Editors: Volodymyr Kindratenko, kindr@ncsa.uiuc.edu
Pedro Trancoso, pedro@cs.ucy.ac.cy

Maximum Performance Computing with Dataflow Engines

By Oliver Pell and Vitali Averbukh

Multidisciplinary dataflow computing is a powerful approach to scientific computing that has led to orders-of-magnitude performance improvements for a wide range of applications.

Tremendous advances in the application of numerical techniques in science have been driven by progress in computer technology. The available computational power and computer memory resources define the scope of the scientific problems that can be addressed, as well as the achievable accuracy level of theoretical modeling and thus also the reliability of the scientific prediction. As modern science strives to address more and more challenging problems on a larger scale using more accurate theoretical models, the coming years will require the development of powerful computational technology at new levels of performance. However, recent years have brought the end of "free" performance gains through processor-frequency scaling, and at the same time, power consumption is now one of the main limiting factors in the design of high-performance computing (HPC) systems. It won't be possible to build HPC systems capable of *exascale* processing (approximately 100 times faster than the world's current fastest supercomputer, which provides 10 petaflops of peak performance) by simply scaling existing technology.

The computer architectures used in modern scientific computing can be roughly divided into general-purpose and problem-specific ones. The foundation for the idea of general-purpose computing was laid by J. von Neumann as early as 1945.[1] Although this is still the primary model of computer architecture more than 60 years later, over the last decade the broad scientific community has begun to appreciate the superiority of problem-specific architectures for solving the most challenging computational problems. Special-purpose computers can be optimized for an application's particular requirements and thus be orders of magnitude more efficient at solving particular problems than their general-purpose brethren.

A number of highly successful applications of dedicated special-purpose computers have been reported in such diverse areas as spin-glass systems,[2] molecular dynamics,[3] and quantum chromodynamics.[4] Importantly, applying special-purpose technology not only accelerated the specific applications by at least an order of magnitude but also, as in the case of the spin-glass model, the improved performance allowed researchers for the first time to access a physically relevant regime of parameters.

Maxeler Technologies is pioneering the development of *Dataflow Computing*, a class of special-purpose computers that are programmable and can be specialized to different applications at different points in time. Dataflow computing has led to orders-of-magnitude lower power consumption and lower datacenter space requirements for a wide range of applications, ranging from 3D finite-difference (FD) partial differential equation (PDE) solvers to Monte Carlo simulations. This article introduces the principles of dataflow computing, describes what dataflow hardware is available, and discusses example applications that have been accelerated with this approach.

What Is Dataflow Computing?

The Maxeler dataflow computing paradigm differs from computing with conventional CPUs (control-flow cores), as Figure 1 shows, and represents an evolution of concepts of dataflow computers[5] and systolic array processors[6] developed in the 1970s and 1980s.

At its heart, a control-flow processor contains a latency-critical loop. Data is read from memory into the processor core, where operations are performed and the results are written back to memory. A dataflow compute engine operates differently. Data is *streamed* from memory onto the chip where operations are performed and data is forwarded directly from one functional unit (a *dataflow core*) to another as the results are needed, without ever being written to the off-chip memory until the chain of processing is complete. Each dataflow core computes only a single type of operation (for example, an addition or multiplication) and is thus simple, so thousands of operations can fit on one chip

