An energy-efficient custom architecture for the SKA1-Low central signal processor

Leandro Fiorin† Erik Vermij† Jan Van Lunteren‡
Rik Jongerius† Christoph Hagleitner‡
†IBM Research, the Netherlands
‡IBM Research – Zurich, Switzerland
{leandro.fiorin,erik.vermij,r.jongerius}@nl.ibm.com {jvl,hle}@zurich.ibm.com

ABSTRACT

The Square Kilometre Array (SKA) will be the biggest radio telescope ever built, with unprecedented sensitivity, angular resolution, and survey speed. This paper explores the design of a custom architecture for the central signal processor (CSP) of the SKA1-Low, the SKA's aperture-array instrument consisting of 131,072 antennas. The SKA1-Low's antennas receive signals between 50 and 350 MHz. After digitization and preliminary processing, samples are moved to the CSP for further processing. In this work, we describe the challenges in building the CSP, and present a first quantitative study for the implementation of a custom hardware architecture for processing the main CSP algorithms. By taking advantage of emerging 3D-stacked-memory devices and by exploring the design space for a 14-nm implementation, we estimate a power consumption of 14.4 W for processing all channels of a sub-band and an energy efficiency at application level of up to 208 GFLOPS/W for our architecture.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors); C.0 [**General**]: Modeling of computer architecture

General Terms

Design

Keywords

HPC, Custom Computing Architectures, Accelerators, SKA.

This work is conducted in the context of the joint ASTRON and IBM DOME project and is funded by the Netherlands Organisation for Scientific Research (NWO), the Dutch Ministry of EL&I, and the Province of Drenthe.

CF'15, May 18 - 21, 2015, Ischia, Italy

http://dx.doi.org/10.1145/2742854.2742855.

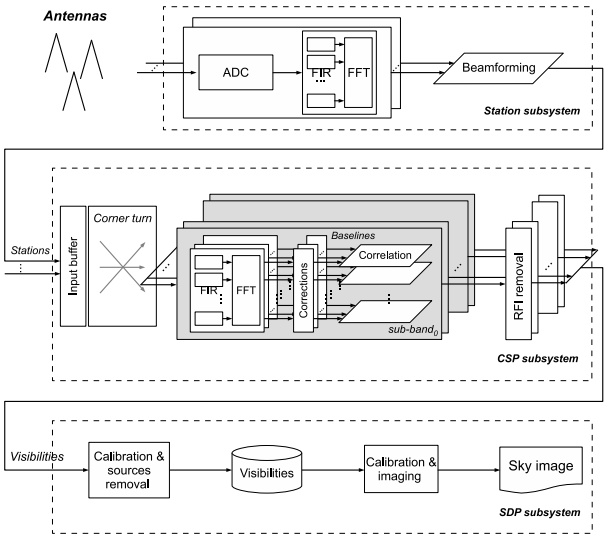

Figure 1: Overview of the processing steps for the SKA1-Low, using the continuum imaging science case [26].

1. INTRODUCTION

The Square Kilometre Array (SKA) will be the biggest radio telescope in the world, with unprecedented sensitivity, angular resolution, and survey speed. Collectively, the antennas composing the SKA are expected to gather 14 exabytes and store one petabyte of data every day, requiring exa operations per second for the processing [19]. In this work, we focus on the implementation of the *central signal processor* (CSP) of the SKA1-Low, the SKA's aperture array instrument that will be built in the first phase of the deployment of the SKA and consisting of 512 stations, each containing 256 dual-polarized antennas [19]. We refer in particular to a two-stage signal channelization implementation in which the signal-processing load is distributed between the stations and the CSP (see Figure 1 for details) [19]. This configuration is similar to the one implemented in LOFAR, the low-frequency aperture array built in the Netherlands [13].

Figure 1 shows an overview of the processing steps for the SKA1-Low, in the case of the *continuum imaging science case* [19]. The processing flow can be divided in three parts: the *station processing*, the *central signal processor*, and the *science data processor* (SDP) [1].

[1]We refer the interested reader to [18,26] for further details.

Table 1: CSP application parameters.

CSP requirements		
Parameter	Description	Value
N_{stat}	# transmitting stations	512
N_{band}	# sub-bands (coarse-grained channels)	512
N_{beam}	# beams per station	1
N_{pol}	# polarizations	2
f_{stat}	Station samples frequency (Mega samples/s)	360
N_{ch}	# channels for frequency sub-band	512
N_{taps}	# FIR filter taps	8
N_{dump}	# samples integrated at the correlation step	1200
Ops_{fir}	# operations per sec. - FIR (Tera ops/s)	11.06
Ops_{FFT}	# operations per sec. - FFT (Tera ops/s)	16.59
Ops_{correc}	# operations per sec. - corrections (Tera ops/s)	5.16
Ops_{corr}	# operations per sec. - correlation (Peta ops/s)	1.51

This paper focuses on the study and evaluation of an energy-efficient custom architecture for the CSP. Current proposals for its implementation rely on the use of high-performance homogeneous supercomputers [18] or on GPU-based accelerated architectures [5], and hardware solutions based on FPGAs [7]. We present a first quantitative study for the implementation of a custom hardware architecture for processing the CSP algorithms. By taking advantage of emerging 3D-stacked memory devices and by exploring the design space for a 14-nm implementation, we evaluate that our architecture consumes 14.4 W for processing all channels of a sub-band, achieving an energy efficiency at application level of up to 208 GFLOPS/W.

The remainder of this paper is organized as follows: Section 2 describes the CSP and presents an analysis of its main algorithms and of their requirements. Section 3 introduces the proposed architecture and discusses the advantages of the new data organization. In Section 4, the proposed architecture is evaluated. Related work is presented in Section 5, and Section 6 concludes the paper.

