ISSN: 2186-8476, ISSN: 2186-8468 Print
Vol. 2. No. 1. March 2013

MODELS FOR DSP APPLICATIONS ON TILED MANY-CORE ARCHITECTURES

G. C. Ononiwu[1], G. A. Chukwudebe [2], M. C. Ndinechi [3], E. N. C. Okafor[4], F. K. Opara[5]

Department of Electrical/Electronic Engineering,
Federal University of Technology, Owerri, Imo State.
NIGERIA.
ononiwugordon@yahoo.co.uk

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. An energy model has been developed, making it possible to optimize energy usage within a target budget, while still meeting the set latency constraints. A means of returning feedback has also been presented. Subsequent work will concentrate on determining how the resultant tool can be employed in DSP application modeling and analysis.

Keywords: Many-core, Energy, Latency, Generic, Hardware, Ranking.

INTRODUCTION

Many-cores architectures are very important because they offer a useful alternative to the traditional hardware for implementing Digital Signal Processing (DSP) applications based on several tradeoffs. The tradeoffs include system flexibility, shorter time to market constraints, length of device life cycles, hardware capacity, design productivity, energy consumption and cost of implementation. Traditional hardware solutions like ASICS and FPGAs score very well in terms of capacity and energy consumption. However, they lag far behind programmable many-core solutions in terms of system flexibility, time to market, longer device life cycles and cost of implementation. These advantages are in part due to the nature of the hardware. However, improvements can be made in developer productivity. Improving the productivity of developers will result in a reduction in the time it takes to get the products to the market. This is a very important quality in today's fast moving consumer devices market.

For Many-core architectures, however, the increase in the number of cores has resulted in an increase in the complexity of developing applications. It is estimated that by the year 2022, the average chip will have 80 times the number of cores in a 2007 chip [1]. This necessitates an improvement in developer tools in order to have a more productive environment. A lot of effort has already been done in this area. However, the tools that are available are mostly proprietary in nature. This makes it difficult for product developers to carry out cross platform comparisons at the earliest stage of product development. Without such comparisons, there is difficulty in determining which commercially available hardware will best meet the required Quality of Service (QOS).

Previous work has concentrated on the functionality of the application with far less attention being paid to the non-functional requirements. Nevertheless, where possible, performance is the target. In this work, we are not only interested in the performance properties of an application, but argue that energy, another non – functional property, should be targeted by the system. The reason for this is not farfetched. Energy has always been a mainstream issue when discussing embedded systems. However, domains well outside the embedded systems industry now regard energy efficiency as a main stream quality issue. Almost all the recent hardware developments incorporate extensive energy saving features that can only be maximally utilized when targeted by the compiler.

This research work has focused on developing models and methods for energy estimation on generic tiled many-core processors. 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 DSP tasks that execute on it. A rank based method can then be used to rank the distinct mappings of an application to the processor on a scale based on their ability to meet with pre-specified timing constraints, and their relative estimated energy consumption levels.

We focus on using synchronous data flow (SDF) [2] 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. This work aims to model the dynamic nature of the processing that is involved, providing a means of studying the run time behavior of the application.

In this paper, we discuss the set of models that allow the tool to carry out a dynamic energy analysis of the application. Also included is a graph based intermediate representation (IR), which maps the modeled application to the target machine, and an abstract interpreter, that is used to provide feedback to the application developer (could also provide a feedback to an auto tuner). 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 up with the end-to-end latency requirement of the application.

Sheduling techniques for dataflow graphs on parallel processing platforms has been extensively researched in the last 30 years. Effort has either concentrated on mapping task graphs to recourses or maximizing throughput by iteratively adjusting the schedule or both.

Bokhari [3] developed an algorithm for mapping dataflow task graphs onto a linear array of processors while Sih and Lee [4] proposed an algorithm which is a single step heuristic that takes inter – processor communication overhead into account during clustering. This algorithm aimed at producing a feedback for iteratively tuning the schedule.

