Optimization of Calibration Algorithms on a Manycore Embedded Platform

Nicolas Sourbier, Jean-François Nezan, Cyril Tasse, Julien Hascoet

Optimization of Calibration Algorithms on a Manycore Embedded Platform

SOURBIER Nicolas
Univ. Rennes, INSA
IETR, CNRS UMR 6164
Rennes, France
Nicolas.Sourbier@insa-rennes.fr

NEZAN Jean-François
Univ. Rennes, INSA
IETR, CNRS UMR 6164
Rennes, France
jnezan@insa-rennes.fr

TASSE Cyril
Observatoire de Paris, CNRS
PSL University
Meudon, France
cyril.tasse@obspm.fr

HASCOET Julien
Kalray
Montbonnot-Saint-Martin, France
jhascoet@kalray.eu

***Abstract*—This paper presents the porting and the optimization of full polarization, direction independent calibration algorithm for radio-interferometry, on an embedded many-core platform. In astronomy, calibration algorithms consist of solving for the unknown complex antenna gains using a known model of the sky. Calibration is a key computation to provide images of the sky at good quality and high resolutions. In the context of the Square Kilometer Array (SKA) project, real-time and low power execution of the calibration is challenging. In this paper, we show that the CohJohnes algorithm provides good properties for being executed efficiently on the new generation of many-core embedded platforms. Experimental results are provided using the Kalray MPPA Bostan platform running 288 64-bit VLIW cores and delivering up to 845 GFLOPS at 12W.**

***Index Terms*—Signal Processing, Astronomy, Calibration, SKA project, Many-core Embedded Systems, Optimizations**

I. Introduction

Calibration is a key computation to go from the radio interferometer measurement to the evaluation of the signal produced by a source in the universe. The gain calibration aims to converge theoretical gains toward the unknown complex antenna gains using a sky model. In the context of the Square Kilometer Array (SKA) project, the calibration algorithm will have to compute the data from a huge number of antenna. For instance, the SKA1 Low subproject will deliver 157 terabytes per second, enough to fill up 35000 DVD every second. Furthermore the computations must be done on site, as close as possible to the antenna to decrease the cost of the data transfers. As a result, the calibration algorithm must provide the best trade-off in terms of the resulting image quality, computation time and energy consumption.

After decades of exponential growth, the processing capability of individual Processing Elements (PEs) have leveled-off, due to complexity and power consumption considerations. In order to cope with the rising complexity of modern embedded applications, Multiprocessor Systems-on-Chips (MPSoCs) have now increased their processing capabilities by integrating more and more PEs, a wide variety of memory architectures, and complex on-chip interconnects. The MPPA processor from Kalray consists of a set of 16 clusters communicating through a Network-on-Chip (NoC), where a cluster designates a set of 16 PEs sharing a local memory of 2 MBytes. Programming MPSoCs with hundreds of heterogeneous PEs, complex memory architectures, and NoCs remains a challenge for embedded system designers but at the same time it delivers high energy efficiency compared to conventional solutions.

Gain calibration is a complex non-linear least square problem and is solved here by the approach of Levenberg-Maquardt (LM) based on the Radio-Interferometric Measurement Equation (RIME [1]) to compute direction independent (DI) gain calibration. Furthermore, this study will rely on the Wirtinger optimization presented in [2] that allows faster computations due to matrix simplifications. The sparse properties of matrices allows to drastically reduce the algorithmic complexity of some computations such as inversions or matrix products.We show here how the properties of the CohJohnes algorithm can be exploited to efficiently execute the calibration on modern MPSoCs. Results are obtained on an Bostan MPPA platform in term of speed and energy consumption.

In section II we present the MPPA platform and its architecture. You will find there more details about the data communication on the chip, the clustered architecture and the organization of the shared memory. Section III is about the Antenna calibration algorithms, related works and optimizations used to solve the calibration problem. The RIME equation are first reminded. The scalar polarization calibration is then briefly explained followed by the fully polarized case. Section IV shows how we adapt the calibration algorithms to the MPPA platform, present the structure of the program and the results in terms of speed and energy consumption.

