# Programming Real-time Autofocus on a Massively Parallel Reconfigurable Architecture using Occam-pi

Zain-ul-Abdin*, Anders Ahlander† and Bertil Svensson*
*Centre for Research on Embedded Systems (CERES)
Halmstad University, Halmstad, Sweden
†Business Area Electronic Defence Systems, Saab AB, Gothenburg, Sweden

***Abstract*—Recently we proposed occam-pi as a high-level language for programming massively parallel reconfigurable architectures. The design of occam-pi incorporates ideas from CSP and pi-calculus to facilitate expressing parallelism and reconfigurability. The feasability of this approach was illustrated by building three occam-pi implementations of DCT executing on an Ambric. However, because DCT is a simple and well-studied algorithm it remained uncertain whether occam-pi would also be effective for programming novel, more complex algorithms.**

**In this paper, we demonstrate the applicability of occam-pi for expressing various degrees of parallelism by implementing a significantly large case-study of focus criterion calculation in an autofocus algorithm on the Ambric architecture. Autofocus is a key component of synthetic aperture radar systems. Two implementations of focus criterion calculation were developed and evaluated on the basis of performance. The comparison of the performance results with a single threaded software implementation of the same algorithm show that the throughput of the two implementations are 11x and 23x higher than the sequential implementation despite a much lower (9x) clock frequency. The two designs are, respectively, 29x and 40x more energy efficient.**

## I. Introduction and Motivation

Embedded signal processing systems are facing the challenges of increased computational demands. Massively parallel reconfigurable architectures, which can be configured to form application-specific hardware, offer high degree of parallelism to meet the application requirements. However, the applicability of reconfigurable processor arrays in embedded and high performance computing is constrained by the need for mastering low-level structural description languages and the lack of support in these languages for expressing dynamic reconfiguration. Clearly, all these challenges need to be taken care of by using an appropriate programming model.

We proposed to use a concurrent programming model that allows the programmer to express computations in a productive manner by matching it to the target hardware using high-level constructs, and the language is further supported by a compiler for providing portability across different hardware resources. Occam [2] is a programming language based on the Communicating Sequential Processes (CSP) [1] concurrent model of computation and was developed by Inmos for their microprocessor chip Transputer. However, CSP can only express a static model of the application, where processes synchronize through communication over fixed channels. In contrast, the pi-calculus [3] allows modeling of dynamic constructions of channels and processes, which enables the dynamic connectivity of networks of processes. Thus, occam-pi [4], combining CSP with pi-calculus, seems to be an interesting approach to programming of run-time reconfigurable systems. A program in occam-pi is a composition of processes in which communication between the processes is managed by unbuffered message passing channels. Occam-pi allows the programmer to structure concurrent programs in such a way that they can not only be more easily understood but also more effectively use the concurrency features of the underlying hardware.

In earlier work, we have demonstrated the feasability of using the occam-pi language to program an emerging massively parallel reconfigurable architecture by implementing a 1D-DCT algorithm [5] [6]. In this paper we demonstrate the applicability of the approach by programming compute-intensive parts of Synthetic Aperture Radar (SAR) systems. SAR image forming has been recognized as an application requiring real-time performance [7]. A case study has been performed by implementing autofocus criterion calculations on the Ambric array of processors. In particular we have used the dynamic parallelism feature of occam-pi in the form of replicated parallel processes.

Ambric is an asynchronous array of so called brics, each composed of two pairs of Compute Unit (CU) and RAM Unit (RU) [8]. The CU consists of two 32-bit Streaming RISC (SR) processors, two 32-bit Streaming RISC processors with DSP extensions (SRD), and a 32-bit reconfigurable channel interconnect for interprocessor and inter CU communications. The RU consists of four banks of RAM along with a dynamic channel interconnect to facilitate communication with these memories. The Am2045 device has a total of 336 processors in 42 brics. Using the proprietary tools the individual objects are programmed in a sequential manner in a subset of the java language, called aJava, or in assembly language [9]. Objects communicate with each other using hardware channels without using any shared memory. Each channel is unidirectional, point-to-point, and has a data path width of a single word. The individual software objects are then linked together using a proprietary language called aStruct.

The rest of the paper is organized as follows: Section II describes the occam-pi language basics and in particular extensions for supporting reconfigurability. Section III provides an overview of the compiler framework. Section IV

978-0-7695-4301-7/11 $26.00 © 2011 IEEE
DOI 10.1109/FCCM.2011.20

describes the SAR system and the significance of the autofocus algorithm. Section V presents the implementation methodology and the two design approaches. Section VI discusses the implementation results of the case study, and the paper is concluded with some remarks and future work in Section VII.

## II. Occam-pi LANGUAGE

Occam-pi [4] language is known for its simplicity, minimal run-time overhead and power to express parallelism. Occam-pi has built in semantics for concurrency and interprocess communication. Occam-pi can be regarded as an extension of classical occam to include the mobility feature of the pi-calculus. The mobility feature is provided by the dynamic, asynchronous communication capability of the pi-calculus. It is this property of occam-pi that is useful when creating a network of processes in which the functionality of processes and their communication network changes at run-time.