On the other hand, Banerjee et al. [5] developed a throughput maximizing scheme using a two step method in which the first step used an iterative scheduling algorithm to determine the tradeoffs between clustering and parallelism. In the next step, the granularity was determined through an iterative refinement technique.

Others have approached the problem by attempting to find a schedule that satisfies a timing or throughput constraint. Examples can be seen in the work of Choudhry et al. [6] and Aiken and Nicolau [7]. Both have decomposed the graph into serial and parallel sections and found optimal assignment of processors to these sections so that the response time is minimal for a given throughput constraint.

Similar to this work, Bengtsson and Svensson [8] [9] have used a two step strategy, which are independent of each other. The first step consists of clustering and the second step consists of scheduling the clusters on the processor. However, while they clearly target performance as a means of tuning the schedule, we have increased the scope of their work by focusing on energy as the basis for iteratively improving the schedule once a set timing constraint has been met.

In this paper, we present the models for mapping an application to a target processor, and the models for carrying out the analysis of this application. These models are presented in section 2. Section 3 presents the functions that compute the cost of implementing tasks on a target processor based on the parameters of the machine. In section 4, the graph based intermediate representation, and the feedback mechanism is presented. Section 5 concludes the paper.

IMPLEMENTATION

In this section we present the model set which consists of an application model which describes the processing requirements of the application, an energy model which computes the energy consumed in carrying out the task based on the particular many-core machine and a machine model that describes the memory and computational resources of the many-core processor. However, we first give brief notes on the target processor.

The Target Processor

The target processor will be an array – structured multiple instruction, multiple data (MIMD) type machine with homogeneous tightly coupled cores. We assume that it is implemented on a Complementary Metal Oxide Semiconductor (CMOS) circuit. We also assume that this processor is capable of performing dynamic speed and voltage scaling (DVS) on a per – core basis. This means that the cores will be modeled with a local clock that is timed with reference to the global system clock. Also, we only consider a distributed memory architecture where all the cores are in control of their own local memory space. This allows them to take advantage of the locality of data that is typical with the data flow domain.

The network of cores is implemented on a mesh structure which provides a decentralized but transparent communication scheme. The structure should implement message passing and should be warm hole routed. This makes it easier to predict communication timing and notice its effects on end to end latency constraints.

The Model Set

The resources of an abstract cored architecture can be described using two models; a Machine model and an Energy model. The Machine model describes the attributes of the machine in terms of what the machine can offer an application. On the other hand, the energy model describes the power resources of the machine. Both models make use of a distinct set of functions when calculating the cost of carrying out atomic tasks. While the machine model makes use of performance functions represented by M, the energy model makes use of a set of energy functions represented by E. Therefore, a many-core processor is modeled by giving values to the parameters of M and by defining the functions F(M) and E (M).

An application model completes the model set by providing a means of capturing the resource requirements of the application.

Application Model

The synchronous data flow (SDF) model of computation is used to model the application. It is made up of a network of actors (which are blocks of computation) and communication channels (which are the only means of communicating between actors). Actors fire as soon as there is enough data or tokens consumed from the communication channel, therefore, the whole process is data synchronized. For a more detailed description of SDFs and their firing properties, see [3].

Each actor is represented by a tuple $< r_p, r_m, R_s, R_r >$

ISSN: 2186-8476, ISSN: 2186-8468 Print
Vol. 2. No. 1. March 2013

Where

- r_p is the worst case execution time for the block of code in the actor.
- r_m is the requirement on local memory in words.
- $R_s = [R_{s1}, R_{s2}, R_{s3},……., R_{sn}]$

  It is a sequence where R_{si} is the number of words produced on channel i each firing.
- $R_r = [R_{r1}, R_{r2}, R_{r3},……., R_{rn}]$

It is a sequence where R_{rn} is the number of words received from channel j each firing.

Because we are not interested in making a cycle accurate simulation of the processing, we make use of worst case estimates on time and memory requirements.