Notations and frequently used symbols

Symbol	Description
$\widehat{g_i}$	Gain vector at the iteration i
$J\mid_{\widehat{g_i}}$	Jacobian of $\widehat{g_i}$
$J\mid^{H}_{\widehat{g_i}}$	Hermitian Transpose of the Jacobian
H	Hessian matrix
λ	damping coefficient
v_m	visibility vector
$h(\widehat{g_i})$	data vector of $\widehat{g_i}$
Na	number of antenna
Nbl	number of baselines
Nd	number of directions
Nt	number of time-frequency points
Np	number of polarizations

II. Manycore Embedded Platform Programming

The Kalray MPPA (Multi-Purpose Processing Array) many-core architecture is the state-of-the-art in terms of many-core embedded architecture. It is a great opportunity to execute calibration algorithms in SKA in real-time and at low power. The MPPA2-256 many-core processor integrates 256 VLIW application cores and 32 VLIW management cores operating at 400MHz on a single chip and delivers more than 80 GOPS/W.

The 256 VLIW application cores are grouped in 16 compute clusters connected with each other by a dual Network-On-Chip (NoC). Each cluster has 2 MB of locally shared memory called SMEM and available for application code and data. In addition to the 16 compute clusters, the MPPA includes 2 Input/Output clusters that communicate with the external world through high-speed interfaces such as the PCIe Gen3 and Ethernet 10 Gbits/s. Those I/O clusters also integrate quad-cores with the same VLIW architecture and are connected to up to 4GB of external DDR3 memory. Each MPPA core implements a 32-bit VLIW architecture which issues up to 5 instructions per cycle, corresponding to the following execution units: branch & control unit (BCU), ALU0, ALU1, load-store unit (LSU), multiply-accumulate unit (MAU) combined with a floating-point unit (FPU). We use the MPPA POSIX-Level

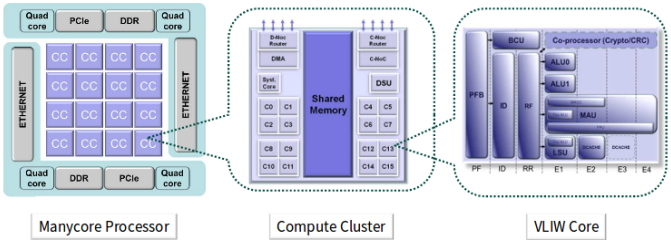

Fig. 1. MPPA-256 Bostan processor architecture.

programming model [3] in which the distribution of code and data across the compute clusters are explicitly managed. With this model, each compute cluster can execute an independent process started from its own binary executable. Inside each compute cluster, up to 15 additional POSIX threads can be created as a first thread already runs the main program. This programming model offers control over the memory footprint and supports thread-level parallel execution within each cluster with one thread per core. The POSIX-Level programming model also manages data transfers by using the POSIX synchronous and asynchronous file operations. These operations activate the dual NoC for internal communications and the high speed interfaces (Ethernet, PCIe Gen3, NoCX) for communications with the external world.

The programming of such an architecture is challenging due to the memory architecture (shared memory inside clusters and distributed memory between clusters) and the small size of the SMEMs compared to the large data to compute. The 80 GOPS/W are reached if the algorithm can be mapped and scheduled on the 256 cores without waiting times due to data transfers. A great care on the algorithm selection and implementation strategies are required.

III. Antenna calibration background

A. RIME formalism and Calibration Algorithm

The RIME equation is obtained by correlating data coming from two radio-telescopes. The RIME is presented in [4] and details about Antenna calibration can be found in [5]. Each antenna A_i delivers complex voltages V from an electromagnetic field E. G are the complex gains we want to compute in the calibration process, B represents the Bias effects and k is the complex exponential term.

$$V_i = \vec{E_{Ai}} = \exp\left(2j\pi(\Omega t + kx_i)\right) = G_i B_i * k_i \vec{E} \quad (1)$$

The calibration is computed using the visibilities which are correlation between two antenna (A_p and A_q) :

$$V_{A_pA_q} = < V_p * V_q^H > \quad (2)$$

$$= G_{Ap} B_{Ap} k_{Ap} \vec{E}\vec{E^H} * k_{Aq}^* * B_{Aq}^H G_{Aq}^H \quad (3)$$

Traditional non-linear least squares problem are solved using iterative algorithms like Gauss-Newton (GN) or Levenberg-Marquardt (LM) algorithms. Those algorithms have been extended to complex functions using a formalism called the Wirtinger derivative in which the conventional real Jacobian becomes a complex.