### A. Basic Constructs

The hierarchical modules in occam are composed of procedures and functions. The primitive processes provided by occam include assignment, input process (?), output process (!), skip process (SKIP), and stop process (STOP). In addition to these there are also structural processes such as sequential processes (SEQ), parallel processes (PAR), WHILE, IF/ELSE, CASE, and replicated processes [2]. The feature of creating replicated parallel processes helps in managing the amount of parallel resources used in the given hardware architecture.

A process in occam contains both the data and the operations it is required to perform on the data. The data in a process is strictly private and can be observed and modified by the owner process only. The communication between the processes is handled via channels using message passing, which helps in avoiding interference problems. In contrast, in occam-pi the data can be declared as MOBILE, which means that the ownership of the data can be passed between different processes. Compared to the channel definition in classical occam, the channel type definition has been extended to include the direction specifiers, Input (?) and Output (!). Thus a variable of channel type refers to only one end of a channel. The channel types added to occam-pi are considered as first class citizens in the type system. A channel direction specifier is added to the type of a channel definition and not to its name. Based on the direction specification, the compiler performs its usage checking both outside and within the body of the process. Channel direction specifiers are also used when referring to channel variables as parameters of a process call.

Let us now take a look at an occam-pi program that computes raise to the power 8 of integers. The main process invokes three instantiations of a process square, which are executed in parallel, as shown in Figure 1. The inputs to the main process are passed through input channel-end in and the results are retrieved from output channel-end out. The square process contains a sequential block that takes one input value, computes its square and passes the resulting value at its output channel.

```
PROC main(CHAN INT in?,
                  out!)
  CHAN INT a,b:
  PAR
    square(in?, a!)
    square(a?, b!)
    square(b?, out!)
:
```

```
PROC square(CHAN INT c?,
                    d!)
  INT x,y:
  SEQ
    c ? x
    y = x * x
    d ! y
:
```

Fig. 1. An Occam-pi program.

### B. Language Extensions to Support Reconfigurability

In this section, we will describe the semantics of the extensions in the occam-pi language, such as mobile data and channels, dynamic process invocation, and process placement attributes. These extensions are used in the programming model to express the different configurations of hardware resources, whose reconfiguration at run-time can be controlled by using dynamic process invocation and process placement attributes.

*1) Mobile Data and Channels:* The assignment and communication in classical occam follows the copy semantics, i.e., for transferring data from the sender process to the receiver both the sender and the receiver maintain separate copies of the communicated data. The mobility concept of the pi-calculus enables the movement semantics during assignment and communication, which means that the respective data has moved from the source to the target and afterwards the source loses the possession of the data. In case the source and the target reside in the same memory space, then the movement is realized by swapping of pointers, which is secure and no aliasing is introduced.

In order to incorporate mobile semantics into the occam language, the keyword MOBILE has been introduced as a qualifier for data types [10]. The definition of the MOBILE types is consistent with the ordinary types when considered in the context of defining expressions, procedures and functions. However the mobility concept of MOBILE types is applied in assignment and communication. The syntax of mobile data variables and channels of mobile data is given as:

```
MOBILE INT x:
CHAN OF MOBILE INT c:
```

The modeling of mobile channels is independent of the data types and structures of the messages that they carry.

**Mobile Assignment:** Having defined the syntax of mobile types, we are now going to illustrate the movement semantics as applied in the case of the assignment operation. Let us consider the assignment of a variable *'y'* to *'x'*, where *'x'* initially has a value *'v0'* and *'y'* has an initial value of *'v1'*. According to the copy semantics of occam, *'x'* will acquire the value *'v1'* after the assignment has taken place and *'y'* will retain its copy of value *'v1'*. Instead, applying the movement

semantics for mobile assignment, *'x'* will acquire the value *'v1'* after the assignment has taken place but the value of *'y'* will become undefined.

**Mobile Communication:** Mobile communication is introduced in the form of mobile channel types, and the data communicated on mobile channels has to be of the mobile data type. Channel type variables behave similarly to the other mobile variables. Once they are allocated, communicating them means moving the channel-ends around the network. In terms of pi-calculus it has the same effect as if passing the channel-end names as messages.

**MOBILE parameter:** Passing parameters in an ordinary `PROC` call consisting of mobile types does not introduce any new semantics implications and is treated as renaming when mobile variables are passed to either functions or processes.

*2) Dynamic Process Invocation:* For run-time reconfiguration, dynamic invocation of processes is necessary. In `occam-pi` concurrency can be introduced not only by using the classical `PAR` construct but also by dynamic parallel process creation using forking. Forking is used whenever there is a requirement of dynamically invoking a new process that can execute concurrently with the dispatching process. In order to implement dynamic process creation in `occam-pi`, two new keywords, `FORK` and `FORKING`, are introduced [11]. The scope of the forked process is controlled by the `FORKING` block in which it is being invoked.

