A MANY-CORE ENERGY AND LATENCY ESTIMATOR

Author's Details:Ononiwu, G. C., Chukwudebe, G. A., Ndinechi, M. C., Okafor, E. N. C.
Department of Electrical and Electronic Engineering, Federal University of Technology, Owerri, Nigeria.

Abstract *-Many-core processors can now be used as an alternative hardware platform for implementing embedded media devices. However, a lack of generic tools for application development may hamper their rate of adoption by industry. This work has contributed towards the solution by providing an abstraction from the many design constraints facing application developers. A Many-core Energy and Latency Estimator (MELE) has been designed to improve the programmer's ability to iteratively map data flow applications to a target machine. A set of models included in the tool transforms an application mapped to an abstract machine into its Intermediate Representation. An Abstract Interpreter which runs on the Ptolemy II modeling platform is used to return feedback from the Intermediate Representation to the programmer. A case study has been used to showcase the use of MELE in analyzing the mapping of data flow applications. The case study has also been used to explain how a rank based system can arrive at the most suitable mapping of an application to the processor based on energy and latency costs. Results from the case study shows that the use of a greater number of cores in the processing does not necessarily result in the highest ranked mapping. Also, some mappings take too long to arrive at their steady state processing cost value. This may result in a lower Quality of Service (QOS) for the application. Based on these findings, suggestions have been made for further work.*

Keywords*: Many-core, Energy, Latency, Hardware, Ranking*

I. INTRODUCTION

High performance embedded stream processing demands real-time performance and low power expenditures.Traditionally, time and power constraints were satisfied using hardwired logic in the form of ASICs and FPGAs. Such technology made it possible to meet the performance and power requirements of real time applications at the cost of limited flexibility and a high development cost.

As it stands today, ASICs are 50 times more efficient than programmable processors [1]. This efficiency gap makes ASICs the solution of first choice for high-volume embedded systems. They are preferred when performance and energy efficiency are of primary concern and when unit costs resulting from high Non Recurring Engineering (NRE) costs can be driven down with higher volumes. However, algorithms and standards are evolving at a faster pace than ever before and freezing them in hardwired logic exacerbates the flexibility problem. Also, as signal processing applications get even more complex, it is necessary to have a more generalised hardware implementation, while restricting the implementation of the (more) complex algorithms to software.

Solutions have therefore been recently developed in the form of highly parallel, much more power efficient programmable processors. These processors have in some cases reduced the efficiency gap between ASICs and programmable processors to within 3 times. They incorporate tens if not hundreds of distributed processing units, otherwise regarded as cores, which have the potential of increasing to the thousands in the not too distant future. Typically they communicate through Network on Chip (NOC) communication structures and maintain localized memories in the cores. These processors are now widely regarded as many - core processors and they are distinct from their Multi-core counterparts because of their distributed memory structure.

However, to be able to use these parallel machines efficiently, application developers need tools that can assist them in exploiting the increasing number of cores. These tools must be domain specific in order to make the most impact. They should enable programmers to easily develop applications on Many - core processors, and at the same time make it possible to scale down the overall energy consumed by the application.

This research has focused on developing a generic Many-core Energy and Latency Estimator (MELE). Generic in the sense that once a processor meets certain pre-specified conditions,

its machine parameters can be extracted and used to generate estimates for Digital Signal Processing (DSP) tasks that execute on it. The resulting tool can be used to analyze the mapping of DSP task graphs to Many-core processors. A rank based method that can be used to rank the distinct mappings of an application to a processor has also been developed. The aim is to identify the mapping that uses the least resources (from a set of mappings) in terms of energy, while still meeting a pre-specified latency constraint. We focus on using synchronous data flow (SDF) models to describe the DSP application, given the fact that SDFs are very suitable for modeling signal flows. The fact that the schedule is determined in advance of the run time environment, limits the cost of run-time supervision [2].

In this paper, we present a functional description of the MELE tool, with an emphasis on how it can be used to analyze a set of distinct mappings of an application to a generic machine. The models, cost methodology, Intermediate Representation, and the Abstract Interpreter used by MELE have already been researched and published in [3]. It is important to note that the tool does not aim at providing a cycle accurate simulation of the processing; rather its function is to guide the programmer on how to optimize the mapping in order to minimize resource use, and still meet with the stipulated end-to-end latency requirement of the application, based on the desired Quality of Service (QOS).
The rest of the paper is organized as follows; section II presents the related work; in section III, a brief description of the Many-core Energy and Latency Estimator (MELE) is presented; section IV presents a case study that describes the use of MELE and the ranking process; section V describes the results and finally, section VI concludes the paper.