2. CENTRAL SIGNAL PROCESSOR

Figure 1 shows an overview of the main processing steps of the SKA1-Low CSP, taking as example the configuration implemented in LOFAR, the low-frequency aperture array built in the Netherlands [13]. At the stations, dual-polarized signals from each antenna are digitized and sent to poly-phase channelizers, which creates a number of frequency sub-bands. Signals of every band are delayed appropriately by multiplying them with a complex phase shift, and added to the corresponding band from the other antennas to create a so-called beam.

At their arrival at the CSP, station data are copied in a memory buffer and a coarse-grained inter-sample time delay is introduced in the beam data to correct for the different relative stations distance and align the beams in time and phase. Station data arrive separated over N_{band} coarse-grained channels or sub-bands, thanks to the first channelization phase performed during station processing.

In the second step, in the stage called *corner turn*, data are redistributed. Data arrive at the CSP *per station*, and the corner turn redistributes them to have data from all stations for a single frequency sub-band in the processing node. This is done because data need to be processed *per sub-band*. After the corner turn, a new channelization step is performed in which signals are split into even narrower frequency channels. After the channelization step, data are multiplied with complex coefficients to correct for artifacts introduced by the filter banks in the station processing.

At this point, samples from every station in the same frequency channel are correlated to detect statistic coherence in the received signals - otherwise too weak to be distinguished from the noise - and integrated over a small period of time to reduce the output size. The result data of this step, called *visibilities*, are processed by removing any sample disrupted by radio-frequency interference.

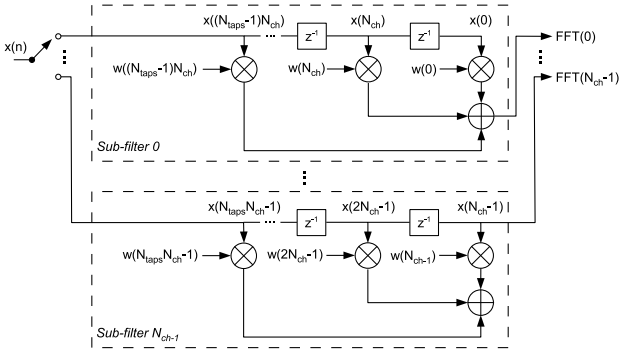

Figure 2: Block diagram of the CSP poly-phase filter.

In this work, we focus on the most computing-intensive algorithms of the CSP, i.e., the *channelization*, the *corrections*, and the *correlation* (grey area in Figure 1). In the remainder of the paper, we will refer to the parameters and requirements summarized in Table 1[2].

2.1 Algorithms profile

2.1.1 Channelization

In the channelization, every frequency sub-band is split into N_{ch} narrower frequency channels by trading time resolution for frequency resolution. This is performed through a poly-phase filter consisting of N_{ch} N_{taps}-taps finite impulse response sub-filters (FIR) and an N_{ch}-point complex-to-complex Fourier transform (FFT) taking as input the outputs of the sub-filters. As a reference, Figure 2 shows a block diagram of the CSP poly-phase filter. Both FIR and FFT algorithms operate on complex samples. Every channel-tap combination in the sub-filters has a different real weight, but all stations and polarizations share the same weight. Given the parameters summarized in Table 1 and considering a Radix-2 Cooley-Tukey implementation for the FFT, the aggregate computing requirements of this step for processing all N_{band} sub-bands are 27.6 tera operations per second, with an input bandwidth of approximately 5.9 terabits per second [13].

2.1.2 Phase shift and bandpass correction

Both phase shift and the bandpass corrections are implemented by multiplying each sample with complex (phase shift) or real (bandpass) coefficients [18] after the channelization. The computing requirements of these two steps are approximately 5.2 tera operations per second. In the case of the phase shift correction, additional computing power may be needed calculating the coefficients, which changes over different time windows. For the sake of simplicity, in the remainder of the paper, we will not discuss the execution of the correction steps, as their computing requirements are negligible with respect to the requirements of the channelization algorithms, and in particular of the correlation algorithm.

[2]At the time of the submission, the specification of the SKA1-Low are not yet fully finalized. Values reported in this work represent an educated guess of the worst-case scenario requirements [19].

2.1.3 Correlation

In this step, simultaneous samples of each pair of stations (baselines) are correlated by multiplying a sample with the complex conjugate of the sample of all other stations, and integrated over a period of time (the *dump time*). For each baseline, four visibilities are calculated, i.e., correlating each one of the two baseline's polarizations with itself and the other. This is performed for each one of the $N_{band} \times N_{ch}$ frequency channel in which the signal spectrum has been divided by the channelization phases. The correlation shows a computing requirement of 1.5 peta operations per second (approximately 2 orders of magnitude higher than the one shown by the channelization step), making it the most expensive computing step of the CSP.

An additional difficulty in implementing the correlation step is that on conventional computer architectures, the channelization and corrections steps in general produce data in an order unsuitable for the correlation step [24], which may influence the performance (time-wise) and power efficiency of the entire CSP.

3. PROPOSED MICRO-ARCHITECTURE

As presented in Section 2, the CSP is composed of several algorithms with small repetitive kernels that process data with relatively little locality and with streaming behavior. Although the computing requirements are enormous, the application is embarrassingly parallel. Given the characteristics of the application, we propose a customized Single-Instruction Multiple-Data (SIMD) micro-architecture suitable for satisfying the energy efficiency requirements of the CSP, while still maintaining a good level of flexibility.

