Real-time implementation of the Kirchhoff plate equation using finite-difference time-domain methods on CPU

Real-Time Implementation of the Kirchhoff Plate Equation using Finite-Difference Time-Domain Methods on CPU

Zehao Wang, Miller Puckette, Tom Erbe
Department of Music
University of California, San Diego
La Jolla, CA, USA
{zehaowang, msp, tre}@ucsd.edu

Stefan Bilbao
Acoustics and Audio Group
University of Edinburgh
Edinburgh, United Kingdom
sbilbao@ed.ac.uk

ABSTRACT

In this paper, we develop real-time applications including virtual instruments and plate reverb using the Kirchhoff plate model with loss and tension terms. Finite-difference time-domain (FDTD) methods are employed, and they are implemented on central processing units (CPUs) and optimized by loop unrolling or advanced vector extensions (AVX), enabling these programs to be executed in real time at fast speeds. applications such as virtual plate instruments and plate reverb using FDTD schemes are developed as puredata (Pd) externals which can serve as objects in puredata, a real-time graphical programming environment for audio and graphics. In these applications, multiple inputs (excitation or audio signal) and outputs whose positions are free to change in real time are allowed, and physical parameters can be dynamically manipulated in real time, allowing users the ability to produce both realistic sound and new sounds not possible to generate in the real world. Additionally, these Pd externals can also be used as modules to build Pd patches, which provides more possibilities for experimental artists.

1. INTRODUCTION

Physical modeling sound synthesis is an application of numerical simulation techniques, and those algorithms, especially models based on partial differential equations (PDEs), are in general computationally intensive compared with earlier abstract synthesis methods. Finite-difference time-domain (FDTD) schemes are one of the major numerical approaches to solving PDEs in electromagnetics [1] and acoustics [2] and have been used for sound synthesis for some time [3]. For most musical instruments, a coupled set of partial differential equations, describing the displacement of the instruments' vibrating components like strings, bars, membranes, or plates, will be numerically solved through different methods of approximation, for example, finite-element methods, finite-volume methods, meshfree methods, or finite-difference methods we used in this paper. One major advantage of PDE-based physical models is that users can manipulate the interpretable control parameters even throughout the synthesis process, which allows them to generate novel sounds using parameters with real-world meanings.

The main drawback to using these algorithms is their relatively large computational expense. However, Recent advances in both algorithm design and computer hardware have the potential to enable the generation of sound synthesis with complex physical models in real time. One major advance in hardware capability is low-level parallelization on the CPU by single instruction multiple data (SIMD) intrinsics, such as streaming SIMD extensions (SSE) and advanced vector extensions (AVX) intrinsics, which are suitable for accelerating operations in FDTD schemes when numerically solving PDEs [4] at the scale of applications in the field of musical acoustics. There are an increasing number of studies about using different hardwares to perform these algorithms in real time. Researchers have explored the potential of field-programmable gate array (FPGA) [5–7] or graphics processing units (GPUs) [8–10] for fast or real-time implementation of FDTD-based simulation algorithms, although the performance is acceptable, such hardware is not suitable for real-time audio plugin or software development which is highly used in music production and live performance. Recently, researchers also studied the use of CPUs for fast [10] or real-time [11, 12] sound simulation using FDTD methods. However, the use of AVX intrinsics on CPUs for accelerating physical modeling sound synthesis is still an under-explored area.

For developing real-time audio software or plugins, one can use the well-known C++ framework, JUCE [13]. Max/MSP [14] and Puredata [15] externals are also possible options and both are heavily used by experimental musicians in conjunction with existing objects in their own patches.

In this paper, we will implement a simple FDTD scheme of the Kirchhoff thin plate equation on rectangular regions, which is used to synthesize the sound of plates or shells (when curved) [3], and introduce the optimization techniques used to accelerate the operations of FDTD schemes, which are used in physical modeling sound synthesis through PDEs. As applications, the design of simple Pd externals of this plate model with multiple dynamic inputs and outputs is also discussed. Numerical results show the speed of our optimized implementation methods by loop unrolling and/or SIMD parallelization using AVX intrinsics, which are capable to be performed in real time.

2. MODEL

The plate model described in this section, assumed made of steel, is described by a number of physical and numerical parameters, provided, for reference, in the tables below.

Symbol	Value	Description	Units
L	0.4	length parameter	m
ϵ	1.5	domain aspect ratio	1
ρ	7.86×10^3	material density	kg/m^3
H	0.0021	plate thickness	m
E	2.06×10^{11}	Young's modulus	Pa
ν	0.3 ($< 1/2$)	Poisson's ratio	1
T	1	tension/meter	N/m
σ_0, σ_1	$5.5, 0.001$	damping parameter	1/s, m^2/s
c_0	1	amplitude of $c_{\rm rc}$	m
$r_{\rm hw}$	0.3	half-width of $c_{\rm rc}$	m
SR	44100	sample rate	Hz
k	1/SR	time step interval	s
h, h_x, h_y	0.0437	grid spacing	m

2.1 Kirchhoff Thin Plate Equation

The Kirchhoff model of a uniform thin isotropic plate [16] is defined as

$$\rho H u_{tt} = -D\Delta\Delta u, \tag{1}$$

where $u(x, y, t)$ is the plate deflection in a transverse direction at time t; here, subscripts t indicate partial differentiation with respect to time t, and Δ is the two-dimensional Laplacian operator (and $\Delta\Delta$ is the biharmonic operator). ρ is a material density, H is the plate thickness, and the flexural rigidity D is defined by

$$D = \frac{EH^3}{12(1-\nu^2)}$$

in terms of Young's modulus E and Poisson's ratio ν. When spatially scaled with respect to a length parameter L, the Kirchhoff model may be written as

$$u_{tt} = -\kappa^2\Delta\Delta u, \tag{2}$$

where

$$\kappa^2 = \frac{D}{\rho H L^4}.$$

The followings are two sets of boundary conditions (clamped and simply supported, respectively) over the boundary ∂U of the domain U:

$$u = \frac{\partial}{\partial \mathbf{n}} u = 0 \qquad \text{clamped,} \tag{3a}$$

$$u = \Delta_{\rm n} u = 0 \qquad \text{simply supported,} \tag{3b}$$

where $\frac{\partial}{\partial \mathbf{n}}$ and $\Delta_{\rm n}$ denote the first-order and second-order scalar derivative in the normal direction of the boundary ∂U.

