Scalable parallelism using dataflow programming

Data flows and dataflow programming are attractive candidates for both modeling and designing parallel computing systems for base stations and mobile terminals.

> ANDERS CARLSSON, JOHAN EKER, THOMAS OLSSON AND CARL VON PLATEN

Dataflow programming was invented to address the issue of parallel computing. Dataflow models are well suited to describe many forms of computation, particularly in the area of digital signal processing.

This article presents case studies about an MPEG-4 video decoder and a multi-standard OFDM receiver. Hardware and software have been synthesized from the dataflow models, and experimental results are presented.

**Introduction**

Support for multiple wireless standards, such as GSM, WCDMA and LTE, is becoming an essential feature for wireless telephony systems. Similarly, mobile terminals must support multiple media formats, such as different MPEG-4 profiles. Keeping power consumption and production costs at attractive points in the design space, while meeting the throughput requirements of such diverse functionality, is a considerable challenge. The trend is to move from fixed-function hardware to configurable and programmable solutions with a higher degree of shared resources and increased software content. Although parallel software is difficult to develop, let alone verify, the so-called “power wall” makes parallel software deployed on parallel architectures a necessity.

Models for functional programming and dataflow programming – the focus of this article – have the potential to facilitate parallel programming. To make this concept practical, however, development tools must evolve. Signal processing is often modeled as flows of data. A circuit diagram, for instance, is a form of dataflow description, especially at the block diagram level. Many forms of computation are well suited to dataflow description and implementation. Some common examples include complex media coding, network processing, imaging and digital signal processing, and embedded control.

Digital radio is one application domain that can be successfully modeled using data flows. Dataflow modeling facilitates component isolation and “pluggable” processing chains. Performance-enhancing functionality, such as hybrid automatic repeat request (HARQ), can be implemented as an optional plug-in. Given that each new generation of mobile terminals is expected to support a larger number of radio standards, more functionality is being implemented in software or programmable hardware. Building a processing chain from pluggable modules – either standard-specific or universal with parameters – is therefore an attractive proposition.

Dataflow modeling is not only a means of creating transceiver implementations. The use of dataflow modeling can also reveal bottlenecks from a lack of parallelism inherent in a radio standard. This is especially useful when developing new standards. To support a wider range of maximum communication speeds, next generation radio standards will be scalable in terms of capacity and processing requirements. Ideally, it should be possible to build a high-capacity transceiver by duplicating the hardware of a baseline transceiver. To do this, the potential parallelism should scale in a linear way with processing requirements.

Parallel programming is not a new proposition or challenge. At the end of the 1960s, researchers were struggling to make the transition to paral-
level computing. At that time, processing performance doubled about every 18 months, a trend that continued for the next three decades. For the most part, increasing clock frequencies drove this development. As a result of the rapid advance in performance, sequential programs proved sufficient for many computing disciplines, but this computer hardware development trend is over. Ever-increasing clock frequencies came to an abrupt end in 2005 due to, among other things, fundamental problems associated with power supply and heat.

Future advances in performance are expected to come from greater use of parallelism, which is facilitated by improved transistor density. Multicore computers are an example of this development, and forecasts for other sectors point in the same direction. For portable consumer electronics, the International Technology Roadmap for Semiconductors (ITRS) 2009 edition foresees exponential growth in parallelism, resulting in hundreds of parallel processing elements within five to ten years.

To leverage this development, the performance of an application or system needs to be scalable through parallelism. However, most applications have been built to accommodate the tradition of constantly increasing clock frequencies and are inherently sequential and thus not susceptible to parallelization, presenting a challenge to the software industry. In contrast, the telecom sector is in a good position, thanks to its practice of designing parallel systems. The task is to make efficient use of increasing parallelism, rather than enabling parallel execution.

Several research centers have been set up and many projects have been launched to address the challenges of multicore processors. Examples include the Parallel Computing Laboratory (ParLab) at the University of California at Berkeley, the Pervasive Parallelism Laboratory (PPL) at Stanford University and the HiPeac European Network of Excellence.1-2 Collaborations with external partners include ACTORS3 and DSL4DSP4 Ericsson also actively researches in this area.