Figure 3 presents an overview of the proposed custom architecture. Several 3D-stacked memory devices, such as Micron's Hybrid Memory Cube (HMC) [11, 17], are connected to the accelerator die through high-speed Serializer/Deserializer (SerDes) I/O links. The accelerator consist of several homogeneous cores, connected by an on-chip interconnection network. Figure 3 shows the CSP custom core micro-architecture template. Each core is composed of m functional units (FUs). Each functional unit is composed of its own private register file (RF) and of an execution unit (represented in the figure by the Complex Fused Multiply-Add - CFMA). Data are transferred from/to the memory exploiting wide path accesses into/from the register files of the functional units. An additional register file (the Shared Scalar Register File - SSRF) is used for storing scalar values shared by all m functional units. As will be explained in Section 3.3, all CSP algorithms can be expressed in form of a multiplication of a vector with a scalar value. The SSRF serves as a buffer into which values are written as vector to exploit the wide memory access of the memory, and read as scalar. The core is controlled by a state machine (micro-controller) that manages a predetermined sequence of storage, communication, and computation steps [23]. All functional units work at lock step under the management of the micro-controller, performing the same operation on different data. As also explained in Section 3.3, each core operates interdependently on its own set of data.

3.1 Functional unit

The analysis of the algorithms in Section 2 shows that the CSP operations are dominated by the correlation. This function can be implemented, by including also the addition due to the integration over the dump time, as a Fused Multiply Add (FMA) between two incoming complex samples and the temporary result of the integration.

The complex FMA (CFMA) can also serve as functional unit to implement the other algorithms of the CSP. The FIR operation contains N_{taps} real to complex multiplications and $N_{taps} - 1$ complex additions, corresponding to $4N_{taps} - 2$ real floating-point operations. By using a CFMA, the FIR can be expressed as N_{taps} CFMAs, equivalent to $8N_{taps}$ real floating-point operations, leading to a theoretical utilization of the functional unit of approximately 47% for N_{taps} equal to 8.

The FFT can be similarly expressed in terms of CFMAs [14, 16]. For instance in the case of a Radix-2 Cooley-Tukey implementation, a butterfly between complex operands requires 10 floating-point operations, and it can be implemented as two CFMAs. The theoretical utilization of the functional units in the FFT is therefore equal to 62.5%.

In the case of a phase shift and bandpass correction (if separated from the channelization), the utilization is equivalent to 75% and 25%, respectively, because of the simple multiplication to complex coefficients in the first case and to real ones in the second case.

Even if the theoretical utilization is sub-optimal for FIR, FFT, and corrections, the computational weight of the correlation makes the overall system-level utilization of the functional units close to 100%. Therefore, possible advantages of using functional units customized for the specific algorithms are not justified by the negligible improvement they could bring at system level.

Each CFMA is provided with a private register file, which allows a full throughput in the utilization of the CFMA to be achieved, i.e., a number of interleaved threads equal to the number of pipeline stages of the execution unit is supported. Given the high parallelism of the algorithms, it is possible to schedule threads without being influenced by data dependencies [8]. To achieve a least-energy solution, we assume two different clocks for the CFMA and the register file, trading parallelism versus pipelining in the access to the register file [8, 16].

3.2 Micro-controller

The micro-controller controls the entire processing path, i.e., the reading of operand values from memory, transferring these to the functional units, triggering the subsequent processing of these operands, and writing the results back to memory [16, 23]. The micro-controller uses a new type of low-level programming model that allows us to program the memory accesses, data transfer and computation in a more tightly coupled fashion. This allows us to remove a large part of the logic overhead (e.g., buffering) that typically resides between the memory circuits in which the data resides and the functional units that process the data, and which is inherently part of conventional general-purpose processing architectures. By also integrating selected memory controller functions, in particular address mapping and access scheduling, the micro-controller can further improve the overall performance by optimizing the bandwidth utilization and power efficiency of the memory system.

The programming model of the micro-controller is based on two main concepts. The first one is a programmable state machine, called B-FSM [22], which makes it possible to support efficient multi-way branch capabilities in the micro-controller. This allows us to evaluate many conditions in parallel, including loop conditions as part of address generation, and access scheduling related conditions.

The second enabling technology is a programmable mapping scheme that can be used to adapt the mapping of addresses over the memory banks, with the objective to im-

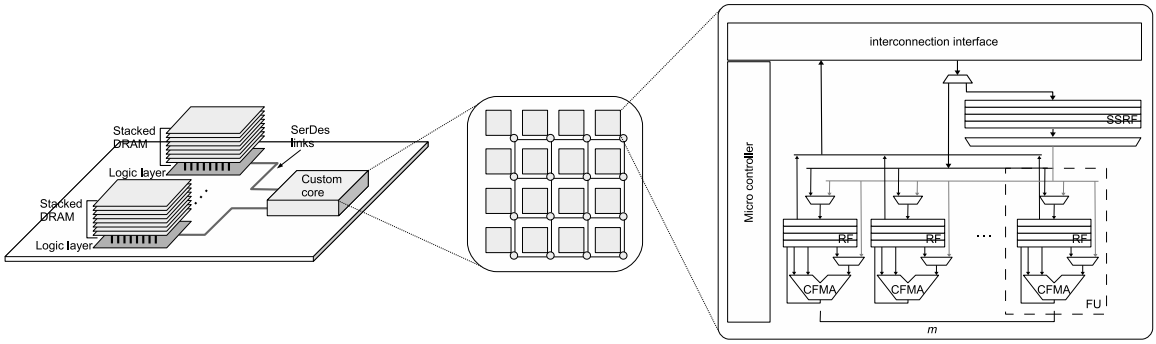

Figure 3: Overview of the proposed accelerator architecture. From left to right: socket, chip, core.

prove memory bandwidth utilization based on improved access interleaving to reduce bank contention [21].