II. RELATED WORK

Most performance prediction tools evolved for uni-processor systems. This is understandable as multiprocessing platforms did not join the mainstream for most applications till the early 2000s. Back in the 1970s and 1980s, multi computing platforms were stuck in labs as experimental machines and early architectural tools were limited to performance prediction. One dilemma that has presented itself to researchers has been the choice between; low-level circuit and verilog simulation tools, which have excellent accuracy but are far less useful for making architectural decisions and also take far too much time in returning results; and the less accurate architecture-level tools which are required at higher levels. One major disadvantage of the low level tools is that they require a complete design before they can be used; however, these simulations are required at a high level to be able to design the application.

CACTI was the first complete toolset developed for power, area and timing estimation for computer architecture research with a focus on RAM based structures [4]. It is an integrated system that combines memory and cache access time, area, performance and power models. The aim was to provide the user with a means of determining the tradeoffs between time, power and area based on mutually consistent assumptions. It is available in two forms; a C++ code version and a web based version. Recent versions of the tool now include support for SRAM and DRAM based caches as well as ordinary memory arrays. CACTI uses device models based on the industry – standard ITRS roadmap, to calculate device parameters at different technology nodes. It takes advantage of optimization features in determining a configuration with minimal power consumption for given area and timing constraints.

Another tool that has been widely used for processor power estimation is Wattch [5]. Wattch is an architectural level power analysis tool developed at Princeton University as an open source development. Just like the product of this work, it aimed to be used at the early stages of the design process to provide a means of determining the many hardware configurations based on tradeoffs that need to be made. It calculated dynamic power dissipation using switching events obtained from an architectural level simulation and capacitance models of the micro architecture. Like this work, it has used a simple linear scaling model; however, its scaling is based on the 0.8µm technology which is highly inaccurate when

making predictions for current and future deep submicron technology modes [5].

Orion 2.0 [6] is a power – area simulator for interconnection networks. It includes models for area, dynamic power and gate leakage but does not consider timing and short circuit power. It makes use of an architecture level regression analysis model for different router components based on energy numbers obtained from simulations. The capacitance of each component is derived using architectural parameters. The activities at each cycle are then used to trigger calculations of network power.

Another interesting tool is MCPAT (Multi-core Power, Area, and Timing) [7]. It is an integrated power, area and timing tool for multithreaded and multi-core processors, released in 2009. It also has support for many-core processor configurations and was designed to work with a variety of performance and thermal simulators over many technology generations (ranging from 90nm, 22nm and beyond). It can be used as a high-level architectural level tool, and similar to this work, it comes with an advantage of having the capacity to allow the user specify low-level details when they deviate from the default values. At the micro-architectural level, McPAT is able to model in-order and out-of-order processor cores, shared caches, integrated memory controllers, and similar to this work, multi-domain clocking. At the circuit and technology levels, the tool has supports for modeling critical-path timing, area modeling, dynamic, short-circuit and leakage power modeling for each of the device types forecast in the ITRS roadmap. These include bulk Complementary Metal Oxide Semiconductor (CMOS) and double-gate transistors.

Bengtsson and Svensson carried out a cycle accurate modeling of the RAW processor using the concept of *timed configuration graphs* [8]. This is very interesting because similar to this work; the timed configuration graphs were used to preserve information for the abstract interpreter. This research work actually started off were the modeling of the RAW stopped. However, their focus in modeling the RAW was on mapping task graphs to the cores and simulating the performance of such mappings. This work extends that concept by incorporating an energy model and variable speed cores. Unlike the many – core model this work has in mind, the RAW processor does not have the ability to vary either its speed or its frequency.

III. SYSTEM DESCRIPTION

The SDF description of the application serves as the input to the Many-core Energy and Latency Estimator (MELE) tool. The actors that make up the SDF application are placed on the abstract cores of the abstract machine in line with the mapping selected by the developer. Mapping is the distribution of actors to the cores of the processor. The selected machine configuration for the simulation is used to configure the abstract machine. This is also regarded as an input to the system and is collected through two editable input forms; a machine model form for collecting the performance parameters of the machine; and an energy model form for collecting the electrical parameters of the machine. Given a reconfigurable machine, it is possible to switch to different configurations of the same machine especially when it comes to the operating speed and voltage of the participating cores. The block diagram describing the MELE tool is presented in

Figure 1.