We implement here the CohJones algorithm described in [2] and based on the LM algorithm using the Jacobian matrix in the Wirtinger framework. More details are available in [6]. In this approach, visibilities are written v_{pq} or v_m for a baseline pq, at time t and frequency ν such as :

$$v_{pq} = v_m = Vec(V_{(pq)t\nu}) \quad (4)$$

$$= \sum_d (\overline{J_{qt\nu}^d} \otimes J_{pt\nu}^d) Vec(S_d) k_{(pq)t\nu}^d \quad (5)$$

Where d represents the directions, J the Jacobian matrix, S_d is the four-polarization sky matrix and $k = exp(-2i\pi(ul + vm + w(n-1)))$ with $n = \sqrt{1 - l^2 - m^2}$.

Results presented here focus on a single direction ($d = 1$) and are time frequency independent. Furthermore, we consider only one source in the center of the observation field ($l = m = 0$) so the complex exponential $k = 1$. This simplification avoids the calculation of the exponential but does not interfere with the Levenberg-Maquardt algorithm. Eq.5 becomes :

$$v_{pq} = (\overline{J_{qt}} \otimes J_{pt}) Vec(S) \quad (6)$$

where J_p is the 2 X 2 Jones matrix of the antenna p.

The CohJones algorithm being iterative, a distinction is made between the measured visibilities v_m obtained with eq.4 and the estimated visibilities v_i at an iteration i. The calibration gain vector $\widehat{g_i}$ at iteration i is computed using Eq.7 and converge toward the G_i gains of eq.1.

$$\widehat{g_{i+1}} = \widehat{g_i} + (H\mid_{\widehat{g_i}} + \lambda \cdot diag(H\mid_{\widehat{g_i}}))^{-1} J\mid_{\widehat{g_i}}^H (v_m - v_i) \quad (7)$$

$$with\ H\mid_{\widehat{g_i}} = J\mid_{\widehat{g_i}}^H J\mid_{\widehat{g_i}} \quad (8)$$

B. Considering a scalar polarization

We can then rewrite eq.5 for a scalar polarization :

$$v_{pq} = \overline{J_q} \times J_p \tag{9}$$

where J_p is a scalar term.

1) The Jacobian matrix: The Wirtinger Jacobian matrix $J_{v_{pq},gw}$ is onerous to compute due to its dimensions. In the DI problem, its size is : $N_p N_{bl} n_t$ x $2N_a N_p$. and is, for a given baseline:

$$J_{v_{pq},g_w} = [J_{v_{pq},g} \quad J_{v_{pq},\overline{g}}] \tag{10}$$

Hopefully the Wirtinger optimization allows to compute and use only half of that matrix. The matrix below details the structure of the Jacobian (in the case of a three antenna calibration) that we will be dealing with.

$$J_{v,g_w} = \begin{bmatrix} a & 0 & 0 \\ b & 0 & 0 \\ 0 & c & 0 \\ 0 & d & 0 \\ 0 & 0 & e \\ 0 & 0 & f \end{bmatrix} \tag{11}$$

Where the a to f components are complexes. In fact, for a scalar polarization, one line of the Jacobian for the antenna a can be written as:

$$[J_{v_{pq},g}] = \begin{cases} \overline{g_q} \ for \ a = p \\ 0 \ otherwise \end{cases} \tag{12}$$

2) Inverting the Hessian Matrix: As the Hessian $H = J^H J$ is a diagonal matrix. The diagonal coefficients are no longer complex but real values. This matrix will be easily inverted with an algorithmic computation of $\mathcal{O}(Na)$. Furthermore we have that equation from [2] :

$$v = (J_{vg} \mid_{\overline{g}}) g \tag{13}$$

Injecting equation 13 in equation 7, and having $H = diag(H)$ we obtain the hereafter algorithm:

$$\widehat{g_{i+1}} = \lambda(\lambda+1)^{-1}\widehat{g_i} + (\lambda+1)^{-1} H \mid_{\widehat{g_i}}^{-1} J_{v,g} \mid_{\widehat{g_i}}^{H} v_m \tag{14}$$

C. Considering the fully polarized case

In the fully polarized case, eq.6 cannot be simplified as eq.9 and J_p is the $2X2$ Jones matrix of the antenna p.

