A Faust Implementation of Coupled Finite Difference Schemes

David Südholt (dsudho20@student.aau.dk) [1], **Riccardo Russo** (riccardo.russo19@unibo.it) [1,3], and **Stefania Serafin** (sts@create.aau.dk) [2]

[1]*CREATE*, **Aalborg University**, Copenhagen, Denmark
[2]*Multisensory Experience Lab, CREATE*, **Aalborg University**, Copenhagen, Denmark
[3]**University of Bologna**, Bologna, Italy

ABSTRACT

Physical models using finite difference schemes (FDS) are typically implemented using mutable data structures. The FDS library of the Faust programming language, where such data structures are not available, is instead based on a cellular automaton approach. This paper proposes a mechanism by which multiple one-dimensional FDS based on the Faust FDS library approach can be coupled together. The coupling is achieved by composing the various FDS algorithms in parallel and modifying the Faust FDS library routing to calculate the connection forces. The mechanism is demonstrated by coupling multiple stiff string models to a bridge, modeled as an ideal damped bar.

1. INTRODUCTION

Physical modeling is an approach to sound synthesis in which the behavior of an instrument is described by a system of equations, the solution of which generates a time-domain waveform [1]. Finite difference schemes (FDS) [2] refer to an approach to solving these equations using finite-difference time domain (FDTD) methods. While FDS are more computationally demanding than other synthesis methods, they offer a close correspondence between model parameters and physical characteristics of the instrument, and impose few restrictions on the nature of the modeled behavior. This makes them a powerful tool to simulate a variety of different interactions [3–5], including those that go beyond what is possible in the physical world.

Faust (Functional AUdio STreams) [6] is a functional programming language for real-time audio processing and sound synthesis. It is designed to write digital signal processing (DSP) specifications that can be compiled to a wide range of backends such as C++, Java, WebAssembly etc. The generated code can either run as a a standalone application or be used in a larger context.

An intuitive implementation of a FDS algorithm in an imperative language would typically involve representing a discrete grid in a mutable data structure and updating it time step by time step. This approach is not possible in Faust, for the lack of arrays or similar data structures. Recently, a library [7] was added to the official Faust distribution that allows for the implementation of FDS algorithms by employing an approach similar to cellular automata. In this paper, we propose a mechanism by which FDS algorithms constructed using this library can be coupled together.

In section 2, we provide an overview over the FDS method. Section 3 describes a theoretical formulation for coupling one-dimensional FDS algorithms that is suitable for implementation in a functional language. Section 4 explains how this formulation is used to implement coupling with the Faust FDS library. Finally, we reflect on the results and discuss future work in section 5.

2. 1-D FINITE DIFFERENCE SCHEMES

In the FDS method, the system to be modeled is usually described by partial differential equations (PDE), which are then discretized and approximately solved with FDTD methods. This section will first demonstrate this process with a simple example, and then present a general formulation for coupling one-dimensional FDS with rigid connections. The methods presented here are based on [2].

2.1 1-D Wave Equation FDS

The simplest example of a FDS algorithm is the 1-D wave equation, where the displacement $u(x,t)$ at a location $x \in \mathcal{D} = [0, L]$ (where L is the length of the system in meters) at time $t \geq 0$ is characterized by

$$\partial_t^2 u = c^2 \partial_x^2 u\,, \tag{1}$$

where ∂_x and ∂_t refer to partial differentiation with respect to x and t, and the wave speed is denoted by c. To obtain an FDS algorithm, we introduce a grid function u_l^n with $n \in \mathbb{N}$ and $l = 0, \ldots, N$, where N is the number of grid points in space. The grid approximates the continuous function u as $u_l^n \approx u(lh, nk)$, where h is the distance between the grid points in space and k is the size of the time step, usually given by the external sampling rate f_s as $k = 1/f_s$.