3.3 Proposed data organization and algorithms execution

To exploit the high available parallelism and the potential of the proposed SIMD architecture, we organize data to better fit the wide line of FUs. As observed, the correlation is the computationally most intensive algorithm, and data should be delivered to the functional units in a way that maximizes their utilization, while minimizing data movement. In this work, we refer to a correlation implementation based on the *input-buffered* architecture [6]. All input samples of the antennas processed by a specific core are first copied to a memory that holds enough samples for the integration over the dump time. The samples of one pair of antennas (baseline) are multiplied and summed to the temporary result until all samples in the dump time have been processed. Then, the result is delivered and a new baseline is processed, continuing in this way until all pairs of antennas have been correlated.

We can assume data to be delivered at the input buffer of the CSP in the form:

$$data\_in[stations][time][polarizations]$$

in which the *time* dimension includes the results of the first channelization, i.e., the "distribution" of the time samples over multiple sub-bands. Similarly to the LOFAR implementation [18], we rely on a corner turn and perform a first parallelization of the computation over the sub-bands (see Figure 1). After the corner turn, we assume data organization at the input of each node processing a sub-band in the form:

$$data\_in[time][stations][polarizations]$$

3.3.1 FIR and FFT execution

The FIR and the FFT are performed by accessing data in memory per column instead of per row, in a way similar to what is described in [9] for the CELL processor. Each functional unit is therefore responsible for processing samples of one of the polarizations of a single station, both for the FIR and the FFT. Figure 4 shows a block diagram of the execution of the FFT over our micro-architecture in the case of an in-place Radix-2 Cooley-Tukey FFT implementation, optimized for CFMAs [14]. Twiddle factors are pre-calculated and organized in memory to optimize their utilization in the successive stages of the FFT calculation. We exploit the parallelism available in the core to perform the algorithms over multiple stations/polarizations at the same time, and use wide accesses to retrieve sample data from the memory.

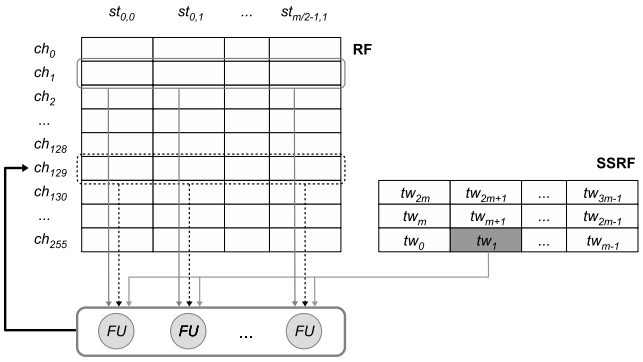

Figure 4: Proposed input data organization for the FIR and the FFT algorithm. The picture also sketches how data are processed by the proposed micro-architecture when implementing the FFT.

In the figure, $st_{i,j}$ is the $i-th$ station with polarization j; ch_i is the $i-th$ channel generated during the FIR step, and tw_i is the $i-th$ twiddle factor used for calculating the FFT butterfly.

Every functional unit works on its own column of data. As the core's functional units work in parallel and at lock step, all the data in the retrieved memory block/line is processed at the same time, therefore efficiently exploiting the wide line of functional units. Moreover, in the case of the FFT, by accessing data in this way, there is no need for special hardware for shuffling data or for performing any transpose operation, and input data locality can be exploited in the core by implementing a decomposition in smaller FFTs [9] or/and by using higher radix orders.

In the execution of the FIR and the FFT, each basic step implies that each FU multiplies its input sample data with a value common to all FUs (the tap weights in the case of the FIR, and the twiddle factors for the FFT). The SSRF is used to store these values and provide them to the FUs.

3.3.2 Correlation execution

By distinguishing between different channelization batches over the time, data after the channelization are provided to the correlation in the form:

$$data\_in[time][channels][stations][polarization]$$

Figure 5 shows the same diagram as in Figure 4, but in the case of the correlation. For each time sample t_i, a line of station data is copied on the SSRF, and, one station data at the time, multiplied with the line of station data retrieved for the line of FUs. Temporary results of the integrations

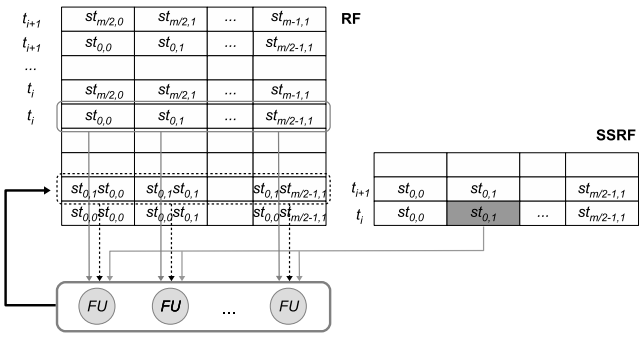

Figure 5: Proposed input data organization for the correlation. The picture also sketches how data are processed by the proposed micro-architecture when implementing the correlation (one channel).

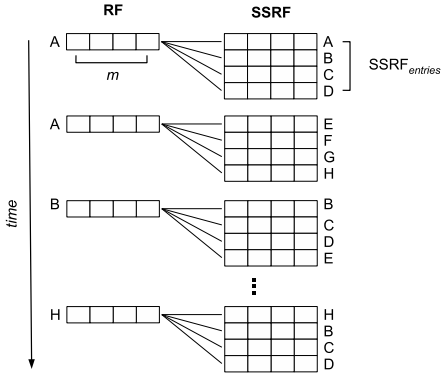

Figure 6: Overview of an optimized input data scheduling on the proposed architecture for the correlation algorithm.

are stored in the RFs until all time samples within the dump time for the calculated correlations have been processed and added to the temporary results. This implementation requires the RF to have enough local storage to store the intermediate results of the integrations, i.e., given a line of m FUs and a number of SSRF entries equal to $SSRF_{entries}$, each RF should be big enough to store $m \times SSRF_{entries}$ temporary integration results.