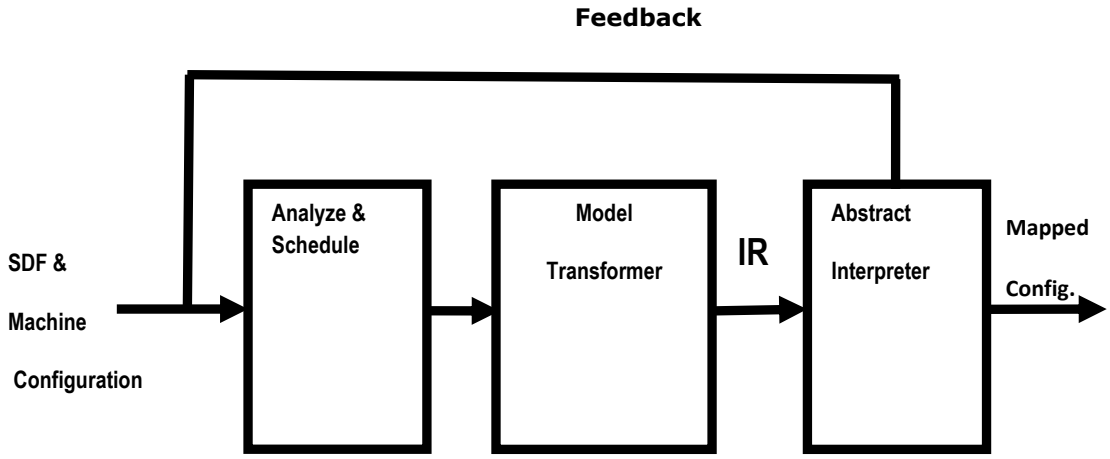

Fig.1: Block diagram representation of the Many-core Energy and Latency Estimation tool.

A. **ANALYZE AND SCHEDULE**: This is the first block of the MELE system. This module is responsible for providing the system with information regarding the SDF application and its mapping (placement) to the cores. It is this module of the system that first determines how many cores have been used for the mapping, how many communication channels are involved, and for each channel, which core is the source and which is the destination. All these are regarded as the top level map of the SDF application. Each core and channel is assigned its unique identifier, known as an index. It is these core indices and their corresponding channel indices that are stored in data structures as the top level map of the application. After the top level map of the application has been determined, the processing requirement of the SDF model is determined. For each actor, a worst case execution time is determined, which is equivalent to the time it takes to execute the operations in the actor, and stored in data structures for use in later stages of the tool. Also, the scheduler uses this stage to determine the repetition vector (firing sequence) of the application which is used to describe its schedule.

B. **MODEL TRANSFORMER**: The model transformer module serves as the engine of the tool. This is the point where a set of models and their related functions transform the mapped SDF application to an Intermediate Representation of the application. The set of model consist of an application model, a machine model and an energy model. The application model abstracts the details of the application that relate to memory requirements, communication requirements and worst case execution times of the task blocks. These details are presented to the abstract machine in the model transformer module. The parameters received from the application model are used in determining the cost of carrying out tasks in terms of energy and latency. On the other hand, the machine model presents the parameters of the machine to the system and has a set of performance functions associated to it. These functions calculate the cost of carrying out tasks. The same goes for the energy model. A detailed presentation of the models and their associated functions (energy and performance) can be found in [3]. The output of the model transformer module is an Intermediate Representation of the SDF application.

C. **INTERMEDIATE REPRESENTATION**: The intermediate representation (IR) is implemented as an abstract graph in a graph data structure in Java with vertices (nodes) and their corresponding edges (links). Each core or memory forms a vertex in the IR while each communication channel forms an edge in the IR. For each core, a vertex core is setup which has its own local timing. The vertex core records a sequence of discrete events that each bears a cost. The memory vertex can only record events that relate to global memory reads or writes. The cost of each event is calculated based on the machine and energy models and stored in the data structure. On the other hand, the channels introduce a delay that is equivalent to their individual weights. This is stored in edges of the IR. The sum total of the data structure formed at this point is regarded as the Intermediate Representation (IR).