The parameters that are allowed for a forked process can be of `VAL` type or `MOBILE` type. The parameters of a forked process follow the communication semantics instead of the renaming semantics adopted by parameters of ordinary processes.

- `VAL` data type: whose value is copied to the forked process.
- `MOBILE` data type and channels of `MOBILE` data type: which are moved to the forked process.

*3) Process Placement Attribute:* Having presented the extensions in the `occam-pi` language, we now introduce the placement attribute, which is inspired by the *placed parallel* concept of `occam`. The placement attribute is essential in order to identify the location of the components that will be reconfigured in the reconfiguration process. The qualifier `PLACED` is introduced in the language followed by two integers to identify the location of the hardware resource where the associated process will be mapped. The identifying integers are logical numbers which are translated by the compiler to the physical address of the resource.

## III. Compilation Methodology

In this section we will give a brief overview of a method for compiling `occam-pi` programs to reconfigurable processor arrays. The method is based on implementing a compiler backend for generating native code.

### A. Compiler for Ambric

When developing a compiler for Ambric, we have made use of the frontend of an existing open-source *Translator from Occam to C from Kent* (Tock) [12]. The compiler is divided into front end, which consists of phases up to machine independent optimization, and back end, which includes the remaining phases that are dependent upon the target machine architecture. In this case, we have extended the frontend for supporting `occam-pi` and developed a new backend, targeting Ambric, thus generating native code in the proprietary languages `aJava`, assembly, and `aStruct`.

*1) Frontend:* The frontend of the compiler, which analyzes the `occam-pi` source code, consists of several modules for parsing and for syntax and semantic analysis. We have extended the parser and the lexical analyzer to take into account the additional constructs for introducing mobile data types, dynamic process invocation and process placement attributes. We have also introduced new grammar rules corresponding to these additional constructs to create Abstract Syntax Trees (AST) from tokens generated at the lexical analysis stage. Steps for resolving names and type checking are performed at this stage. The frontend also tests the scope of the forking block and whether the data passed to a forked process is of `MOBILE` data type, thus fulfilling the requirement for communication semantics.

In order to support the channel end definition, we have extended the definition of channel type to include the direction whenever a channel name is found followed by a direction token, i.e., '`?`' for input and '`!`' for output.

*2) Ambric backend:* The Ambric backend is further divided into two main passes. The first pass generates declarations of `aStruct` code including the top-level design, the interface and binding declarations for each of the composite as well as primitive objects corresponding to the different processes specified in the `occam-pi` source code. Thus each process `occam-pi` is translated to a primitive object, which can then be executed on either the SR or SRD processor of Ambric. Before generating the `aStruct` code, the backend traverses the AST to collect a list of all the parameters passed in procedure calls specified for processes to be executed in parallel. This list of parameters, along with the list of names of procedures called, is used to generate the structural interface and binding code for each of the parallel objects.

The next pass makes use of the structured composition of the `occam-pi` constructs, such as `SEQ`, `PAR`, and `CASE`, which allows intermingling processes and declarations and replicating constructs like (`SEQ`, `PAR`, `IF`). In case of the `FORK` construct, the backend generates the background code for managing the loading of the successive configuration from the local storage and communicating it to the concerned processing elements. The translation of `REAL` data type to fixed-point numbers and the subsequent generation of code for fixed-point arithmetics is also performed by the backend.

Floating-point representation is supported in the `occam-pi` language (in the form of `REAL` data types); however, it is not supported by Ambric architecture. Thus, a transformation from floating-point numbers to fixed-point numbers has been developed and added to this pass of the Ambric backend. The supported arithmetic operations are explained as follows:

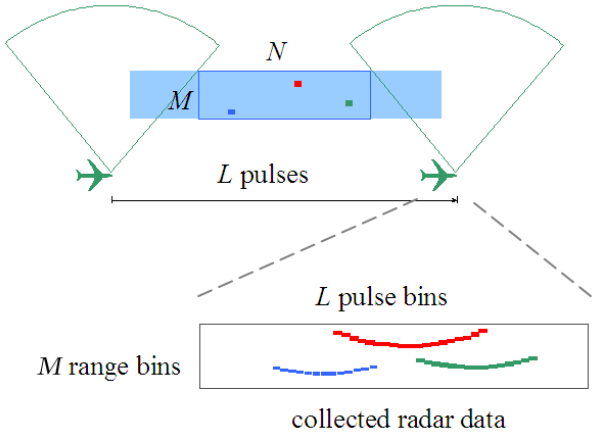


Fig. 2. Simplified illustration of stripmap SAR.