1) The Jacobian matrix: the a to f components described in equation 11 are now 4 x 4 blocks. The shape of one block is described in the hereafter matrix 15. More details about the Jacobian can be found in [4].

$$J_{v_{pq},gw} = \begin{bmatrix} a & 0 & b & 0 \\ 0 & a & 0 & b \\ c & 0 & d & 0 \\ 0 & c & 0 & d \end{bmatrix} \tag{15}$$

2) Inverting the Hessian Matrix: In the fully polarized case, this matrix is block diagonal with hermitian blocks that present the same structure as the matrix 15 with $c = \overline{b}$ and each block corresponds to a specific antenna. In practice, the coefficients b and c have a smaller amplitude than a and d. It is then possible to make the approximation $H \approx diag(H)$ allowing to calibrate data using the algorithm described in equation 14.

Due to floating points approximations, we chose not to approximate the calculation of H^{-1}. As the inversion of a 4 x 4 matrix is a bit onerous, we rely on its shape and properties to show that as long as H is invertible each block of H^{-1} can be written as:

$$\alpha \begin{bmatrix} d & 0 & -b & 0 \\ 0 & d & 0 & -b \\ -c & 0 & a & 0 \\ 0 & -c & 0 & a \end{bmatrix} \tag{16}$$

with $\alpha = \frac{1}{da - |b|^2}$.

IV. MPPA Implementation and Results

A. Mapping the calibration on the MPPA cores

Using the property that it is possible to separate data over antenna, we built a two level master-slave software. The I/Os processor enslaves the clusters, then the main core of a cluster enslaves the other cores of its cluster. The three following graphs display the structure of the program and the parallelism. Considering a scalar polarization and for one

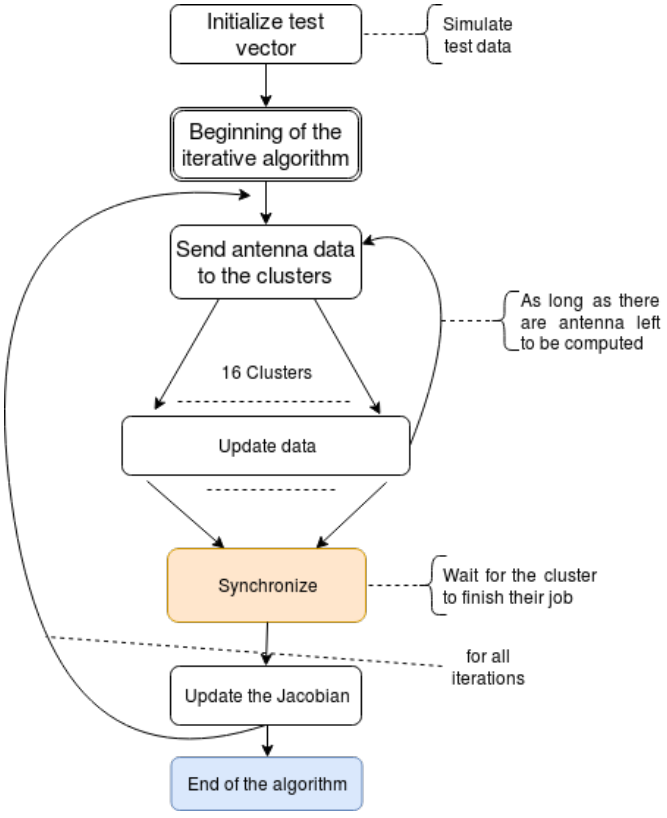

Fig. 2. Program execution on the IO/s

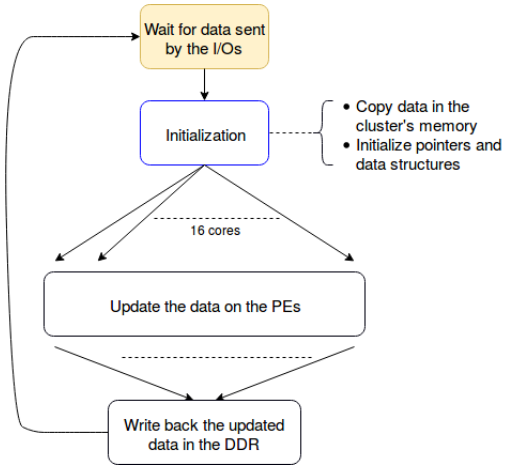

Fig. 3. Program execution on each Clusters

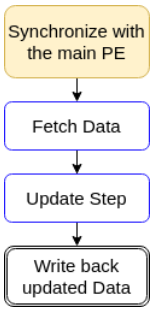

Fig. 4. Program execution on the PEs

iteration, indexes corresponding to antennas are sent from the I/Os to the clusters and each cluster fetches the data pertaining to the indexes. Then the main PE of the cluster dispatches the data corresponding to one antenna on each core and begin the computation. At the end of the execution, the updated part of the vector is written back into the DDR.

