

Accelerating 3D Finite Difference

Maxeler dataflow solutions currently deliver the fastest Reverse Time Migration implementations in the industry, with performance predicted to more than double with each new silicon generation. Dataflow technology provides a massive level of parallelism compared to CPUs, with fast on-chip memories ideally suited to 3D finite difference problems. Maxeler dataflow engines (DFEs) have large on-board DRAM memories, with up to 384GB of DFE DRAM in each node, removing the data bus as a bottleneck in an accelerated system. Maxeler MaxGen systems are domain-specific compilers that enable geoscientists to harness the power of Maxeler dataflow solutions (Figure 1) without learning low-level design. MaxGenFD, for 3D finite difference applications, enables rapid development of accelerated seismic processing applications such as Forward Modeling, Reverse Time Migration and Waveform Inversion.

MaxGenFD handles the complexities facing any finite difference implementation such as managing very large data sets, boundary conditions and domain decomposition across multiple compute elements with halo exchange. In addition, the compiler removes the need for the geoscience programmer to perform low-level optimizations such as customizing data-types and generating optimized stencil descriptions for hardware.

MaxGenFD is implemented as a layer on top of Maxeler MaxCompiler, the general-purpose dataflow programming system for HPC applications. This allows the user to retain the full programming power of MaxCompiler, while MaxGenFD provides pre-defined libraries to implement common features of finite difference applications.

MaxGenFD Programming Model

In a typical software finite difference application, the code loops through many timesteps of a core kernel that performs wave propagation. Broadly, the computation in the kernel can be broken down into derivative calculations (convolving one or more of the input wavefields with a stencil operator), wave equation (computing the next wavefield state based on the input wavefields and the derivatives), and boundary conditions. Convolution with the finite difference stencil usually dominates both

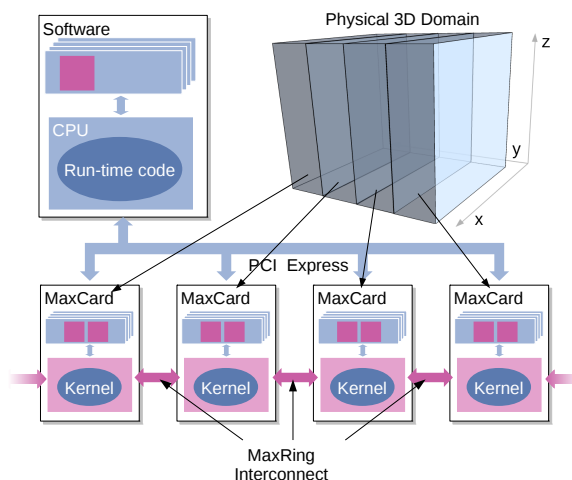


Figure 1: Dataflow finite difference with 3D domain decomposed over multiple dataflow engines connected via a MaxRing interconnect.

operation count and computation time.

Figure 2 shows the architecture of a MaxGenFD dataflow application compared to a software-only implementation. In the pure software implementation, the CPU executes the kernel computation and the compute node's main memory contains the wavefield and earth model data. When using MaxGenFD, the core kernel of code, including the critical convolutions, has been reimplemented as a specialized finite difference Kernel (FDKernel) and is now running in a dataflow engine. The FDKernel behaves as a function with some number of wavefield inputs, an earth model input (possibly with multiple parameters), and some number of wavefield outputs. Wavefields and earth models are now stored (possibly in compressed form) in the DRAM memory on the DFE. In addition, the FDKernel can have some number of CPU inputs and outputs that can be used to receive stimulus data from the CPU and to return output data from the DFE during the execution of a timestep.

To accelerate an application using MaxGenFD, the user splits the application into three components:

- FDKernel (written in Java)
- FDKernel configuration (written in Java)
- Runtime (written in C, C++ or FORTRAN)

The FDKernel is a mathematical description of a specialized function that, given appropriate input data, performs some part of a finite difference calculation (usually a timestep, or part of a timestep).

