MEETING ENGINEER EFFICIENCY REQUIREMENTS IN HIGHLY PARALLEL SIGNAL PROCESSING BY USING PLATFORMS

Anders Åhlander¹, Anders Åström², Bertil Svensson¹ and Mikael Taveniku³

1. Centre for Research on Embedded Systems, Halmstad University, Halmstad, Sweden
2. Airborne Radar Division, Ericsson Microwave Systems AB, Mölndal, Sweden
3. XCube Communication, Westford, MA, USA
Email: anders.ahlander@ide.hh.se, anders.astrom@ericsson.com, bertil.svensson@ide.hh.se, micke@x3-c.com

ABSTRACT
One of the driving forces behind the development of new, highly parallel architectures is the need for embedded high-performance computing. The development of advanced applications on such architectures is, however, potentially connected to high costs. Cost-effective development requires tools and processes that provide engineer efficiency, in this case tools and processes that help the developer master the application complexity. Related to engineer efficiency are the important concepts of system sustainability and flexibility. To address these issues, a platform approach can be taken. The platform should offer an understandable and stable development model, and at the same time give the possibility to take advantage of the rapid technology development, including the use of new parallel architectures. Thus it must support multiple hardware targets, and the development model should decouple application development from mapping aspects. Two radar signal processing examples, one compute-intensive STAP and one data-intensive SAR, are used to illustrate the need. The GEPARD platform is presented as an example of our approach, and we argue that the described platform is a good fit for advanced signal processing development, facilitating the desired engineer efficiency, sustainability, and flexibility.

KEY WORDS
High performance applications, parallel and distributed architectures, tools for application development, abstract platforms, radar signal processing.

1 Introduction and motivation
Many of the most performance-demanding signal processing applications (such as advanced radar signal processing) require dedicated, or at least application-oriented, architectural solutions and they offer significant challenges in terms of algorithm complexity. At the same time they are developed only to a limited set of customers.

To cope with the computational demands, the applications will typically have to run on highly parallel computer architectures [1][2][3][4]. Today, there are a number of such stream computing systems on the market, where systems from Mercury Computer Systems [5] and SKY Computers [6] are examples. Basically, they all rely on high-performance COTS microprocessors, e.g., Motorola PowerPC, with an interconnect network, such as Mercury RACE or SKYChannel, together with a software platform for application development. In addition to the established, commercial systems there is a multitude of new and emerging parallel architectures, such as the Stanford University Imagine [7] and the MIT RAW [8], aimed at demanding streaming applications.

The applications represent major investments in system software, as well as in optimization of algorithms. A cost-effective development of complex applications requires tools that put engineer efficiency first. In other words, the hardware and software platforms, as well as the application development process itself, must be designed to maximize efficiency of the engineers developing the application.

In many cases the application must be supported over many years of lifetime, i.e., there is a requirement to acquire sustainability. Therefore, the compute model needs to accommodate several technology cycles (preferably technology independent) and it should be decoupled from any specific vendor or interface. Another factor that comes into play in developing systems with a long life span is that it is vitally important to be either 100% sure that the chosen vendors will be there for the life of the program, or that there is a clear and “easy” path to replace a vendor’s system with another’s.

A third requirement is the demand for flexibility. Since different end-users require different variations of the system, it must be easy and efficient to enable various options as well as testing and deploying them. There is also a growing demand that the system, while working in real time, can be configured – or reconfigure itself – in order to adapt to the situation.

All these factors lead us to propose that a modular, platform based approach should be taken when developing advanced, industrial signal processing systems. The need
for cost reduction in the software, hardware and application design processes speaks even more in favor of this approach. In Section 2 we look closer into these issues to get a deeper understanding of what the general requirements on such a platform are. Then, in Section 3, we characterize the specific demands of radar signal processing. In Section 4, as an illustration of the approach, we introduce GEPARD, a new virtual platform concept for a variety of parallel architectures, and describe the user view of such a system. We demonstrate the feasibility of the approach in Section 5 by mapping two radar signal processing tasks on the platform. Finally, the paper ends with a related work and a conclusions section.

