemes is that they yield solely $2\cdot 3^d$-point stencils. They can be implemented matrix-free for Jacobi, Gauß-Seidel-type, or conjugate gradient solvers [17]. This is, on the one hand, important if the algorithm is memory bounded as there is no memory overhead to hold the matrix—a property that gains impact with new supercomputers released. On the other hand, it is particularly important if the grid changes permanently due to full multigrid or dynamic adaptivity as there is no runtime overhead for the matrix (re-)assembly. The stencils can be written down as linear combination

$$L(h) = C\cdot h^d M + C\cdot h^{d-2} A$$

acting cell-wisely on the unknowns of the "younger" cell face and

$$R(h) = C\cdot h^d M + C\cdot h^{d-2} A$$

acting on the "older" unknowns and also considering the right-hand side, C being a generic constant that might depend on the time step size. M and A are the standard d-dimensional finite element mass and stiffness stencils without h scaling with the shape and test function being hat functions in space.

$$\begin{aligned}
&\frac{1}{6h}\begin{bmatrix} 0 & 0 & 0 \\ h^2-6 & 4h^2+12 & h^2-6 \\ h^2 & 4h^2 & h^2 \end{bmatrix} \\
=&\underbrace{\frac{h}{6}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 4 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \frac{1}{h}\begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}}_{L(h)} - \underbrace{\frac{h}{6}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 4 & 1 \end{bmatrix}}_{R(h)}
\end{aligned}$$

DOI https://doi.org/10.4208/nmtma.m12si07 | Generated on 2024-12-19 04:50:30

switches from a d-dimensional to a $d+1$-dimensional space-time stencil representation and shows an implicit Euler for $d=1, \kappa=1$ with the upward direction being the time axis. There is no additional parameter for the time step size involved here, since all grid cells of the spacetree are hypercubes and, consequently, the time step size equals the mesh size h. If we squeeze the bounding box of the space-time domain, we choose different temporal and spatial mesh sizes, and the time step size and h do not coincide. While that provides additional flexibility for the time stepping scheme, it does not change the fact that the spacetree prescribes a standard coarsening in time. Again, this is a pro from a storage point of view since coarse grids are significantly smaller than fine grids. From a solver point of view, this standard coarsening in time has to be studied.

Finally, the adaptivity (in time) benefits directly from the space-time formulation. The constraint

$$\forall v_1 = (x,\ell)_1, \quad v_2 = (x,\ell)_2, \quad u_{v_1} \notin \mathbb{U} : \; u_{v_1} = u_{v_2}, \quad \ell_2 > \ell_1 \wedge u_{v_2} \in \mathbb{U} \tag{3.2}$$

gives us a trivial injection of the fine grid solution representation to the coarser levels, i.e. it defines a coarsened representation of the current solution on each grid level of the spacetree. At the same time, the space-time definition of the shape functions prescribes an interpolation for the hanging nodes. Both operations, the injection and the interpolation fit to the spacetree space-time formulation. They propose inherently to solve the equation systems with a full approximation storage scheme similar to [6] where the implementation follows [13]: the hanging nodes are interpolated due to the shape functions; the coarse level vertices that do not contribute to the fine grid are set to a value due to the constraint (3.2). Consequently, we evaluate the same stencil for all cells of one grid level that are adjacent to at least one fine grid vertex, and the hanging nodes—both in time and in space—do not require special stencils.

4. Multigrid

The discussion of our multigrid is twofold. We first analyse standard time stepping schemes before we switch to the actual space-time multigrid formulation and exploit the insights we obtained from the time stepping scheme.

4.1. A time stepping experiment

The operators we evaluate on the space-time grid equal a weighted linear combination of an operator stemming from the Laplacian and a mass operator. The sum is a standard elliptic operator. Multigrid for elliptic operators is comprehensively studied, so it is convenient to pick up this experience. In this subsection, we fix a time step size τ whereas the time step size in the k-spacetree later on equals the cell mesh size h. If we solve the one-time-step equation with a geometric Galerkin two-grid scheme where the prolongation P exclusively results from ϕ, the resulting coarse grid equation equals

$$\underbrace{P^T L(h,\tau) P \mathbf{e}_{kh}(\tau)}_{L(k\cdot h,\tau)\mathbf{e}_{kh}(\tau)} = \underbrace{P^T (R(h,\tau)\mathbf{u}_h(0) - L(h,\tau)\mathbf{u}_h(\tau))}_{P^T \mathbf{r}_h(\tau)}$$

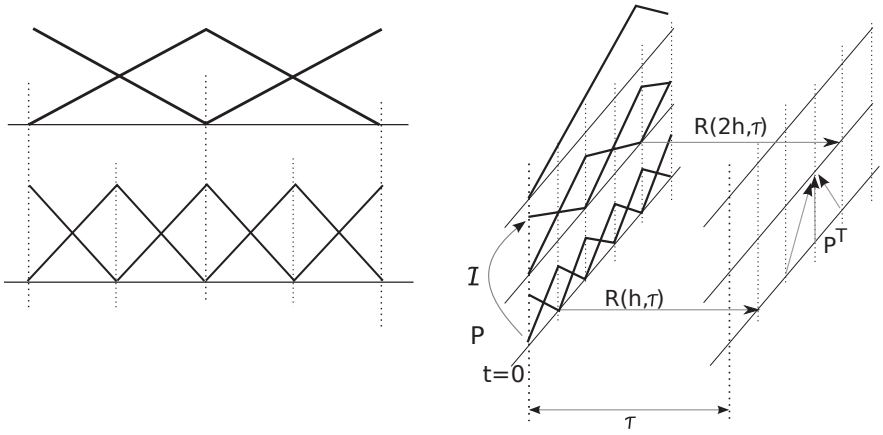

Figure 2: Left: generating system with two levels; Right: the injection I coarsens the solution due to a simple copy of the values ($k = 2$), $R(h)$ determines the right-hand side with respect to h, and the prolongation P and restriction P^T act only on one time slice (the left slice represents the solution $u(0)$, the right slice the time step $u(t)$).

with $\mathbf{e}_{kh}(t)$ approximating the coarse grid error on the time slice $t = \tau$. A recursive extension of the two-grid scheme to a multigrid algorithm is straightforward. The coarse grid operator is again a linear combination of the two elliptic operators with a different scaling in terms of h, and it equals the operator arising from a direct discretisation of the elliptic system on the coarser level. We again implement this geometric multigrid solver matrix-free.