The use of the RF and the SSRF, together with optimized data scheduling, allows us to reduce the number of accesses to the main memory of the accelerator, and therefore the requested data traffic from it. Figure 6 shows a simplified example of how data can be scheduled on the proposed architecture when implementing the correlation. In the example, we consider 8 lines of m station data (A to H), i.e., $8 \times m$ stations to correlate (single polarization, in the example). The figure shows the sequence of data loads on the RFs and on the SSRF. For sake of clarity, Figure 6 shows only the stations involved in the correlation, and not each station's time samples. From the figure, it is straightforward to derive that the number of read accesses to the memory and the input bandwidth is proportional, in the case of the SSRF, to:

$$RDs(mem, SSRF) = \frac{stations}{2m}\left(\frac{stations}{m} + 1\right) \qquad (1)$$

In the case of the RFs, the number of read accesses is proportional to:

$$RDs(mem, RF) = \frac{stations}{2m \times SSRF_{entries}}\left(\frac{stations}{m} + 1\right) \qquad (2)$$

Note that by reorganizing the data in the way presented before the channelization, there is not need to transpose data before the execution of the correlation, as is currently done in conventional computing architecture [18].

4. EVALUATION RESULTS

In this section, we present the results of the design-space exploration we performed on our architecture in order to find the most energy-efficient configuration.

4.1 Experimental setup

We modeled the CFMA by extracting implementation data for a single-precision floating-point FMA from [8] and scaling them to 14-nm technology, likely to be commercially available in 2018, when the construction of SKA1 will begin. We considered a non-optimized CFMA composed of four FMAs, with twice the number of pipeline stages of the FMA. As our goal is to reduce the total energy, we focused on a low V_{dd} and high V_{th} implementation, i.e., synthesized with a low-power technology library [8]. For modeling the register files and the SSRF, we relied on CACTI-D [20], scaling the results obtained from the tool from 32 nm to 14 nm. On-chip interconnections were modeled on implementation results extracted from [4]. Power and area numbers for the micro-controller are preliminary synthesis results for a 14-nm implementation. Numbers for I/O chip-to-chip data transfers over the backplane were estimated by scaling to 14 nm power and area results obtained for an electrical transceiver implemented in 65 nm and capable of up to 15 Gb/s data rate [2]. For the technology scaling, we referred to the trends reported in [1] and [3] in the case of Low Operating Power technology. For logic, we applied a scaling factor of 0.53x per technology node for the area, and a scaling factor of 0.8x per technology node for dynamic and leakage power. For SRAM devices, from 32 nm to 14 nm, we considered an overall area scaling factor of 0.34x and a power scaling factor of 0.56x. We considered a 30% increase per technology node in the maximum frequency of the design.

We model the 3D-stacked memory modules after a Micron study showing that the energy per bit for access in the first-generation HMC is measured at 10.48 pJ, of which 3.7 pJ/b in the DRAM layers and 6.78 pJ/b in the logic-layer, which includes HMC memory controllers, short-range high-speed Serializer/Deserializer (SerDes) link controllers, a crossbar switch, and additional logic [11,17]. As the logic layer is implemented in 90 nm, we scaled the logic layer access energy to 2.1 pJ/b to take into account of the most recent technology likely to be used in the second-generation HMC [12]. The energy access values obtained are in line with numbers reported in other studies [10]. We assume an additional 1.33 W of consumption in the logic layer for components other than the SerDes links, and a background DRAM array power equal to 10% of the DRAM power consumption at maximum bandwidth [17]. An additional 1 pJ/b has been considered for crossing the short-range links between the core accelerator chip and the HMC devices [11]. Each HMC device has a maximum of four full-duplex 16-lane links with an aggregate link bandwidth of 480 GB/s [12].

4.2 Design-space exploration

As the requirements of the correlation dominate the application, we focus on them the system-level design. Performance and costs of executing the channelization algorithms on the architecture will be evaluated afterwards.

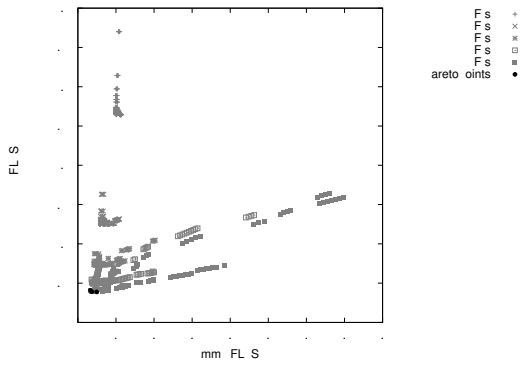

Figure 7: Results of the parameters design-space exploration for the proposed architecture.

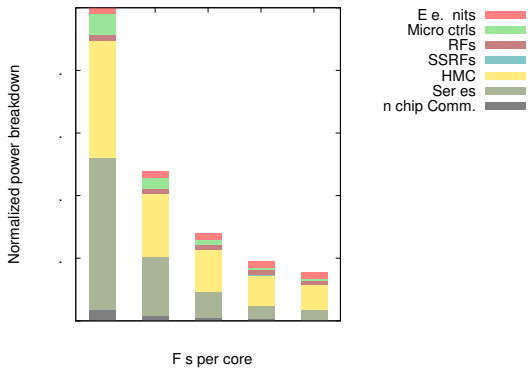

Figure 8: Power breakdown for different core configurations.