Machine Model

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

$$M = < (x, y), p, b_g, g_w, g_r, o, s_o, s_l, c, h_l, r_l, r_o, s_f, f, w_l, I_{Leakage} > \qquad (1)$$

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.
- b_g is global memory bandwidth, in words per clock cycle.
- g_w is the penalty for global memory write, in words per clock cycle.
- g_r is the penalty for global memory read, in words per clock cycle.
- o is the software overhead for initiation of a network transfer, in clock cycles.
- s_o is core send occupancy, in clock cycles, when sending a message.
- s_l is the latency for a sent message to reach the network, in clock cycles.
- c is the bandwidth of each interconnection link, in words per clock cycle.
- h_l is network hop latency, in clock cycles.
- r_l is the latency from network to receiving core, in clock cycles.
- r_o is core receive occupancy, in clock cycles, when receiving a message.
- s_f stands for scaling factor, and is used to determine the speed of the core with reference to the global operating frequency of the machine. s_{fi} ε S_F where i corresponds to the number of the core as can be determined by the cores x, y coordinates.

- f denotes the maximum operating frequency at which a core can operate.
- w_l is the average wire length between cores.
- $I_{Leakage}$ is the leakage current per core.

Energy Model

In a processor, power is the rate of consumption of electrical energy and energy is the sum total of all the electrical energy consumed over the entire period.

$$P = W/T \tag{2}$$

$$E = P * T \tag{3}$$

E, P, W and T are the Energy, power, amount of work and time interval respectively.

In order to model the power consumed in the processor, we choose to approach the problem from two angles. First, model the power consumed by the cores and then separately provide a model of the power consumed by the interconnection network. A combination of the two will provide us with a model for the power consumed in the processor at a given point in time.

We estimate the power consumed by the entire many-core processor to be the sum of the dynamic power produced due to switching activity in the cores and also along the interconnection network when it is toggled, the short circuit current power which occurs during gate signal transitions, and the power that results from leakage current. Therefore

$$Power = P_{dyn} + P_{short} + P_{Leakage} \tag{4}$$

We can neglect the short circuit power because it is usually insignificant when compared to the rest.

Therefore,

$$Power \sim P_{dyn} + P_{Leakage} \tag{5}$$

Core Power Consumption

CMOS circuits dissipate power by charging the various load capacitances whenever they are switched. In one complete cycle, current flows from the source to charge the load capacitance and then flows from the charged load capacitance to the ground during discharge. Therefore, in one complete charge-discharge cycle, a total charge of:

$$Q = C_L V \tag{6}$$

This is the equivalent charge transferred from the source to the ground. Multiply this by the switching frequency to get the current used per second.

Therefore,

$$I = C_L V f \tag{7}$$

However, Electric power is given by the equation:

$$P = IV \tag{8}$$

Therefore, multiplying equation 7 by V, gives:

$$P = C_{Eff} V^2 f \tag{9}$$

Equation 9 gives the effective power dissipated by a CMOS device. Since most gates do not switch at every clock cycle, they are often accompanied by a factor, b, called the activity factor. Therefore, dynamic power dissipation can be rewritten as

$$P_{dyn} \approx bC_{Eff} V^2 f \quad (10)$$

Where b is an activity factor which relates to the rate of 0/1 transitions that occur within core, C_{Eff} is the effective cumulative capacitance, V is the base supply voltage and f is the clock frequency.

$$P_{Leakage} = VI_{Leakage} \quad (11)$$

Leakage power is also known as idle power or static power. It takes place when the processor is both in its idle and active states. It's as a result of the constant leakage of current from the transistors that make up the circuit even when the transistors are not switching.

Therefore,

$$\text{Power} \approx bC_{Eff} V^2 f + VI_{Leakage} \quad (12)$$

However, the ability to slow down the core for slack reclaiming has to be included in the model. This introduces the s_f machine parameter which is used to model the variability in the speed of cores vis-à-vis the global processor speed. Therefore, the equation becomes:

$$\text{Power} \approx (s_f)bC_{Eff} V^2 f + VI_{Leakage} \quad (13)$$

Where s_f is the scaling factor and is specific to a particular core.