D. **ABSTRACT INTERPRETATION**: This module takes in the IR that was generated in the Model Transformer as its input, and imposes the rules of the architecture on it. Since all the nodes are virtual cores in the IR that store timing information inside them based on local timing, what the interpreter does is to look at them from a global perspective. Starting from global time 0, it determines which core is processing and which core is waiting for input from other cores. It builds up a chronological history of all events that have taken place from time 0. This way, it is able to present the global timing and energy view to the programmer, making it possible for the programmer to receive feedback from the interpreter.

IV. CONFIGURATION SETTINGS

The configuration setting is made up of the parameters of the machine being modeled. It is used to describe the common resources of the machine that is available to the application. It is at the core of the tools portability. All a user needs do is set the parameters according to the particular machine that is being modeled. These parameters are used to define abstract performance and energy functions that compute the cost of various configurations of the application. Description of the machine is given by;

$$M = < (x, y), p, b_g, g_w, g_r, o, s_o, s_l, c, h_l, r_l, r_o, sf, f, w_l, I_{Leakage}, V> \quad (3.4)$$

Where

- x, y are the number of rows and columns of cores. They describe the exact location of the core in the processor array.
- p is the processing power (instruction throughput) of each core, in operations per clock cycle. For the class of many – core processors being modeled by this work, 1 will be represent the default value because these are simple cores.
- b_g is global memory bandwidth, in words per clock cycle. All memory being used by the application that is not local to the core will form part of b_g.
- g_w is the penalty for global memory write, in words per clock cycle. This cost will come into play once an application writes to the global memory.
- g_r is the penalty for global memory read, in words per clock cycle. This cost will come into play once an application reads from the global memory.
- o is the software overhead for initiation of a network transfer, in clock cycles per frame. It is the cost in clock cycles of preparing the data to be written into the output buffers of the core in preparation for a network transfer operation.
- s_o is core send occupancy, in clock cycles, when sending a message. It is the cost of writing into the output buffers of the core when a network operation is taking place.
- s_l is the latency for a sent message to reach the network, in clock cycles. It is the average time it takes for the sent message to be injected into the network.
- c is the bandwidth of each interconnection link, in words per clock cycle. It describes the carrying capacity of the link.

- h_l is network hop latency, in clock cycles. It is the latency involved when moving a message between two cores.
- r_l is the latency from network to receiving core, in clock cycles. It is the average time it takes for the data to be extracted from the network by the core.
- r_o is core receive occupancy, in clock cycles, when receiving a message. It is the cost of writing data into the core buffers when data is received from the network.
- s_f is the scaling factor used to determine the speed and voltage level of the core with reference to the global operating frequency and voltage of the machine. $Sf_i \varepsilon S_F$ where i corresponds to the number of the core as can be determined by the x, y coordinates of the core.
- f denotes the global operating frequency of the processor.
- w_l is the average wire length between cores.
- $I_{Leakage}$ is the leakage current per core.
- V is the supply voltage.

The test that follows is aimed at displaying the use of the MELE tool in analyzing mappings of an application to a generic processor at the earliest time in the product development cycle. This test will be needed at a time when no processor has been selected for the application, and the results will aid the selection process. For the test, the following configuration setting has been selected based on equation 3.4.

M = < (4, 4), 1, 1, 1, 6, 2, 5, 1, 1, 1, 1, 3, (1:10), (100MHZ), (10nm), (0.001mA), 1.2V>

These are hypothetical values for a generic many-core processor. They were taken from the experiments used to model the RAW processor in the work carried out by Bengtsson and Svensson [8]. Also, the parameter values that relate to power, wiring and the frequency of the generic processor have been taken from the ITRS report predictions [9].

V. CASE STUDY: A SEQUENCE TO ARRAY CONVERTER

The case study presents a Sequence-to-Array Converter application implemented in the Ptolemy II modeling environment. All the communication in this application is flowing in one direction. This means that different mappings of the application will not affect the routing of the communication. What is at stake here is the optimum number of cores that are needed to give the best results and the optimum configuration in terms of the running speed of the cores. The block diagram of the application displayed in fig 2.