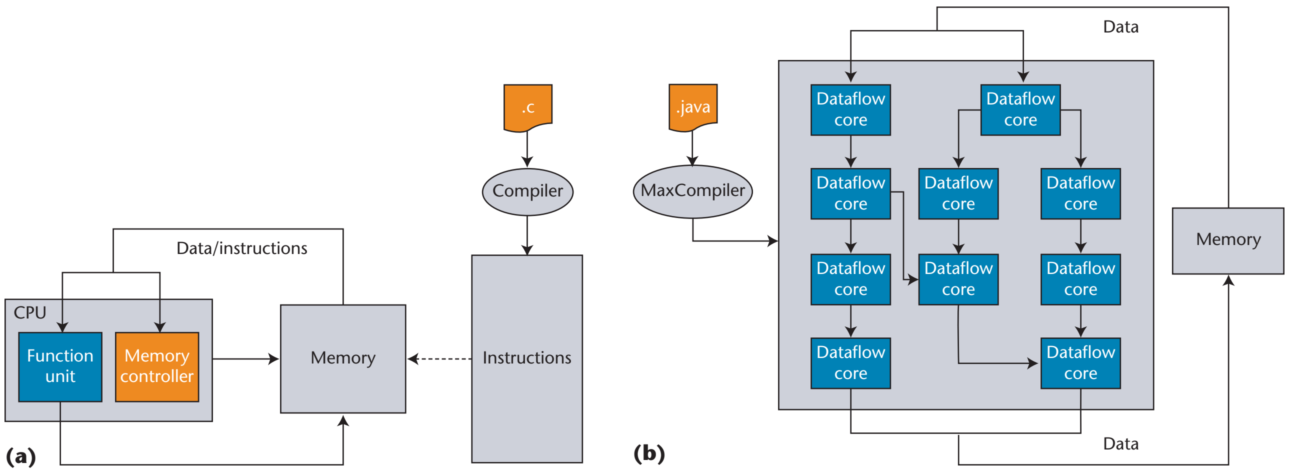


Figure 1. Computing with (a) a conventional control-flow core compared to (b) dataflow cores. A control-flow processor contains a latency-critical loop at its heart. Data is read from memory into the processor core, where operations are performed and the results are written back to memory. A dataflow compute engine operates differently. Data is streamed from memory onto the chip where operations are performed, and data is forwarded directly from one functional unit (a dataflow core) to another as the results are needed, without ever being written to the off-chip memory until the chain of processing is complete.

with every dataflow core computing simultaneously.

Unlike on the control-flow core, where operations are computed at different points in time on the same functional units ("computing in time"), the complete dataflow computation is laid out spatially on the chip ("computing in space"). Dependencies in the dataflow are resolved statically at compile time, and because there are no new dependencies present at runtime, the whole dataflow engine can be deeply pipelined (1,000 to 10,000 stages). Every stage of the pipeline computes in parallel and the dataflow architecture maintains a throughput of one result per cycle. One analogy for moving from control flow to dataflow is the Ford car manufacturing model, where expensive, highly skilled craftsman (control-flow cores) are replaced by a factory line, moving cars through a sea of single-skilled workers (dataflow cores).

The dataflow engine structure itself represents the computation, thus there's no need for instructions per se; they're replaced by operation executing units. Because there are no instructions there's no need for instruction-decode logic. A static dataflow machine doesn't require techniques such as branch prediction (since there aren't any branches) or out-of-order scheduling (parallelism is explicit). And because data is always available on chip for as long as it's needed, general-purpose caches aren't needed—the minimum amount of buffering memory is utilized automatically as necessary. This saves silicon area and power budget. By eliminating these extraneous functions, the chip's full resources are dedicated to performing computation. At a system level, the dataflow engine handles computation of the large-scale streaming operations in an application while dynamic events and control are managed by control-flow core(s).

Figure 2. Rack of MPC-C series compute nodes. More than 40 of these one-unit (1U) systems can fit in a standard rack.

The key to the dataflow advantage is a multidisciplinary approach to scientific computing, where a team of natural and computer scientists and engineers work together to *codesign* a system all the way from the formulation of the computational problem down to getting the best possible hardware for its solution. One example of this is the design of finite-difference stencils,[7] where different stencil designs are shown to be most efficient in dataflow implementations compared to using conventional CPUs.