Interconnection Network Power Consumption

J. Hu et al. [10] proposed a model for evaluating the power consumption of the interconnection network based on the concept of the bit energy metric (E_{bit}) which is defined as the average energy consumed when one bit of data is transported through the interconnection network, from one point A to another point B.

$$E^{AB}_{bit} = n_{hops} \text{ x } E_{Sbit} + (n_{hops} - 1) \text{ x } E_{Lbit} \quad (14)$$

Where E_{Sbit} and E_{Lbit} represent the energy consumed by the switch and the links between the cores.

This model was further developed by Wolkotte et al. [11], through their work, in determining the exact amount of energy consumed when a bit goes through a router and wires. This is given by,

$$E_{ps} = 0.98 \text{ x } N_{hops} + (0.39 + 0.12 \text{ x wire\_length}) \text{ x } (N_{hops} - 1) \quad (15)$$

for packet switched networks and

$$E_{cs} = 0.37 \text{ x } N_{hops} + (0.39 + 0.12 \text{ x wire\_length}) \text{ x } (N_{hops} - 1) \quad (16)$$

for circuit switched networks.

Where N_{hops} and wire_length correspond to the number of rounting turns, and the average wire length between the routers.

COST METHODOLOGY

In order to compute the cost of carrying out tasks, the models make use of two sets of functions that perform the actual computation of the costs. The machine model uses a set of performance functions, while the energy model uses a set of energy functions. This section presents both sets.

PERFORMANCE FUNCTIONS

F is a set of abstract functions describing the performance of computations, global memory transactions and local communication:

$F(M) = < t_p, t_s, t_r, t_c, t_{gw}, t_{gr} >$

Where

- t_p is a function evaluating the time to compute a list of instructions.
- t_s is a function evaluating the core occupancy when sending a data stream.
- t_r is a function evaluating the core occupancy when receiving a data stream.
- t_c is a function evaluating network propagation delay for a data stream.
- t_{gw} is a function evaluating the time for writing a stream to global memory.
- t_{gr} is a function evaluating the time for reading a stream from global memory.

The value for each of the performance functions can be derived from the machine parameters as follows:

Compute

The time required to process the computation of a list of instructions is given as:

$$t_p(r_p, p) = (s_f)[r_p/p] \qquad (17)$$

which is a function of the requested number of operations r_p and core processing power p. To calculate r_p, we count all instructions except those related to network send- and receive operations.

Send

The time required for a core to issue a network send operation is expressed as

$$t_s (R_s, o, s_o) = (s_f)[[R_s/\mathit{framesize}] \times o + R_s \times s_o] \qquad (18)$$

Send is a function of the requested amount of words to be sent, R_s, the software overhead o when initiating a network transfer, and send occupancy s_o. The framesize is a machine specific parameter.

Receive

The time required for a core to issue a network receive operation is expressed as:

$$t_r(R_r, o, r_o) = (s_f)[[R_r/\text{framesize}] \times o + R_r \times r_o] \qquad (19)$$

Network Propagation Time

This is expressed as

$$t_c (R_s, d, s_l, h_l, r_l) = s_l + d(x_s,y_s,x_d,y_d) \text{ x } h_l + n_{turns} + r_l \qquad (20)$$

Where s_l and r_l represent the injection and extraction latency respectively, while $d(x_s,y_s,x_d,y_d)$ and h_l represent the number of network hops and network hop latency respectively. $d(x_s,y_s,x_d,y_d)$ is determined from the source and destination coordinates as:

$$d = | x_s - x_d | + | y_s - y_d | \qquad (21)$$

Routing turns add an extra cycle which is captured as n_{turns} and is calculated using the source and destination coordinates.