- The assignment operation converts the constant value on the right side of the operator to the selected fixed-point format. If the selected format of the left side variable does not have enough precision for representing the constant value, then attributes such as saturation, overflow and rounding are performed on the constant.
- The add and subtract operation are applied directly without any loss of accuracy during the operation.
- The multiply operation is implemented as an assembly module and each instance of the multiply operator is replaced by a function call to the assembly module. The word length of the product is equal to the sum of word lengths of the two operands. The multiply module consists of shift operations to align the decimal point and throw away the superfluous sign bit.
- The division operation is also implemented as an assembly module. The divider module consists of shift operations to align the decimal part of the result.

## IV. SAR and Autofocus

SAR systems can be used to create high-resolution radar images from low-resolution aperture data. A SAR system produces a map of the ground while the platform is flying past it. The radar transmits a relatively wide beam to the ground, illuminating each resolution cell over a long period of time. The effect of this movement is that the distance between a point on the ground and the antenna varies over the data collection interval. This variation in distance is unique for each point in the area of interest. This is illustrated in Figure 2 where the area to be mapped is represented by *M*x*N* resolution cells. The cells correspond to paths in the collected radar data. The task for the signal processor is to integrate, for each resolution cell in the output image, the responses along the corresponding path. The flight path is assumed linear.

### A. Image forming

A computationally efficient method for creating the image is the Fast Factorized Back-Projection (FFBP) [13]. In FFBP, the whole image initially consists of a large number of small subimages with low angular resolution. These subimages are iteratively merged into larger ones with higher angular resolution, until the final image with full angular resolution is obtained.

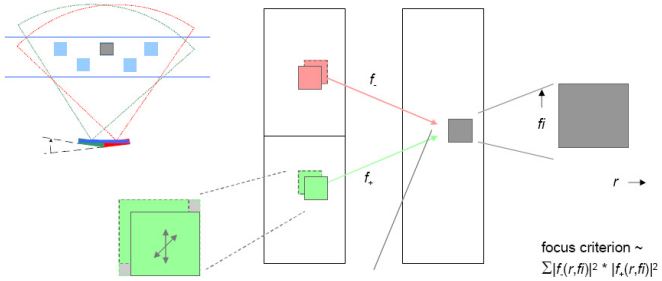


Fig. 3. Illustration of Autofocus and focus criterion.

### B. Autofocus

In reality, the flight path is not perfectly linear. This can, however, be compensated for in the processing. In the FFBP, the compensations typically are based on positioning information from GPS. If this information is insufficient or even missing, autofocus can be used. The autofocus calculations use the image data itself and are done before each subaperture merge. One autofocus method, which assumes a merge base of two, relies on finding the flight path compensation that results in the best possible match between the images of the contributing subapertures in a merge. Several flight path compensations are thus tested before a merge. The image match is checked according to a selected focus criterion as shown in Figure 3. The criterion assumed in this study is maximization of correlation of image data. As the criterion calculations are carried out many times for each merge, it is important that these are done efficiently. Here, the effect of a path error is approximated to a linear shift in the data set. Thus a number of correlations between subimages that are slightly shifted in data are to be carried out. See Section V for more about the calculations. Autofocus in FFBP for SAR is further discussed in [14].

### C. Performance Requirements

The integration time may be several minutes. The computational performance demands are tens or hundreds of GFLOPS. The large data sets themselves represent a challenge but also the complicated memory addressing scheme due to, e.g., changing geometric proportions during the processing. The exact computational requirements are dependent on the chosen detailed algorithms and radar system parameters.

## V. Implementation Methodology

In order to realize the autofocus algorithm on the Ambric platform, the first step in the development process is to determine the dataflow patterns of the algorithm and estimate an approximate amount of resources to be used. The next step is

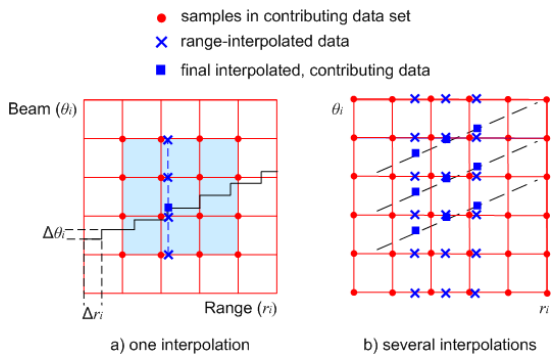


Fig. 4. Usage of cubic interpolation kernel to calculate contributing pixel data points.

to write the `occam-pi` application code in terms of processes based on the dataflow diagram and compose these processes to be executed either in sequence or in parallel to each other. The `occam-pi` application is tested for functional correctness by using the Kent Retargetable occam Compiler (KRoC) [15] run-time system and finally the `occam-pi` application code is compiled by our compiler to the native languages of Ambric. The generated code can then be compiled to generate binaries for the Ambric platform using its proprietary design environment.