In a fully polarized case, the computation is more complex and the number of data is increased so that the 2Mb limit in a cluster do not allow to compute the data of one antenna on each core of a cluster. The matrix operations on the data of one antenna are executed in parallel over the 16 cores of the cluster.

B. Optimizations

As seen before, the calibration involves onerous matrix computation and the MPPA architecture comes with consequent memory constraints. We present here various optimizations to reduce both the complexity and the computation time but also to ensure the portability of the algorithms on the MPPA. We present here the optimizations on the scalar polarization problem but the same have been used for the full-polarization.

1) Optimizations based on the Jacobian shape: Relying on the idea that the Jacobian had the structure presented in eq.11, only the non null components in a vector are stored. As the native Jacobian had to store $Na(Na-1)*Na$ complex floats, we save $Na*64bits$. Furthermore, considering the Jacobian as a vector makes the complexity of $J_{v,g}\,|^H_{\widehat{g}_i}\,v_m$ product going from $\mathcal{O}(Na^3)$ to $\mathcal{O}(Na^2)$.

2) Optimizations around the Hessian: The Hessian H is obtained by the calculation $H\,|_{\widehat{g}_i} = J\,|^H_{\widehat{g}_i}\,J\,|_{\widehat{g}_i}$. Knowing the diagonal structure of the Hessian, we consider it as a vector of Na components instead of a $Na*Na$ matrix. As we consider the Jacobian as a vector, the computation of the Hessian has no longer a complexity in $\mathcal{O}(Na^4)$ but in $\mathcal{O}(Na^2)$. Moreover, considering the Hessian as a 1-D vector saves some memory in a cluster. As long as all the components of H are non-null, this matrix is trivially invertible. Another matrix product is required in this algorithm (the one between H^{-1} -that should be $Na*Na$ elements- and $J_{v,g}\,|^H_{\widehat{g}_i}\,v_m$ -that is a Na elements vector). The complexity of this product is reduced to $\mathcal{O}(Na)$ by the previous optimization.

C. Performances

This part presents, compares and comments MPPA and x86 (intel Core i5-3570 CPU - 3.40GHz x 4) implementations in terms of quality, computation time and energy consumption (estimating the power consumption of an x86 architecture at 50 W). The following table is obtained after 60 iterations.

Na	memory (Mo)	computing time (s)	Energy (J)
non-optimized code on a x86 platform (single core)			
32	0.279	1.039	51.95
64	2.163	4.587	229.35
256	1357	174.364	8718.2
1024	>8600	unable to run	x
non-optimized code on the MPPA (single core)			
32	0.279	927.651	435.6
>62	>1.97	unable to run	x
optimized code for MPPA (256 cores)			
32	0.008	0.063	0.51
64	0.017	0.197	0.87
256	0.066	2.736	8.35
1024	0.262	43.562	140.15

Fig. 5. Program performance in a scalar polarization case

1) Scalar polarization: The figure 5 displays the results in terms of speed and of energy consumption of the calibration of Na antenna while considering a scalar polarization on different platforms and with optimized/unoptimized implementations. The "unable to run" value refers in fact to a saturated memory (RAM for the x86 architecture and Cluster memory for the MPPA). Data separation and reshaping matrices allow us to calibrate up to 6000 antenna on the MPPA. In the unoptimized case, we can estimate the amount of data in one cluster using the formula $memory_{used} = Na^3 + 2Na^2 + Na$. Without taking into account local variables, code, etc... Using 8 Bytes variables leads to fill the memory of a cluster if we try to calibrate 62 antenna or more (52 in practice).