**Current practices**

C and related programming languages are dominant for software in consumer electronics, embedded multimedia and communications equipment. However, C’s control over low-level detail, usually considered a good thing, tends to overspecify programs. Besides specifying algorithms, C specifies how inherently parallel computations are sequenced, how inputs and outputs are passed between algorithms and, at a higher level, how computations are mapped to threads, processors and application-specific hardware.

Using analysis, it is not always possible to recover the original knowledge about the program and the chances for restructuring transformations are limited. Therefore, C is not a good starting point for parallelization.

Tools and frameworks, such as OpenMP5 and the message-passing interface (MPI)6, are used to facilitate the construction of parallel C programs for multicore and multiprocessor systems. These tools are, however, severely limited, as they can only expose parallelism when it has been explicitly identified by a programmer and they do not support programmers in assessing the correctness of parallelization.

In hardware design, the current practice is to create a low-level description, also referred to as Register Transfer Logic (RTL). The abstraction level of RTL tends to be low, which results in long development cycles impacting time to market.

Great gains in hardware efficiency can be made from architectural optimization. For this reason, it is desirable to raise the abstraction level of hardware descriptions, and as a consequence, the use of high-level tools.

Because RTL is inherently parallel, it is easy to translate a parallel description in a higher-level language into RTL. Unfortunately, descriptions in traditional programming languages, such as C, do not map equally well because of the difficulty in resolving the sequential nature of traditional programming languages into parallel hardware.

**Dataflow programs**

A dataflow program is defined as a directed graph, where nodes represent computational units or actors, and arcs represent the flow of data — communication channels that connect the actors. An actor is solely concerned with mapping its input to output. Dependencies are expressed by connecting actors; there is no other source of dependence. These properties make dataflow programs very flexible in terms of partitioning and sequencing of computations.

It has been shown that dataflow models offer a representation that can effectively support the tasks of parallelization and vectorization, thus providing a practical means of supporting multiprocessor systems and utilizing vector instructions.5-6 Interestingly, as dataflow programs grow in size they tend to expose more parallelism.

In parallel computing, a distinction is made between

- parallelism that scales with the size of the problem, known as data parallelism and
- parallelism that scales with the size of the program, known as task parallelism.

Scaling an algorithm over larger amounts of data is a relatively well-understood problem that applies to dataflow programs, as well as other programming models. While a dataflow program has a straightforward parallel composition mechanism, it is difficult, for example, to compose a C program, whose parts execute concurrently without interference.

There are several classes of dataflow programs. These classes differ in expressiveness and analyzability. At one end of the spectrum are Kahn process networks, which can express any kind of computation.9 Though not all the interesting properties of these networks can be established by program analysis (Kahn process networks are Turing complete). At the other end are synchronous dataflow networks, for which static schedules and bound memory requirements can be determined in order to make statements about the absence of deadlock.7 The drawback is limited expressiveness, as synchronous dataflow cannot express control flow, such as input-dependent iteration.

Constrained dataflow models, such as synchronous dataflow, can be synthesized into particularly efficient code, whereas more general forms of dataflow programs, such as Kahn pro-
actions. The untyped Select actor (see the port patterns, which in this exam-
ple, corresponds to one token on ports A and B.

```
action Add() int A, int B => int Out
  action A:[a], B:[b] => Out:[sum]
  var int sum = a + b
end
```

An actor can take any number of actions. The untyped Select actor (see below) reads and forwards a token from port A or B, depending on the evaluation of guard conditions.

```
action Select() bool selector, int A, int B => int Out
  action selector:[s], A:[a] => [a] guard s end
```

The above examples are quite trivial, as typical actors can be several hundred lines of code. Sorting, for example, can be implemented with a pair of actors (one that contains two actions and another that contains four actions) together with a network that links several instances in a chain according to the amount of data to be sorted.