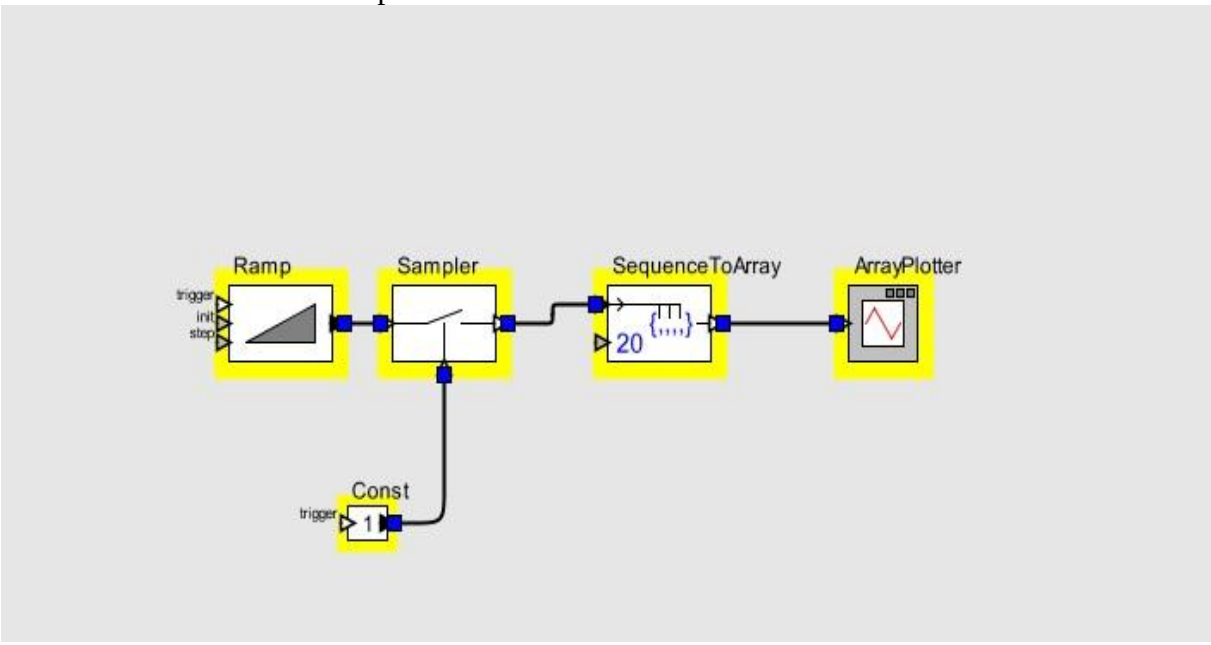

Fig 2: The Sequence-to-Array Converter application

The application consists of the following actors:

- A sequence source ramp.
- A Sampler.
- A generic source constant.
- A sequence to array.
- An array plotter

In total, the application has 5 actors. The ramp actor serves as a signal source whose output is sampled and sequenced to an array. The output of the sequence to array is plotted on an array plotter display.
The application is presented as input to the tool. Also, a machine configuration and a latency constraint are provided. In this case we assume the configuration presented in section IV and a latency constraint of 330 clock cycles. Note that the constraint is based on the target Quality Of Service (QOS) to be delivered by the application.

PLACEMENT: Using a 4x4 arrangement of cores, results in a 16 core machine. Each actor is regarded as an atomic entity that must not be subdivided. Then at a maximum, the application can take up to five cores. However, the generic source constant is too small to be in one core alone. This is due to the fact that it will result in an unbalanced load across the cores because its computational needs are by far less than that of any other actor. Therefore, at a minimum, a single core is used, while at max, Quad cores are used. Table 1 gives a breakdown of the distribution of the actors in the application to the cores.

Table 1: Breakdown of process task distribution for the Sequence-to-Array converter application.

Actor	Single Core Mapping (SCM)	Dual Core Mapping (DCM)	Triple Core mapping (TCM)	Quad core Mapping (QCM)
Ramp	Core 1	Core 1	Core 1	Core 1
Generic Constant	Core 1	Core 1	Core 1	Core 2
Sampler	Core 1	Core 1	Core 1	Core 2
SequenceToArray	Core 1	Core 2	Core 2	Core 3
ArrayPlotter	Core 1	Core 2	Core 3	Core 4

The list of possible scaling factor settings for the mapping of the application to the machine is given in table 2.