Differentiation is approximated by introducing *forward, center* and *backward* difference operators. The first-order time operators are given by

$$\delta_{t+}u_l^n = \frac{u_l^{n+1} - u_l^n}{k}, \; \delta_{t\cdot}u_l^n = \frac{u_l^{n+1} - u_l^{n-1}}{2k} \text{ and } \delta_{t-}u_l^n = \frac{u_l^n - u_l^{n-1}}{k}. \tag{2}$$

Second-order differentiation can be approximated by combining first-order operators:

$$\delta_{tt}u_l^n := \delta_{t+}\delta_{t-}u_l^n = \frac{u_l^{n+1} - 2u_l^n + u_l^{n-1}}{k^2} \tag{3}$$

The spatial difference operators $\delta_{x+}, \delta_{x\cdot}, \delta_{x-}$ and δ_{xx} are defined analogously.

These operators can be used to discretize a PDE such as equation (1):

$$\delta_{tt}u_l^n = c^2\delta_{xx}u_l^n \tag{4}$$

Expanding the operators and rearranging the equation yields an explicit update equation, which we can use to calculate the state of the grid at time step $n+1$:

$$\begin{aligned} \frac{1}{k^2}u_l^{n+1} &= \frac{1}{k^2}\left(2u_l^n - u_l^{n-1}\right) \\ &+ \frac{c^2}{h^2}\left(u_{l+1}^n - 2u_l^n + u_{l-1}^n\right) \end{aligned} \tag{5}$$

For reasons of numerical stability, h and k cannot be chosen independently, but are linked by a stability condition. In the case of the 1-D wave equation, this condition is given by $h \geq ck =: h_{\min}$. The number of points N can then be chosen as $N = \lfloor L/h_{\min} \rfloor$, from which the grid spacing follows as $h = N/L$. This results in the highest number of grid points permitted by the stability condition.

2.2 General FDS Formulation

We will now consider a general FDS algorithm given by

$$\ell u_l^n = J_l(x_e)f_e^n \,. \tag{6}$$

Here, f_e^n refers to an external excitation force applied to location x_e. The linear spreading operator J_l distributes input at a location $x = x_i$ between the nearest grid point $l_i = \lfloor x_i/h \rfloor$ to the left of x_i and nearest grid point $l_i + 1$ to the right. Writing $\alpha_i = x_i/h - l_i$, it is defined as

$$J_l(x_i) = \frac{1}{h}\begin{cases} 0 & l < l_i \\ (1-\alpha_i) & l = l_i \\ \alpha_i & l = l_i + 1 \\ 0 & l > l_i + 1 \end{cases} . \tag{7}$$

In this paper, we only consider operators ℓ such that expanding it will always result in an explicit update equation that is a linear combination of grid points in the spatial neighborhood of u_l^{n+1} at time steps n or earlier. We can then write the state of the grid at time step n in vector form $\mathbf{u}^n = [u_1^n, \ldots, u_{N-1}^n] \in \mathbb{R}^{N-1\times 1}$, and state update equation for the general FDS algorithm as:

$$a\mathbf{u}^{n+1} = \sum_{\tau=0}^{T}\mathbf{B}^{(\tau)}\mathbf{u}^{n-\tau} + \mathbf{j}_e f_e^n \,, \tag{8}$$

where T is the number of past states required for the update equation, $\mathbf{j}_e$ is a column vector representing the spreading operator defined in equation (7), and the scalar a and the coefficient matrices $\mathbf{B}^{(\tau)} \in \mathbb{R}^{(N-1)\times(N-1)}$ for $\tau = 0, \ldots, T$ result from expanding the difference operators contained in ℓ.

The top and bottom rows of the matrices $\mathbf{B}^{(\tau)}$ also determine the boundary conditions of the FDS algorithm.

The right-hand side of equation (8) contains all information about how the previous state of the grid and the externally supplied excitation force influences $\mathbf{u}^{n+1}$. For ease of notation, we will summarize this information by defining an auxiliary variable $\mathbf{q}$ such that

$$\mathbf{q}^n := \sum_{\tau=0}^{T}\mathbf{B}^{(\tau)}\mathbf{u}^{n-\tau} + \mathbf{j}_e f_e^n \,, \tag{9}$$

therefore, the update equation (8) becomes: $a\mathbf{u}^{n+1} = \mathbf{q}^n$.