A simple, perceptually correct plate model with loss and tension is shown as follows,

$$u_{tt} = -\kappa^2\Delta\Delta u + \gamma^2\Delta u - 2\sigma_0 u_t + 2\sigma_1 \Delta u_t, \tag{4}$$

where $\gamma = \sqrt{T/\rho H}$, σ_0 with $\sigma_1 = 0$ indicates frequency-independent damping, and if $\sigma_1 \neq 0$, increasing damping at higher frequencies is modeled. Note that it is possible to rewrite σ_0 and σ_1 in terms of two chosen values of decay constant T_{60} at specified frequencies [1] .

[1] If $\sigma_1 = 0$, we have $\sigma_0 = 6 * \log 10/T_{60}$.

2.2 Excitation

A raised cosine distribution could be used as an all-purpose forcing condition to model both plucks and strikes [3]:

$$c_{\rm rc}(x, y) = \begin{cases} \frac{c_0}{2}\left(1 + \cos\left(\pi d_i(x, y)/r_{\rm hw}\right)\right), & d_i(x, y) \le r_{\rm hw} \\ 0, & d_i(x, y) > r_{\rm hw} \end{cases} \tag{5}$$

where $d_i(x, y) = \sqrt{(x - x_i)^2 + (y - y_i)^2}$, c_0 is the amplitude, $r_{\rm hw}$ is the half-width, and (x_i, y_i) is the center of the excitation. One can also couple other complex excitation models like mallets and bows with the plate model to make it more musically interesting and plausible. Each excitation term could be added to the plate models as an extra term [2]

$$X(x, y, t) = e_{\rm exc}(x, y)F(t), \tag{6}$$

where $e_{\rm exc}(x, y)$ is a distribution representing the spatial extent of the excitation, and $F(t)$ is a force divided by the total plate mass. For virtual instruments, one can use $c_{\rm rc}$ as the excitation with $F(t) = \mathbf{1}(t = 0)$; a direct choice of the excitation distribution for plate reverb is the Dirac function, i.e., $e_{\rm exc}(x_0, y_0) = \delta(x - x_0, y - y_0)$ with the input waveform as the force term $F(t)$. For multiple inputs, one can simply add multiple Xs to the right-hand side of equation (4).

A single output is defined as

$$m(x_{\rm o}, y_{\rm o}, t) = u(x_{\rm o}, y_{\rm o}, t) \tag{7}$$

is normally drawn from the transverse displacements of the plate at given coordinate $(x_{\rm o}, y_{\rm o})$.

3. NUMERICAL SCHEMES AND IMPLEMENTATION

In this paper, these equations will only be solved on a rectangular domain $U = \{x, y | 0 \le x \le L_x,\ 0 \le y \le L_y\}$. Assume the indices of grid points in U for discretization are $GP = \{(l, m) \mid l = 0, 1, \dots, N_x,\ m = 0, 1, \dots, N_y\}$. However, to involve the fourth-order difference operators, we need to create a "virtual" boundary defined as $u_{l,m}$, where the (l, m)-pairs satisfying $(l+1)(l - N_x - 1)(m + 1)(m - N_y - 1) = 0$, $l = -1, 0, 1, \dots, N_x + 1$, $m = -1, 0, 1, \dots, N_y + 1$. Thus, the indices for grid points with both natural and virtual boundaries are $EGP = \{(l, m) \mid l = -1, 0, 1, \dots, N_x + 1,\ m = -1, 0, 1, \dots, N_y + 1\}$. Define the set for internal grid points as $IGP = \{(l, m) \mid l = 1, 2, \dots, N_x - 1,\ m = 1, 2, \dots, N_y - 1\}$, the set for natural boundaries as $NBD = GP \backslash IGP$, the set for virtual boundaries as $VBD = EGP \backslash GP$ and the set for boundaries as $BD = NBD \cup VBD$. Grid spacings for x-axis and y-axis are $h_x = L_x/N_x$ and $h_y = L_y/N_y$, respectively, and here we assume $h_x = h_y$, i.e., equal grid spacing. A demonstration for the partition of grid points is shown in Fig. 1.

[2] For the Kirchhoff thin plate (1), just add it to the right-hand side of the equation.