Table 2: *List of all possible scaling factor settings for the Sequence-to-Array Converter Application.*

Mapping	Abreviation	Scaling factor setting
Single Core Mapping	SCM	(1)
Dual Core Mapping Un-Optimized	DCM Un-Op	(1,1)
Dual Core Mapping Optimized	DCM Op	(2,2)
Triple Core Mapping Un-Optimized	TCM Un-Op	(1,1,1)
Triple Core Mapping Optimized	TCM Op	(1,2,3)
Quad Core Mapping Un-Optimized	QCM Un-Op	(1,1,1,1)
Quad Core Mapping Optimized	QCM Op	(2,1,1,2)

IV. RESULTS

A summary of the output from the system for the mappings are presented in tables 3.

Table 4.10: *Summary of results of all mappings (Sequence-to-Array Converter).*

Mapping	SCM	DCM Un-Op	DCM Op	TCM Un-Op	TCM OP	QCM Un-Op	QCM Op
Total periodicity	1070	162	324	192	315	210	288
Total Energy Consumed (µj)	30.82	3.65	0.46	3.84	1.75	3.12	2.62

The output for SCM puts it way beyond the latency constraint of 330 clock cycles. There is also no need to further optimize it because this only increases its latency. The DCM, TCM and QCM cases all had latencies well below the set latency constraint. Therefore, there was a need to further slowdown the cores in order to reduce the energy consumed. The scaling factor setting for slowing down the cores had to be carefully selected in such a way that the latency constraint was not breached. The values that have been used gave a latency that is as close as possible to the constraint. The ranking is now presented in table 4. The SCM mapping is not included in the ranking because it did not meet the constraint.

Table 4*: Ranking of the mapping of the Sequence-to-Array Converter application.*

Mapping	Energy Consumed (µj)	Rank
DCM Op	0.46	1
TCM Op	1.75	2
QCM Op	2.62	3
QCM Un-OP	3.12	4
DCM Un-Op	3.65	5
TCM Un-Op	3.84	6

Based on this ranking, the Optimized Dual Core Mapping (DCM Op) ranked best for this application. Further study shows that some mappings take too long to arrive at their steady state values. This can be seen in figures 3 and 4.