Due to the generating system (Fig. 2), it is easy to implement a full approximation storage (FAS) scheme with an injection I acting on the solution, i.e. I represents the realisation of the constraint (3.2). Let I copy values to the coarser grid if there are two vertices at the same position in space. The implementation of this operator is trivial due to the construction rule of the k-spacetree. The fine grid equation of the two-grid FAS algorithm then remains

$$\begin{aligned} \mathbf{b}_h(\tau) &:= R(h,\tau)\mathbf{u}_h(0), \\ L(h,\tau)\mathbf{u}_h(\tau) &= \mathbf{b}_h(\tau) \end{aligned}$$

and is subject of μ_1 presmoothing steps. Yet, the FAS yields the modified coarse grid problem

$$\begin{aligned} \mathbf{u}_{k\cdot h}(\tau) &\leftarrow I\mathbf{u}_h(\tau), \\ \hat{\mathbf{u}}_h(\tau) &\leftarrow \mathbf{u}_h(t) - P\mathbf{u}_{k\cdot h}(\tau), \\ \hat{\mathbf{r}}_h(\tau) &\leftarrow \mathbf{b}_h(t) - L(h,\tau)\hat{\mathbf{u}}_h(\tau), \\ \mathbf{b}_{k\cdot h}(\tau) &\leftarrow P^T\hat{\mathbf{r}}_h(\tau), \end{aligned}$$

$$\begin{aligned} L(k\cdot h,\tau)\mathbf{u}_{k\cdot h}(\tau) &= \mathbf{b}_{k\cdot h}(\tau), \\ \mathbf{u}_h(\tau) &\leftarrow \hat{\mathbf{u}}_h(\tau) + P\mathbf{u}_{k\cdot h}(\tau). \end{aligned}$$

DOI https://doi.org/10.4208/nmtma.m12si07 | Generated on 2024-12-19 04:50:30

The back projection in the last line is typically followed by μ_2 postsmoothing steps. This representation follows [6] who introduced the notion of the hierarchical residual $\hat{\mathbf{r}}$. The recursive extension to a multigrid V-cycle $V(\mu_1,\mu_2)$ is again straightforward.

With a hierarchical generating system, with a multiscale representation of the solution due to I, and with a well-defined space-time interpolation P, local a posteriori error estimators relying on different grid resolutions (refinement history) [14,15] fall naturally into place. In our experiments, we however restrict to a pure h error estimator that refines the grid wherever the second derivative of the approximate is large: The ten percent of the vertices with the largest hierarchical surplus $\hat{\mathbf{u}}_h(t)$ measured in the maximum norm are marked and the algorithm then refines the surrounding cells. The coarsening is realised the other way round.

A full multigrid (FMG) algorithm follows from this scheme directly. We introduce a third order interpolation P_{FAS}. Then, we set up a sufficiently coarse grid, perform one $V(\mu_1,\mu_2)$-cycle, prolongate the approximate due to P_{FAS}, and continue iteratively until the finest grid is reached. For each step of this FMG, we need an appropriate right-hand side

$$\mathbf{b}_{h\cdot ik}(t) = R(h\cdot ik,\tau)\cdot \underbrace{I\cdot I\cdot \ldots I}_{k \text{ times}}\ \mathbf{u}_h(0) \tag{4.1}$$

that we define in terms of the injection and the mesh-size dependent right-hand side operator (Fig. 2).

Experiments with $V(\mu_1,\mu_2)$ cycles and $FMG(\mu_1,\mu_2)$ for some problems with sufficiently smooth solutions reveal two insights:

- The bigger the time step the bigger the "speedup" of a V-cycle compared to a simple Jacobi solver, if both solvers start with $\mathbf{u}_h(0)$ as initial guess for $\mathbf{u}_h(\tau)$. If the time step τ is embarrassingly small, the number of required Jacobi iterations equals or even underruns the number of multigrid cycles, i.e. Jacobi is computational cheaper.
- If the time step exceeds a certain threshold, FMG converges faster than the V-cycles. Otherwise, the V-cycles are faster in terms of total operations.

Consequently, there is a small range $0 < \tau \leq \tau_0$ where plain Jacobi is the fastest solver. This range is followed by a second range $\tau_0 < \tau \leq \tau_1$ where the V-cycle is the best choice. For a time step size $\tau > \tau_1$, FMG with (4.1) outperforms the two other solvers. This is a well-known fact that is due to the mass operator acting as Tikhonov regularisation term—the weight of the Laplacian $L(h,\tau)$ in the overall operator grows with τ.

The reason for this behaviour also becomes obvious, if we decompose the initial condition spatially into its Fourier modes. Then, the unperturbed initial condition evolves in time as

$$u(t) := \sum_k \tilde{u}_k(t) e^{-i\pi kx}. \tag{4.2}$$

It is noted that

$$\Delta u(t) = -\pi^2 \sum_k k^2 \tilde{u}_k(t) e^{-i\pi kx} \quad \text{and} \quad \partial_t u(t) = \sum_k (\partial_t \tilde{u}_k(t)) e^{-i\pi kx}$$

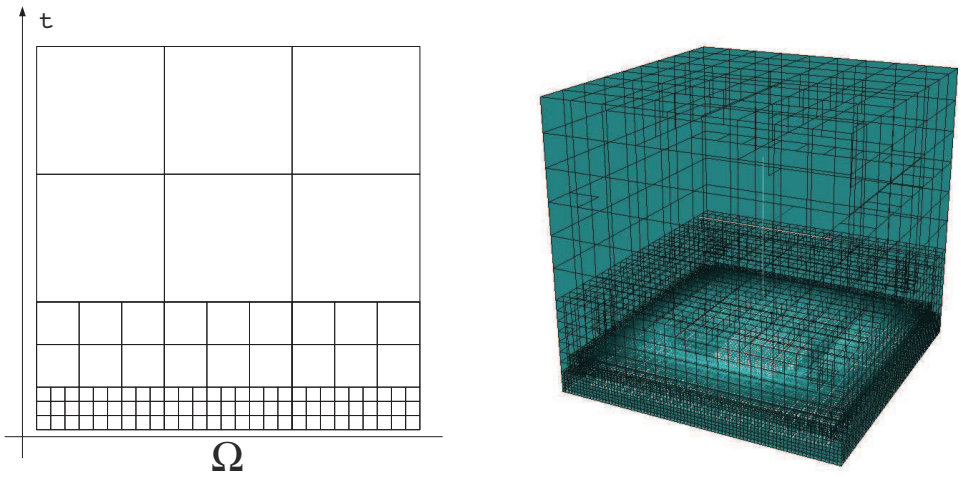

Figure 3: Left: Starting grid for the time-F-cycle; Right: Space-time grid for $d = 2$ after several F-subcycles with adaptive refinement.

imply the ordinary differential equation $\partial_t \tilde{u}_k(t) = -\pi^2 k^2 \ \tilde{u}_k(t)$, i.e. $\tilde{u}_k(t) = C \ e^{-\pi^2 k^2 \ t}$. The evolution of frequency components of the solution in time is the faster the bigger the frequency. Small stimuli of the solution at $t = 0$ are damped faster than low frequency stimuli. If the time step is very small, the difference of $\mathbf{u}_h(0)$ and $\mathbf{u}_h(\tau)$ comprises consequently primarily fast oscillations—low frequency, long-term behaviour cannot be resolved. This difference $u(t) - u(0)$ (the initial error in the equation system) dominated by high frequencies is damped efficiently by Jacobi. For large time steps, the constant extrapolation of $\mathbf{u}_h(0)$ to $\mathbf{u}_h(\tau)$ however also comprises low frequency components and, consequently, a multigrid solver then is of value.