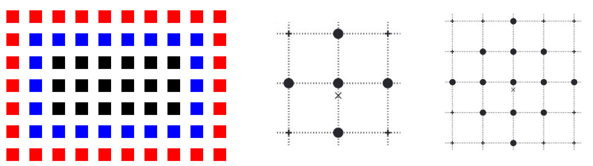

Figure 1. (a) $N_x = 7, N_y = 4$, red: VBD, blue: NBD, black: IGP; (b,c): stencils of $\delta_{\Delta\boxplus}$ and $\delta_{\Delta\boxplus,\Delta\boxplus}$, respectively

3.1 Grid functions and finite difference operators

Let $u_{l,m}^n$ denote the numerically updated u at time step n on grid point $(l,m) \in EGP$. First, we define temporal operators can be defined as follows,

$$e_{t\pm}u_*^n = u_*^{n\pm1}, \qquad 1u_*^n = u_*^n, \tag{8a}$$

$$\delta_{t\pm} \triangleq \pm\frac{1}{k}(e_{t\pm} - 1), \quad \delta_{t\cdot} \triangleq \frac{1}{2k}(e_{t+} - e_{t-}), \tag{8b}$$

$$\delta_{tt} = \delta_{t+}\delta_{t-} = \frac{1}{k^2}(e_{t+} - 2 + e_{t-}), \tag{8c}$$

The following are spatial operators [3] :

$$e_{x\pm}u_{l,m}^n = u_{l\pm1,m}^n \tag{9a}$$

$$\delta_{x\pm} \triangleq \pm\frac{1}{h}(e_{x\pm} - 1), \quad \delta_{x\cdot} \triangleq \frac{1}{2h}(e_{x+} - e_{x-}) \tag{9b}$$

$$\delta_{xx} = \delta_{x+}\delta_{x-} = \frac{1}{h^2}(e_{x+} - 2 + e_{x-}) \tag{9c}$$

$$\delta_{\Delta\boxplus} = \delta_{xx} + \delta_{yy} = \frac{1}{h^2}(e_{x+} + e_{x-} + e_{y+} + e_{y-} - 4) \tag{9d}$$

$$\delta_{\Delta\boxplus,\Delta\boxplus} \triangleq \delta_{\Delta\boxplus}\delta_{\Delta\boxplus} = \delta_{xx}\delta_{xx} + \delta_{yy}\delta_{yy} + 2\delta_{xx}\delta_{yy} \tag{9e}$$

Stencils or footprints for discrete Laplacian and biharmonic operators $\delta_{\Delta\boxplus}, \delta_{\Delta\boxplus,\Delta\boxplus}$ are shown in Fig. 1. The above difference operators are used to approximate differential operators in PDEs and derive schemes for numerical solutions. Interpolation operators and discrete spreading functions for approximating the Dirac delta function defined as follows are also needed when we numerically solve PDEs.

A zeroth-order (westward/southward) interpolant $I_0(x_\mathrm{o}, y_\mathrm{o})$ operating at position $(x_\mathrm{o}, y_\mathrm{o})$ is defined by

$$I_0(x_\mathrm{o}, y_\mathrm{o})u = u_{l_\mathrm{o},m_\mathrm{o}}, \tag{10}$$

where $l_\mathrm{o} = \mathrm{floor}(x_\mathrm{o}/h)$, $m_\mathrm{o} = \mathrm{floor}(y_\mathrm{o}/h)$. Another choice that is more accurate is the bilinear interpolant,

$$\begin{aligned} I_1(x_\mathrm{o}, y_\mathrm{o})\,u = &(1-\alpha_{x,\mathrm{o}})(1-\alpha_{y,\mathrm{o}})\,u_{l_\mathrm{o},m_\mathrm{o}} \\ &+ (1-\alpha_{x,\mathrm{o}})\,\alpha_{y,\mathrm{o}}u_{l_\mathrm{o},m_\mathrm{o}+1} \\ &+ \alpha_{x,\mathrm{o}}(1-\alpha_{y,\mathrm{o}})\,u_{l_\mathrm{o}} \\ &+ 1, m_\mathrm{o} + \alpha_{x,\mathrm{o}}\alpha_{y,\mathrm{o}}u_{l_\mathrm{o}+1,m_\mathrm{o}+1} \end{aligned}, \tag{11}$$

[3] y-counterparts of the first three operators could be defined similarly.

where $\alpha_{x,\mathrm{o}} = x_\mathrm{o}/h - l_\mathrm{o}$ and $\alpha_{y,\mathrm{o}} = y_\mathrm{o}/h - m_\mathrm{o}$.

Spreading grid functions $J_0(x_i, y_i)$ and $J_1(x_i, y_i)$, operating at position (x_i, y_i) which approximate the 2D Dirac delta function $\delta(x - x_i, y - y_i)$, may be similarly defined as the duals to these interpolants as follows,

$$J_{l,m,0}(x_\mathrm{i}, y_\mathrm{i}) = \frac{1}{h^2}\begin{cases} 1, & l = l_\mathrm{i}, m = m_\mathrm{i} \\ 0, & \text{else} \end{cases}$$

$$\begin{aligned} &J_{l,m,1}(x_\mathrm{i}, y_\mathrm{i}) \\ &= \frac{1}{h^2}\begin{cases} (1-\alpha_{x,\mathrm{i}})(1-\alpha_{y,\mathrm{i}},) & l = l_\mathrm{i}, m = m_\mathrm{i} \\ (1-\alpha_{x,\mathrm{i}})\,\alpha_{y,\mathrm{i}}, & l = l_\mathrm{i}, m = m_\mathrm{i}+1 \\ \alpha_{x,\mathrm{i}}(1-\alpha_{y,\mathrm{i}}), & l = l_\mathrm{i}+1, m = m_\mathrm{i} \\ \alpha_{x,\mathrm{i}}\alpha_{y,\mathrm{i}}, & l = l_\mathrm{i}+1, m = m_\mathrm{i}+1 \\ 0, & \text{else} \end{cases} \end{aligned} \tag{12}$$

where $l_\mathrm{i} = \mathrm{floor}(x_\mathrm{i}/h)$, $m_\mathrm{i} = \mathrm{floor}(y_\mathrm{i}/h)$, $\alpha_{x,\mathrm{i}} = x_\mathrm{i}/h - l_\mathrm{i}$, and $\alpha_{y,\mathrm{i}} = y_\mathrm{i}/h - m_\mathrm{i}$.