Latency in Clock Cycles

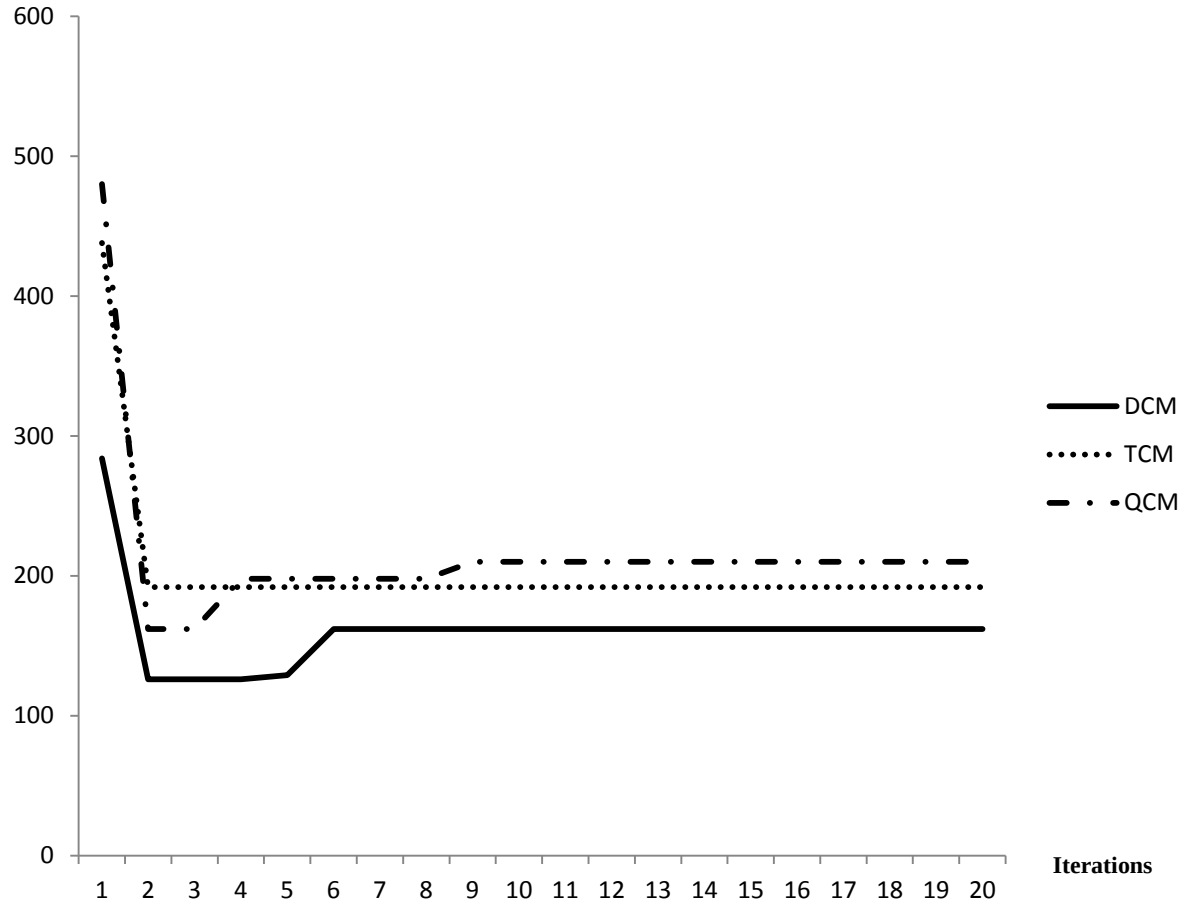

Fig. 3: Latency against Iterations for the Sequence-to-Array Converter application (Un-Optimized).