CAL enables several ways of expressing and controlling the flow of data. Two actors may never share a state and their only means of communication is via the actor ports. The runtime system is responsible for how and when an actor is scheduled. An actor is simply a specification that describes the actions that will occur in response to the presence of data. The CAL actor model allows a complete separation between the scheduling of actors and the algorithm that an actor's network specifies. The same network can be scheduled in a variety of ways, all resulting in the same functional result, but with different timing and computational properties.

ACTOR

```
actor Add() int A, int B => int Out
  action A:[a], B:[b] => Out:[sum]
  var int sum = a + b
end
```

An actor can take any number of actions. The untyped Select actor (see below) reads and forwards a token from port A or B, depending on the evaluation of guard conditions.

```
action Select() bool selector, int A, int B => int Out
  action selector:[s], A:[a] => [a] guard s end
```

The above examples are quite trivial, as typical actors can be several hundred lines of code. Sorting, for example, can be implemented with a pair of actors (one that contains two actions and another that contains four actions) together with a network that links several instances in a chain according to the amount of data to be sorted.

CAL enables several ways of expressing and controlling the flow of data. Two actors may never share a state and their only means of communication is via the actor ports. The runtime system is responsible for how and when an actor is scheduled. An actor is simply a specification that describes the actions that will occur in response to the presence of data. The CAL actor model allows a complete separation between the scheduling of actors and the algorithm that an actor's network specifies. The same network can be scheduled in a variety of ways, all resulting in the same functional result, but with different timing and computational properties.

Reconfigurable video coding
Multimedia processing can easily be modeled using data flows. A recent development within MPEG makes CAL a suitable language for experiments in this area. The MPEG reconfigurable video coding (RVC) framework is a new ISO standard that aims to provide video codec specifications at the level of library components, instead of monolithic algorithms. The basic idea is to specify a decoder by selecting components (or actors) from a standard library of coding algorithms. The ability to dynamically configure and reconfigure codecs calls for new methodologies and new tools for describing the bitstream syntaxes and parsers of the new codecs.

The RVC framework uses the CAL actor language to specify the standard library and instantiation of the RVC decoder model. RVC is composed of two ISO/IEC specifications.11-12 For more information about RVC, see Overview of the MPEG reconfigurable video coding framework.13

Globally asynchronous locally synchronous (GALS)
A coarse-grain asynchronous architecture, such as GALS, makes it possible to exploit benefits of synchronous, as well as asynchronous design methodologies.14 Ordinarily, clock skew in synchronous digital designs is limited by the implementation of a balanced clock tree. The goal of a balanced clock tree is to make all clock delays equal. However, achieving equal delays in the clock tree becomes increasingly difficult as system complexity increases.

GALS designs reduce clock skew constraints and give smaller clock trees, because clocking is not a global issue in GALS. GALS might reduce induced noise from the clock to the analog domain, as noise from digital parts comes from several smaller clock domains rather than from one large clock domain. The switching noise from clock nets is therefore spread out over time. Additionally, the lack of global synchronization makes it possible to save energy in the clock net.

There is a strong coupling between GALS and dataflow descriptions. The use of first in, first out (FIFO) channels to communicate between processes in dataflow descriptions can translate directly into a GALS specification.
dataflow processes are implemented as synchronous blocks and the FIFO channels are implemented as asynchronous communication channels.

CAL dataflow networks support multiple clock domains, with up to one clock domain per actor. Replacing the FIFO channels between clock domains with asynchronous FIFO channels transforms the dataflow into a GALS network, making the CAL dataflow an attractive candidate for modeling GALS architectures. Because GALS designs consist of modules with a well-defined interface, GALS is also conducive to modular design. Work to try to transform CAL dataflow to a GALS network is ongoing.

**Case studies**

The following case studies illustrate different applications of dataflow programming, where both the hardware and software have been synthesized from CAL dataflow models.

**OFDM inner receiver**

A digital radio receiver can be divided into an analog front-end, a digital front-end, the inner receiver and the outer receiver. The inner receiver, as shown in Figure 1, demodulates the baseband signal, converting analog waveforms to codeword probabilities. In this example, parts of an inner receiver were implemented for OFDM-based radio systems, to test the dataflow methodology for digital radio.