3. COUPLING FORMULATION

We now move on to considering M different FDS algorithms with individual update equations $a_m\mathbf{u}_m^{n+1} = \mathbf{q}_m^n$ for $m = 1, \ldots, M$. These systems need not be homogeneous and might be characterized by different operators ℓ_m, as long as they are linear. In particular, they might have different lengths L_m and grid spacings h_m, but share the time step k. If two such systems are *coupled*, it means that a point along one system is connected to a point along the other, and they will influence each other through a connection force.

Here, we formulate the modular connection mechanisms covered in detail in [8] in a way that can be easily integrated into the Faust FDS implementation. For this purpose, we define a coupling as a 4-tuple (r, s, x_r, x_s) for $1 \leq r, s \leq M$. This denotes that the r-th system is coupled *above* the s-th system, where $x_r \in [0, L_r]$ and $x_s \in [0, L_s]$ refer to the respective coupling locations. The connection force $f_{r,s}^n$ acts positively on system r and negatively on system s. Given a set of couplings $\mathfrak{C}$, this is reflected by the update equation

$$\begin{aligned} a_m\mathbf{u}_m^{n+1} = \mathbf{q}_m^n &+ \sum_{(r,s,x_r,x_s)\in\mathfrak{C}:r=m} \mathbf{j}_{m,x_r} f_{r,s}^n \\ &- \sum_{(r,s,x_r,x_s)\in\mathfrak{C}:s=m} \mathbf{j}_{m,x_s} f_{r,s}^n \,. \end{aligned} \tag{10}$$

Here, the additional subscript m for the spreading vectors $\mathbf{j}$ reflects the fact that the value α_i (see equation (7)) depends on the grid spacing of the m-th FDS algorithm.

It remains now to calculate the connection force $f_{r,s}^n$. The force depends on the nature of the connection; here, we assume a rigid connection, which is characterized by $u_r(x_r, t) = u_s(x_s, t)\forall t$. To apply this to the discrete model, we introduce the linear interpolation operator I, using the same notation as with J_l:

$$I(x_i)u^n = (1-\alpha_i)u_{l_i}^n + \alpha_i u_{l_i+1}^n \tag{11}$$

A rigid connection can then be enforced by requiring

$$\mathbf{i}_{r,x_r}\mathbf{u}_r^{n+1} \overset{!}{=} \mathbf{i}_{s,x_s}\mathbf{u}_s^{n+1} \, \forall (r,s,x_r,x_s) \in \mathfrak{C}\,, \tag{12}$$

where $\mathbf{i}$ denotes a row vector such that multiplication with the grid results in interpolation as defined in equation (11).

Assuming that the coupling points are sufficiently distant from each other that their associated grid points do not overlap, i.e. no grid point is affected by two or more couplings, allows us to solve for the coupling forces individually by expanding equation (12):

$$\begin{aligned} \mathbf{i}_{r,x_r} a_r^{-1}\left(\mathbf{q}_r^n + \mathbf{j}_{r,x_r} f_{r,s}^n\right) &= \\ = \mathbf{i}_{s,x_s} a_s^{-1}\left(\mathbf{q}_s^n + \mathbf{j}_{s,x_s} f_{r,s}^n\right) \end{aligned} \tag{13}$$

Observing that

$$\begin{aligned} \mathbf{i}_{m,x_m}\mathbf{j}_{m,x_m} &= \frac{1}{h_m}\left((1-\alpha_m)^2 + \alpha_m^2\right) \\ &= h_m \|\mathbf{j}_{m,x_m}\|_2^2\,, \end{aligned} \tag{14}$$

we can solve equation (13) for $f_{r,s}^n$:

$$f_{r,s}^n = \frac{a_s^{-1}\mathbf{i}_{s,x_s}\mathbf{q}_s^n - a_r^{-1}\mathbf{i}_{r,x_r}\mathbf{q}_r^n}{a_r^{-1}h_r\|\mathbf{j}_{r,x_r}\|_2^2 + a_s^{-1}h_s\|\mathbf{j}_{s,x_s}\|_2^2} \tag{15}$$

A system of coupled FDS algorithms can then be simulated by

1. Calculating $\mathbf{q}^n$ according to equation (9),
2. Calculating the connection forces according to equation (15),
3. Applying the forces to the connection points and updating the grid state according to equation (10).