3.2 A simple explicit FDTD scheme for the Kirchhoff thin plate

For the linear plate model with loss (4), its explicit scheme could be written as follows,

$$\delta_{tt}u = -\kappa^2\delta_{\Delta\boxplus,\Delta\boxplus}u + \gamma^2\delta_{\Delta\boxplus}u - 2\sigma_0\delta_{t\cdot}u + 2\sigma_1\delta_{t-}\delta_{\Delta\boxplus}u, \tag{13}$$

when written out in full,

$$\begin{aligned} u_{l,m}^{n+1} = &[(2 - 20\mu^2 - 4\psi - 8\sigma_1\xi)u_{l,m}^n \\ &+ \left(8\mu^2 + \psi + 2\sigma_1\xi\right)(u_{l,m+1}^n + u_{l,m-1}^n \\ &+ u_{l+1,m}^n + u_{l-1,m}^n) - 2\mu^2(u_{l+1,m+1}^n + u_{l+1,m-1}^n \\ &+ u_{l-1,m+1}^n + u_{l-1,m-1}^n) - \mu^2(u_{l,m+2}^n + u_{l,m-2}^n \\ &+ u_{l+2,m}^n + u_{l-2,m}^n) - (1 - \sigma_0 k - 8\sigma_1\xi)u_{l,m}^{n-1} \\ &- 2\sigma_1\xi(u_{l,m+1}^{n-1} + u_{l,m-1}^{n-1} + u_{l+1,m}^{n-1} + u_{l-1,m}^{n-1})] \\ &/\,(1 + \sigma_0 k) \end{aligned} \tag{14}$$

for $(l,m) \in IGP$, where $\mu = \frac{\kappa k}{h^2}$, $\psi = \frac{\gamma^2 k^2}{h^2}$, and $\xi = \frac{k}{h^2}$ are the scheme parameters.

3.2.1 *Matrix-form update rules*

Let $[u^n]$ be the by-column(y-axis) flattened vector of $u_{l,m}^n$ restricted on IGP, i.e.,

$$[u^n]_{(l-1)(N_y-1)+m} = u_{l,m}^n,$$

and $\tilde{u}^n$ be the grid matrix, where $\tilde{u}_{m,l}^n = u_{l,m}^n$, normally $(l,m) \in IGP$ so $\tilde{u}^n \in \mathbb{R}^{(N_y-1)\times(N_x-1)}$, but we can also extend it to other grid points in BD and VBD. Thus, the matrix-form update rules for (14) have the following general formula:

$$[u^{n+1}] = S[u^n] - T[u^{n-1}], \tag{15}$$

where $S, T \in \mathbb{R}^{(N_y-1)(N_x-1)\times(N_y-1)(N_x-1)}$ are given as follows for different boundary conditions:

$$\begin{aligned} S &= \tfrac{2I-\mu^2(L^2+P)+(\psi+2\sigma_1\xi)L}{1+\sigma_0 k} \\ T &= \tfrac{(1-\sigma_0 k)I+2\sigma_1\xi L}{1+\sigma_0 k} \end{aligned}, \quad \text{clamped}, \tag{16a}$$

$$\begin{aligned} S &= \tfrac{2I-\mu^2 L^2+(\psi+2\sigma_1\xi)L}{1+\sigma_0 k} \\ T &= \tfrac{(1-\sigma_0 k)I+2\sigma_1\xi L}{1+\sigma_0 k} \end{aligned}, \quad \text{simply supported}, \tag{16b}$$

here

$$L = \begin{bmatrix} A & I & & & & 0 \\ I & A & I & & & \\ & \cdots & \cdots & & & \\ & & \cdots & \cdots & & \\ & & & I & A & I \\ 0 & & & & I & A \end{bmatrix}, \in \mathbb{R}^{NN\times NN},$$

$$P = \begin{bmatrix} 2N & 0 & & & & 0 \\ 0 & N & 0 & & & \\ & \cdots & \cdots & & & \\ & & \cdots & \cdots & & \\ & & & 0 & N & 0 \\ 0 & & & & 0 & 2N \end{bmatrix}, \in \mathbb{R}^{NN\times NN}, \tag{17}$$

where $NN = (N_y-1)(N_x-1)$, L is the discrete Laplacian operator, $I \in \mathbb{R}^{(N_y-1)\times(N_y-1)}$ is the identity matrix, and

$$A = \begin{bmatrix} -4 & 1 & & & & 0 \\ 1 & -4 & 1 & & & \\ & \cdots & \cdots & & & \\ & & \cdots & \cdots & & \\ & & & 1 & -4 & 1 \\ 0 & & & & 1 & -4 \end{bmatrix}, \in \mathbb{R}^{(N_y-1)\times(N_y-1)},$$

$$N = \begin{bmatrix} 1 & 0 & & & & 0 \\ 0 & 0 & 0 & & & \\ & \cdots & \cdots & & & \\ & & \cdots & \cdots & & \\ & & & 0 & 0 & 0 \\ 0 & & & & 0 & 1 \end{bmatrix}, \in \mathbb{R}^{(N_y-1)\times(N_y-1)}.$$

3.2.2 Numerical boundary conditions

According to (3), we can obtain the following numerical boundary conditions,

$$u = \delta_{x\pm}u = \delta_{y\pm}u = 0 \quad \text{clamped}, \tag{18a}$$

$$u = \delta_{xx}u == \delta_{yy}u = 0 \quad \text{simply supported}, \tag{18b}$$

where the direction of difference operators is related to the direction of the normal vector n.