Streamed Global Memory Read

This is the propagation time when streaming data from the global memory to a receiving core. It is expressed as

$$t_{gr}(r_l, d, h_l) = r_l + d(x_s,y_s,x_d,y_d) \text{ x } h_l + n_{turns} \qquad (22)$$

It is a function of the receiving core latency, r_l, the network hop distance, d, and the network hop latency, h_l.

Streamed Global Memory Write

This is the propagation time when streaming data to the global memory from a sending core. It is expressed as

$$t_{gw}(s_l, d, h_l) = s_l + d(x_s,y_s,x_d,y_d) \text{ x } h_l + n_{turns} \qquad (23)$$

It is a function of the send latency, s_l, the network hop distance, d, and the network hop latency, h_l.

ENERGY FUNCTIONS

With this in mind we can now start the process of computing the energy cost of events taking place in the many-core.

The complete set of E are described as follows:

$$E(M) = <e_p,e_s,e_r,e_c,e_{gw}e_{gr}> \qquad (24)$$

- e_p is a function that evaluates the energy consumed when a sequence of instructions is computed at a core.
- e_s is a function that evaluates the energy consumed by the core when it prepares to send a stream of data.
- e_r is a function that evaluates the energy consumed by a core as it receives a stream of data.
- e_c is a function that evaluates the energy consumed due to network propagation delays.
- e_{gw} is a function that evaluates the energy cost of writing a data stream to the global memory.
- e_{gr} is a function that evaluates the energy cost of reading a data stream from the global memory.

The value for each of the energy functions can be derived from the machine parameters as follows:

Compute

The energy required to process the computation of a list of instructions is sum of the total power consumed during the duration of the entire computation and this includes the leakage power. It is represented by the equation:

$$e_p(r_p,p,s_f,b,CEff,V,f) = [b \text{ x } C_{Eff}((s_f)^{-1}V^2)] \text{ x } t_p + VI_{Leakage}(t_p/f) \qquad (25)$$

t_p represents the number of clock cycles used to process the computation, s_f is the speed scaling factor, V represents the operating voltage, f represents the operating frequency, $p \varepsilon M$ represents the core processing power in operations, and b represents the activity factor which is always 1 when the system is active and 0 when it is not switching. $I_{Leakage}\, \varepsilon\, M$ is a measure of the average leakage current per core.

Send

The energy required for a core to issue a network send operation is expressed as:

$$e_s(s_f,V,f,t_s) = [bC_{Eff}\ ((s_f)^{-1}V^2f)\ x\ (t_s)] + [VI_{Leakage}]\ x\ (t_s/f) \qquad (26)$$

t_s represents the number of clock cycles used to issue a network send operation.

Receive

The energy required for a core to perform a network receive operation is expressed as:

$$e_r(s_f,V,f,t_r) = [bC_{Eff}\ ((s_f)^{-1}\ V^2f)]x\ t_r + [VI_{Leakage}]\ x\ [t_r/f] \qquad (27)$$

t_r represents the number of clock cycles used to receive a network message.

Network Propagation Energy

This is the energy required to move a message on the interconnection network, from one core to another. Equation 22 must be taken into account since the model is restricted to packet switched networks. Therefore,

e_c = (Total number of bits)(Bit Energy consumed at the switch + Bit Energy consumed on the wire link)+ energy leaked due to various latencies

$$(s_l + r_l + n_{turns}) \qquad (28)$$

Equation 25 will eventually take the form:

$$e_c(d, w_l, s_l, r_l) = \{R_s\}\{0.98\ x\ d(x_s,y_s,x_d,y_d) + [(\ 0.39 + 0.12\ x\ w_l\ )\ x\ [d(x_s,y_s,x_d,y_d) - 1]]\} + (s_l + r_l + n_{turns})VI \qquad (29)$$

where w_l, s_l and r_l are, the average wire length between cores, the send latency and the receive latency accordingly. d is arrived at from equation 6. R_s represents the size of the communication.