The calibration time and the energy consumption of a calibration is not linearly scaled up with the number of antenna. This is due to the size of the matrices that we have to compute with. Indeed, if we take the Jacobian's case, dealing with 6 antenna will lead to a matrix with 18 elements (6 with our optimization) and calibrating 4 antenna will lead to a Jacobian of 48 elements (12 with optimizations). Obviously, bigger matrices lead to bigger computation and requires more resources. This justifies all the computational shortcuts and the reshaping of the matrices. The optimized code on the MPPA applies a separation over the antenna. It leads to have in each cluster and for each antenna, matrix sizes that vary linearly with Na. However, using this software architecture don't lead to a linear energy consumption or runtime. As each cluster is designed to calibrate 16 antenna (one antenna per PE) at its maximum, only two clusters are required to calibrate 32 antenna but the 16 clusters start and fetch data 4 time to calibrate 1024 antenna over one iteration. Moreover, updating the Jacobian matrix in the IO is an heavy computation that also increase the energy consumption and calibration time independently of the clusters. With the parameter Na fixed to 256, the measured speed-up between 1 core on 1 cluster and the 256 cores over one iteration of the MPPA is **33**. This low value is due to the fact that starting one PE on one cluster will grant him direct access to the DDR as the NoC will not be used by any other cluster.
The here after figure 6 displays a comparison of the number of operations per Joules between a x86 architecture and the MPPA.

Na	architecture	op/J
32	x86	37
	MPPA	2341
64	x86	16
	MPPA	2526
256	x86	2
	MPPA	1233

Fig. 6. Number of operations per Joules on different platforms

The following table 7 sums up the cost of 1 iteration in terms of time and energy consumption with several different started cluster/PE configurations (sorted by energy consumption) for a fixed number of antenna.

PE	clusters	iteration time (ms)	Energy (J)
16	1	35.83	0.097
16	4	8.99	0.107
8	8	4.82	0.127
1	1	76.39	0.132
1	16	5.80	0.155
4	16	3.00	0.167
16	16	2.45	0.207

Fig. 7. Energy consumption and calibration time depending on the number of cluster and PE started

Although we know the link between energy consumption and computation time, this table is a good indicator to find out an optimal configuration to run our program. We can easily extract from this table a linear coefficient between the iteration time and the number of started clusters. By choosing the number of started clusters and PE, we are able to regulate the energy consumption to get closer to the real-time constraint.

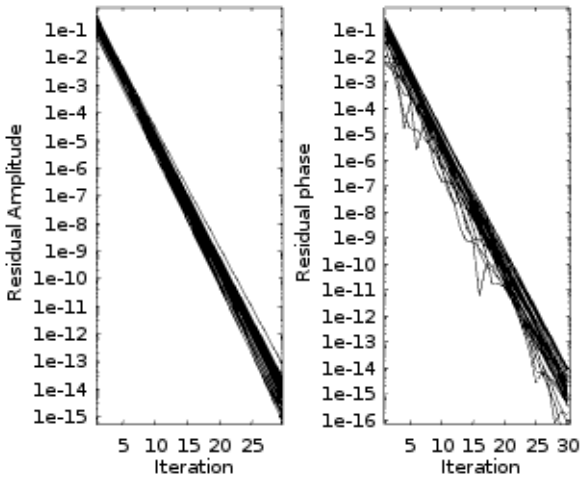

Fig. 8. Amplitude and phase of the difference between the estimated gains at iteration x and the true gain vector in a 40 antenna scalar calibration case

We can use this plot to deduce a better design of our system. As the SKA project requires a specific precision (around 10^{-7}), we can then choose the optimal number of iterations to reduce both computation time and energy consumption.

2) Full polarization: The figure 9 displays the results in terms of speed and energy consumption of the calibration of Na antenna while considering a fully polarized problem using the MPPA and a x86 architecture. The comparison with an unoptimized program on the MPPA is not relevant as without optimization, the memory of a cluster would be filled with 20 antenna.

Na	memory (Mo)	computing time (s)	Energy (J)
non-optimized code on a x86 platform (single core)			
32	4.36	28.08	1404
64	34.2	124.66	6233
128	271	659.109	32955
optimized code for MPPA (64 cores)			
32	0.002	0.53	1.93
64	0.004	2.09	6.25
128	0.008	8.34	24.23
512	0.033	133.3	410.85
1024	0.066	533.2	795.58

Fig. 9. Program performance in a fully polarized case

Here, we chose to start only 64 cores of the MPPA (16 clusters of 4 PE) to reduce the size of the data transfers and

focus more on the computation. One antenna is computed at a time on one cluster and we use data parallelism to compute the four different polarizations on four PEs. The speedup on the iterative part of a 256 antenna calibration is **147**.