The first part of these boundary conditions, $u = 0$, means $u_{l,m} = 0$ for $(l,m) \in NBD$. In another part of boundary conditions with difference operators, the virtual boundary should satisfy the following equations, respectively:

$$u_{l,m} = 0 \quad \text{clamped}, \tag{19a}$$

$$\begin{aligned} u_{-1,m} &= -u_{1m}, \\ u_{(N_x+1),m} &= -u_{(N_x-1),m}, \\ u_{l,-1} &= -u_{l,1}, \\ u_{l,(N_y+1)} &= -u_{l,(N_y-1)} \end{aligned} \quad \text{simply supported}, \tag{19b}$$

for $(l,m) \in VBD$.

3.2.3 Numerical stability condition

Using von Neumann analysis [3], we can derive the stability condition of scheme (14) as follows,

$$h \geq \sqrt{\sigma_1 k + \sqrt{\sigma_1^2 k^2 + 16\kappa^2 k^2}}. \tag{20}$$

3.2.4 Numerical excitation and output

The discrete versions of the excitation (6) and output (7) are defined as follows, respectively,

$$X^n(x_i, y_i) = J_p(x_i, y_i)F(nk), \tag{21}$$

$$m^n(x_\text{o}, y_\text{o}) = I_p(x_\text{o}, y_\text{o})u^n, \tag{22}$$

where J_p is a p-th order spreading function which could be chosen from J_0 and J_1 defined in (12), and I_p is a p-th order interpolation function which could be chosen from I_0 and I_1 defined in (10) and (11), respectively. For applications, multiple excitations and outputs are allowed by adding multiple Xs to the scheme and reading out multiple ms at proper time steps.

3.3 Implementation

In the real-time implementation in C/C++, all matrices and vectors are using double-precision array data type, and matrices are stored by flattening them by column. All code should be compiled with at least -O3/-Ofast and -mavx2 [4] -march=native flags.

3.3.1 Matrix-free temporal difference updates

Consider the numerical schemes (14) in Section 3.2, they are all explicit schemes which means no linear systems are required to be solved for the temporal difference updates. We can either use a sparse-matrix form update (15) for all grid points or use matrix-free updates (14) for each grid point in loops. For sparse-matrix form update, the loop unrolling or parallelization is used for sparse matrix-vector multiplication which will be described later. Here we choose matrix-free temporal difference updates for the real-time implementation since updates in this form can have the fastest speed among all methods we mentioned in this paper, numerical results and comparison are presented later. The general pseudo-code for matrix-free temporal difference updates is shown in Algorithm 1. And the optimization technique is to unroll [5] or parallelize using AVX intrinsics [6] every for-loop in Algorithm 1, and use AVX's fused operations like fused multiply-add (fmadd) instead of two separate operations if supported. Demonstration for these optimization techniques is shown in Fig. 2.

[4] For AVX2. Choose -mavx for AVX and -mavx512f for AVX512.

[5] In our implementation, the number of iterations unrolled for a single iteration is 8.

[6] If double precision is used, num_simd = 2 for AVX, 4 for AVX2, and 8 for AVX512. In my implementation, num_simd is set to 4 and AVX2 with double precision is used.

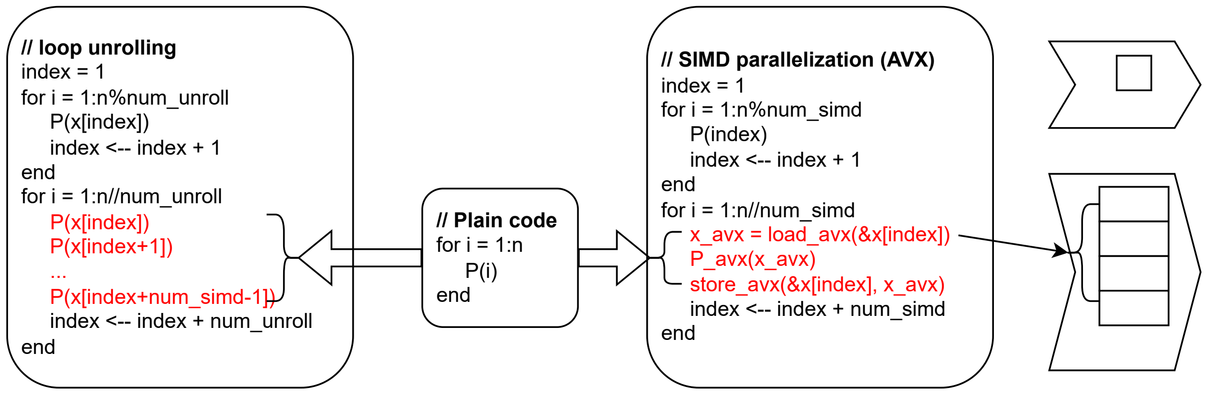

Figure 2. Demonstration for loop unrolling and SIMD parallelization. P means the operation for each time step (including all parameters and coefficients from some data), and x is the array-type data used for and updated by the operation. For the operation using SIMD parallelization, several entries of the data need to be loaded to some consecutive memory addresses, then the single instruction will be applied to these loaded entries simultaneously, and finally, store the result back to the data. Here for AVX2, the number of these double-precision entries for each iteration (num_simd) is 4.
The first for-loop in each box indicates the remainder of the total number to be unrolled which cannot be unrolled or parallelized as a whole.