The next section describes how this behavior can be implemented using the Faust FDS library.

4. FAUST IMPLEMENTATION

A detailed explanation of the concepts behind the Faust FDS library can be found in [7] or in the official documentation. [1] We will summarize how the FDS library implements one-dimensional FDS algorithms, and then discuss how we can implement coupling using the previously derived formulation. The code and an example using the mechanism to couple guitar strings over a bridge can be found at `https://github.com/dsuedholt/coupled-fds-faust`. Functions with the `fd.` prefix are part of the Faust FDS library, while those without are part of the proposed coupling implementation.

4.1 One-Dimensional FDS Algorithms in Faust

The basic building block of an FDS algorithm in Faust is a *scheme point*. In a general, uncoupled FDS algorithm, a scheme point calculates u_l^{n+1} as a linear combination of its neighboring grid points in space and time. A scheme point takes as input a force signal (e.g. an excitation), some

[1] `https://faustlibraries.grame.fr/libs/fds/`

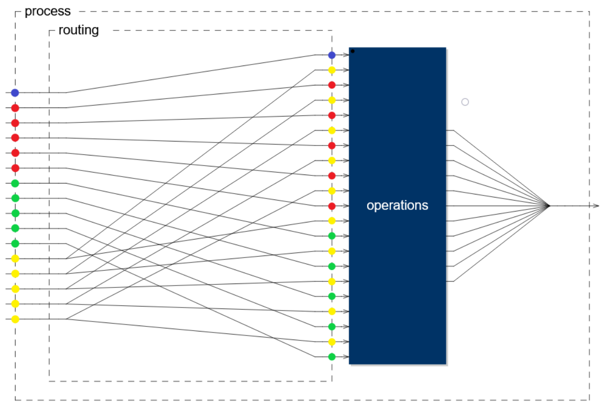

Figure 1. A scheme point with $R = 2$, $T = 1$ and $D = 1$. It expects a force at the input marked in blue, the weights for the current timestep at the inputs marked in red, the weights for the previous timestep at the inputs marked in green, and finally the neighboring grid points u_{l-R}^n to u_{l+R}^n at the inputs marked in yellow. The routing section rearranges the inputs so that the operations block can perform the linear combination from equation 16. The grid points at the yellow inputs are duplicated T times and fed into delays of appropriate length to correctly apply the weights. Finally, the output of the individual calculations is summed together.

coefficients and the grid points at time step n, and outputs the value of the grid at u_l^{n+1} as a linear combination of the grid points using the supplied coefficients.

The function `fd.schemePoint(R,T,D)` takes as input a neighborhood radius R, indicating how many spatial neighbors to each side a grid point depends on, a time coefficient T, indicating how many steps back in time are needed, and the dimensionality D of the FDS algorithm. We can rewrite the update equation (8) as

$$au_l^{n+1} = \sum_{\tau=0}^{T}\left(\sum_{i=-R}^{R} \mathbf{B}_{(l,l+i)}^{(\tau)} u_{l+i}^n + f_l^n\right) \tag{16}$$

where $\mathbf{B}_{(i,j)}^{(\tau)}$ refers to the entry in the i-th row and j-th column of the coefficient matrix $\mathbf{B}^{(\tau)}$, and f_l^n refers to an externally supplied force signal for the grid point l. Figure 1 illustrates the construction of a scheme point under these conditions.

The function `fd.buildScheme1D(points,R,T)` stacks a number of scheme points on top of each other, and `fd.model1D(points,R,T,scheme)` builds the proper routing of input forces, coefficients, and feedback around the scheme points, so that the output of the scheme at each time step is the current value of the grid.

4.2 Implementing the Coupling Mechanism

We will now build on the Faust FDS functionality to realize coupling. Instead of `fd.model1D`, we introduce a function `system1D` that puts together multiple schemes constructed by `fd.buildScheme1D` and couples them according to supplied coupling information. The individual schemes first calculate $\mathbf{q}^n$, which is then routed into a function `forceUpdate`; where, for each coupling, the affected grid points are interpolated and the resulting forces are calculated and spread back to the grid points. Finally, the output is divided by a to obtain the grid values at $n+1$. Like in `fd.model1D`, the output of the system is always routed back to its own input at the next time step using the Faust feedback operator.