Injection I yields a frequency decomposition of the solution on the hierarchical generating system. The coarser the grid the smoother the solution components that still are represented. At the same time, the hierarchical surplus $\hat{\mathbf{u}}$ denotes the high frequency contributions. FMG (4.1) derives the solution level after level, i.e. component-wise in terms of Fourier modes. First, low frequency components are projected to the next time step. The solver then derives a (low frequency) solution to this problem. With a smooth representation of the solution, it proceeds with higher frequencies. FMG with V-cycles as building block is more expensive than few pure V-cycles in terms of total computations. While this overhead is bounded by a constant, it nevertheless dominates if the difference of the solution and the preceding time step comprises only few low-frequency errors.

4.2. Full multigrid in time

The latter rationale justifies Parareal implementations that choose a Fourier-based time integrator for the coarse time predictor (see, e.g., [5]). This paper realises a similar procedure for the space-time grid.

We start with an unbalanced k-spacetree that resolves the initial condition as fine as

DOI https://doi.org/10.4208/nmtma.m12si07 | Generated on 2024-12-19 04:50:30

possible. Everywhere else, the k-spacetree is as coarse as possible. Such an adaptive space-time tree resolves the initial condition accurately (Fig. 3). On this grid, we apply one V-cycle followed by an error estimator evaluation that refines the grid in space and time. Here, the solution of the V-cycle is projected to the refined grid with a projection P_{FAS}. In accordance with Section 4.1, P_{FAS} exhibits tensor product style, its temporal interpolation equals P's temporal operator discussed below, but it is of higher order in space. We continue recursively. Such a scheme resembles nested iterations on the space-time grid, where we start with the maximal resolution on the initial condition hyperface of the space-time hypercube. It is a temporal full multigrid.

Compared to the one-time-step studies before, the linear operators belonging to $L(h)$ and $R(h)$ remain the same. The restriction P^T also remains the same, as we apply a strictly spatial elliptic multigrid scheme. Both the prolongation P and the higher order prolongation P_{FAS} however are augmented by an additional contribution forward time, i.e. they do not only act in space but also linearly on the successing time steps. A standard weightening stencil for $d = 1, k = 2$

$$P = \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 \\ 2 & 4 & 2 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{becomes} \qquad \hat{P} = \frac{1}{4}\begin{bmatrix} C_{\hat{P}} & 2C_{\hat{P}} & C_{\hat{P}} \\ 2 & 4 & 2 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.3}$$

with the upward direction representing the time axis and a well-suited $0 < C_{\hat{P}} \leq 1$ ($C_{\hat{P}} = 1$ mirrors a linear solution propagation in time). This reflects the fact that an update of the solution on any coarse level affects also all the upcoming time steps. Due to the modified prolongation, the solver is not a textbook Galerkin multigrid scheme anymore. Yet, the coarse grid operator

$$L(k \cdot h) = P^T L(h) \hat{P}$$

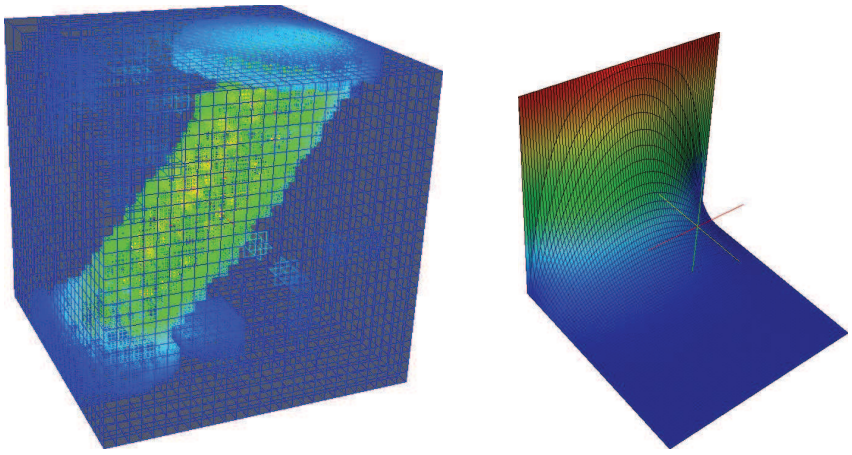

Figure 4: Left: $2+1$-dimensional spacetree (time axis from bottom to top) where a heat source moves over the two-dimensional surface; grid stems from a time F-cycle with dynamic adaptivity criterion; Right: a bar with constant temperature cools down due to homogeneous boundary conditions; $d = 1$ and time axis along from top left to bottom right.

DOI https://doi.org/10.4208/nmtma.m12si07 | Generated on 2024-12-19 04:50:30

still equals the direct spatial discretisation as P^T acts only spatially and the coarse operator collapses in time.

The standard coarsening in time picks up the insight that for very small time steps Jacobi is faster than a multigrid algorithm: time slices next to the initial condition are processed by a Jacobi or m-grid algorithm with m being small: only in regular time slots, several-grid smoothing is applied. It depends on the slice's position in time how many coarser grid lines coincide with this slice and, hence, how many coarser grid corrections are performed. The start layout of the grid also picks up the insight that for large time steps a full multigrid is faster than other solvers. While Jacobi, e.g., is applied on the first time step due to this start grid and the standard coarsening in time, the terminal time slice is tackled by full multigrid with the right-hand side (4.1). In-between, there is all sort of mixtures of Jacobi and nested m-grid V-cycles. The dynamic adaptivity that refines both in space and time picks up the idea that a fine time resolution is important if the solution in space-time changes rapidely, i.e. where a fine spatial mesh is required. Whereever hanging nodes in time occur, the spacetree paradigm is equivalent to adaptive time stepping. Whereever a load balancing scheme cuts the space-time domain along the time direction and makes the computing nodes handle the non-overlapping subdomains exchange the boundary only after each iteration, the spacetree paradigm is related to wavelet relaxation schemes [12]: each node updates "its" solution on a time-space subinterval, and then exchanges and updates the vertices along a time submanifold. As there might be hanging vertices along this manifold, the different subdomains are processed with different time meshes.

5. Results

We study our space-time multigrid algorithm for different settings on the $d+1$-dimensional hypercube. All experiments comprise homogeneous boundary conditions, i.e. $g = 0$. The first two examples simulate a cooling process where the initial condition equals

$$u_0 = \prod_{i\in\{1,\ldots,d\}} \sin(\pi x) \quad \text{with } f = 0, \quad \text{or} \tag{5.1}$$

$$u_0 = 1 \qquad\qquad \text{with } f = 0, \tag{5.2}$$

and the solution u becomes smaller with increasing time due to the lack of a stimulus f. The system cools down (Fig. 1 and Fig. 4, right). An analytical solution for the first initial condition results directly from (4.2). The third experiment starts with a zero initial condition. Then, a heat source is rotating around the center of the $d \leq 2$-dimensional hyperface or the center of one of the cube's faces respectively and heats up the system while the homogeneous boundary conditions again cool down the whole system (Fig. 1).

$$\begin{aligned} &\partial_t u - \kappa\Delta u = f, \\ &u_0 = 0,\ \kappa = 0.2,\ u|_{\partial\Omega} = 0 \text{ and } f \in \{0,1\}. \end{aligned} \tag{5.3}$$

The stimulus f is modelled by a small circle. Such a setup enters a transient phase at startup and then passes over into a periodic solution, i.e. a steady cycle.