abbr. of platform	machine	operating system	supported instruction sets
MBA	MacBook Air 2020 with 1.1 GHz 4-core Intel i5	MacOS 12	AVX, AVX2
MBP	MacBook Pro 2021 with 10-core M1 Max	MacOS 12	N/A
PC_Linux	AMD Ryzen 7 5800X 8-core 4.7 GHz	Ubuntu 22.04 LTS	AVX, AVX2
PC_Win	AMD Ryzen 7 5800X 8-core 4.7 GHz	Windows 11	AVX, AVX2

Table 1. Systems and hardware for numerical experiments

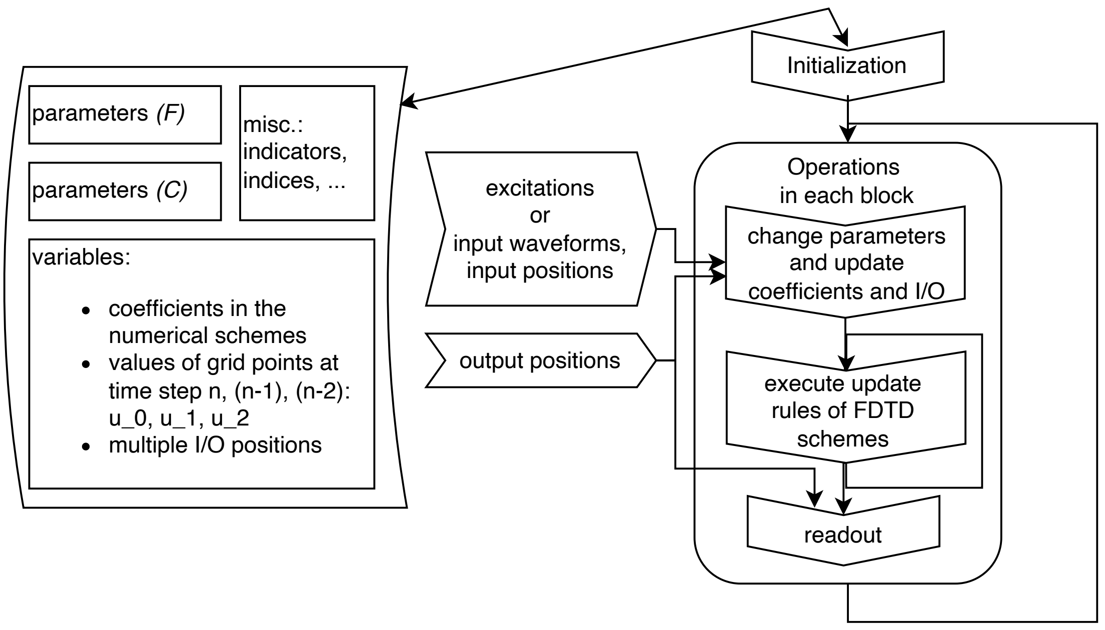
Initialization
parameters (F)
parameters (C)
misc.: indicators, indices, ...
variables:
coefficients in the numerical schemes
values of grid points at time step n, (n-1), (n-2): u_0, u_1, u_2
multiple I/O positions
excitations or input waveforms, input positions
output positions
Operations in each block
change parameters and update coefficients and I/O
execute update rules of FDTD schemes
readout

Figure 3. A diagram for Pd externals with algorithms using FDTD schemes.

Algorithm 1 Matrix-free temporal difference updates

Input: All required parameters for coefficients Θ, values on all grid points $\in EGP$ for time step n and $n-1$: $\tilde{u}^1$, $\tilde{u}^2$

Output: values on all grid points $\in EGP$ for time step $n+1$ and n: $\tilde{u}$ (the current time step),$\tilde{u}^1$ (the previous time step)

```
function MFTemporalDifferenceUpdates($\Theta, \tilde{u}^1, \tilde{u}^2$)
    Initialize a zero matrix $\tilde{u}$ on all grid points in $EGP$
    Update for all grid points in $IGP$
    for $(l,m) \in IGP$ in some order[7] do
        $\tilde{u}_{l,m} \leftarrow f_{(l,m),\Theta}(\tilde{u}^1, \tilde{u}^2)$    ▷ $f_{(l,m),\Theta}$ is some linear functions of $\tilde{u}^1, \tilde{u}^2$ restricted on some grid points centered in $(l,m)$ with coefficients $\Theta$ are from (14)
    end for
    Set the boundary for $\tilde{u}$
    for $(l,m) \in VBD$ do    ▷ Ignore $NBD$ since all values on grid points in $NBD$ are 0
        $\tilde{u}_{l,m} \leftarrow$ some boundary conditions like (18, 19)
    end for
    return $\tilde{u}, \tilde{u}^1$
end function
```

3.3.2 *Alternative implementations*

Here, we'll briefly introduce the implementation methods for matrix-form update rules derived in Section 3.2.1. Consider formulas (16) of S and T in (15), the most computationally intensive part is $L^2[u^t]$ and $L[u^t]$. For these multiplications, the following implementation methods optimized for the structure of L are proposed to achieve fast speeds. The sparse-matrix format used here is adapted from the Compressed Sparse Row (CSR) format, but we don't need to store all entries of the sparse matrices since there are only a few constant values appeared in the matrices, and the number of those constant values are independent of the size of the matrices [8] . Notice that, L^2 could be regarded as a single matrix or a composition of two matrices, which means we have two possible approaches for a given implementation method.