For the purpose of the `forceUpdate`, we rewrite equation (15) as

$$f^n_{r,s} = \beta_r \mathbf{i}_{r,\chi_r} \mathbf{q}^n_r + \beta_s \mathbf{i}_{s,\chi_s} \mathbf{q}^n_s \, , \tag{17}$$

with

$$\beta_r = \frac{-a_r^{-1}}{a_r^{-1} h_r ||\mathbf{j}_{r,\chi_r}||_2^2 + a_s^{-1} h_s ||\mathbf{j}_{s,\chi_s}||_2^2} \, , \tag{18a}$$

$$\beta_s = \frac{a_s^{-1}}{a_r^{-1} h_r ||\mathbf{j}_{r,\chi_r}||_2^2 + a_s^{-1} h_s ||\mathbf{j}_{s,\chi_s}||_2^2} \, . \tag{18b}$$

The input required by `system1D` to couple M schemes together are the coefficients to calculate $\mathbf{q}^n_m$, the factors a_m, the set of couplings $\mathfrak{C}$ and the coefficients β_r, β_s to calculate the coupling forces. An illustration of this is given in figure 2. Figure 3 shows an overview over how the signals are routed to allow for the force calculation.

An important distinction between `fd.model1D` and `system1D` is that the coupling implementation requires the coefficients $\mathbf{B}^{(\tau)}$ and the scalar factor a to be supplied separately, while the Faust library function expects the coefficients to already be scaled by a^{-1}.

4.3 Example: Strings over a Bridge

We will now demonstrate how this mechanism can be used in practice to couple different FDS together. We want to model three strings coupled to a connecting bridge, resulting in $M = 4$ different FDS. The schemes $m = 1, 2, 3$ model the strings using the stiff string operator ℓ_{s_m}, defined as [2]

$$\begin{aligned} \ell_{s_m} = &\rho_m A_m \delta_{tt} - T_m \delta_{xx} + E_m I_m \delta_{xx}\delta_{xx} \\ &+ 2\rho_m A_m \sigma_{0,m} \delta_{t-} - 2\rho_m A_m \sigma_{1,m} \delta_{t-}\delta_{xx} \, . \end{aligned} \tag{19}$$

Here, $\sigma_{0,m}$ is a general and $\sigma_{1,m}$ a frequency-dependent damping parameter, and E_m is Young's modulus, describing the material stiffness of the string. T_m refers to the string tension, and ρ_m to its density. Given the string radius r_m, $A_m = \pi r_m^2$ is the cross-sectional area of the string and $I_m = \pi r_m^2/4$ is the moment of inertia.

The scheme $m = 4$ models the bridge as an ideal bar. The corresponding operator ℓ_b can be obtained from ℓ_{s_m} by setting the tension to 0.

The values of the coefficient matrices $\mathbf{B}_m$ follow from expanding $\ell_{s_m} u^n_l$ (omitting the m subscripts for simplicity):

$$\mathbf{B}^{(0)}_{(i,j)} = \frac{1}{h^4} \begin{cases} -EI & |i-j| = 2 \\ Th^2 + 4EI + \frac{2\rho A \sigma_1 h^2}{k} & |i-j| = 1 \\ -2Th^2 - 6EI - \dfrac{2\rho A \sigma_0 h^4}{k} & \\ \quad - \dfrac{4\rho A \sigma_1 h^2}{k} + \dfrac{2\rho A h^4}{k^2} & i = j \\ 0 & \text{else} \end{cases} \tag{20}$$

$$\mathbf{B}^{(1)}_{(i,j)} = \begin{cases} -\frac{2\rho A \sigma_1}{h^2 k} & |i-j| = 1 \\ \frac{2\rho A \sigma_0}{k} + \frac{4\rho A \sigma_1}{h^2 k} - \frac{\rho A}{k^2} & i = j \\ 0 & \text{else} \end{cases} \tag{21}$$

and $a = \rho A/k^2$. Simply taking over the coefficients from the expanded FDS algorithms and "clipping" them at the matrix boundaries implements the clamped boundary condition, equivalent to introducing virtual points u^n_{-1}, u^n_{N+1} and setting $u^n_{-1} = u^n_0 = u^n_N = u^n_{N+1} = 0$. This boundary condition is employed for the bridge.