4.2.1 Correlation

We explored different configurations of the architectural template presented in Section 3. While focusing on performing our operations on 64-bit-wide complex operands (32 bits for both the real and the imaginary part), we varied the number m of FUs per core, the number of SSRF entries, and the corresponding size of the RFs. As every FU practically operates on its own set of data independently of the other FUs, we considered each core's RF either as big memories with wide accesses or as m smaller memories with narrow accesses. We explored different memory implementation technologies considering for RFs either SRAM or embedded DRAM (eDRAM).

Each CFMA, implemented as a 6 pipeline stage unit, operates at 433 MHz. Each core is therefore able to deliver a maximum throughput of $m \times 433$ MHz. Evaluations were performed by considering that our accelerator can process all N_{ch} channels of one sub-band. In those configurations in which a single chip was not able to satisfy the design constraints (e.g., the maximum allowed core-memory bandwidth), additional area and power overhead were considered for a multi-chip implementation.

To allow a streaming behavior of the application and overlap execution with memory transfers, we doubled the size of all memory components. Given the requirements in Table 1, the minimum amount of memory needed for a streaming input-buffered implementation of the correlation is equal to 9.375 GB. We considered therefore a minimum of three 4-layered 4-GB HMC devices, for a total capacity of 12 GB per accelerator.

Figure 7 shows the results of the design-space exploration performed on our custom architecture, by considering the inverse of the number of GFLOPS per Watt and the inverse of the number of GFLOPS per unit of area (mm^2) as design objectives. The power evaluation includes all the components modeled. Area values refer only to the accelerator die. Table 2 summarizes the main characteristics of the points in the Pareto curve with minimum area and minimum power, α and β, respectively. For both points, the number m of functional units per core is equal to 64 and the number of cores in the chip is 14. The throughput from the HMC for the α and the β point is equal to 68 GB/s and 81 GB/s, respectively. The different throughputs are due to the difference in the $SSRF_{entries}$ parameters and to the influence that $SSRF_{entries}$ has on the number of memory accesses. The throughput to the HMC is in both cases equal to 18 GB/s. In the table, $SSRF_{entries}$ is the number of SSRF entries, and RFs_{size} is the total size of the RFs in the core. RF_{config} specifies the memory implementation technology used for evaluating the memory component and whether it employs narrow (N) or wide (W) accesses. The utilization of the functional units is equal to 95.1%.

As noted, every core works on its dedicated memory space. The amount of main memory needed per core for implementing the correlation is equal to approximately 686 MB. The size of a 4-GB HMC vault, i.e., a vertical memory slice in the stacked DRAM which is managed by a dedicated controller, is equal to 128 MB [12]. By allocating six vaults to each core, the memory access requests can be served in parallel by the controllers without contention penalties.

Figure 8 shows the power breakdown in the case of the correlation while varying the number of FUs per core. The histograms are normalized with respect to the configuration using 4 FUs per core. For each core configuration, we chose the point that minimizes the product $power^2 \times area$. As modeled by Equations 2 and 1, by increasing the number m of FUs per core, the power reduces approximately as $1/m$. This is mainly due to the reduction of the number of micro-controllers needed and to the reduction of the communication between memory and cores, which decreases the memory access power, the HMC links power, and the on-chip communication power. By exploiting the locality of the station data in the correlation using wider cores, the number of accesses to the memory is significantly reduced. Figure 8 also shows that the main contribution to the power consumption is due to the accesses to the main memory. Together with the SerDes communication cost, the memory accesses account for around 73% of the power budget for the most energy-efficient configuration. As the computing capabilities needed are the same for each configuration, the power consumption due to RFs and execution units is approximately constant for all configurations. Both power consumptions are dominated by the dynamic components. Given the relatively small size of the SSRF, its contribution to the overall power consumption is negligible. By using 64-wide cores, we are able to save approximately 85% in power with respect to a 4-wide core configuration.

In Figure 9, we analyze the variation of the power consumption of the architecture components by varying the size of the SSRF and the RFs. We show results for the configuration with 8 functional units per core that minimizes the $power^2 \times area$ product. The configuration with minimal power (SSRF equal to 32) corresponds to the point at which the reduction in the memory and communication power equals the increase of the power of the RFs. In fact,

Table 2: Characteristics of the Pareto point with minimum power (α) and minimum die area (β). Values are reported for correlating all stations for each N_{ch} channels of one sub-band.

Point	Power (W)	Area (mm^2)	GFLOPS/W	$SSRF_{entries}$	RFs_{size}/core (KB)	RF_{conf}
α	11.47	14.54	257	4	268	SRAM, N
β	12.04	9.73	245	2	134	eDRAM, N

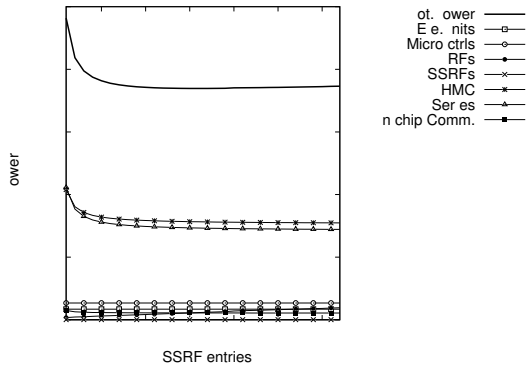

Figure 9: Power consumption while varying the size of the SSRF (8 FUs per core).