Dataflow Compute Hardware

Dataflow engines (DFEs) must be integrated with conventional CPUs, networking, and storage to build a balanced compute system at a cluster level. Within compute racks, the individual Maxeler Maximum Performance Computing (MPC) dataflow compute nodes come in two varieties: MPC-C series and MPC-X series (see Figures 2 through 4). The MPC-C

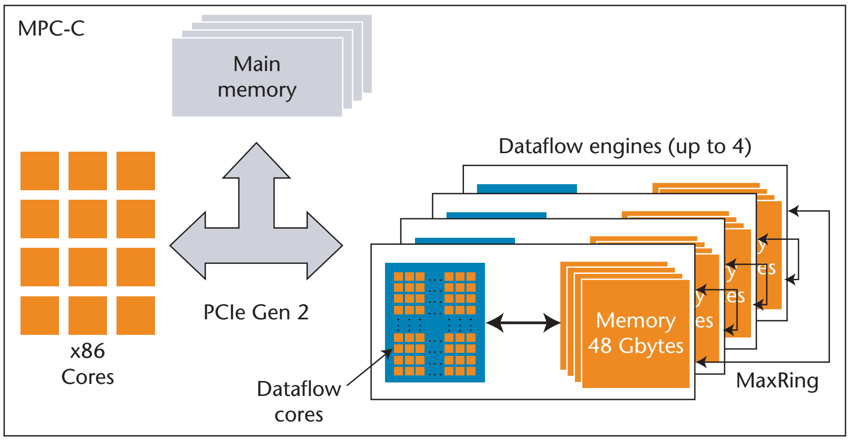


Figure 3. MPC-C series architecture. The nodes contain x86 CPUs directly attached to dataflow compute engines via a second-generation PCI Express bus (PCIe Gen 2).

series are compute nodes containing x86 CPUs directly attached to dataflow compute engines via PCI Express. The MPC-X series are stand-alone dataflow compute nodes that connect to CPU-only nodes in a system via Infiniband. Both the X and C series are dense one-unit (1U) systems where more than 40 can fit in a standard rack. Dataflow engines within a node have an additional high-bandwidth, low-latency direct interconnect between the chips called *MaxRing*, which is utilized for fast communication between neighboring compute elements.

The MPC-C series allows a standalone deployment of dataflow technology with a fixed combination of tightly coupled CPUs and dataflow compute engines, while the MPC-X series allows for a heterogeneous system with a tailored balance of different compute technologies, since the balance of CPUs to DFEs is flexible at runtime. For example, if an application runs on a fixed number of CPU cores and continuously utilizes one or more DFEs, a C series platform could be the best fit; while if the application has several stages (some of which use DFEs) or might require a varying number of DFEs, depending on the particular problem it's running, the X series might be more appropriate.

Multiple applications can make use of the dataflow engines within a system. The system management software (MaxelerOS running within standard Linux) manages the dataflow engines, allocating them to waiting processes as they're freed by other applications—time-slicing the same way that the operating system would for conventional CPUs. This means that the user can concentrate on coding an application for maximum performance without worrying about low-level resource management (for more information, see the sidebar "Programming Dataflow Systems").

Application Acceleration with Dataflow

Dataflow computing doesn't suit all situations; however, a wide range of applications can be effectively accelerated with this approach. Speedups of more than 20× compared to multicore servers have been achieved for applications ranging from large-scale, complex Monte Carlo simulations to irregular, financial tree-based PDE solvers, 3D finite-difference (FD) and finite-element (FE) solvers, and genome processing. In industry, dataflow computing has seen increased adoption in the finance[8] and oil and gas[9] sectors.