We have implemented two versions of the same algorithm, with a different degree of parallelism exploited by the two approaches. We have used a parameterized approach for both of the designs, so the amount of parallelism can be varied by using the construct of replicated `PAR` of `occam-pi` based on parameters such as area of interest ($N$) and number of pixels processed per interpolation kernel ($M$). In addition there are the other parameters of degree of shift and degree of tilt which are to be passed to the algorithm. Both design approaches take as an input, two 6x6 blocks of image pixels from the area of interest of the contributing image. Cubic interpolation based on Neville's algorithm [16] is performed in the range direction followed by the beam direction to estimate the value of the contributing pixels along the tilted lines, and the resulting subimages are to be correlated according to the autofocus criterion. Figure 4(a) illustrates how an interpolated value is computed from samples in the contributing data set, and Figure 4(b) indicates how intermediate results of interpolation can be reused. Each pixel data comprises two 32-bit floating-point numbers corresponding to the real and imaginary components. These floating-point numbers are represented by the `REAL` data type in `occam-pi` and the `REAL` data values are translated to Q16 format fixed-point representation. For fixed-point arithmetics specialized assembly language code is inserted in place of arithmetic operations in the generated code by the compiler backend. Following is a description of the two design approaches:

### A. Design-I

In the first design we have used six range interpolators to calculate the cubic interpolation along the six rows of pixel data in one of the two input pixel blocks, as shown in Figure 5. The input pixel data is fed to the range interpolators through two splitters, which route the pixel data values received from the source distributor block. Since there are no arithmetic computations performed by the source and splitter blocks, when mapped on the Ambric array these blocks are implemented on SR processors.

The range interpolators perform the same operation on different rows of pixel data. During the first iteration each range interpolator takes data values corresponding to four pixels from their inputs and performs the cubic interpolation; then the resultant interpolated pixel data values are passed to the beam interpolation stage. Each range interpolator is implemented on a set of three SRD processors which are connected in a pipeline manner. Since the computed results of the different range interpolators are to be used by multiple beam interpolators, some of the range interpolator blocks have multiple outputs and the resulting interpolated data is copied to these multiple outputs.

The next stage in the dataflow diagram is to perform the cubic interpolation in the beam direction. Three beam interpolators are implemented to perform the beam interpolation on the resulting output of the range interpolation stage. Similar to the range interpolator, each beam interpolator block is also composed of three SRD processors connected in a pipeline. Each beam interpolator takes four inputs from four different range interpolators and it receives its input data values corresponding to four range interpolated pixels on these input ports. The resulting data values of the beam interpolation stage is passed to the three correlators. Each correlator is implemented on one SRD processor and takes pixel data values from each of the pixel data blocks, calculates their correlation and passes the result to the summation block to calculate the final autofocus criterion. The summation block is implemented on a single SRD processor. Three iterations of the range interpolation, beam interpolation, correlation and summation stages are performed in order to compute the autofocus criterion for the entire 6x6 image block.

The parameters of area of interest ($N$) and number of pixels processed per interpolation kernel ($M$) have been used in the construct of replicated `PAR` of `occam-pi` to control the resource usage as shown in Figure 6. Figure 6 also illustrates the generation of static interconnections between different processes based on the given parameters.

### B. Design-II

The second design uses three times the number of range and beam interpolators as compared to the first design, as shown in Figure 7, so that only one iteration of execution of each of the stages will result in computation of the autofocus criterion for the complete 6x6 pixel block. However, due to the limitation of the maximum number of SRD processors available on the Am2045 chip that we are using as a target for realization, we

```
PROC autofocus(VAL INT N, M, VAL REAL xintr, xinti, CHAN REAL dinp0?, dinp1?, CHAN REAL res!)
  [(N/M)*2] CHAN REAL doutp:
  [N*2] CHAN REAL32 soutp:
  [N*4] CHAN REAL routp:
  [N] CHAN REAL boutp:
  [N/2] CHAN REAL coutp:
  PAR
    datadist(N,dinp0?,doutp[0]!,doutp[]!)
    PAR i=0 FOR ((N/M)-1)
      PAR j=0 FOR ((N/M)-1)
        PAR
          split(N,doutp[(i*2)+j]?,soutp[(i*3)+(j*3)]!,soutp[((i*3)+(j*3))+1]!, soutp[((i*3)+(j*3))+2]!)
          rangeintp1(xintr,xinti,soutp[(i*6)+(j*5)]?,routp[(i*12)+(j*11)]!)
          rangeintp2(xintr,xinti,soutp[((i*6)+(j*3))+1]?,routp[((i*12)+(j*8))+1]!,routp[((i*12)+(j*8))+2]!)
          rangeintp3(xintr,xinti,soutp[((i*6)+(j*1))+2]?,routp[((i*12)+(j*3))+3]!,routp[((i*12)+(j*3))+4]!,
                     routp[((i*12)+(j*3))+5]!)
      PAR j=0 FOR (N/M)
        beamintp(xintr,xinti,routp[(i*12)+(j*4)]?,routp[((i*12)+(j*4))+1]!,routp[((i*12)+(j*4))+2]!,
                 boutp[(i*3)+j]!)
    PAR j=0 FOR (N/M)
        corr(boutp[i]?,boutp[i+3]?,coutp[i]!)
    corrsum(coutp[0]?,coutp[1]?,coutp[2]?,res!)
:
```

Fig. 6. Simplified illustration of Autofocus criterion calculation Design-I implementation in `Occam-pi`.