A basic OFDM inner receiver consists of a synchronizer, frequency error compensator, fast Fourier transformer (FFT), channel estimator, equalizer and demodulator. Of these parts, the implementation comprised the frequency compensation, synchronizer and FFT.

1. the synchronizer, which estimates time position and frequency error with a coordinate rotation digital computer (CORDIC) rotator to compensate for digital frequency error; and
2. FFT – in this case, a configurable FFT that supports a maximum symbol length of 8k samples.

The specification language used was CAL. The dataflow description was synthesized to an off-the-shelf FPGA-based development platform using Open Dataflow (OpenDF) and OpenForge tools. Test data was streamed over Ethernet to the development board and the result was displayed on an attached VGA display. An on-chip processor handled the Ethernet streaming and data display. The synthesized hardware is capable of processing 50 Megasamples/s, which is sufficient for real time. For more details on the implementation, see *Reconfigurable OFDM Inner Receiver Implemented in the CAL Dataflow Language.*

It was possible to reach the stated performance goals, but the CAL implementation used more resources than a comparable RTL implementation, to a large extent due to the relatively crude tools. Essentially, there is a one-to-one relationship between the CAL code and the RTL, placing the burden of optimization on the RTL toolchain. Several sources of overhead have been identified in the CAL to RTL translation, and work is ongoing to drastically reduce this.

In another case study, an MPEG-4 decoder was specified in CAL and implemented on an FPGA. The code generated from the CAL specification outperformed a reference in VHDL (a hardware description language), achieving better performance with less hardware resources. The CAL implementation required significantly less development effort.

CAL encourages reasoning relating to interfaces and structure. Contributing factors are: strict isolation of the actor’s internals, the asynchronous token interfaces and the hierarchical network modelling. These features also limit the impact of actor modifications.

As a result, it is comparatively easy to restructure a system until it meets the desired performance requirements. Ultimately, this leads to significantly reduced developer effort for a given functionality, with minimal or no penalty in area or performance; particularly for large and complex problems.

However, for small systems with regular structure and wide data paths, like the FFT, the overhead of the flow control and action scheduler appear to overshadow the structural gains.

**Multicore MPEG-4 decoder**

In a second case study, an MPEG-4 simple profile decoder was implemented in software on a 200MHz quad-core ARM11 MPCore, using the code-generation tools developed within the ACTORS project. The dataflow model was taken from the RVC toolbox. The main functional units – themselves hierarchical compositions of actor networks (Figure 2) – consisted of:

![Dataflow model of the MPEG-4 simple profile decoder.](image-url)
Efficient realization of a dataflow model on a multicore architecture involves the challenge of how to partition the workload. While the semantics of the dataflow model allow any of the actors to be deployed on any of the cores without violating data dependencies, the way actors are partitioned impacts greatly on system performance. Two (sometimes conflicting) objectives that need to be considered are balanced load over the cores and minimal communication cost. Partitioning can be performed statically (offline) or dynamically (at runtime).

Naive execution of a dataflow model is associated with significant overhead. In the first, naive, attempt, each actor was executed in a separate thread, synchronized with FIFOs in shared memory. This way, the scheduler of the operating system (SMP Linux) managed load dynamically. In a second attempt, the actors were partitioned statically and each core executed a single worker thread. This approach improved performance by several orders of magnitude. Clearly the first approach suffered badly from the overhead of context switches. Efforts are being made to pursue the idea of partitioning actors dynamically, but as yet, a competitive solution has not been devised.

Figure 3 shows the results of partitioning the MPEG decoder onto one through four cores. Good speed-ups were achieved by increasing the number of cores. The partitions were, however, found manually, meaning that this solution remains unexplored for the most part. More advanced tool support would further speed up the search process and greatly improve the quality of partitions.

There is room for improvement when it comes to frame rate. The current solution is still immature, compiling actors separately and treating them as separate entities, even at runtime. Given the fine level of granularity to which the MPEG decoder is specified, the overhead of this execution model is high relative to the useful work it performs. The Model Compiler, a recent development of the ACTORS project, allows several actors to be synthesized jointly. For an earlier position on the model compiler, see For classic work in this area see references 7 and 20. It is anticipated that joint synthesis of actors will significantly improve performance by reducing the overhead associated with data transport and scheduling.