Here, we'll discuss one particular application in detail: FD wave modeling. Finite difference is a major numerical method used to solve

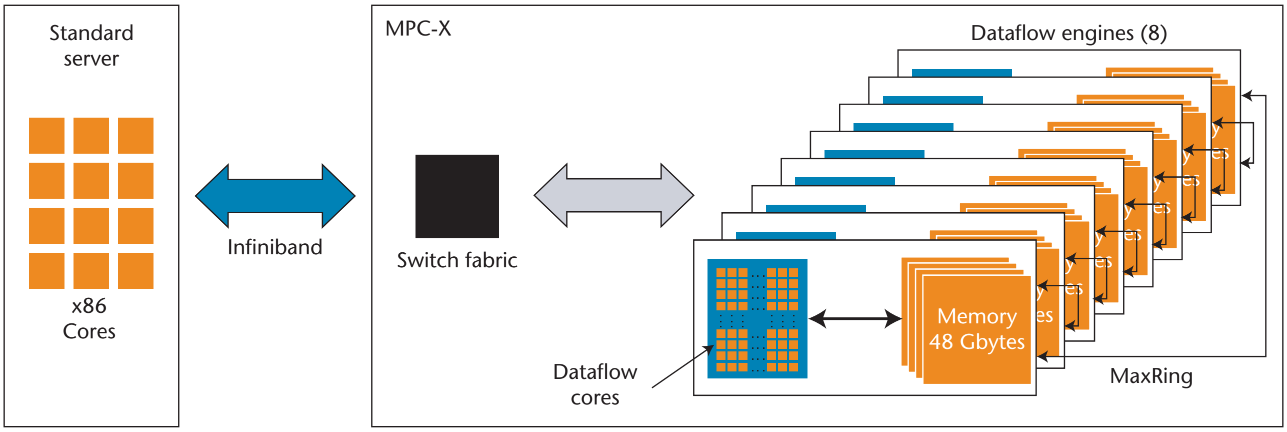


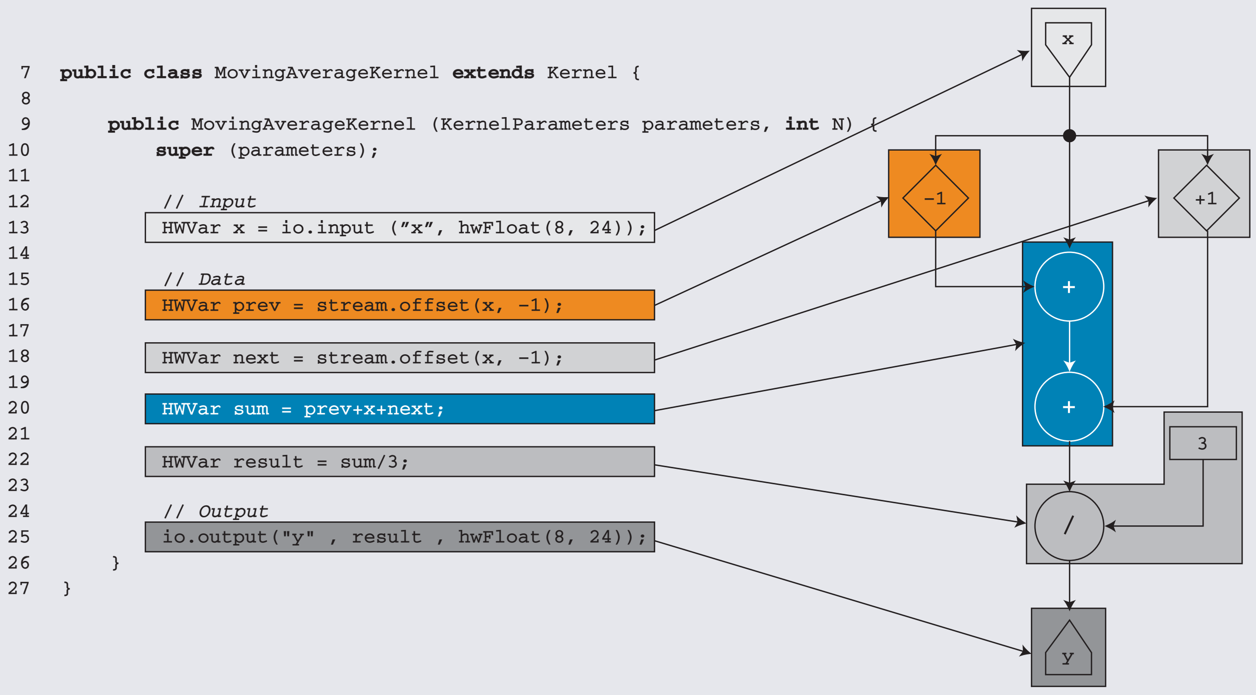
Figure 4. MPC-X series architecture. These are stand-alone dataflow compute nodes that connect to CPU-only nodes in a system via Infiniband.