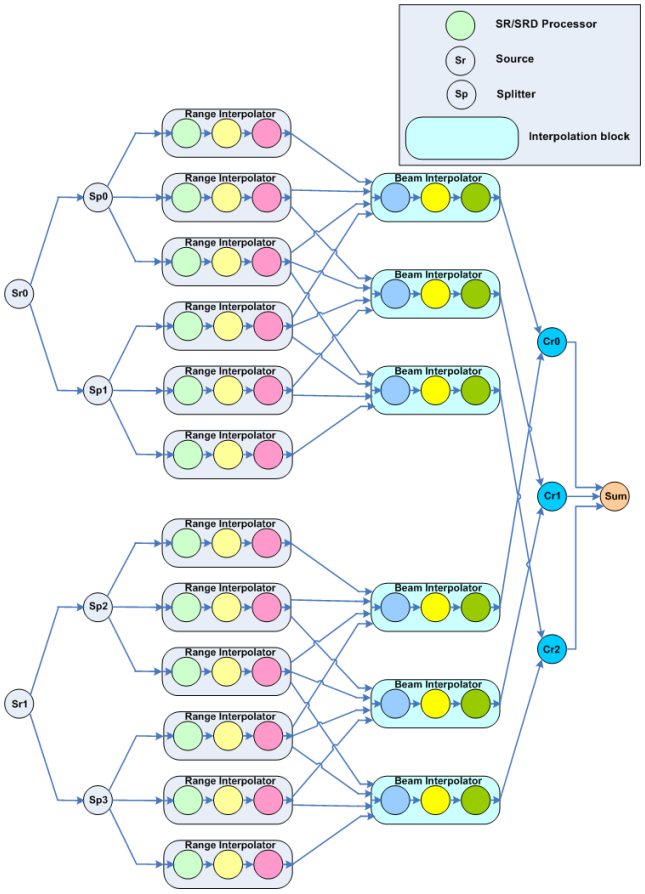


Fig. 5. Dataflow diagram of Design-I.

have to reduce the number of pipelined processors within each of the range and beam interpolation blocks to two.

The increase in the number of range interpolators is also reflected in the increase in the number of splitters, so there are six splitters used to feed the 18 range interpolators performing the cubic interpolation in the range direction of one of the input 6x6 pixel blocks. The six splitters are fed by a single source through two source distributors, because the number of output ports on an SR processor cannot exceed five. Similar to the first design, the source and source distributors are executed on SR processors. The dataflow patterns from the range interpolators to the beam interpolators, from the beam interpolators to the correlators, and further on to the summation stage are similar to the previous design except that we have separate resources for each iteration of interpolation and correlation stages.

## VI. Implementation Results and Discussion

The implementation results are achieved by realizing both the above mentioned designs on Ambric Am2045 architecture and executing them on a GT board containing one Am2045 chip being operated at 300 MHz clock. We have used the performance harness library provided by Nethra Imaging Inc. to obtain cycle accurate performance measurements. We have also obtained results of a sequential version of the same algorithm by executing it as a single threaded application on Intel i7-M620 CPU operating at 2.67 GHz. Table I presents the resources consumed in terms of number of used SRD processors, SR processors, and RU banks, alongwith the percentage of total amount of available resources.

The greater number of SRD processors used as compared to the SR processors is due to the fact that most of the blocks involve complex arithmetics which cannot be performed on SR processors, and also due to the limited instruction memories of the SR processors, which in this case makes them useful only

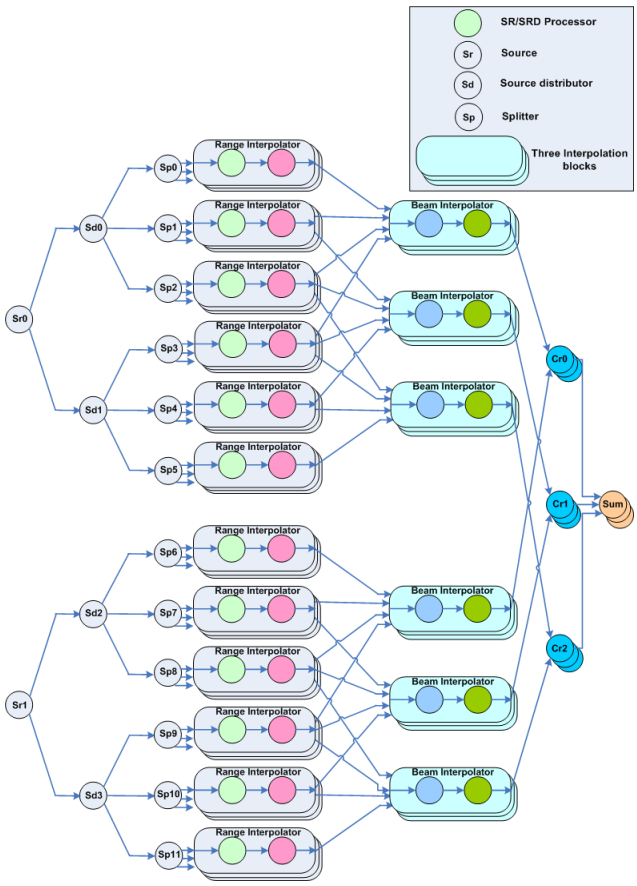


Fig. 7. Dataflow diagram of Design Approach-II.