2 Cost-effective application development

2.1 Engineer efficiency

Obtaining engineer efficiency in signal processing application development is a question of mastering complexity. The increase in complexity comes from both an increase in computational load and a more complex interaction among the involved processes. The aggregated complexity in the engineering process is not necessarily proportional to the amount of data or the number of operations on it. It is more dependent on the amount of processing per piece of data and the interactions dependent on that processing. For instance, going from traditional MTI (moving target indicator) radar signal processing to STAP (space-time adaptive processing), which implies an increased number of operations per data item and more complex mathematics, is definitely considered an increase in complexity. On the other hand, going from 16-channel to 32-channel STAP (constant number of operations per data) means a substantial increase in computational load, but does not (significantly) increase the complexity for the design engineer.

For any particular application in this area, as it evolves over time the computational complexity grows approximately an order of magnitude per decade. Typically, over the lifetime of the application, we also see a growing complexity in the algorithms, an increased number of working modes as well as a more dynamic use of the available modes. Thus, the challenges in taking in and understanding the whole application become more real, and there is a risk that this results in a prolonged development process. However, the rapid technology development may provide the key to solving the problem. According to Moore’s law, which is anticipated to hold at least until 2010, we see a doubling in VLSI capability every 18 months. Per decade, this is two orders of magnitude. This is actually a ten times faster growth than the complexity growth in a given application during the product life-time. In other words, hardware performance will improve relative to the complexity of the problem. As the application evolves over time, it will become more complex from an application standpoint (refined algorithms and functions), but at the same time it will be increasingly easy to implement the computations in computing hardware. Over time this means that a particular application can evolve from a pure special purpose hardware implementation through specialized hardware architectures, to an off the shelf multicomputer and, eventually, to pure commodity hardware.

Thus, technology improvement makes it possible to trade hardware performance for reduced development time by using higher levels of abstraction.

2.2 Sustainability

In radar systems, as well as in many other signal processing applications, there may be a mismatch between the lifetime of a technology generation and the lifetime of the application. A typical radar application has a life span of 15 to 20 years, which translates to many generations of technology. This has several interesting consequences – opportunities as well as problems: First, the customer may expect functional growth in accordance with the current level of technology. Second, components used to implement the system may become unavailable. Third, new hardware provides potential for significant savings in cost per unit. All these factors imply that it is crucial to be able to easily move an application from one hardware platform to another, as well as to add functionality.

In order to design a sustainable system, the following issues should be considered.

- **Technology insertion.** Over the product lifetime it will be necessary to upgrade the software (in terms of functionality) and refresh the hardware (either for obsolescence or cost/size improvement).
- **Scalability.** It must be easy to scale up or down in order to tolerate different hardware implementations of the system. Implementations may vary in terms of number of processors (which affects the mapping of the calculations) as well as in terms of what type of hardware devices are used (such as field-programmable gate arrays, digital signal processors, processor arrays, and microprocessors). Even if it is hard to reach seamless scalability, upgrading technology needs to be relatively easy and predictable.
- **Layered application development.** Application development should rely on a modern, sustainable codebase, in which a clear separation of hardware features from the application requirement is made. This means that the view provided in the application design should be independent of the hardware design. Likewise, mapping of software modules onto hardware should be done in an application-independent way.
- **General-purpose vs. special-purpose hardware.** Certain functions can be implemented in accelerators as long as there is a clear path (and a clean interface) to replace the hardware when it becomes obsolete.

The issue of sustainability is, of course, tightly connected to engineer efficiency, because the demands must be met in a cost-effective way. In the following sections it will be
demonstrated how the above issues are supported by the suggested platform concept.

2.3 Flexibility
The demands for flexibility must be met. For instance, it should be possible to enable software options for different products or for different users – for example, certain functions may not be present in some versions, and other versions may have tweaks to parts of the algorithms depending on customer specific requirements. Other variations include using different types and numbers of sensors. In order to keep reasonable bounds on development time and cost with these variations, efficient testing and deployment of the systems is essential.