Programming Dataflow Systems

Maxeler dataflow engines are programmed in Java combined with Maxeler's language extensions to drive dataflow generation. The implementation of a dataflow application in a Maxeler accelerated system comprises two distinct elements: multiple *kernels* and one *manager*, written in Java. Kernels are hardware data-paths implementing the arithmetic and logical computations needed within an algorithm. The manager describes the logic that orchestrates dataflow between kernels and off-chip I/O in the form of streams. Separating computation and communication into kernels and a manager is beneficial, because it enables logic for kernels to be deeply pipelined without running into the synchronization problems that plague parallel programming on control-flow cores.

Kernel programs describe a dataflow graph, defining input and output streams and instantiating and connecting operators to these streams. The compiler transforms the kernels into low-level hardware and generates a hardware dataflow implementation (the `.max` file), which the developer can link into a CPU application using the standard GNU development tool chain.

The example of a moving average kernel can be used to show the structure of a kernel graph. The application computes a three-point moving average over a sequence of *N* data values. In a software implementation, an array would be used to hold the data and would be scanned through with a loop to compute the three-point average for each index. Figure A shows an implementation as a dataflow kernel. Here the current, previous, and next values in the input stream are accessed using *stream offsets* relative to the current position in the stream, and the three values are then averaged. The dataflow kernel operates at the sustained processing rate of one result per cycle; if there's silicon area available, it could be further parallelized to compute multiple outputs per cycle.



```
public class MovingAverageKernel extends Kernel {

    public MovingAverageKernel (KernelParameters parameters, int N) {
        super (parameters);

         // Input
         HWVar x = io.input (”x”, hwFloat(8, 24));

         // Data
         HWVar prev = stream.offset(x, -1);

         HWVar next = stream.offset(x, -1);

         HWVar sum = prev+x+next;

         HWVar result = sum/3;

         // Output
         io.output("y" , result , hwFloat(8, 24));
    }
}
```

Figure A. Source code and corresponding graph for a simple moving average dataflow kernel. The current, previous, and next values in the input stream are accessed using stream offsets relative to the current position in the stream, and the three values are then averaged.

PDEs such as the wave equation. In particular, for the geosciences, medical imaging, and physics simulations, explicit FD is an elegant and regular algorithm that affords efficient implementation within the dataflow paradigm. We compute wave propagation using an explicit FD time-marching approach, starting from the acoustic wave equation:

$$\frac{\partial^2 u}{\partial t^2} = v^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right),$$

where u is pressure and v is acoustic velocity. We can use a Taylor expansion to approximate these derivatives and calculate the result as a convolution. The full-wave propagation computation requires the storage of two 3D pressure volumes for the current and previous state and a model volume for velocity. At each time step, the current and previous states are