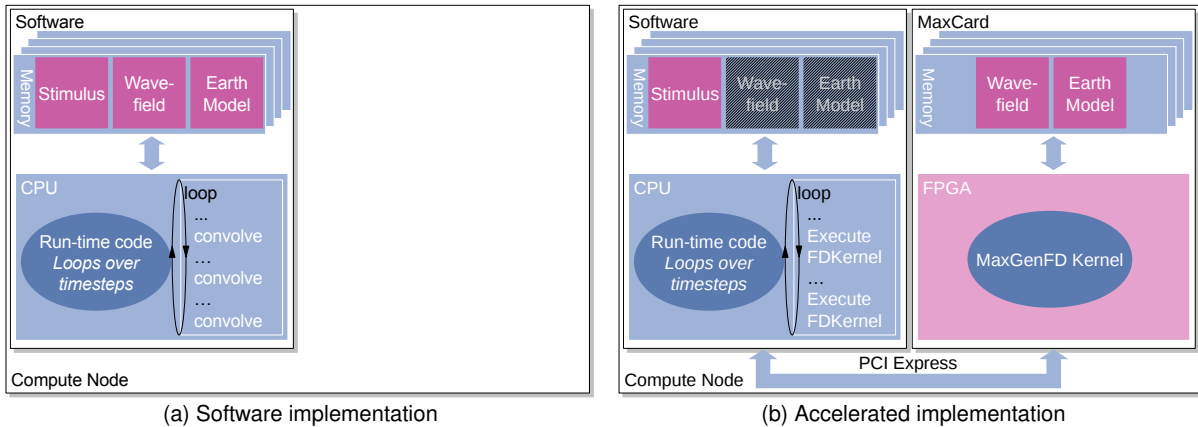


Figure 2: Software and dataflow finite difference implementations. In the dataflow implementation, the earth model and wavefields are stored in the DRAM of the dataflow engine(s).

The FDKernel configuration configures the performance options for the FDKernel at build time. This allows a single mathematical FDKernel description to be used to generate multiple different DFE configurations with different performance characteristics.

The runtime integrates the dataflow kernel into a complete software application. The runtime code drives the FDKernel and decides what input data to provide to the FDKernel at any point in time. The software can also set scalar input values for the FDKernel to configure specific functionality, for example to set different boundary conditions for different surfaces of the problem domain.

By partitioning the application in this way, functionality that is fixed (e.g. the shape of the finite difference stencil) is hard-coded into the dataflow engine and highly optimized, however parts that need to change on a timestep-to-timestep basis (e.g. which input wavefields should be read) are configured in software.

Acoustic Forward Modeling

To introduce MaxGenFD, we consider an example of an Acoustic Forward Modeling code. We model an isotropic medium with variable velocity, described by the equation:

$$\frac{\partial^2 p}{\partial t^2} = v^2 \nabla^2 p + s(t)$$

As a boundary condition, we use an absorbing “sponge” region at the edges of the domain. A finite difference implementation of this equation can be described in pseudo-code as in Figure 3.

```

for t = 1 to tmax:
  for i = 0 to X*Y*Z-1:
    l = convolve(curr[i], stencil)
    next := 2 * curr[i] - prev[i]
           + vv[i] * l
    next[i] := next + source[i]
  apply_boundary_sponge(curr, next)
  swap_buffers(prev, curr, next)
  
```

Figure 3: Pseudo-code for a simple finite difference modeling implementation.

The FDKernel

In our example, the timestep is implemented as an FDKernel using MaxGenFD. The source code is shown in Figure 4. *IsotropicModelingKernel* is a Java class that extends the MaxGenFD base class *FDKernel*. Notice that there are no implementation-specific details defined in the Kernel: these are set separately at compile-time in an *FDConfig* object for the build (as shown in Figure 5), allowing the FDKernel to be completely generic.

Figure 8 shows how the FDKernel executes each timestep. The current wavefield (p_t) is read from the dataflow engine DRAM into the FDKernel and convolved. The wave equation uses the results of the convolution; the current and previous wavefields (p_t and p_{t-1}) and the earth model (read from the DFE DRAM); and the source data for the current timestep (s_t) (streamed from the CPU). The result (p_{t+1}) is written back to the DFE DRAM.