Furthermore, there is the requirement that the system needs to be configurable for the mission at hand or to meet changing demands on the fly. This may, for example, include special modes depending on the environment, expected scenarios, etc., all this while working in real-time.

2.4 Using platforms
Motivated by the demands for engineer efficiency, sustainability and flexibility, the use of platforms, as described in this paper, is an approach to provide application development environments that give:

- the possibility to take advantage of the rapid technology development as a way to shorten application development time,
- a decoupling of system software from the hardware implementation,
- the availability of multiple implementation options for any given application,
- a simple way to replace or add hardware modules,
- a layered development of application software and support for re-use of software components, and
- scalability in terms of problem size as well as technology development.

3 Radar signal processing
Modern radar signal processing, with its short series and high complexity, is a good example of an application area where cost-efficient development is put in focus.

Signal processing in future-generation active electronically scanned array antenna (AESA) radar is indeed a challenging problem. AESA systems will consist of hundreds of antenna elements and will have to process much larger data quantities than current systems do. Moreover, for airborne applications there are additional requirements of small space and low power consumption. The computers should also be capable of scaling in multiple dimensions, such as the number of input channels and the types of algorithms.

Two extremes, with regard to the memory and performance demands, are the space-time adaptive processing (STAP) and the synthetic aperture radar (SAR) processing. Their positions in the design space are illustrated in Fig. 1. These applications, which in many senses are dimensioning for the signal processor, are further described below.

![Fig. 1. The memory and performance demands in STAP and SAR.](image)

3.1 Space-time adaptive processing
One of the main advantages of the AESA is the capacity to extract not only temporal, but also spatial information, e.g. the direction of incoming signals. By estimating the arrival directions and other properties of unwanted signals in the sidelobes, the signal processing can suppress them by applying complex valued weights to the signals from the antenna channels [9][10]. This type of radar signal processing is generally referred to as STAP.

In its basic operation the STAP algorithm uses the received data to estimate the clutter and jammer interferences and from this estimation compute a set of weights that is applied to the data. As the estimation is based on the data itself, the algorithm is categorized as adaptive. The STAP is massive – hundreds of GFLOPS in sustained performance is required for the critical operations. The data amounts are however moderate, often less than 10 MByte per data block.

A sample STAP chain is shown in Fig. 2. The figure illustrates how the data is processed, step by step. The bars in the data blocks show data subsets in which there are dependencies between the data elements. A subset consists of one or several data vectors on which a linear function is applied. Here the functions FIR (Finite Impulse Response) filtering, matrix-by-vector multiplications, FFTs (Fast Fourier Transforms), and QRDs (QR decompositions), respectively, are applied in the different processing steps.

In the figure typical data size and performance numbers are shown. The computational demands are high and therefore the data set has to be split over several processors which share the load. The data set is split in such a way that all data dependencies in the linear functions are kept within one processor’s memory. This is achieved by following the slicing directions indicated in the figure. As can be seen, the slicing direction may change from one processing step to another, which means that the data has to be reorganized between these steps. This stresses the importance of the interconnection network. A particular challenge considering the data partitioning arises in step 4 where the QRDs are applied on overlapping subsets of the data block. This implies that some data elements have to
be duplicated in the partitioning process.

Fig. 2. Processing in a STAP chain.

3.2 Synthetic aperture radar processing
The SAR application produces a map (in real time) of the ground while the aircraft is flying past it. In contrast to STAP, SAR processing has a rather low operation count per input point, but, on the other hand, operates on massive amounts of data. In SAR the radar transmits a relatively wide beam to the ground illuminating each resolution cell over a long time, during multiple pulses while the aircraft is moving. The effect of this movement is that the distance between a point on the ground and the antenna will change over the data collection interval. This change in distance is different for each point in the area of interest. The task of the signal processing chain is to, for each resolution cell in the output image, integrate the actual data that a target in that particular cell would have. The integration is done along curve segments that differ for each point. There are several algorithms that can be used to compute this integral, the choice is a trade-off between accuracy and computational load.