Streamed Global Memory Read

This is similar to e_c. It is the energy expended when streaming a message on the interconnection network, from the global memory to the core that is making the read request. The only difference comes in the fact that we neglect s_l because the data is streamed from the global memory and not from a core. Therefore, equation 29 will be reduced to:

$$e_{gr}(d, w_l, r_l) = \{R_r\}\{0.98\ x\ d(x_s,y_s,x_d,y_d) + [(\ 0.39 + 0.12\ x\ w_l\ )\ x\ [d(x_s,y_s,x_d,y_d) - 1]]\} + (r_l + n_{turns})\ VI \qquad (30)$$

Streamed Global Memory writes

This is similar to e_c and e_{gr}. It is the energy expended when streaming a message on the interconnection network, from the core that is requesting a write operation to the global memory location where the streams are going to. The only difference comes in the fact that we neglect r_l because the data is streamed to the global memory and not to a core. Therefore, equation 29 will be reduced to:

$$e_{gw}(d, w_l, s_l) = \{R_s\}\{0.98\ x\ d(x_s,y_s,x_d,y_d) + [(\ 0.39 + 0.12\ x\ w_l\ )\ x\ [d(x_s,y_s,x_d,y_d) - 1]]\} + (s_l + n_{turns})VI \qquad (31)$$

Next, the IR is discussed

ISSN: 2186-8476, ISSN: 2186-8468 Print
Vol. 2. No. 1. March 2013

INTERMEDIATE REPRESENTATION (IR) AND ABSTRACT INTERPRETATION

The IR is a graph data structure representing an SDF application, after it has been partitioned and mapped to a specific many-core target. We use the IR as input to an abstract interpreter for evaluating the dynamic behavior of the application when executed on the machine.

The IR $G_M^A(V, C)$ is a description of the application A that has been mapped to the virtual machine, M. V is the set of Vertices (cores or memory) and C represents the set of communication channels or edges. During scheduling, the each SDF sub-graph is assigned a core in M. After constructing the IR, each $v \varepsilon V$ and each $c \varepsilon C$ is assigned costs in terms time and also in terms of energy consumed. The costs are calculated from the parameters of M and the functions of F and E.

A sequence of abstract operations (receive, compute and send) are used to represent the actions of a core at the time it is fired. For each core that has a part of the application mapped to it, the set of activities that have taken place in them from the time the program was instantiated, is preserved in the vertex for that core in terms of a sequence of costs that relate to each activity. Such as:

$$S_t = t_{r1}, t_{r2}, t_{r3} \ldots\ldots\ldots\ldots\ldots t_{rn}, t_p, ts1, ts2, ts3 \ldots\ldots\ldots\ldots t_{sn} \quad (32)$$

For timing.

And in terms of energy:

$$S_e = e_{r1}, e_{r2}, e_{r3} \ldots\ldots\ldots\ldots\ldots e_{rn}, e_p, e_{s1}, e_{s2}, e_{s3} \ldots\ldots\ldots\ldots e_{sn} \quad (33)$$

Memory vertices are also assigned costs in terms of g_r and g_w to account for the cost of reading and writing to the global memory, respectively.

Let the source vertex of channel e be source (e). Then for each incoming edge of a vertex p, a receive operation is added with a cost given as:

$$t_r \mathcal{E} F(M) \quad (34)$$

and

$$e_r \mathcal{E} F(E) \quad (35)$$

Likewise, t_p and e_p are used for a compute operation to calculate the cost of the computation that takes place when an actor fires.

$$t_p \mathcal{E} F(M) \quad (36)$$

and

$$e_p \mathcal{E} F(E) \quad (37)$$

A send operation is added for each outgoing edge of a vertex. Let the sink vertex of channel e be sink (e). The cost of sending tokens on that channel is given as

$$t_s \mathcal{E} F(M) \quad (38)$$

and

$$e_s \mathcal{E} F(E) \quad (39)$$

The functions $t_c \mathcal{E} F(M)$ and $e_c \mathcal{E} F(E)$ signify the weight of the edge. They are both given as:

$$t_c \mathcal{E} F(M) \quad (40)$$

and

$$e_c \varepsilon F(E) \tag{41}$$

The weight of an edge is the same as the communication delay when sending a token between two cores that it connects, and the cost in energy terms of carrying out the communication task. The weight corresponds to the value of t_c.