All the experiments were conducted with the PDE solver framework Peano [17,18] and, thus, restrict to $k = 3$. This code traverses the $(k = 3)$-spacetree in a depth-first manner with a stack automaton and evaluates the operators stricly cell-wisely without assembling any global matrix. For the measurements, we used the BlueGene/P system Shaheen at the King Abdullah University of Science and Technology (KAUST) as test machine. Since scalability, load balancing, and domain decomposition are beyond the scope of this multigrid paper, we restrict ourselves to one node of this machine, i.e. four cores. A damped Jacobi smoother with $\omega = 0.8$ acts as smoother for the multigrid algorithms, the time integrator equals an implicit Euler to avoid instabilities stemming from rough input and stimulus data.

5.1. Memory footprint and memory behaviour

The realisation comes along with a very low memory footprint (Table 1). This memory footprint per vertex depends on d if we use an error estimator, as the evaluation criterion in the realisation analyses the hierarchical surplus along each of the $d+1$ coordinate axes. The code can handle adaptive regular Cartesian grids with $\mathcal{O}(10^7)$ vertices on one node with 4 GByte of main memory. Main memory of that size is for example characteristic for BlueGene/P architectures. That big problems per node are promising candidates for a good parallel efficiency, as the workload that can be handled is reasonably high.

Table 1: Memory footprint per vertex for different solvers. The numbers are given in bytes.

	$d=1$	$d=2$	$d=3$
Explicit Euler (without error estimator)	18	18	18
Implicit Euler (without error estimator)	26	26	26
Implicit Euler (with error estimator)	34	42	50
Crank-Nicolson (without error estimator)	26	26	26
Crank-Nicolson (with error estimator)	34	42	50

Due to the low memory requirements and the strict element-wise, matrix-free processing, the cost per vertex per iteration is independent of the total problem size handled by one node: cache effects are negligible since the cache hit rate never is smaller than 99,95% and, with that simple stencils of fixed size to evaluate, the time to get and write back one vertex to or from the L_2 cache coins the runtime. This is also in accordance with [17, 18], and allows us to specify the algorithm's performance in the abstract metric stencil updates, i.e. how often the algorithm has to apply a stencil such as the operator, prolongation, or restriction.

5.2. A time-stepping experiment

The motivation for our space-time approach in Section 4.1 stems from experiments with standard time stepping. We study the experiments (5.1) and (5.3) for $d = 2$ Table 2

Table 2: Stencil updates for different solvers for the first time step for different mesh sizes. The number in brackets is the number of iterations/(nested) V-cycles. Experiment (5.1) with $d = 2$.

	Jacobi			
τ	28^2	82^2	244^2	730^2
$5.00 \cdot 10^{-7}$	$4.08 \cdot 10^4$ (52)	$3.09 \cdot 10^5$ (46)	$1.19 \cdot 10^6$ (20)	$5.86 \cdot 10^6$ (11)
$1.00 \cdot 10^{-6}$	$4.23 \cdot 10^4$ (54)	$2.82 \cdot 10^5$ (42)	$7.74 \cdot 10^5$ (13)	$1.17 \cdot 10^7$ (22)
$5.00 \cdot 10^{-6}$	$4.39 \cdot 10^4$ (56)	$2.42 \cdot 10^5$ (36)	$9.53 \cdot 10^5$ (16)	$5.54 \cdot 10^7$ (104)
$1.00 \cdot 10^{-5}$	$4.78 \cdot 10^4$ (61)	$2.76 \cdot 10^5$ (41)	$1.67 \cdot 10^6$ (28)	(≥ 200)
$2.50 \cdot 10^{-5}$	$5.33 \cdot 10^4$ (68)	$3.09 \cdot 10^5$ (46)	$4.11 \cdot 10^6$ (69)	(≥ 200)
$5.00 \cdot 10^{-5}$	$5.64 \cdot 10^4$ (72)	$3.36 \cdot 10^5$ (50)	$8.16 \cdot 10^6$ (137)	(≥ 200)
	V(1,1)			
$5.00 \cdot 10^{-7}$	$4.23 \cdot 10^4$ (5)	$3.05 \cdot 10^5$ (4)	$2.06 \cdot 10^6$ (2)	$1.24 \cdot 10^7$ (1)
$1.00 \cdot 10^{-6}$	$5.07 \cdot 10^4$ (6)	$3.05 \cdot 10^5$ (4)	$2.06 \cdot 10^6$ (3)	$1.24 \cdot 10^7$ (2)
$5.00 \cdot 10^{-6}$	$5.07 \cdot 10^4$ (6)	$3.05 \cdot 10^5$ (4)	$2.06 \cdot 10^6$ (3)	$1.24 \cdot 10^7$ (2)
$1.00 \cdot 10^{-5}$	$5.07 \cdot 10^4$ (6)	$2.29 \cdot 10^5$ (3)	$2.06 \cdot 10^6$ (3)	$1.85 \cdot 10^7$ (3)
$2.50 \cdot 10^{-5}$	$5.07 \cdot 10^4$ (6)	$2.29 \cdot 10^5$ (3)	$2.06 \cdot 10^6$ (3)	$2.47 \cdot 10^7$ (4)
$5.00 \cdot 10^{-5}$	$4.23 \cdot 10^4$ (5)	$3.05 \cdot 10^5$ (4)	$3.43 \cdot 10^6$ (5)	$3.09 \cdot 10^7$ (5)
	V(3,3)			
$5.00 \cdot 10^{-7}$	$2.29 \cdot 10^4$ (2)	$2.10 \cdot 10^5$ (2)	$9.49 \cdot 10^5$ (1)	$8.56 \cdot 10^6$ (1)
$1.00 \cdot 10^{-6}$	$2.29 \cdot 10^4$ (2)	$2.10 \cdot 10^5$ (2)	$9.49 \cdot 10^5$ (1)	$8.56 \cdot 10^6$ (1)
$5.00 \cdot 10^{-6}$	$2.29 \cdot 10^4$ (2)	$2.10 \cdot 10^5$ (2)	$9.49 \cdot 10^5$ (1)	$8.56 \cdot 10^6$ (1)
$1.00 \cdot 10^{-5}$	$2.29 \cdot 10^4$ (2)	$1.05 \cdot 10^5$ (1)	$9.49 \cdot 10^5$ (1)	$8.56 \cdot 10^6$ (1)
$2.50 \cdot 10^{-5}$	$2.29 \cdot 10^4$ (2)	$1.05 \cdot 10^5$ (1)	$9.49 \cdot 10^5$ (1)	$8.56 \cdot 10^6$ (1)
$5.00 \cdot 10^{-5}$	$2.29 \cdot 10^4$ (2)	$2.10 \cdot 10^5$ (2)	$1.90 \cdot 10^6$ (2)	$1.71 \cdot 10^7$ (2)
	FMG(1,1)			
$5.00 \cdot 10^{-7}$	$1.31 \cdot 10^5$ (17)	$1.10 \cdot 10^6$ (17)	$6.51 \cdot 10^6$ (13)	$3.39 \cdot 10^7$ (10)
$1.00 \cdot 10^{-6}$	$1.22 \cdot 10^5$ (16)	$1.10 \cdot 10^6$ (17)	$5.83 \cdot 10^6$ (12)	$4.63 \cdot 10^7$ (12)
$5.00 \cdot 10^{-6}$	$1.39 \cdot 10^5$ (18)	$9.52 \cdot 10^5$ (15)	$4.45 \cdot 10^6$ (10)	$7.72 \cdot 10^7$ (17)
$1.00 \cdot 10^{-5}$	$1.39 \cdot 10^5$ (18)	$7.99 \cdot 10^5$ (13)	$5.83 \cdot 10^6$ (12)	$8.33 \cdot 10^7$ (18)
$2.50 \cdot 10^{-5}$	$1.39 \cdot 10^5$ (18)	$5.71 \cdot 10^5$ (10)	$8.57 \cdot 10^6$ (16)	$8.95 \cdot 10^7$ (19)
$5.00 \cdot 10^{-5}$	$1.22 \cdot 10^5$ (16)	$4.95 \cdot 10^5$ (9)	$9.94 \cdot 10^6$ (18)	$8.33 \cdot 10^7$ (18)
	FMG(3,3)			
$5.00 \cdot 10^{-7}$	$6.30 \cdot 10^4$ (7)	$4.74 \cdot 10^5$ (7)	$3.35 \cdot 10^6$ (7)	$2.16 \cdot 10^7$ (7)
$1.00 \cdot 10^{-6}$	$6.30 \cdot 10^4$ (7)	$5.79 \cdot 10^5$ (8)	$3.35 \cdot 10^6$ (7)	$3.02 \cdot 10^7$ (8)
$5.00 \cdot 10^{-6}$	$6.30 \cdot 10^4$ (7)	$4.74 \cdot 10^5$ (7)	$2.40 \cdot 10^6$ (6)	$4.73 \cdot 10^7$ (10)
$1.00 \cdot 10^{-5}$	$7.45 \cdot 10^4$ (8)	$3.69 \cdot 10^5$ (6)	$3.35 \cdot 10^6$ (7)	$4.73 \cdot 10^7$ (10)
$2.50 \cdot 10^{-5}$	$6.30 \cdot 10^4$ (7)	$2.65 \cdot 10^5$ (5)	$4.30 \cdot 10^6$ (8)	$4.73 \cdot 10^7$ (10)
$5.00 \cdot 10^{-5}$	$6.30 \cdot 10^4$ (7)	$2.65 \cdot 10^5$ (5)	$5.25 \cdot 10^6$ (9)	$4.73 \cdot 10^7$ (10)