A special case of SAR is a stripmap mode [11], the computation of which is shown in Fig. 3. The figure is, regarding data dependencies etc., interpreted in the same way as the figure for the STAP case. The SAR application can work on data sets in the order of 80 GByte with data integration times of up to 50 sec. The processing mainly comprises FIR filters, FFTs, and IFFTs (Inverse FFTs). The performance need is tens of GFLOPS. Also in this case the data has to be split over several processors, as indicated in the figure. However, in this case the reason is primarily the large data set. In Steps 1-7 the data slices overlap by (FIR filter length – 1), due to the sliding FIR-filters in Steps 3 and 6.

3.3 Implications for the computer platform
A platform can in this context be seen as a virtual architecture, designed for an easy mapping of applications onto its virtual resources. The two application examples above imply that the execution has to be done on a parallel computer system. The platform must therefore support efficient development and execution of applications on parallel hardware. Since it must be possible to run several different applications concurrently (multi-mode processing), the platform must provide means for advanced execution control.

From a developer’s point of view the platform must offer a straightforward mapping of signal processing applications, including multi-mode processing applications, onto the platform’s virtual processing resources. This includes powerful mechanisms for data splits and reorganizations, and means for describing run-time execution control. It is equally important that the platform’s virtual resources can be mapped on the actual hardware with high utilization of processors, memories and communication devices. It can also be noted that there are some heavy signal processing steps that may require special purpose hardware nodes (e.g. ASICs) for efficiency reasons. This implies that the platform should support the use of a heterogeneous, parallel computer system.

Fig. 3. Processing in a SAR chain.

4 The GEPARD platform concept
GEPARD is a concept developed with the purpose to have a common virtual platform for signal processing in a range of different radar products. The platform is developed for parallel signal processing with the important issues of engineer efficiency, sustainability, and flexibility in focus.

4.1 Requirements
The goal of having a common signal processing platform requires consensus among the developers of different radar products; the virtual platform should for instance

- have support for highly parallel, multi-mode signal processing, including run-time control functionality,
- provide an application programming interface, API, with (i) signal processing functions, e.g. FIR and FFT, (ii) data partitioning and reorganization functions, and
control communication is packet oriented. The resource and communicat
on model of the GEPARD physical parallel computer architectures, and at the same
time is a natural target for signal processing applications. The resource and communicat
on model of the GEPARD concept does not hide the hardware completely, only its
details. This facilitates having both an efficient physical realization and a natural application mapping. The model
is based on independent, distributed nodes in a network, see Fig. 4. The system is heterogeneous, i.e. the nodes can
be based on microprocessors, as well as DSPs, FPGAs, processor arrays or ASICs. Four main node types are identified: controllers, workers, data reorganizers, and I/O nodes. Nodes in a system are unaware of each other, they only communicate with two or several work queues. The data to be processed is placed in the distributed data buffer, which can be reached by all worker nodes. The control communication is packet oriented.

4.2 Resource and communication model
We aim to find a virtual resource and communication model that can be efficiently realized on many different
physical parallel computer architectures, and at the same time is a natural target for signal processing applications.

4.2.1 Basic model components
Control packets. Packets consisting of control information and references to data are sent between the nodes in the system. The control packets have four layers:

- Control layer: A “to do list” of operations on the data, i.e., information that the worker node needs for processing (e.g., operation type, data partitioning information, timing, and queue information).
- Supervision layer: Registrations, task logs etc.
- Processing mode layer: System parameters such as choice of pulse repetition frequency.
- Data layer: Size, organization and reference to data.

The principle for how the control packets are built is shown in Fig. 5. Basically, every data batch that enters the system is tagged with a “to do list” that describes the signal processing to be performed on the batch. This makes it easy to make a mode change between integration intervals.