for data distribution. A signifact number of RU banks are used to store the additional instructions for SRD processors that exceed the internal memory of 256 words. Some of the RU bank memory is also used in implementing FIFO buffers on the channels between different processors to reduce the effect of communication stalls. When going from the first design to the second one, the number of SRD processors should be three times the number of SRD processors used in design-I, but, due to the limited number of available SRD processors, we have to reduce the pipelined processors within each interpolator to two. The use of the performance harness library results in the use of one additional SR and one additional SRD processor, as well as six additional RU banks.

TABLE I
RESOURCES CONSUMED FOR AUTOFOCUS CRITERION CALCULATION.

| | **SRDs** | **SRs** | **RU Banks** |
|---|---|---|---|
| Am2045 Full Capacity | 168 | 168 | 336 |
| Design-I on Ambric | 70(42%) | 24(14%) | 113(34%) |
| Design-II on Ambric | 141(84%) | 28(17%) | 208(62%) |

Table II shows performance and power results: the latency, in cycle count, for producing first correlation output, the throughput, in terms of number of pixels per second on which the given autofocus criterion is computed, the speedup figures for the design realized on Ambric compared to a sequential implementation executed on Intel i7-M620 CPU, and the estimated power consumed by the two parallel and one sequential implementation based on the figures obtained from Am2045 [17] and Intel i7-M620 processor [18] data sheets.

TABLE II
PERFORMANCE AND ESTIMATED POWER RESULTS OF AUTOFOCUS CRITERION CALCULATION.

| **Implementations** | **Latency** (cycles) | **Throughput** (pixels/sec.) | **Speedup** Throughput | **Power** (Watts) |
|---|---|---|---|---|
| Sequential on Intel i7 | - | 21,600 | 1 | 17.5 |
| Design-I on Ambric | 16,497 | 236,386 | 11 | 6.52 |
| Design-II on Ambric | 12,793 | 486,224 | 23 | 9.8 |

The latency results of the design-II depict an improvement in terms of 30% less cycles as compared to design-I. The throughput of the second design is 2.1x times the throughput of the first design and the throughput speedup with respect to the sequential implementation is 11x and 23x respectively for the two designs. With 94 processors which are clocked 9 times slower, a speedup of 11 shows that the design programmed in `occam-pi` is indeed efficient. Ideally the throughput of design-II should have been three times that of design-I, but the use of almost twice the number of processors results in some communication stalls in between the data distribution and interpolation stages, and the effects of the reduced number of pipelined processors within individual interpolators is reflected in the reduction of the throughput. The two designs realized on Ambric consume much less power than the traditional one, and they provide 29x and 40x, respectively, more throughput per watt as compared to the sequential implementation.

We have experienced that the different stages of the cubic interpolation kernel to be executed on the pipelined processors have to be optimized to be able to fit into at most two RU banks of memory for each SRD processor. Otherwise, if it exceeds the size of two RU banks, the placement tool cannot make use of the second SRD processor available in the same compute unit of the Ambric architecture. The optimization is achieved by generating the assembly code for the fixed-point arithmetics used in the cubic interpolation kernel by the compiler backend. Other optimizations implemented in the compiler include scalarization of array variables and exploitation of instruction level parallelism by using the `mac_32_32` instruction in place of successive multiplication and addition instructions.

## VII. CONCLUSIONS AND FUTURE WORK

We have presented an approach of using a CSP based language for programming the emerging class of processor array architectures. We have also described the mobility features of the `occam-pi` language and the extensions in language constructs that are used to express run-time reconfigurablity. The ideas are demonstrated by a working compiler, which compiles `occam-pi` programs to native code for an array of processors, Ambric. An application study of focus criterion calculation is performed and the results corresponding to two different mappings of the said algorithm are presented.

In terms of performance, the two implementations on Ambric outperform the CPU implementation by factors of 11-23, while operating at a clock frequency of 300 MHz as compared to 2.67 GHz. This shows that the designs programmed in `occam-pi` are indeed efficient. The use of a much lower clock frequency together with the switching off of unused cores in the Ambric architecture provides the side advantage of a significant reduction in energy consumption of the two parallel implementations, which is an important factor to consider for embedded systems.