and Table 3. The (spatial) grid both for $t = 0$ and $t = \tau$ is regular, we use an implicit Euler, and the solver stops as soon as the difference of two subsequent approximates of $u(t)$ runs below 10^{-10} in the maximum norm.

If the ratio of time step size τ to mesh size is sufficiently small, the pure Jacobi outperforms the other solvers. For a fixed mesh size, there are time step size domains where different variants of V-cycles are faster than full multigrid or vice versa. While Jacobi's performance degrades with increasing time step size, the difference between the V-cycles and the full multigrid becomes smaller. Finally, if the mesh size is sufficiently small and the time step size is sufficiently big, the full multigrid overtakes the other two solvers, no

Table 3: Stencil updates for different solvers for the first time step for different mesh sizes. The number in brackets is the number of iterations/(nested) V-cycles. Experiment (5.3) with $d = 2$.

	Jacobi			
τ	28^2	82^2	244^2	730^2
$5.00 \cdot 10^{-7}$	$7.21 \cdot 10^4$ (92)	$5.65 \cdot 10^5$ (84)	$3.04 \cdot 10^6$ (51)	$1.17 \cdot 10^7$ (22)
$1.00 \cdot 10^{-6}$	$7.21 \cdot 10^4$ (92)	$4.91 \cdot 10^5$ (73)	$2.44 \cdot 10^6$ (41)	$2.18 \cdot 10^7$ (41)
$5.00 \cdot 10^{-6}$	$6.98 \cdot 10^4$ (89)	$3.29 \cdot 10^5$ (49)	$1.37 \cdot 10^6$ (23)	$9.97 \cdot 10^7$ (187)
$1.00 \cdot 10^{-5}$	$6.82 \cdot 10^4$ (87)	$2.56 \cdot 10^5$ (38)	$2.68 \cdot 10^6$ (45)	(≥ 200)
$2.50 \cdot 10^{-5}$	$6.27 \cdot 10^4$ (80)	$1.75 \cdot 10^5$ (26)	$6.37 \cdot 10^6$ (107)	(≥ 200)
$5.00 \cdot 10^{-5}$	$5.57 \cdot 10^4$ (71)	$1.75 \cdot 10^5$ (26)	(≥ 200)	(≥ 200)
	V(1,1)			
$5.00 \cdot 10^{-7}$	$2.96 \cdot 10^5$ (35)	$2.74 \cdot 10^6$ (36)	$1.78 \cdot 10^7$ (26)	$6.80 \cdot 10^7$ (11)
$1.00 \cdot 10^{-6}$	$2.96 \cdot 10^5$ (35)	$2.67 \cdot 10^6$ (35)	$1.44 \cdot 10^7$ (21)	$7.41 \cdot 10^7$ (12)
$5.00 \cdot 10^{-6}$	$2.79 \cdot 10^5$ (33)	$1.91 \cdot 10^6$ (25)	$7.55 \cdot 10^6$ (11)	$1.24 \cdot 10^8$ (20)
$1.00 \cdot 10^{-5}$	$2.62 \cdot 10^5$ (31)	$1.52 \cdot 10^6$ (20)	$8.92 \cdot 10^6$ (13)	$1.36 \cdot 10^8$ (22)
$2.50 \cdot 10^{-5}$	$2.28 \cdot 10^5$ (27)	$9.91 \cdot 10^5$ (13)	$1.24 \cdot 10^7$ (18)	$1.42 \cdot 10^8$ (23)
$5.00 \cdot 10^{-5}$	$1.94 \cdot 10^5$ (23)	$7.62 \cdot 10^5$ (10)	$1.44 \cdot 10^7$ (21)	$1.36 \cdot 10^8$ (22)
	V(3,3)			
$5.00 \cdot 10^{-7}$	$1.37 \cdot 10^5$ (12)	$1.36 \cdot 10^6$ (13)	$8.54 \cdot 10^6$ (9)	$6.85 \cdot 10^7$ (8)
$1.00 \cdot 10^{-6}$	$1.37 \cdot 10^5$ (13)	$1.26 \cdot 10^6$ (12)	$6.64 \cdot 10^6$ (7)	$6.85 \cdot 10^7$ (8)
$5.00 \cdot 10^{-6}$	$1.26 \cdot 10^5$ (11)	$1.26 \cdot 10^6$ (12)	$3.80 \cdot 10^6$ (4)	$6.85 \cdot 10^7$ (8)
$1.00 \cdot 10^{-5}$	$1.26 \cdot 10^5$ (11)	$9.43 \cdot 10^5$ (9)	$4.75 \cdot 10^6$ (5)	$6.85 \cdot 10^7$ (8)
$2.50 \cdot 10^{-5}$	$1.03 \cdot 10^5$ (9)	$5.24 \cdot 10^5$ (5)	$5.69 \cdot 10^6$ (6)	$6.85 \cdot 10^7$ (8)
$5.00 \cdot 10^{-5}$	$9.14 \cdot 10^4$ (8)	$4.19 \cdot 10^5$ (4)	$6.64 \cdot 10^6$ (7)	$6.85 \cdot 10^7$ (8)
	FMG(1,1)			
$5.00 \cdot 10^{-7}$	$2.58 \cdot 10^5$ (32)	$2.48 \cdot 10^6$ (35)	$2.09 \cdot 10^7$ (34)	
$1.00 \cdot 10^{-6}$	$2.58 \cdot 10^5$ (32)	$2.48 \cdot 10^6$ (35)	$1.96 \cdot 10^7$ (32)	
$5.00 \cdot 10^{-6}$	$2.58 \cdot 10^5$ (32)	$2.25 \cdot 10^6$ (32)	$1.27 \cdot 10^7$ (22)	
$1.00 \cdot 10^{-5}$	$2.58 \cdot 10^5$ (32)	$2.09 \cdot 10^6$ (30)	$9.94 \cdot 10^6$ (18)	
$2.50 \cdot 10^{-5}$	$2.49 \cdot 10^5$ (31)	$1.71 \cdot 10^6$ (25)	$6.51 \cdot 10^6$ (13)	
$5.00 \cdot 10^{-5}$	$2.32 \cdot 10^5$ (29)	$1.33 \cdot 10^6$ (20)	$7.89 \cdot 10^6$ (15)	
	FMG(3,3)			
$5.00 \cdot 10^{-7}$	$1.32 \cdot 10^5$ (13)	$1.21 \cdot 10^6$ (14)	$1.09 \cdot 10^7$ (15)	
$1.00 \cdot 10^{-6}$	$1.32 \cdot 10^5$ (13)	$1.21 \cdot 10^6$ (14)	$9.99 \cdot 10^6$ (14)	