Implementation for matrix-form update rules using sparse block matrix-vector multiplication:

A direct way is to implement the standard sparse block matrix-vector multiplication for $L^2[u^t]$ and $L[u^t]$. Here, L^2 is treated as a single matrix. Each non-zero block of the matrix will be multiplied by the vector and accumulated to the final result, and for the multiplication between each block and the vector, the operations between each row of the block and the vector are explicitly computed without for-loops since the amount of non-zero entries is fixed. The loop-unrolling or SIMD parallelization is applied to the iteration within each block by column. The abbreviation of this implementation is mvmul.

Implementation for matrix-form update rules using convolutions:

[8] For L, we only need to store 2 values $[1, -4]$; for L^2, we only need to store 6 values $[-1, -2, 8, -20, -18, -19]$, where -18 and -19 are for corners and edges, respectively.

Consider the difference operators $\delta_{\Delta\boxplus}$ and $\delta_{\Delta\boxplus,\Delta\boxplus}$ in (13) or the structure of L and L^2, the operations between the difference operators/matrices and vector are similar to convolutions. Thus, we can use operations like convolution kernels to do the sparse block matrix-vector multiplication. For L^2 and L, sizes of the kernels K_{L^2}, K_L are 5×5 and 3×3, respectively, and the kernels are shown as follows,

$$K_L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$K_{L^2} = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -2 & 8 & -2 & 0 \\ -1 & 8 & -20 & 8 & -1 \\ 0 & -2 & 8 & -2 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}. \tag{23}$$

We can also use a composition of two convolutions with a kernel of K_L for the convolution with a kernel of K_{L^2} [9] . For implementation, all operations between the kernel and the vector are explicitly computed without for-loops, and the loop-unrolling or SIMD parallelization is applied to the iteration of the kernel's shifts by column. The abbreviations of the above implementations are conv_K_L^2 and conv_K_L, respectively.

3.4 Numerical results

In this paper, four platforms including three machines and three operating systems listed in Table 1 are used for numerical experiments. All numerical results listed below are calculated for 1 second with simply supported boundary condition, one raised-cosine excitation, and one fixed-position output, a grid size of $N_x = 28$ and $N_y = 19$ which is enough to synthesize quality sound, and other parameters are chosen as described in Sec. 2.

Here we list time costs in seconds for the full plate algorithm in (14, 15, 3.2.1) with different optimization techniques for matrix-free temporal difference updates and sparse matrix operations of matrix-form updates in Table 2. The version of Matlab we used is R2022a, and plain code means no optimization techniques are used. As we can see in the numerical results, the best time costs of each platform are much smaller than the duration of synthesized sound, which means the optimized implementation has the capability to be executed in real time since the most computationally intensive part of the program is the matrix-vector multiplication, allowing proper numbers of multiple (moving) inputs and outputs will not impose significant extra time costs.

[9] If we use a single 5×5 kernel K_{L^2}, we need to manipulate the edges and corners after the convolution according to the boundary condition; if we use a composition of two convolutions with a 3×3 kernel K_L, we don't need to do such manipulations under the simply supported boundary conditions.

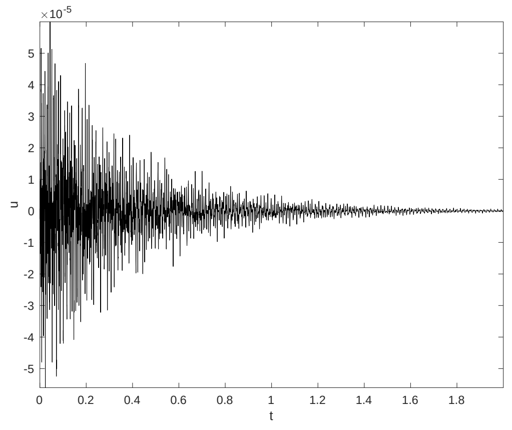

Figure 4. The plot of a 2-second sound example generated by Matlab. Sound examples could be found on this GitHub repo: https://github.com/water45wzh/linear-plate-demo.

	MBA	MBP	PC_Linux	PC_Win
plain code	0.403	0.192	0.200	0.265
loop unrolling	0.326	0.185	0.202	0.252
AVX2	0.372	N/A	0.349	0.393
plain code (-O3)	0.122	**0.031**	0.026	0.040
loop unrolling (-O3)	0.117	0.054	0.052	0.065
AVX2 (-O3)	**0.039**	N/A	**0.025**	**0.034**
Matlab	0.394	0.239	0.191	0.187
mvmul_unroll	0.873	0.825	0.742	0.813
mvmul_AVX2	0.742	N/A	0.612	0.698
mvmul_unroll (-O3)	0.267	0.124	0.168	0.176
mvmul_AVX2 (-O3)	0.224	N/A	0.151	0.157
conv_K_L_unroll	0.671	0.311	0.353	0.385
conv_K_L_AVX2	0.609	N/A	0.414	0.432
conv_K_L_unroll (-O3)	0.167	0.044	0.045	0.045
conv_K_L_AVX2 (-O3)	0.163	N/A	0.054	0.058
conv_K_{L^2}_unroll	0.807	0.374	0.401	0.440
conv_K_{L^2}_AVX2	0.705	N/A	0.475	0.493
conv_K_{L^2}_unroll (-O3)	0.191	0.068	0.067	0.061
conv_K_{L^2}_AVX2 (-O3)	0.179	N/A	0.055	0.058

Table 2. Numerical results of the full plate algorithm with matrix-free temporal difference updates and matrix-form updates. We only test matrix-form updates in Matlab implementation. The first three rows are compiled without -O3 flag. Bold results are the best results for each column.

The plot of a 2-second sound example is shown in Fig. 4.