The Faust FDS library takes in coefficients as input for every single scheme point; therefore, it is possible to use different values at the boundary points to implement the simply supported boundary condition, which is given by: $u^n_{-1} = -u^n_1$, $u^n_{N+1} = -u^n_{N-1}$, $u^n_0 = u^n_N = 0$. This is the condition implemented for the stiff strings.

The strings are coupled above the bridge by defining the couplings [2]

$$\begin{aligned} \mathfrak{C} = \{&(1, 4, 0.1, 0.04), \\ &(2, 4, 0.1, 0.08), \\ &(3, 4, 0.1, 0.12)\} \end{aligned} \tag{22}$$

Together with the factors β to enable the force calculation according to equations (17-18), we have now fully characterized coupling the strings to the bridge using a rigid connection to implement it using the developed methods.

5. CONCLUSION

This paper proposed a general mechanism for coupling one-dimensional FDS algorithms with rigid connections using the Faust FDS library. It was demonstrated on an example that can implement the desired behavior in real-time in the Faust Online IDE.

The method works with explicit, one-dimensional FDS algorithms and makes use of rigid connections and linear interpolation. Future work will involve including support for different types of connections and interpolators, in order to generalize the implementation and include it in the Faust distribution.

To understand how the performance of the Faust FDS library and the proposed coupling implementation compares to other languages and frameworks, a speed evaluation should also be performed. However, due to the large

[2] The Faust code uses zero-based indexing of the schemes as opposed to the one-based indexing in the mathematical formulation

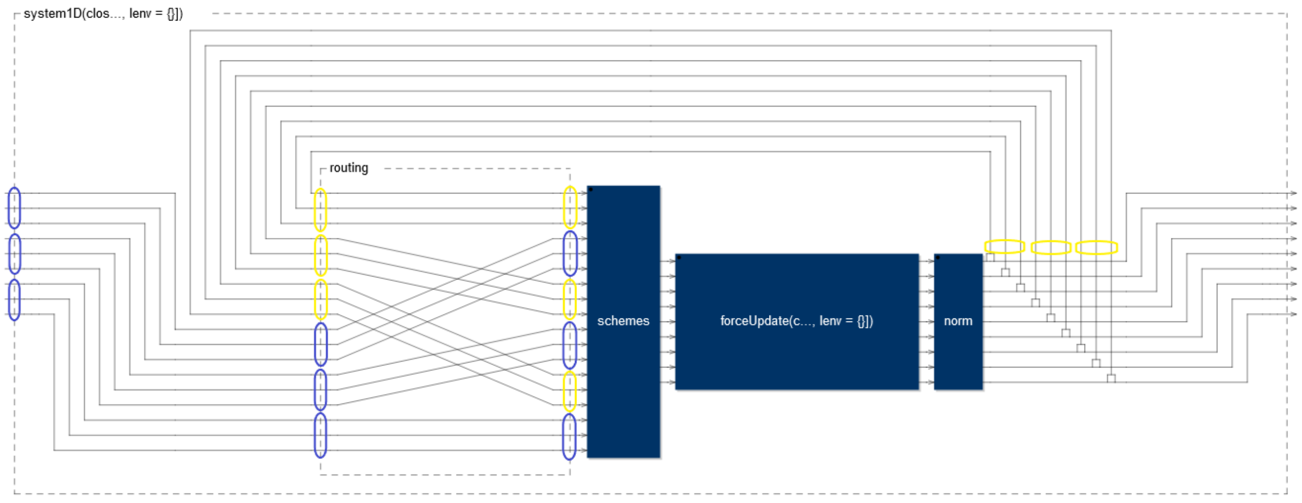

Figure 2. An example of `system1D` combining three FDS with three points each into a coupled system. The inputs of the entire system are the excitation forces for each point of each system in order, marked in blue. The forces are then routed to be interleaved with the grid points from the previous time step, marked in yellow, so that each individual scheme can receive its proper input. The block labeled here as "schemes" performs the calculation of $\mathbf{q}^n$ for all schemes. These values are then passed on to add and subtract the appropriate coupling forces in the "forceUpdate" block, and finally passed to the "norm" block, where division by a_m finalizes the update calculation.