Controllers. Each platform contains at least one controller node. The controller delegates the work to be done in the system. It gets control packets from the Ready queue and puts them in the appropriate Job or Data reorganization queues. The controller could be more or less intelligent. An intelligent controller selects the proper worker queue depending on the individual worker’s load status, instead of just letting the type of task determine the queue. This gives a better utilization of the system, but on the other hand, the controller becomes more complex and gets increasingly loaded when the number of nodes scales up.

Workers. A platform can be equipped with an arbitrary number of worker nodes. The number and types of nodes can be changed over time, which provides for a scalable system that is easy to develop. The worker nodes perform the actual signal processing, by processing packets that hold control information together with references to data. The worker gets a packet from the Job queue, it fetches the data (perhaps from multiple sources) referenced by the packet, and does the processing. After processing, the packet is modified to point to the resulting data in the worker’s outbuffer, and is put in the Ready queue. The packet is also updated to reflect the current position in the processing chain.

There can be several types of workers. A worker could, for instance, be specialized for high-speed FFTs or matrix multiplications, or it could be a general purpose node. There is one Job queue for each type of worker.
Data reorganizers. The platform has data reorganizer nodes. The purpose of this node is to support data partitioning over several workers, which may be needed in heavy signal processing applications. In the Control layer, packets hold data partitioning information, which is received from the application development system. This partitioning information describes the number of sub-batches in each processing step, and how the data is divided. Whenever the controller sees an upcoming change in the data partition it sends the packet to the data reorganizer. The data reorganizer is further described below.

4.2.2 Data reorganization
In a pipelined execution chain, the individual pipeline stages may have to be divided over several nodes. This implies that the original data block has to be split over a number of nodes which process one subblock each. This is not trivial, however, since the subblocks in the different processing stages can belong to different subsets of the original data block. Corner turns, splits and fusions are examples of reorganizations that may have to be carried out between different pipeline stages. Fig. 6 illustrates the principle for how these basic data reorganization operations are realized. It should be noted that, in each specific operation, the Control, Supervision, and Processing mode layers in all packets are identical and remain the same even after the data reorganization.

Fusion. The data reorganizer collects all packets needed for the operation, creates a new packet and puts it in the controller’s ready-queue destroying the old packets. The new packet has references to all the subblocks that form the new block, but as soon as the packet has been processed by a worker it will only have one reference to data - the worker carries out the actual data movement before it processes the data and stores the result.

Split. The data split is basically the opposite of the fusion. The data reorganizer splits one packet into a number of packets with references to their respective subblocks. The subblocks could be created from overlapping volumes in the original data block – the data is then overlapped. Also in this case the workers perform the actual data movements.

Corner turn. In the corner turn operation the slice direction of the initial data block is changed. The new subblocks, i.e., the new slices, consist of data from all the original subblocks, i.e. the old slices. The data reorganizer takes a number of packets and creates new packets, each one having references to parts of all the initial subblocks. Data splits and fusions could be carried out implicitly in the corner turn operation.

5 Signal processing on GEPARD
STAP and SAR are two signal processing applications with diverse requirements; typically, in STAP the extensive performance demand is dimensioning for the signal processor, whereas in SAR the memory requirement is the critical concern. In this section, sample STAP and SAR calculations are mapped onto the GEPARD resource and communication model, and it is shown that the GEPARD model lends itself in a natural way to these demanding types of signal processing.

5.1 Space-time adaptive processing
The sample STAP chain illustrated in Fig. 2 is used in a 32-channel airborne radar system. Due to the computational demands the data block must be split over several worker nodes. We assume that the initial Pulse Compression requires 8 nodes, that the Beamforming and Doppler filtering steps are carried out in sequence by another group of 8 nodes (they are chained in order to avoid unnecessary inter-node communication) and, finally, that the STAP detector step requires as much as 64 parallel nodes.