$5.00 \cdot 10^{-6}$	$1.32 \cdot 10^5$ (13)	$1.10 \cdot 10^6$ (13)	$6.19 \cdot 10^6$ (10)	
$1.00 \cdot 10^{-5}$	$1.32 \cdot 10^5$ (13)	$9.98 \cdot 10^5$ (12)	$5.25 \cdot 10^6$ (9)	
$2.50 \cdot 10^{-5}$	$1.20 \cdot 10^5$ (12)	$8.93 \cdot 10^5$ (11)	$3.35 \cdot 10^6$ (7)	
$5.00 \cdot 10^{-5}$	$1.20 \cdot 10^5$ (12)	$6.84 \cdot 10^5$ (9)	$4.30 \cdot 10^6$ (8)	

matter which number of pre- and postsmoothing steps is chosen. The advantage of pure Jacobi compared to the multigrid solvers is significant for the rough problem of (5.3). If the problem is smoother, the multiscale solvers are the method of choice as a rule of thumb.

For extremely small (compared to the mesh) time steps sizes, Jacobi is the method of choice. For sufficiently big (compared to the mesh) time steps sizes, full multigrid is the method of choice. In-between, the V-cycle is the fastest solver. For a time stepping algorithm, this implies that for very tiny time steps and rough problems, there is no need to tackle the resulting elliptic problem with a multiscale solver. For bigger time steps, multiscale solvers exhibit their superiority.

5.3. Solution propagation

We next study the cooling process from (5.2) for $d = 2$. An implicit Euler acts as time integrator due to the jump of the initial-boundary conditions, and we track the solution at three different points in the space-time domain: $(\frac{1}{2}, \frac{1}{2}, \frac{1}{81})^T$, $(\frac{1}{2}, \frac{1}{2}, \frac{21}{81})^T$, and $(\frac{1}{2}, \frac{1}{2}, \frac{40}{81})^T$. The measurements in Fig. 5 present the normalised solution, i.e. the current approximation divided by the converged result, for the three different points for three different solvers: a classical time stepping with $\tau = 1/80$ applying a V-cycle for each time step's elliptic problem, a V-cycle on the full regular space-time grid with $h = 1/80$ where the restriction and prolongation operator act solely spatially, and a time full multigrid that starts with a minimal adaptive space-time grid (Fig. 3) resolving the initial condition with $h = 1/80$ regularly. This F-cycle continues to refine the whole space-time domain until it ends up with a regular $h = 1/80$ space-time grid.

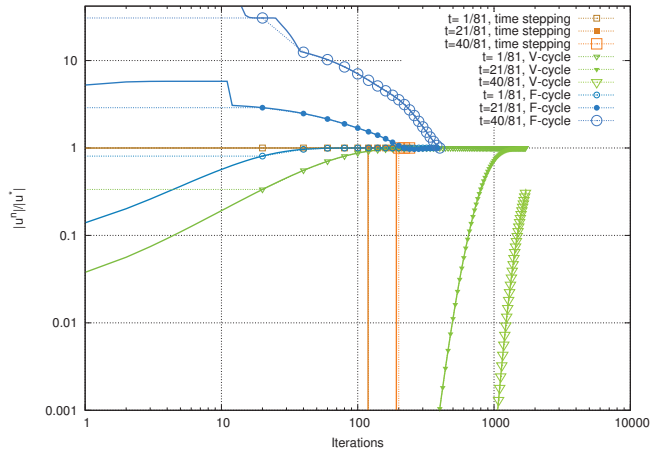

Figure 5: Convergence of three different time integration schemes for three different sample points.

With the time stepping, the value of $(\frac{1}{2}, \frac{1}{2}, \frac{21}{81})$ does not start to converge towards to the real solution before the 20 preceding time steps are solved (to convergence of the 20 time slices, the solver needs approximately 120 V-cycles). For $(\frac{1}{2}, \frac{1}{2}, \frac{40}{81})^T$, it is 40 time steps. With a pure V-cycle and standard coarsening on the whole space-time domain, the test point's convergence is worse, as the individual time slices are exclusively coupled on the finest grid. Due to the standard coarsening, two third of these time slices are processed by a Jacobi solver only: time slice one and two for example are tackled by a pure Jacobi smoother. The initial condition consequently has to be passed through these two slices before it affects the solution on the two-grid slice of time step three. The slower convergence of Jacobi compared to a spatial V-cycle makes the subsequent time slices represent irrelevant equations until the first two Jacobi time slice has converged. This effect is the stronger the further in the future the observation point is. A real V-cycle with standard coarsening on the space-time domain consequently does not pay off. Semicoarsening would resolve this problem but would induce a significant bigger total memory footprint.