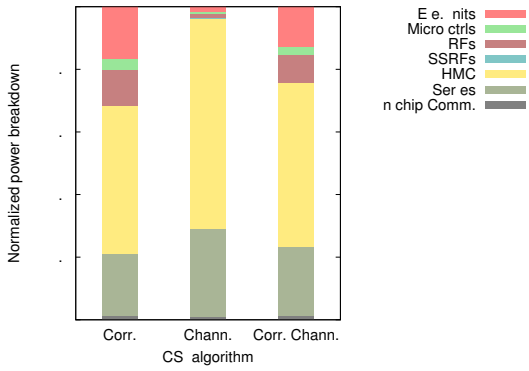

Figure 10: Power breakdown for correlation and channelization.

while the increase of the size of the SSRF decreases the memory-core communication (as shown in Equation 2), it also increases the size of the RFs and consequently the cost associated with accessing its elements.

4.2.2 Channelization algorithms

As can be derived from Table 1, the channelization step requires 12.2 Giga CFMAs per second for each sub-band. A single core with 64 FUs provides sufficient computational power for implementing the channelization. Therefore, we evaluated the execution of the channelization algorithms by considering an additional core with 256 MB of allocated memory space in the system. By implementing the FIR and FFT as described in Section 3.3.1, each single functional unit, working at 433 MHz, processes eight stations.

To exploit the locality of the input samples, we store several of them in the main memory and execute the FIR a sub-filter at the time over those time samples (by taking as reference Figure 2, the order of the results would be FFT(0), FFT(N_{ch}), FFT($2N_{ch}$), ..., FFT(kN_{ch}), FFT(1), FFT($1+N_{ch}$, FFT($1+2N_{ch}$), ..., FFT($1+kN_{ch}$), etc, where k depends on the number of samples stored). This implementation allows us to reduce the number of input samples read from main memory for each FIR execution to one asymptotically. The FFT step is then performed over all results accumulated in the main memory. A Radix-8 version of the FFT for CFMAs was considered [14]. Once the FFT step has been performed, samples are copied into the memory space of the cores implementing the correlation.

Our implementation of the channelization results in an utilization of the core equal to about 44%. The core requires an additional throughput from the HMC devices to core and from core to the HMC devices of approximately 22 GB/s and 18 GB/s, respectively. The execution of the channelization implies at the node level an additional power consumption of 2.23 W.

By spreading the constant power overhead over the operations performed, the energy of executing an 8-taps FIR is equal to about 2.2 nJ, whereas executing a 512-point FFT in this core configuration consumes 1.38 μJ.

Figure 10 compares the normalized power distribution for separate execution of the correlation and the channelization and when considering the two steps together. Accessing the main memory represents the biggest contribution to the power budget. In the case of the channelization, however, this contribution is higher in percentage, because of the relatively small exploitable input data locality of the filtering step, in particular when compared with the execution of the correlation, which presents a more compute-bound behavior [26]. As expected, at application level, the power consumption is dominated by the correlation.

4.2.3 Overall solution

Summarizing the studies presented in the two previous subsections, the CSP can be implemented with 512 nodes (one for each sub-band), each one composed of a 15-core chip with the configuration of point α in Table 2, and three 4-GB HMC devices. Each 15-core chip can theoretically deliver around 3.3 TFLOPS. The evaluated CSP algorithms achieve an overall utilization of 92%. The estimated power consumption for the node is 14.4 W, leading to an energy efficiency of 208 GFLOPS/W at application level. The 15-core chip occupies approximately 15.7 mm^2 of die area. The data throughput from and to the HMC devices is equal to 93 GB/s and 37 GB/s, respectively. One 60 GB/s SerDes link for each HMC is therefore sufficient for providing the required bandwidth. The chip input and output data rate are equal to 5.76 GB/s and 2.46 GB/s, respectively. Four transceiver macros are therefore sufficient for providing the needed I/O bandwidth.

5. RELATED WORK

The main CSP algorithms have been recently studied and evaluated in several computing platforms, with the goal of optimizing their execution by exploiting the characteristics of the systems targeted. As the channelization algorithms are well known and vastly discussed in the literature, we focus in particular on the correlation implementations.

In [24] and [18], the correlation algorithm was implemented on several architectures, i.e., CPUs, GPUs, and the IBM

Blue Gene/P and Cell processors. The authors observed that, given the characteristics of the algorithm, good performance can be achieved when sufficient local storage is available. Both the Cell and the IBM Blue Gene can achieve close to peak performance, whereas CPUs and GPUs reach significantly lower numbers (70% and 40%, respectively), with the Cell the most power-efficient architecture. However by applying a multi-level data-tiling strategy in memory, and by optimizing the overlapping of streaming data transfer and computation, peak performance of up to 80% can be achieved in modern GPUs for a large number of stations [5].

Research by Souza et al. [7] shows that FPGAs are a suitable candidate for running the correlation algorithm, because of its regular and simple structure. The use of custom data widths allows a potential reduction of the power consumption at the cost of a reduced programmability, flexibility, and algorithm scalability.

With regard to the proposed architecture, many different solutions can be found in the literature for accelerating specific kernels in high-performance computing by exploiting the intrinsic parallelism of the application. While systolic arrays were popular in the 80s [15], more recent works have, for example, studied the acceleration of mathematical kernels, such as general matrix-matrix multiplications (GEMM) [16, 25], or the acceleration of frameworks for the management of big-data workloads, such as MapReduce [17].

While similar in some of the aspects to the related work discussed, our paper presents the first system-level evaluation of the implementation of a custom architecture for accelerating the main algorithms of the SKA1-Low central signal processor. By approaching the problem at the application level, it is possible to optimize the memory utilization by reorganizing data for exploiting wide SIMD cores and the wide access and high bandwidth offered by 3D-stacked memories. This feature is supported in our architecture by the SSRF, which can be used in the execution of all algorithms discussed, and by the micro-controller, which exploits the regularity of the data access pattern for optimizing data movements in the memory layers and the functional units.