The system has 32 I/O nodes, one per channel, each one feeding packets into the system. All packets belonging to the same integration interval look the same, apart from the references to their respective subblock of data. The data is processed according to Table 1. The data reorganizations specified in the table are carried out as described in Section 4.2. Note that the chosen number of processing nodes should be considered as an illustration that helps to explain the principle for doing data partitions. The exact type and number of nodes is of course subject to a more detailed analysis.
Table 1 Data reorganizations and functions in a sample STAP chain.

<table>
<thead>
<tr>
<th>Processing step</th>
<th>Data reorganization / Function</th>
<th>Operation (dominant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (in 8 nodes)</td>
<td>Creation of 8 subblocks, sliced by channel, from 32 inchannels</td>
<td>8 fusions 4-&gt;1</td>
</tr>
<tr>
<td></td>
<td>Pulse compression in 8 nodes</td>
<td>FIR</td>
</tr>
<tr>
<td>2,3 (in 8 nodes)</td>
<td>Creation of 8 new subblocks, sliced by range</td>
<td>1 corner turn 8-&gt;8</td>
</tr>
<tr>
<td></td>
<td>Beamforming in 8 nodes</td>
<td>Matrix multiplication</td>
</tr>
<tr>
<td></td>
<td>Doppler filtering (chained)</td>
<td>FFT</td>
</tr>
<tr>
<td>4 (in 64 nodes)</td>
<td>Creation of 64 overlapped subblocks, sliced by Doppler. The subblocks overlap one position on each side.</td>
<td>1 combined corner turn and overlapped split 8-&gt;64</td>
</tr>
<tr>
<td></td>
<td>STAP detection in 64 nodes</td>
<td>QRDs etc.</td>
</tr>
</tbody>
</table>

The signal processing chain described in the table is translated to the (simplified) packet shown in Fig. 7.

Incoming packet from I/O-nodes, after the initial data fusion (1 of 8 packets)

Control layer

Function Param. Chained Status, data reorg./function Partioning, slicing | overlap | #subblocks
FIR ... done/pending **z [0] | 8
MMult ... X pending/pending **y [0] | 8
FFT ... done/pending **y [0] | 8
STAP det. ... function at once) pending/pending **y [1] | 64
in the same node) (partition info from application dev. system)

Supervision layer

Processing mode layer

Data layer

Data block, start | Data block, dim. | Subblock, start | Subblock dimensions
block start | X=128 (pulse) | subblock start | X=128
also an id tag for generated packets | Y=256 (range) | | Y=256
Z=32 (channel) | (info calculated by data reorganizer node.) | (based on info in control and data layers)

Fig. 7. A packet generated in a STAP chain.

In the control layer of the packet, the signal processing chain is reflected as a to-do list. The datablock to be processed is placed in the distributed data buffer and referenced by the data layer. A block belonging to a particular integration interval has a fixed start address in the distributed data buffer throughout the whole signal processing chain. For each step in the to-do list, there is information on how the datablock is divided into subblocks; slicing direction, possible overlap, and number of subblocks. The start addresses to the different subblocks are derived from the start address of the datablock and the partitioning information. Each step also specifies function parameters, such as numbers and lengths of FFTs, i.e., radar mode and data partitioning dependent numbers, and whether the signal processing step is chained or not.

5.2 Synthetic aperture radar

The sample Stripmap SAR processing chain illustrated in Fig. 3 is used in a one-channel sensor system mounted on a relatively slow moving aircraft. The amount of data is big, and the computational load is so high that a large number of processing nodes is necessary. The chosen number of nodes in each step is determined by the size of the data. In Step 1-7 the data slices overlap (FIR filter length – 1), whereas in the succeeding steps they do not.

The signal processing chain is, like in the STAP case, translated to packets. A packet in the SAR processing chain is exemplified in Fig. 8. It can be observed that, in this case, there are several chainable processing steps, which means that data movement can be substantially reduced.