**Conclusions**

The historic trend of ever-increasing clock frequencies has come to an end. In the future, gains in performance will be derived from increased parallelism. Therefore, systems whose performance scales with increased parallelism make an attractive proposition.

A particular problem with current practices is over-specification. Software tends to be specified in such a way that computations are serialized, failing to expose the potential of parallelism. Low-level specification of hardware (such as RTL) is costly to develop and the specification of timing complicates the refactoring of designs.

Dataflow models are well suited for describing many forms of computation, particularly in the digital signal processing domain. As such, dataflow programming is an interesting approach that can be applied to highly parallel computing systems found in mobile terminals and base stations.

A dataflow program is composed of computational units called actors that may but not necessarily need to execute in parallel. This very flexible programming model allows for exploration of the design space. In particular, a dataflow program can be partitioned into:

- separately timed hardware components – which is useful in the context of partly asynchronous GALS architectures; or
- separate software threads to be executed on a multicore CPU.

Dataflow models have been used in two case studies to specify an MPEG-4 video decoder and a multi-standard OFDM receiver. The hardware and software were synthesized from the models. Much of this work was performed within the European FP-7 ACTORS and MultiBase projects, using tools and methodologies made available by the OpenDF initiative. MPEG RVC models were used in the video decoder case study, while the OFDM model was developed as part of this work.

The ability to make rapid design iter-
ations was of great value when developing hardware from the OFDM model. It was possible to explore an array of design ideas on the architectural level in limited development time. The performance targets were met, but the resource requirements (corresponding to silicon area) exceeded those of a comparable RTL design. The software, which was synthesized from the MPEG RVC model, could be partitioned so that performance scaled with the number of available cores. However, performance on a single core is much lower than that of a comparable implementation in C.

The case studies illustrated some interesting properties of dataflow programming. In particular, the models could be partitioned and refactored in a flexible manner. Further improvement of the OpenDF tools is required to make the synthesized code more efficient. It appears that an increased multiplexing of hardware resources and a reduction of the execution overhead in software would greatly improve the usefulness of the tools.

Anders Carlsson

is a senior researcher in the Modem Hardware Group at Ericsson in Lund, Sweden. He received his Lic.Eng. in industrial electrical engineering from Lund University, Sweden in 1998. He then joined ENEA, where he worked on DSP algorithms and realtime scheduling. He joined Ericsson Research in 2007, where he has been working on realtime multicore scheduling and dataflow programming. He is currently a member of the European FP-7 research project MultiBase.

Carl von Platen

is a senior researcher in the Device Software Group at Ericsson Research in Lund, Sweden. His research interests include development tools for embedded systems, code generation and code optimization. He received an M.Sc. in mathematics from Lund University in 1992. He is currently leading the work package on dataflow compilation in the European FP-7 research project ACTORS.

Thomas Olsson

is a senior researcher in the Modem Hardware Group at Ericsson in Lund, Sweden. He received his Ph.D. in ASIC design from Lund University in 2004. In 2004 he joined Ericsson in Lund, and in 2005, he joined Ericsson Research. Here he has worked with digital circuit design and digital system implementation, primarily for OFDM systems such as LTE. He is currently a member of the European FP-7 research project MultiBase.

Johan Eker

is a senior researcher at Ericsson Research in the Device Software Group in Lund, Sweden. He has a Ph.D. in automatic control from Lund University (1999). After a brief stint in industry, he joined the Ptolemy group at University of California, Berkeley in 2001, where he co-designed the CAL Actor Language, which was recently adopted by ISO/MPEG. He has been with Ericsson Research since 2003 and is co-coordinator of the European FP-7 project ACTORS, which focuses on dynamic resource management and multicore programming.

References

3. www.actors-project.eu
4. dsi4dsp.inf.elte.hu
5. The OpenMP API Specification for Parallel Programming, http://openmp.org
21. www.multibase-project.eu
22. www.opendf.org