Eventually, the IR will be a single data structure that links up all the vertices via edges, with the vertices storing the set of all the activities that have taken place in the core that it represents from time 0. Also, the edges that link the vertices represent the communication channels which connect the core and storing information relating to the bandwidth of the channel in form of the weight of the edge. The IR now serves as input to the abstract interpreter.

An abstract interpreter, implemented on the Ptolemy II modeling framework, is used to draw meaningful conclusions from the IR. It takes in the IR 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.

CONCLUSION

In the last decade, emerging parallel processors have come within the performance reach of ASICs. Effort should now be concentrated on developing adequate abstractions from the hardware that will improve on software design productivity. This is needed in order to take full advantage of the performance and energy reduction that such systems can achieve. Useful abstractions can come from a better programmer environment through the provision of domain specific languages, tools and simulation environments. This will reduce the time it takes to design new products and improve on quality by taking into account certain non – functional properties of a system.

This work has implemented models for DSP application development on tiled many-core processors. Three sets of models have been presented; an application model, a many – core machine model and an energy model. An Intermediate representation has been used to concretize the actions of these models. The abstract interpreter that interprets the IR has been designed to run on top of the Ptolemy II environment. It should be able to provide a developer with the ability to analyze successive mappings on a tiled processor array, ranking them according to specified end – to – end latency and energy constraints.

Future work will be carried out; using case studies to show how these models can be used to analyze DSP applications. Also, a ranking system will be developed for ranking the various ways of mapping of the application to the processor.

ISSN: 2186-8476, ISSN: 2186-8468 Print
Vol. 2. No. 1. March 2013

REFERENCES

[1] International Technology Roadmap for Semiconductors. (2007). International technology roadmap for semiconductors—System drivers, [Online].Available:http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_System-Drivers.pdf.

[2] Lee, E. A. & Messershmitt, D. G. (1987). Static Scheduling of Synchronous Data Flow Programs for Signal Processing, *IEEE Transactions on Computers,* C – 36(1).

[3] Bokhari, S. H. (1988). Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers.*

[4] Sih, G. C. & Lee, E. (1993). A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. on Parallel and Dist. Systems,* 4(2).

[5] Banerjee, S. et al., (1995). Macro pipelining based scheduling on high performance heterogeneous multiprocessor systems. *IEEE transactions on signal processing*.

[6] Choudhry et al. Optimal processor assignment for a class of pipelined computations, IEEE transactions on parallel and distributed systems. April 1994.

[7] Aiken. & Nicolau, A. (1988). Optimal loop parallelization. *In Proceedings of the ACM Conference on Programming language Design and Implementation.*

[8] Bengtsson, J. (2008). A set of Models for Manycore Performance Evaluation Through Abstract Interpretation of Timed Configuration Graphs, *School of IDE,* Tech. Rep. IDE0856.

[9] Bengtsson, J. & Svensson, B. (2006). A Configurable Framework for Stream Programming Exploration in Baseband Applications, In Proc. Of 1^{th} int'l Workshop on High – level Parallel Programming Models and Supportive Environments in conjuction with the Int'l Parallel and Distributed Processing Symp, (IPDPS 2006), Rhodes, Greece.

[10] Hu, J. & Marculescu, R. (2003). Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures, *Proc. IEEE DATE*, pp. 688–693.

[11] Wolkotte, P. T., Smit, G. J. M., Kavaldjiev, N., Becker, J. E. & Becker, J. (2005). Energy Model of Networks-on-Chip and a Bus. *In the Proceedings of the International Symposium on System-on-Chip*.