Incoming packet, after the initial overlapped data split (1 of 100 packets)

Control layer

Function Param. Chained Status data reorg./function Partitioning | slicing | overlap | #subblocks
Mult ... done/pending **y [15] | 100
FIR ... X done/pending **y [15] | 100
FFT ... X done/pending **y [15] | 100
2D FIR ... X done/pending **y [15] | 100
FFT ... X done/pending **y [15] | 100
FFT ... pending/pending **x [0] | 10
Resample, FI... pending/pending **y [0] | 10
IFFT ... X done/pending **y [0] | 10
IFFT ... pending/pending **x [0] | 10

Supervision layer

Processing mode layer

Data layer

Data block, start | Data block, dim. | Subblock, start | Subblock dimensions
block start | X=10000 (range) | subblock start | X=10000
also an id tag for generated packets | Y=10000 (azimuth) | | Y=130 (w. overlap)
Z=1 (channel) | (same data block) | Z=1 (same data block)

Fig. 8. A packet in the SAR processing chain.

5.3 Summary

The two described signal processing application examples show that the translation from an algorithm to control layer packets is direct and clear; the signal processing chain is translated to a “to-do list” together with data partitioning information. The data partitioning, which is dependent on the properties of the actual system, can be easily changed if the system is changed. It is easy to describe a system that executes both modes simultaneously, according to the principle in Fig. 5. The job queuing mechanisms in the resource and communication model help in achieving good load balancing.

6 Related work

The concept and need for a “platform” is not new, on the contrary there have been several efforts to provide tools for signal processing on parallel hardware. Some of the concepts were developed as part of the DARPA RASSP program [12], which was an initiative to develop techno-
gies for Rapid Prototyping of Application Specific Signal Processors and focused on improving the process for developing high-performance embedded digital signal processors.

Cost-effective development and efficient execution of applications on parallel, high-performance computer systems is in the focus of the DARPA High Productivity Computing Systems (HPCS) program [13]. The program has for instance created a system assessment framework in which the productivity of a computer system is measured in terms of the ratio between its utility and cost.

Realization of an efficient signal processing framework is facilitated by better understanding of component based systems such as those developed under the DARPA Polymorphic Computing Architectures (PCA) program [14] and the Software Defined Radio (SDR) forum [15]. One goal in PCA is to develop adaptable, “polymorphic”, middleware for embedded computing applications. The SDR forum defines models for software radio and a streaming component architecture (SCA) for how these should be implemented. The Streaming Component Environment, SCE [16], developed independently and in conjunction with both SCA and PCA, is an implementation of a modern component based framework for data-streaming applications. SCE will be a good base for the implementation layers of the GEPARD platform.

7 Conclusion and future work

We have described the need for engineer efficiency, sustainability and flexibility in high-performance signal processing, especially in advanced radar systems. Typical characteristics of these systems are low volume, few customers, complex application development, and a long system lifetime. We observed that some algorithms fit certain devices better than others, and that the best implementation of a system will change over its lifetime. This further emphasized the need for a hardware independent model for application development. To address these challenges we suggest a layered platform, where the developer defines the application using control and transformation functions, which then independently map onto a heterogeneous computational and communication model. To the user of the suggested platform, GEPARD, the underlying hardware architecture is abstracted to a resource and communication model. The model does not hide the hardware completely, since it allows the user to control data mappings and computations.

The application examples showed how two different signal-processing chains were mapped onto this model. The first application had a high computational demand and a small data set, while the second application emphasized data and data redistribution.

We conclude that the platform concept represented by GEPARD is a good candidate for advanced signal processing development, giving both a straightforward application mapping and potential for efficient execution on parallel hardware. The support for multiple hardware targets and a development model that decouples detailed mapping aspects from application leads to a significant increase in engineering efficiency for development as well as for technology insertion. Future studies will focus on a component based API, as well as mapping issues, such as performance efficiency, on parallel hardware.

Acknowledgements

The following persons have made significant contributions to the GEPARD platform concept: Stefan Andersson, Per Ekström, Per Ericsson, and Kurt Lind at Ericsson Microwave Systems AB.

References