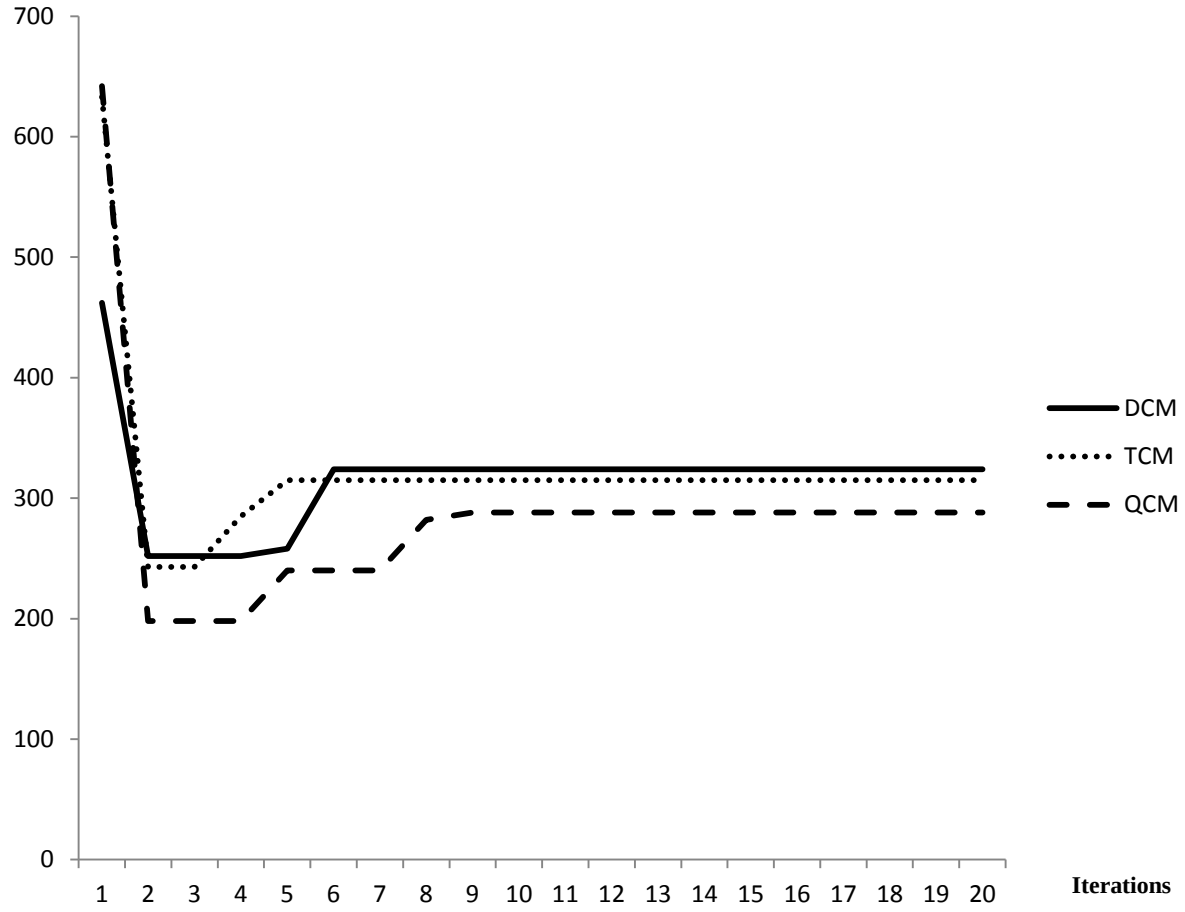

Fig. 4: Latency against Iterations for the Sequence-to-Array Converter application (Optimized).

In figure 3, the dual core mapping took 6 iterations before arriving at its steady state value and the same happened in figure 4. This is in contrast to the TCM mapping in figure 3 which took only 2 iterations to reach steady state conditions. This can have a significant impact on the Quality of Service (QOS) provided by the application. However, the level of impact will be determined only after a careful study. This may necessitate some adjustment to the ranking methodology.

V. CONCLUSION

The objectives of this work which are to develop a tool for analyzing the dynamic execution cost of DSP tasks on many-core processors in terms of latency and energy consumption have been achieved.

A case study has been used to show case the Many-core Energy and Latency Estimator (MELE). Results from these experiments show that the use of a greater number of cores in the processing does not necessarily result in the highest ranked mapping. This is because the cost of synchronization grows as the number of cores used in the processing increases. Also, some mappings take too long to arrive at their steady state processing costs. This will most likely have a negative effect on Quality of Service.

These findings support the pre-design objectives of the Many-core Energy and Latency Estimator (MELE), in that they show that it can be used by a programmer to analyze the mapping and receive useful insights into the relative cost of carrying out the various tasks involved in the application. These can then be used to rank the mapping and thereby predict the best mapping and configuration for the application.

One major improvement that can be made going forward is to the scaling factor methodology. At the moment, a linear method has been used to model for multi-speed cores. It has been implemented this way because at the moment, there is not enough information on how multi-

speed cores have been implemented in reality. Very few processors have this capability and for the most recent processors like the Kalray's MPPA and the Anemone many-core processors, no work has been published to date. In the future, more information will be available and this work could be modified to take them into account.

Also, the Ranking method should be improved by taking account of the time it takes a particular mapping to reach its steady state convergence time value. However, this should be done after a careful study has been carried out on its effect on product quality.

REFERENCES

1. Dally, W. J., Balfour, J., Black-Shaffer, D., Chen, J., Harting, C. R., Parikh, V., Park, J., and Sheffield, D. (2008). Efficient Embedded Computing. *IEEE Computer*, July, pp 27-32.
2. Lee, E. A. and Messershmitt, D. G. (1987). Static Scheduling of Synchronous Data Flow Programs for Signal Processing. IEEE Transactions on Computers, Vol. C – 36, No. 1, January.
3. Ononiwu, G. C., Chukwudebe, G. A., Ndinechi, M. C., Okafor, E. N. C., Opara, F. K. (2013). Models for DSP applications on tiled Many-Core Architectures. Asian Journal of Natural and Applied Sciences, ISSN: 2186-8476, Vol 2. No. 1. Pp 70-81.
4. Thoziyoor, S., Ahn, J., Monchiero, M., Brockman, J., and Jouppi, N. (2008). A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies.In the Proceedings of the International symposium on Computer Architecture (ISCA).
5. Brooks, D., Tiwari, V., and Martonosi, M. (2000). Wattch: a framework for architectural-level power analysis and optimizations. In the Proceedings of the International symposium on Computer Architecture (ISCA), June.
6. Kahng, A., Li, B., Peh, L. –S., and Samadi, K. (2009). ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. DATE.
7. Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., Jouppi, N. P. (2009). McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. MICRO 2009.
8. Bengtsson, J., and Svensson, B. (2009). Manycore Performance Analysis using Timed Configuration Graphs. In the Proceedings of the international Symposium on Systems, Architectures, Modeling and Simulation (SAMOS IX 2009), Samos, Greece.
9. International Technology Roadmap for Semiconductors, "International technology roadmap for semiconductors—System drivers," 2007 [Online]. http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_System-Drivers.pdf