4. APPLICATION: PD EXTERNALS

In this section, I will briefly describe the methodology of building Pd externals with algorithms using FDTD schemes. The first thing is to classify all parameters into two categories: parameters in F that are fixed throughout the runtime and parameters in C that may change throughout the runtime, shown as follows,

$$F = \{\mathrm{SR}, k, h, h_x, h_y, L, \epsilon\},$$

$$C = \{\rho, H, E, \nu, T, \sigma_0, \sigma_1, in, out\},$$

where in and out are all positions for excitations and outputs, respectively.

Pd externals will be loaded as objects in Pd patches. There are two types of objects in Pd, *control* objects and *tilde* objects. The control objects carry out their function sporadically, as a result of one or more types of events [15]. The tilde objects compute audio samples, which are continuous streams of numbers. The audio portion of the patch is always running, whether MIDI messages arrive or not. The externals we build here are tilde objects. And tilde objects are processed block by block, where blocks serve as audio buffers, and the number of samples in each block is normally 64, 128, 256, or 512 according to Pd settings.

When a tilde object is loaded in Pd patches, its *new* module and *setup* module will be called and the object will thus be initialized with a class containing all variables and parameters (like F and C) required to be stored or changed throughout the runtime. All updates from the numerical schemes need to be executed block by block in a pointer of the *perform* function, and it needs to be called in the *dsp* method by *dsp_add* which is a module for handling the audio signal stream in this object and will be added to the *setup* module. A diagram for demonstration is shown in Fig. 3. Two Pd externals are built using the above methodology, one is thinplate∼, a virtual plate instrument with 8 excitation positions and 8 outputs and another is platereverb∼, a plate reverb with 8 inputs and 8 outputs. A diagram of Pd externals with algorithms using FDTD schemes is shown in Fig. 3.

5. CONCLUSIONS

In this paper, the implementation of FDTD schemes for the Kirchhoff thin plate equation with different optimization techniques and the development of corresponding Pd externals with multiple movable inputs and outputs has been presented. Numerical comparisons between different implementation approaches and optimization techniques are performed on different platforms, showing that SIMD parallelization using AVX intrinsics for matrix-free temporal difference updates achieves the fastest speed which is much smaller than the threshold of real-time execution.

Future work may involve using the proposed optimization strategy for other synthesis algorithms with similar sparse-matrix operations, like 2-D digital waveguide meshes [17], and developing real-time implementations of models with nonlinearity, such as the von Kármán plate model [3, 18], which could synthesize more perceptually correct sound. Another possible direction is to design more user-friendly applications using FDTD schemes that can interact with users through some hardware and software controllers, which means a mapping between the models' control parameters and the controllers' keys, buttons, knobs, and sliders need to be carefully designed.

Acknowledgments

The authors are grateful to the reviewers for their constructive suggestions for revision.

6. REFERENCES

[1] D. M. Sullivan, *Electromagnetic simulation using the FDTD method.* John Wiley & Sons, 2013.

[2] D. R. Bergman, *Computational acoustics: theory and implementation.* John Wiley & Sons, 2018.

[3] S. Bilbao, *Numerical sound synthesis: finite difference schemes and simulation in musical acoustics.* John Wiley & Sons, 2009.

[4] D. Kusswurm, *Modern Parallel Programming with C++ and Assembly.* Springer, 2022.

[5] E. Motuk, R. Woods, S. Bilbao, and J. McAllister, "Design methodology for real-time fpga-based sound synthesis," *IEEE Transactions on signal processing*, vol. 55, no. 12, pp. 5833–5845, 2007.

[6] E. Motuk, R. Woods, and S. Bilbao, "Parallel implementation of finite difference schemes for the plate equation on a fpga-based multi-processor array," in *2005 13th European Signal Processing Conference.* IEEE, 2005, pp. 1–4.

[7] F. Pfeifle and R. Bader, "Real-time finite difference physical models of musical instruments on a field programmable gate array (fpga)," in *Proc. of the 15th Int. Conference on Digital Audio Effects (DAFx-12)*, 2012, pp. 17–21.

[8] S. Bilbao, A. Torin, P. Graham, J. Perry, and G. Delap, "Modular physical modeling synthesis environments on gpu," in *ICMC*, 2014.

[9] C. J. Webb, "Computing 3d finite difference schemes for acoustics–a cuda approach," Master's thesis, University of Edinburgh, 2010.

[10] C. J. Webb, "Parallel computation techniques for virtual acoustics and physical modelling synthesis," Ph.D. dissertation, The University of Edinburgh, 2014.

[11] M. G. Onofrei, S. Willemsen, and S. Serafin, "Real-time implementation of a friction drum inspired instrument using finite difference schemes," in *2021 24th International Conference on Digital Audio Effects (DAFx).* IEEE, 2021, pp. 168–175.

[12] S. Willemsen, N. S. Andersson, S. Serafin, and S. Bilbao, "Real-time control of large-scale modular physical models using the sensel morph," in *16th sound and music computing conference.* Sound and Music Computing Network, 2019, pp. 151–158.

[13] ROLI, "Juce." [Online]. Available: https://juce.com

[14] C. '74, "Max/msp." [Online]. Available: https://cycling74.com/products/max

[15] "Pd manual." [Online]. Available: http://msp.ucsd.edu/Pd_documentation/

[16] P. M. Morse and K. U. Ingard, *Theoretical acoustics.* Princeton university press, 1986.

[17] C. Chafe, "Extensions to the 2d waveguide mesh for modeling thin plate vibrations."

[18] T. von Kármán, "Festigkeitsprobleme im maschinenbau," *Encyklopädie der Mathematischen Wissenschaften*, vol. 4, no. 4, pp. 311–385, 1910.