MaxGenFD leverages the streaming computation model of MaxCompiler, so all of our inputs to and

```
public class IsotropicModelingKernel extends FDKernel {
    public IsotropicModelingKernel(FDKernelParameters p) {
        super(p);
        Stencil stencil = fixedStencil(-6, 6, coeffs, 1/8.0);
        FDKernel curr = io.wavefieldInput("curr", 1.0, 6);
        FDKernel prev = io.wavefieldInput("prev", 1.0, 0);
        FDKernel dvv = io.earthModelInput("dvv", 9.0, 0);
        FDKernel source = io.hostInput("source", 1.0, 0);

        FDKernel l = convolve(curr, ConvolveAxes.XYZ, stencil);

        FDKernel sponge = boundaries.sponge(50);
        prev = prev * sponge;
        FDKernel next = curr * 2 - prev + dvv * l + source;
        next = next * sponge;

        io.wavefieldOutput("next", next);
    }
}
```

Figure 4: FDKernel for isotropic modeling.

```
FDConfig config = new FDConfig(DefaultType.FLOAT);
config.setParallelPipelines(4);
config.setDefaultType(hwFloat(8,24));
config.setWaveFieldType(StorageType.compressed16);
```

Figure 5: FDKernel configuration for isotropic modeling.

outputs from the Kernel are streams. We do not specify where streams should get their data explicitly: we declare our streams using input and output types (e.g. wavefield, earth model, CPU input), which allows MaxGenFD to connect the streams appropriately.

The FDKernel uses a 13-pt stencil which is declared using a special *Stencil* construct. If the stencil coefficients contain symmetry, MaxGenFD will automatically optimize the operations that are actually performed to minimize multiplications. We apply the stencil in all three axes using a single call to the *convolve* method.

The boundary condition is created using MaxGenFD's boundary conditions library. A 50-pt absorbing "sponge" region is generated by the call to *boundaries.sponge* as a multiplicand that is multiplied with the two wavefields. Because the FDKernel description is a dataflow program, all points in the problem domain are multiplied by the sponge value, even if they are in the interior of the domain, but the boundary library will set the sponge coefficient to 1.0 when outside the absorbing region. Notice that the way the sponge is applied is subtly different to the software pseudo-code: applying the sponge to the previous wavefield be-

```
for(t = 1; t < tmax; t++) {
    // Set-up timestep
    if (t < tsrc) {
        source = generate_source_wavelet(t);
        maxlib_stream_region_from_host(maxlib, "source", source
            srcx, srcy, srcz, srcx+1, srcy+1, srcz+1);
    }
    maxlib_stream_from_dram(maxlib, "curr", curr_ptr);
    maxlib_stream_from_dram(maxlib, "prev", prev_ptr);
    maxlib_stream_earthmodel_from_dram(maxlib, dvv_array);

    maxlib_stream_to_dram(maxlib, "next", next_ptr);

    maxlib_run(maxlib); // Execute timestep

    swap_buffers(prev_ptr, curr_ptr, next_ptr);
}
```

Figure 6: Runtime software for isotropic modeling.

fore the wave equation allows us to compute the sponge for the wavefield in timestep $t+1$ rather than timestep t , meaning that the dataflow engine only needs to write one wavefield output back to memory, rather than two, giving increased performance.

Integrating with the software application

Figure 6 shows the CPU code that executes each timestep for the isotropic modeling example. For each timestep, the CPU sets up the computation it would like the FDKernel to perform, selecting which wavefields should be streamed to/from the inputs and outputs of the FDKernel, whether a source stimulus should be added, etc. Once the set-up is complete, the CPU instructs the FDKernel to run the computation. The source wavelet is generated and streamed from the CPU for each timestep. The output from the calculation is streamed back into the DFE DRAM. At

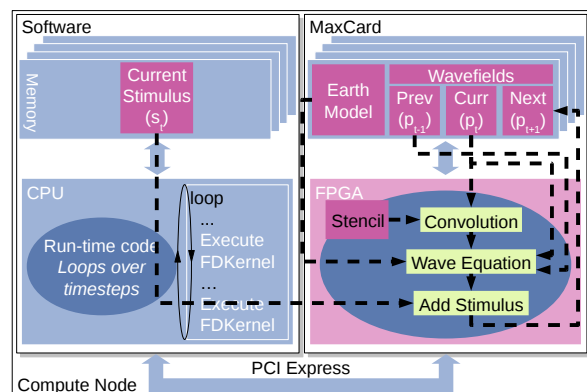


Figure 8: Simple example Kernel showing inputs and outputs of the convolution and wave equation on each time step.