From the programmability point of view, it is observed that the explicit concurrency of `occam-pi` with the ability to describe computations that reside in different memory spaces, together with the dynamic process invocation mechanism, makes it suitable for mapping applications to massively parallel reconfigurable architectures. The `occam-pi` language is based on well-defined semantics, and its simplicity, static compilation properties, minimal run-time overhead and power to express parallelism help in the task of parallelization. The existence of the `REAL` data type in `occam-pi` and the introduced conversion of the floating-point arithmetics to fixed-point by the compiler backend also reduces the overall burden on the programmer, compared to manually implementing the fixed-point arithmetics. Furthermore, the support for expressing dynamic parallelism in the form of replicated `PAR` construct enables the compiler to perform resource-aware compilation in accordance with the application requirements. The reconfigurability support allows effective reuse of resources, and the placement attributes allow processes to be co-located, which gives a potential to avoid unnecessarily expensive communication. In addition to the language features, our proposed methodology of testing the functionality of the application in `occam-pi` language before compiling the generated native code using the Ambric design environment reduces the turnaround time for implementing various design alternatives quite significantly. We believe that raising the abstraction level for the programmer, while not compromising the performance benefits, will be the key to success for the adoption of the emerging reconfigurable architectures in the mainstream computing industry.

Future work will focus on developing applications in `occam-pi` language to exploit the run-time reconfiguration capability of the target hardware and on extending the compiler framework to target other reconfigurable architectures such as PACT XPP, picoarray, and Element CXI.

## ACKNOWLEDGMENT

The authors would like to thank Nethra Imaging Inc. for giving access to their software development suite and hardware board. This research is part of the CERES research program funded by the Knowledge Foundation and the ELLIIT strategic research initiative funded by the Swedish government. The support from the European Union Artemis project SMECY is also acknowledged.

## REFERENCES

[1] Hoare, C.A.R.: *Communicating Sequential Processes.* Prentice-Hall. (1985).

[2] Occam® 2.1 Reference Manual, SGS-Thomson Microelectronics Limited. (1995)

[3] Milner, R., Parrow, J., and Walker, D.: *A Calculus of Mobile Processes, Part I.* Information and Computation, 100, (1989).

[4] Welch, P.H., and Barnes, F.R.M.: *Communicating mobile processes: Introducing occam-pi.* Lecture Notes in Computer Science, Springer Verlag. 175-210 (2005).

[5] Zain-ul-Abdin, and Svensson, B.: Using a CSP based programming model for reconfigurable processor arrays. *Proceedings of International Conference on Reconfigurable Computing and FPGAs, ReConFig'08.* (2008)

[6] Zain-ul-Abdin, and Svensson, B.: Specifying run-time reconfiguration in processor arrays using high-level Language. *Presented at the 4th HiPEAC Workshop on Reconfigurable Computing.* (2010) [Available online at; http://hh.diva-portal.org/]

[7] Ahlander, A., Hellsten, H., Lind, K., Lindgren, J., and Svensson, B.: Architectural challenges in memory-intensive, real-time image forming. *Proceedings of International Conference on Parallel Processing, ICPP'07.* (2007)

[8] Jones, A. M., and Butts, M.: TeraOPS hardware: A new massively-parallel MIMD computing fabric IC. *In Proceedings of IEEE Hot Chips Symposium.* (2006)

[9] Butts, M., Jones, A. M., and Wasson, P.: A structural object programming model, architecture, chip and tools for reconfigurable computing. *Proceedings of 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines.* (2007).

[10] Welch, P.H., and Barnes, F.R.M.: *Prioritised dynamic communicating processes: Part I.* Communicating Process Architectures, IOS Press. 321-352 (2002).

[11] Welch, P.H., and Barnes, F.R.M.: *Prioritised dynamic communicating processes: Part II.* Communicating Process Architectures, IOS Press. 353-370 (2002).

[12] Tock: Translator from Occam to C by Kent. "https://www.cs.kent.ac.uk/research/groups/sys/wiki/Tock", [Online; accessed 8th July, 2008]

[13] Ulander, L.M.H., Hellsten, H., and Stenstrom, G.: *Synthetic-aperture radar processing using fast factorized back-projection.* IEEE Transactions on Aerospace and Electronic Systems, Vol. 39(3). 760-776 (2003).

[14] Hellsten H., Dammert P., and Ahlander A.: Autofocus in Fast Factorized Backprojection for processing of SAR images when geometry parameters are unknown. *Proceedings of 2010 IEEE International Radar Conference.* (2010).

[15] KRoC: Kent Retargetable occam Compiler. "http://www.cs.kent.ac.uk/projects/ofa/kroc/", [Online; accessed 10th July, 2010]

[16] Neville, E.H.: *Iterative interpolation.* Journal of Indian Math Society, Vol. 20. 87-120 (1934).

[17] Am2045 Data Book, Ambric Inc. (2007)

[18] Intel® Core™ i7-600, i5-500, i5-400, and i3-300 Mobile Processor Series Datasheet, Intel Corporation. (2010)