The full multigrid in time gives a trend for the test points immediately, as it takes the

multiscale behaviour of the solution explicitly into account. However, the guess in the test points overshoots the solution significantly. This is due to the fact that we chose a linear time prolongation in (4.3), i.e. $C_{\hat{P}} = 1$, which is far too optimistic. Instead, the time prolongation should reproduce the exponential decrease derived from (4.2).

If an application outside of the solver is interested in rough guesses or trend delivered on time, the holistic full multigrid with our adaptive space-time grid is of value. This effect is the stronger the later the sample point's time. The result after a few iterations is only a good guess if the solution to the experiment is sufficiently smooth. Furthermore, a code used for such rough guesses should take into account a more sophisticated interpolation operator. However, the value of the technique in principle is demonstrated here.

5.4. Adaptivity and total runtime

Finally, we study the overall time to solution for the cooling process (5.1) and the rotating heat source with our simple adaptivity criterion for $d = 2$. In the experimental setups we always started with a regular grid for the initial time step, and the time stepping applied $V(1,1)$-cycles to solve the individual elliptic problems.

The cooling process yields a very smooth solution in time (Fig. 4), and, hence, the

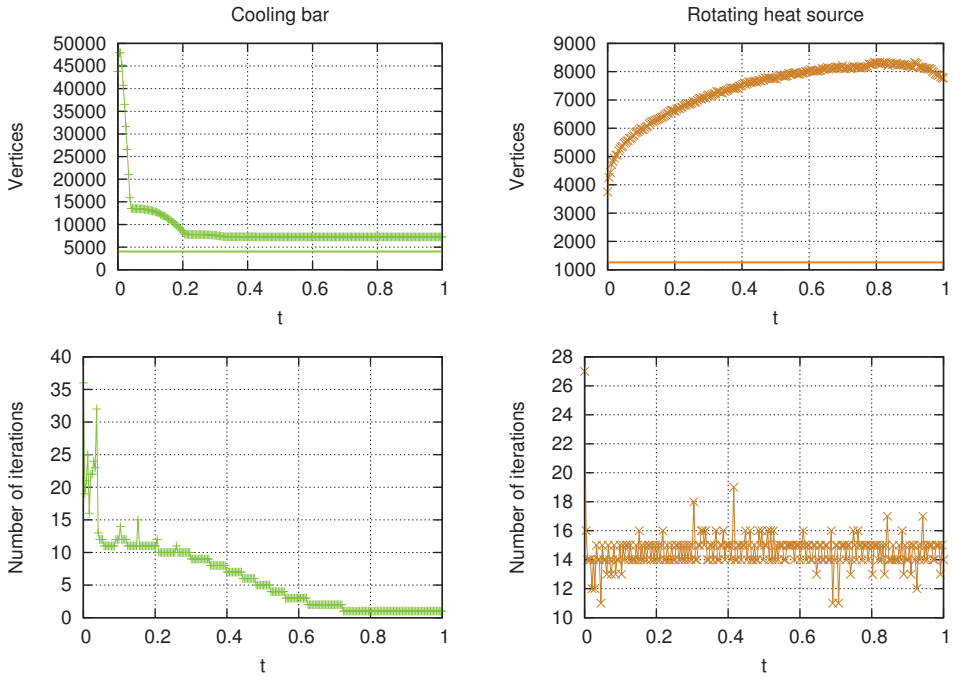

Figure 6: Number of vertices and number of iterations per time step for a time stepping scheme with dynamic grid refinement in space. The horizontal threshold in the upper diagrams gives the number of vertices divided by 243 for the same experimental setting on a space-time grid, i.e. divided by the number of time steps.

DOI https://doi.org/10.4208/nmtma.m12si07 | Generated on 2024-12-19 04:50:30

time stepping scheme reduces the overall number of vertices with the time advancing. Furthermore, the number of iterations required per time step decreases (Fig. 6 as example for one setup with initial mesh size $h = \frac{1}{243}$), as the solution diffuses. For the example studied, the full multigrid yields a grid with 977240 vertices, i.e. the number of vertices broken down to a single time step of the time stepping is by a factor of two smaller.

The rotating heat source yields a solution where the region of interest that requires a high resolution does not disappear in time and, furthermore, moves throughout the simulation (Fig. 3 for a snapshot and Fig. 4 for the space-time behaviour). Both the time stepping and the space-time full multigrid adopt the space-time grid around the regions of fast solution changes, i.e. around the stimulus. The number of vertices for the time stepping trends to roughly 8000 (Fig. 6). While the number of vertices then remains in the same order, the adaptivity criterion permanently removes vertices from the simulation and adds additional vertices around the heat source. In contrast to the cooling bar, the number of iterations required per time step does not decrease in time: the algorithm uses the solution of the last time step as initial guess for the solution of the elliptic problem, and this guess always lacks the information that the heat source already has moved on. A space-time simulation run with the same experiment parameters yields a grid with 306972 vertices, i.e. the number of vertices broken down on a single time step of the time stepping is by a factor of seven smaller than for the time stepping.

In a second experimental setup, the time stepping is allowed to evaluate a dynamic time step size criterion and adopt the time step size. This criterion mirrors the space-time adaptivity criterion. For the cooling process, the time stepping increases this step size as the changes within one time interval become smaller and smaller. This is reasonable, as the initial stimulus (initial condition) is damped further and further. Such a grid plotted as grid sequence in time equals exactly the grid the space-time full multigrid derives. It is similar to the discretisation in Fig. 3. For the rotating heat source, the time stepping cannot increase the step size as the changes from one time interval to the next are not decreasing. The time stepping has to use the same time step size permanently while the space-time approach refines the grid in space and time only round the heat stimulus.

In terms of runtime, i.e. stencil evaluations, the time stepping outperforms the space-time multigrid for the cooling process: If we select a start grid with $h = \frac{1}{27}$, the space-time multigrid required 4.36 times more stencil evaluations than the corresponding time stepping scheme. If we select a start grid with $h = \frac{1}{81}$, the space-time multigrid required 6.76 times more stencil evaluations than the corresponding time stepping scheme. The difference grew with more accurate mesh sizes, so we stopped the measurements. The space-time multigrid cannot cope with a well-tailored time stepping for such smooth problems.

In terms of runtime, i.e. stencil evaluations, the time stepping also outperforms the space-time multigrid for the rotating heat source: For a minimum mesh size of $h = \frac{1}{27}$, the time stepping is faster than the full multigrid by a factor of 7.46. For a minimum mesh size of $h = \frac{1}{81}$, the time stepping is faster than the full multigrid by a factor of 3.85. For a minimum mesh size of $h = \frac{1}{243}$, the time stepping is faster than the full multigrid by a factor of 2.10. The runtime gap closes with finer mesh sizes. Consequently, the space-time

multigrid is advantageous if and only if the number of vertices broken down on a time stepping time step is significantly smaller (the solution is not smooth in time), if and only if the time stepping scheme cannot choose large time steps due to some solution regions with a very high activity (the solution is not smooth in space), and if and only if the time stepping has to do a significant number of V-cycles per time slice.