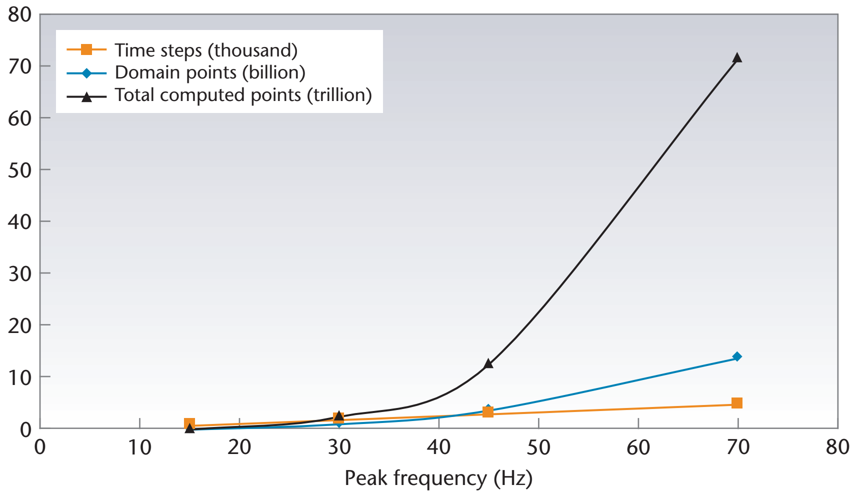


Figure 5. Impact of increased modeling frequency on memory and computation costs. In wave modeling, the amount of computation required increases with the fourth power of the wave frequency.

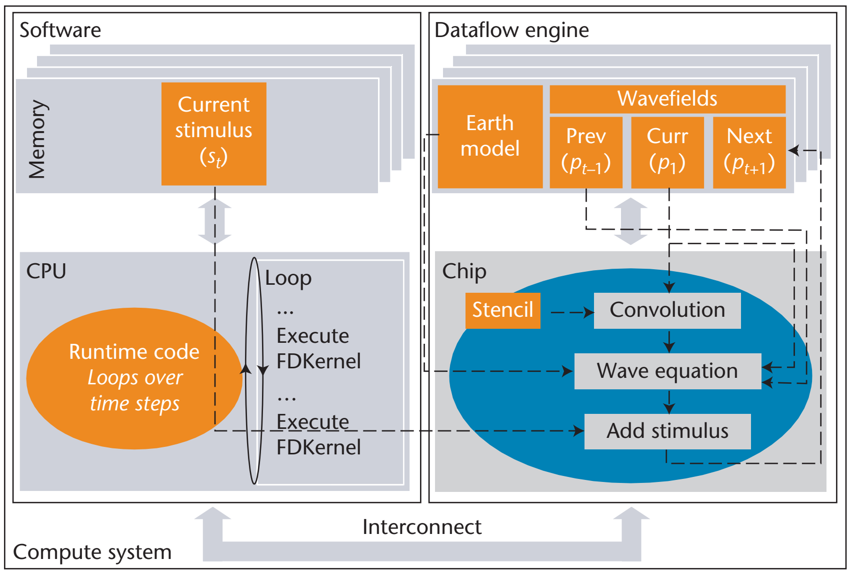


Figure 6. Structure of an accelerated wave-modeling application. The CPUs in the system retain control of the application and instruct the dataflow engine(s) to compute each time step.

computed upon to produce a next-state pressure field. More complex physics (for example, modeling elasticity or anisotropy) operate with the same basic principle but require more data to be stored and more computation.

One of the challenges in wave modeling is that the amount of computation required increases with the fourth power of the wave frequency (see Figure 5), which makes high-performance implementations particularly vital to model high frequencies. Modeling at high frequencies (for example, 70 Hz) can easily require hundreds of gigabytes of memory.

In dataflow-accelerated wave modeling, the CPUs in the system retain control of the application, and instruct the dataflow engine(s) to compute each time step, as Figure 6 shows. Pressure and velocity data volumes reside in the local memory of each dataflow engine and are streamed through the dataflow implementation of the modeling kernel, which computes the next time-step pressure field. Stimulus data can be sent from the CPU to the dataflow engine to be added to points in the field at runtime, and data can also be read out each time step to visualize or store to disk.