6. CONCLUSIONS

In this paper, we propose, for the first time, the implementation of a custom architecture for accelerating the main algorithms of the SKA1-Low central signal processor. After discussing the main characteristics of the algorithms and the challenges offered by the size of the problem, we propose an accelerator architecture that takes advantage of the bandwidth and low-power operations of new 3D-stacked memory devices. Our evaluations show that our architecture, thanks to a data organization that supports the execution of algorithms over wide SIMD cores, is able to process all 512 channels of a sub-band for just 14.4 W, achieving an energy efficiency at application level of up to 208 GFLOPS/W.

7. REFERENCES

[1] International Technology Roadmap for Semiconductors 2012 Update, 2012.

[2] G. Balamurugan, J. Kennedy, G. Banerjee, et al. A Scalable 5-15 Gbps, 14-75 mW Low-Power I/O Transceiver in 65 nm CMOS. *IEEE J. Solid-State Circuits*, 43(4):1010–1019, April 2008.

[3] R. Borkar, M. Bohr, and S. Jourdan. Advancing Moore's Law in 2014 - The Road to 14 nm. Intel Presentation, August 2014.

[4] G. Chen, M. Anders, H. Kaul, et al. A 340mv-to-0.9v 20.2Tb/s Source-Synchronous Hybrid Packet/Circuit-Switched 16x16 Network-on-Chip in 22nm Tri-Gate CMOS. In *ISSCC '14*, pages 276–277, Feb 2014.

[5] M. A. Clark, P. C. L. Plante, and L. J. Greenhill. Accelerating Radio Astronomy Cross-Correlation with Graphics Processing Units. *CoRR*, abs/1107.4264, 2011.

[6] L. R. D'Addario. Low-Power Correlator Architecture for the Mid-Frequency SKA, Memo 133. Technical Report, Jet Propulsion Laboratory, California Institute of Technology, 2011.

[7] L. de Souza, J. Bunton, D. Campbell-Wilson, et al. A Radio Astronomy Correlator Optimized for the Xilinx Virtex-4 SX FPGA. In *FPL'07*, pages 62–67, Aug 2007.

[8] S. Galal and M. Horowitz. Energy-Efficient Floating-Point Unit Design. *IEEE Trans. Comput.*, 60(7):913–922, July 2011.

[9] J. R. Geraci and S. M. Sacco. A Transpose-free In-place SIMD Optimized FFT. *ACM Trans. Archit. Code Optim.*, 9(3):23:1–23:21, Oct. 2012.

[10] B. Giridhar, M. Cieslak, D. Duggal, et al. Exploring DRAM Organizations for Energy-Efficient and Resilient Exascale Memories. In *SC '13*, pages 1–12, Nov 2013.

[11] J. Jeddeloh and B. Keeth. Hybrid Memory Cube New DRAM Architecture Increases Density and Performance. In *VLSIT '12*, pages 87–88, June 2012.

[12] Hybrid Memory Cube Consortium. Hybrid Memory Cube Specification 2.0. Nov. 2014.

[13] R. Jongerius, S. Wijnholds, R. Nijboer, and H. Corporaal. An End-to-End Computing Model for the Square Kilometre Array. *Computer*, 47(9):48–54, Sept 2014.

[14] H. Karner, M. Auer, and C. W. Ueberhuber. Top Speed FFTs for FMA Architectures, 1998.

[15] T. Lippert, N. Petkov, P. Palazzari, and K. Schilling. Hyper-Systolic Matrix Multiplication. *Parallel Computing*, 27(6):737–759, 2001.

[16] A. Pedram, J. McCalpin, and A. Gerstlauer. Transforming a Linear Algebra Core to an FFT Accelerator. In *ASAP '13*, pages 175–184, 2013.

[17] S. Pugsley, J. Jestes, H. Zhang, et al. NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads. In *ISPASS '14*, pages 190–200, March 2014.

[18] J. W. Romein, P. C. Broekema, J. D. Mol, and R. V. van Nieuwpoort. The LOFAR Correlator: Implementation and Performance Analysis. In *PPoPP '10*, pages 169–178, New York, NY, USA, 2010. ACM.

[19] SKA Organisation. Square Kilometre Array. "`http://www.skatelescope.org/`".

[20] S. Thoziyoor, J. H. Ahn, M. Monchiero, et al. A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies. In *ISCA '08*, pages 51–62, Washington, DC, USA, 2008. IEEE Computer Society.

[21] J. van Lunteren. Towards Memory Centric Computing: a Flexible Address Mapping Scheme. In *CCECE '99*, vol.1, pages 385–390, May 1999.

[22] J. van Lunteren. High-Performance Pattern-Matching for Intrusion Detection. In *INFOCOM '06*, pages 1409–1421, 2006.

[23] J. van Lunteren. Memory-Driven Near-Data Acceleration and its application to DOME/SKA. 2014 HPC User Forum, September 2014.

[24] R. V. van Nieuwpoort and J. W. Romein. Correlating Radio Astronomy Signals with Many-Core Hardware. *Int. J. Parallel Progr.*, 39(1):88–114, 2011.

[25] S. Vangal, J. Howard, G. Ruhl, et al. An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS. *IEEE J. Solid-State Circuits*, 43(1):29–41, Jan 2008.

[26] E. Vermij, L. Fiorin, R. Jongerius, C. Hagleitner, and K. Bertels. Challenges in Exascale Radio Astronomy: Can the SKA Ride the Technology Wave? *Int. J. High Perform. Comput. Appl.*, 29(1):37–50, Feb 2015.