6. Conclusion and outlook

Our space-time approach tackling the heat equation exploits the multi-frequency representation which our full approximation storage scheme in combination with the k-spacetree yields. With the whole problem at hand, the workload to be distributed among a parallel computer exceeds the workload of a time stepping scheme by magnitudes. This makes the approach a promising candidate for massively parallel systems. Here, our algorithm queues into a long tradition of time parallelisation approaches. The algorithm's adaptivity in space and time makes it related to local time stepping schemes, and the implementation of algorithms for steady cycle problems benefits from the fact that initial and terminal condition are available at the same time. Due to the multiscale prolongation of the solution in time, the algorithm is well-suited for challenges where a first guess of the solution's behaviour is important. Such requirements arise, e.g., in the context of computational steering, fluid-structure interaction, or calibration. With the whole space-time discretisation at hand, the algorithmic paradigm finally is a promising candidate for all kinds of reverse problems where it is important to be able to go back in time.

The runtime of the algorithm at the first glance is disappointing, as the time stepping outperforms the holistic approach. However, three aspects put this shortcoming into perspective: First, we compared solely sequential runtimes. Parallel scalability is not taken into account. Second, for sufficiently rough problems where the adaptivity gains impact the performance gap closes. Third, the multigrid algorithm is far from being tuned. Actually, here is a substantial backlog demand for well-chosen prolongation operators in time.

The latter aspect is one of three natural extension points of this work. First, all the multigrid ingredients such as prolongation in time, smoother, windowed relaxation, and better choice of shape functions have to be studied in detail. Also, the whole aspect of semi-coarsening may not be neglected on the long run. However, it is important to derive schemes that coarse in time aggressively, as many applications and unanswered questions demand for longer and longer simulation time intervals—the growth in time steps exceeds the improvement in the spatial resolution. Besides the multigrid aspects, we second have to study the parallel scalability of the approach. Here, the derivation of tailored load balancing metric, heuristics, and strategies is of special interest. Any load balancing approach benefits from the fact that the space-time formulation yields more workload than a standard time stepping scheme. However, the fact that the workload "grows along the time axis" as the solver continues to derive the space-time solution deserves special attention for the load balancing. It either has to take this phenomenon into account explicitly or has to perform a permanent rebalancing. The heat equation is an important building block of many applications. Future work, finally, has to extend this model to more complex

settings, application environments, and to new classes of problems where the space-time discretisation shows its strengths. The *h*-adaptivity provided by the spacetree inherently simplifies the handling of complicated computational domains. However, many problems such as problems with jumping material coefficients or domain boundaries that are not aligned with the coordinate axes require the solver to be able to handle different stencils for different vertices as well. In such a case, the matrix-free realisation can be preserved, while the need to store a stencil per vertex—perhaps in combination with a BoxMG-like [4] approach—increases the memory footprint of the solver. This is turn reduces the maximum size of a problem that can be handled in its space-time formulation on a single core—an effect that has to be evaluated in future work.

Acknowledgments This publication is partially based on work supported by Award No. UK-c0020, made by the King Abdullah University of Science and Technology (KAUST).

References

[1] M. R. Benioff and E. D. Lazowska. *Report to the President. Computational Science: Ensuring America's Competitiveness*. President's Information Technology Advisory Committee, 2005.

[2] A. Borzì. Multigrid methods for parabolic distributed optimal control problems. *J. Comput. Appl. Math.* 157(2), pp. 365–382, 2003.

[3] Ch. Brandenburg, F. Lindemann, M. Ulbrich, and S. Ulbrich. Advanced numerical methods for pde constrained optimization with application to optimal design in navier stokes flow. In S. Engell, A. Griewank, M. Hinze, G. Leugering, R. Rannacher, V. Schulz, M. Ulbrich, and S. Ulbrich, editors, *Constrained Optimization and Optimal Control for Partial Differential Equations*, to appear. Birkhäuser Verlag, 2010.

[4] J. E. Dendy. Black box multigrid. *J. Comput. Phys.* 48(3), pp. 366–386, 1982.

[5] M. Gander and S. Vandewalle. On the Superlinear and Linear Convergence of the Parareal Algorithm. In O. B. Widlund and D. E. Keyes, editors, *Domain Decomposition Methods in Science and Engineering XVI*, volume 55 of *LNCS*, pp. 291–298. Springer-Verlag, Berlin Heidelberg, 2007.

[6] M. Griebel. *Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hiearchischen-Transformations-Mehrgitter-Methode*, volume 342/4/90 A. SFB-Bericht, Dissertation, Technische Universität München, 1990.

[7] M. Griebel. *Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen*. Teubner Skripten zur Numerik. Teubner, Habilitation, Technische Universität München, 1994.

[8] M. Griebel and D. Oeltz. A Sparse Grid Space-Time Discretization Scheme for Parabolic Problems. *Computing*. 81(1), pp. 1–34, 2007.

[9] W. Hackbusch. Parabolic multi-grid methods. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Sciences and Engineering VI*, pp. 189–197. North-Holland, 1984.

[10] G. Horton. The time-parallel multigrid method. *Commun. Appl. Numer. M.* 8(9), pp. 585–595, 1992.

[11] G. Horton and S. Vandewalle. A Space-Time Multigrid Method For Parabolic PDEs. Technical report, Universität Erlangen, 1993.

[12] J. Janssen and S. Vandewalle. Multigrid waveform relaxation on spatial finite element meshes: The discrete-time case. *SIAM. J. Numer. Anal.*, 33, pp. 456–474, 1993.

[13] S. F. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations*. SIAM, 1989.

[14] J.M. Melenk and B.I. Wohlmuth. On residual-based a posteriori error estimation in hp-fem. *Adv. Comput. Math.* 15, pp. 311–331, 2001.
[15] W. F. Mitchell and M. A. McClain. A survey of *hp*-adaptive strategies for elliptic partial differential equations. In T. E. Simos, editor, *Recent Advances in Computational and Applied Mathematics*, pp. 227–258. Springer-Verlag, Netherlands, 2011.
[16] H. van der Ven. An adaptive multitime multigrid algorithm for time-periodic flow simulations. *J. Comput. Phys.* 227(10), pp. 5286–5303, 2008.
[17] T. Weinzierl. *A Framework for Parallel PDE Solvers on Multiscale Adaptive Cartesian Grids*. Verlag Dr. Hut, 2009.
[18] T. Weinzierl and M. Mehl. Peano – A Traversal and Storage Scheme for Octree-Like Adaptive Cartesian Multiscale Grids. *SIAM. J. Sci. Comput.* 2010. accepted.
[19] D. E. Womble. A Time-Stepping Algorithm for Parallel Computers. *SIAM. J. Sci. Stat. Comp.* 11(5), pp. 824–837, 1990.