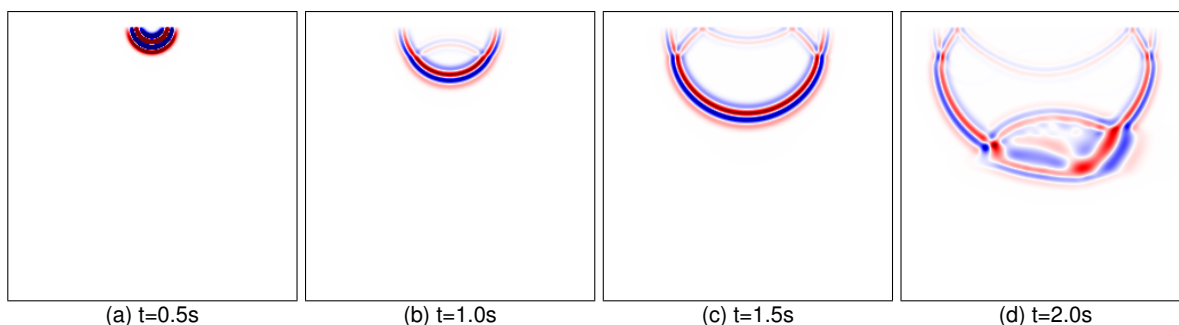


Figure 7: Output from using MaxGenFD to model wave propagation through the SEG salt model.

the end of each timestep, the wavefield buffers are swapped.

While this example has been shown with three wavefield buffers for simplicity (previous, current and next), it is possible to use only two buffers, writing the results from a timestep directly into the previous wavefield buffer.

Figure 7 shows snapshots of output from running the accelerated modeling on the SEG salt model.

Optimization

The performance of a MaxGenFD accelerated application is dependent on the compute performance of each dataflow engine, the memory bandwidth per DFE and the number of DFEs available in the system.

FDKernel descriptions provide both parallelism within a single computational pipeline and the instantiation of multiple computational pipelines running in parallel. The FDKernel description is the same regardless of the level of parallelization, with the user specifying the number of computational pipelines that should be created during the dataflow engine build process in an *FDConfig* object.

MaxGenFD allows the data representations for each type of data to be configured to optimize the DFE resources required for each compute pipeline, maximizing the number of parallel pipelines that can fit onto a single DFE. MaxGenFD generates annotated source code showing resource utilization for each line of user code, allowing the user to identify code for optimization.

To optimize use of memory bandwidth, MaxGenFD can add compression and decompression function blocks at the beginning and end of the FDKernel computation pipeline. On-chip (de)compression

has no run-time cost, provided there is sufficient silicon area available, and earth models and wavefields can be automatically compressed and decompressed on the fly, which increases the amount of memory bandwidth available by the compression factor. MaxGenFD provides a number of in-built compression schemes that have been verified for use in seismic modeling and migration applications.

Debugging

For verifying the correctness of FDKernels, MaxGenFD includes the FDSimulator, a specialization of the MaxCompiler simulator that allows users to simulate FDKernels quickly to verify their functionality and to tune the results. Using the Maxeler Kernel simulation engine, small problem sizes can be run for many timesteps purely in simulation (e.g. 64x64x64 domains), allowing the mathematical correctness of operations described to be verified before building dataflow engine hardware.

Once the behavior has been verified, then the design can be built in hardware and run with a large domain. In hardware, debugging facilities are still available in the form of hardware exceptions (detecting overflow, underflow, divide by zero and invalid operations) and MaxDebug, a graphical tool for debugging data flow scheduling in the whole system.

Training and Education

Maxeler Technologies provides full documentation and educational material for the use of MaxGenFD, and runs workshops on implementing accelerated 3D finite difference applications, as well as general use of MaxCompiler and creating complete dataflow supercomputing solutions.