Na	architecture	op/J
32	x86	5.47
	MPPA	3979
64	x86	2.46
	MPPA	2457
128	x86	0.93
	MPPA	1268

Fig. 10. Number of operations per Joules

The figure 11 displays the accuracy of the computing on a 10 antenna calibration for the real and imaginary part of each antenna at each polarization. We can see on these plot that the algorithm takes more time to converge. This is partly due to a mathematical shortcut that we use to perform singular value decomposition to take the first antenna as a reference.

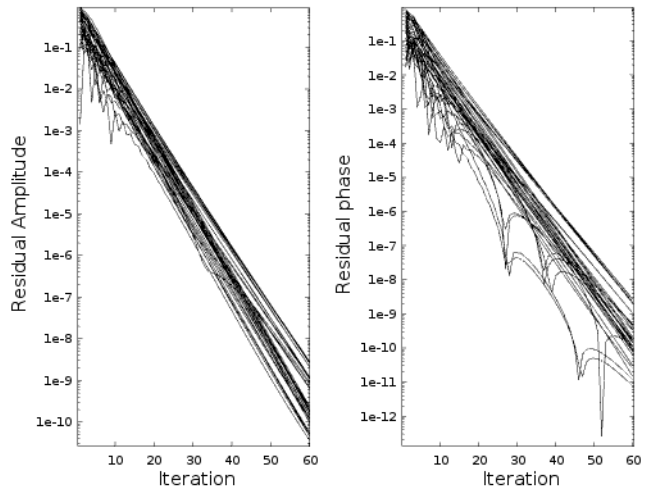

Fig. 11. Amplitude and phase of the difference between the estimated gains at iteration x and the true gain vector in a 10 polarized antenna case

The time spent for an iteration (the parallelized step) take a negligible amount of time in comparison with the whole program. This is mainly due to the computation of the Jacobian that requires the whole Gain vector to be computed. As this gain vector is wide ($Np * Na$ elements of 8 bytes and the optimized Jacobian has also $Na * (Na - 1) * Np$ elements), migrating the computation of this matrix on the clusters is difficult and don't really improve the performances as the NoC is overloaded all the time.

V. CONCLUSION

This paper shows that calibration algorithms are efficiently executed on a MPPA many-core platform in terms of computing time and energy consumption. By studying the various equations, the shapes of the matrix and their properties, we have been able to significantly reduce the time of computation and thus to increase our performances. Numerous efforts have been made to reduce the size of the data, to ensure the computation on the 16 clusters of the chip. Hopefully properties of data separation and the computation on matrices lead to an obvious parallelism that is suitable for the MPPA.

This study was a first approach to the calibration problem and the final algorithms will take several time-frequency blocks and calibration's direction. The architecture of the MPPA is adapted for the computation of integers and the next generation of MPPA will improve the floating point performances. Future research will evaluate and compare fixed point and floating point implementations.

Even if there is still a lot to be done on these algorithms, we show here that the optimizations and the properties of the MPPA platform are relevant solutions for the SKA project on the calibration algorithms.

REFERENCES

[1] S. R. J. Hamaker J. P., Bregman J. D. 1996.

[2] C. Tasse., "Applying wirtinger derivatives to the radio interferometry calibration problem," 2014.

[3] B. D. D. Dinechin and G. L. C. L. B. O. J. R. T. S. al., Pierre Guironnet de Massas, *A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor*. June 2013.

[4] O. Smirnov, "Revisiting the radio interferometer measurement equation," *arXiv*, 2011.

[5] C. O.M. Smirnov, "Radio interferometric gain calibration as a complex optimization problem," *arXiv*, 2015.

[6] R. Hunger, "An introduction to complex differentials and complex differentiability," *TUM*, 2007.

[7] S. Kosnac, "Kalray mppa many-core processors," 2015.

[8] W. Wirtinger *Mathematische Annalen*, 1927.

[9] K. Madsen, H. B. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems (2nd ed.)," 2004.

[10] B. D. de Dinechin, "Engineering a manycore processor platform for mission-critical applications," 2015.