Multiple dataflow engines can work together on a single modeling problem, by splitting the domain in one dimension into multiple subdomains and assigning each subdomain to different engines. At the edge of each subdomain, engines must exchange boundary data with their neighbors, and they can do this using the direct MaxRing interconnect. Because MaxRing is a point-to-point interconnect, the communication bandwidth scales as the number of engines increases.

The dataflow approach offers significant performance benefits over traditional CPUs, but the precise impact varies depending on the size of the problem to be tackled. Small problems that run in a short amount of time show a smaller benefit from dataflow acceleration. For example, an acoustic model with variable velocity and density[10] showed speedups of 80× for small problems (using 8 million cells) but more than 200× for larger problems (1 billion cells). For very large-scale problems, the ability to decompose across multiple dataflow engines compared to using message passing interface (MPI) communication on a large cluster can magnify dataflow's benefits. In another example, a single CPU server utilizing 16 dataflow engines to compute on a 14-billion-point domain delivered an equivalent performance to more than 1,800 conventional CPU cores.[11]

Dataflow computing can deliver orders-of-magnitude improvements in space and power consumption for a wide range of applications; and dataflow compute engines can be balanced with other kinds of compute resources in a cluster environment (such as CPUs and storage). The benefits we've seen show considerable promise for achieving the potential of exascale computing.

References

1. M.D. Godfrey and D.F. Hendrey, "The Computer as von Neumann Planned It," *IEEE Annals of the History of Computing*, vol. 15, no. 1, 1993, pp. 11–21.
2. F. Belletti et al., "Ianus: An Adaptive FPGA Computer," *Computing in Science & Eng.*, vol. 8, no. 1, 2006, pp. 41–49.
3. M.C. Herbordt et al., "Computing Models for FPGA-Based Accelerators," *Computing in Science & Eng.*, vol. 10, no. 6, 2008, pp. 35–45.
4. G. Goldrian et al., "QPACE: Quantum Chromodynamics Parallel Computing on the Cell Broadband Engine," *Computing in Science & Eng.*, vol. 10, no. 6, 2008, pp. 46–54.
5. J.B. Dennis, "Data Flow Supercomputers," *Computer*, vol. 13, no. 11, 1980, pp. 48–56.
6. H.T. Kung, "Why Systolic Architectures?" *Computer*, vol. 15, no. 1, 1982, pp. 37–46.
7. H. Fu et al., "Accelerating 3D Convolution Using Streaming Architectures on FPGAs," *Proc. Soc. Exploration Geophysicists* (SEG), SEG, 2009; www.onepetro.org/mslib/servlet/onepetropreview?id=SEG-2009-3035.
8. S. Weston et al., "Rapid Computation of Value and Risk for Derivatives Portfolios," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 8, 2012, pp. 880–894.
9. O. Lindtjorn et al., "Beyond Traditional Microprocessors for Geoscience High-Performance Computing Applications," *IEEE Micro*, vol. 31, no. 2, 2011, pp. 41–49.
10. T. Nemeth et al., "An Implementation of the Acoustic Wave Equation on FPGAs," *Proc. Soc. Exploration Geophysicists* (SEG), SEG, 2008; www.onepetro.org/mslib/servlet/onepetropreview?id=SEG-2008-2874.
11. D. Oriato et al., "FD Modeling Beyond 70Hz with FPGA Acceleration," presentation, Soc. Exploration Geophysicists High-Performance Computing (SEG HPC) Workshop, 2010.

Oliver Pell is the vice president of engineering at Maxeler Technologies. His research interests include high-performance computing technology and scientific computing. Pell received an MSc in advanced computing from Imperial College London. Contact him at oliver@maxeler.com.

Vitali Averbukh is a lecturer in the Department of Physics at Imperial College London and a scientific consultant at Maxeler Technologies. His research interests include electron structure and dynamics, attosecond physics, and ab initio quantum chemistry. Averbukh received a PhD in chemistry from the Technion–Israel Institute of Technology. Contact him at v.averbukh@imperial.ac.uk or vaverbukh@maxeler.com.