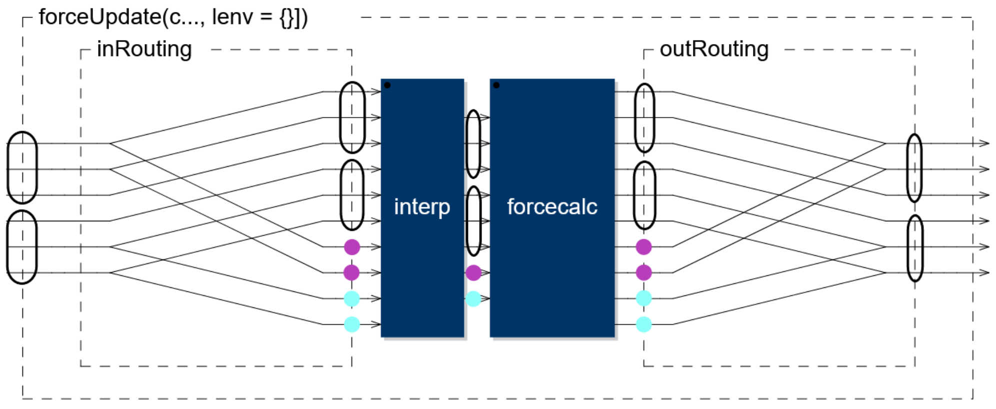

Figure 3. An example of `forceUpdate` for two FDS with three grid points each and one coupling. For each coupling, the four grid points belonging to the coupling (two per system) are added to the current scheme values $\mathbf{q}^n$ in parallel, and the "interp" block interpolates the two points per system into one single value, which are then used in "forcecalc" to calculate the coupling force. The values $\mathbf{q}^n$ are simply passed through. The outputs of "forcecalc" are then the outputs of applying the spreading operator to the calculated force; and multiplying the values for the lower system with -1. Then these four values per coupling can simply be added back onto their respective grid points.

variety of optimization options and backends of the Faust compiler, this exceeds the scope of this paper.

6. REFERENCES

[1] S. Bilbao, C. Desvages, M. Ducceschi, B. Hamilton, R. Harrison, A. Torin, and C. Webb, "Physical Modeling, Algorithms and Sound Synthesis: The NESS Project," *Computer Music Journal*, vol. 43, no. 2-3, pp. 15–30, Jun. 2020.

[2] S. Bilbao, *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. Wiley Publishing, 2009.

[3] S. Bilbao and A. Torin, "Numerical Modeling and Sound Synthesis for Articulated String/Fretboard Interactions," *Journal of the Audio Engineering Society*, vol. 63, no. 5, pp. 336–347, May 2015.

[4] S. Willemsen, S. Serafin, S. Bilbao, and M. Ducceschi, "Real-time Implementation of a Physical Model of the Tromba Marina," in *Proceedings of the 17th Sound and Music Computing Conference*, June 2020.

[5] M. G. Onofrei, S. Willemsen, and S. Serafin, "Real-Time Implementation of a Friction Drum Inspired Instrument using Finite Difference Schemes," in *Proceedings of the 24th International Conference on Digital Audio Effects (DAFx20in21)*, September 2021.

[6] Y. Orlarey, D. Fober, and S. Letz, "FAUST : an Efficient Functional Approach to DSP Programming," in *New Computational Paradigms for Computer music*, 2009, pp. 65–96.

[7] R. Russo, S. Serafin, R. Michon, Y. Orlarey, and S. Letz, "Introducing Finite Difference Schemes Synthesis in FAUST: A Cellular Automata Approach," in *Proceedings of the 18th Sound and Music Computing Conference*, June 2021.

[8] S. Bilbao, M. Ducceschi, and C. Webb, "Large-Scale Real-Time Modular Physical Modeling Sound Synthesis," in *Proceedings of the 22nd International Conference on Digital Audio Effects (DAFx-19)*, September 2019.