ISSN: 2186-8476, ISSN: 2186-8468 Print
Vol. 1. No. 2. June 2012

ENERGY AND PERFORMANCE MODELING OF DATAFLOW APPLICATIONS ON MANY-CORE ARCHITECTURES

Ononiwu, Gordon Chiagozie*

Department of Electrical and Electronic ,Engineering,
Federal University of Technology, PMB 1526, Owerri, Imo State,
NIGERIA
ononiwugordon@yahoo.co.uk

ABSTRACT

Many-core processors will provide system developers on embedded platforms with the best means of achieving system flexibility, shorter time to market, longer device life cycles and overcome the hardware productivity gaps of familiar hardwired and semi-hardwired implementations. However, the lack of portable tools for application development may hamper the rate of their adoption by industry. This work aims to contribute towards the solution by providing an abstraction from the many design constraints facing application developers. Models have been designed to improve our ability to iteratively map data flow applications to the target machine. Furthermore, an energy model has been introduced, making it possible to optimize energy usage within a target budget, while still meeting the set performance constraints.

Keywords: Many-core, Energy, Performance, Hardware

INTRODUCTION

As it stands today, a progression towards the many – core processing technology is now accepted as a means of overcoming the scalability issues facing multi – core technology. However, its use is still restricted to the signal processing domain, where there is a natural match. In the future, it is expected to be the dominant programmable architecture for high performance computing. For this to happen, an aggressive effort at providing tools for application and compiler development has to be undertaken. Multi-core technologies, even with their inherent scaling bottlenecks, are still the mainstream programmable processing technology because the paradigm is well understood and compiler technology is most developed. However, in a few years' time we are expected to reach the limit in the expected performance of multi-cores. Looking at the international technology roadmap, 2007 [1], it is clear that this may already be happening in some domains.

* Gordon Ononiwu., was a guest PhD student at the Center for Research on Embedded Systems (CERES), Halmstad University, Halmstad, Sweden. He is now with the Department of Electrical and Electronics Engineering, Federal University of Technology, Owerri, Imo state, Nigeria.(Mailing address: EEE Department, FUTO, Box 1526, Owerri, Imo State. e-mail: ononiwugordon@yahoo.co.uk).
This work was funded in part by the Swedish Institute (SI), Halmstad University and the Federal University of Technology, Owerri. Support also came from Jerker Bengsson and Bertil Svensson, both of Halmstad University, Sweden.

Leena and Luna International, **Oyama, Japan.**

This work argues for improved tools on many-core architectures through the continuous advancement in languages and compilers. Even in the signal processing domain, a lot of effort is still needed. One such effort targets the efficient allocation of parallel hardware resources through the provision of improved heuristics.

For real time systems, there is also the need to provide the compiler with performance feedbacks that will make it easier to develop applications that meet specified timing constraints, thereby improving developer productivity and reducing time to market constraints.

Real time systems provide us with a unique opportunity, because the correctness of the task depends on both the functional and non-functional properties of the application. Existing compilers are not adequate because they mostly target only the functional properties. When a feedback is obtained early in the development circle, adjustments can be made without wasting time needlessly.

We are not only interested in performance properties but argue that energy, another non – functional property, should also be targeted by the compiler. The reason for this is not farfetched. Energy has always been a mainstream issue when discussing embedded systems. However, the overbearing nature of the discussion now means that domains well outside the embedded systems industry now regard energy efficiency as a main stream issue. Almost all the recent hardware developments incorporate extensive energy saving features that can only be maximally utilized when targeted by the compiler.

This work provides a tool which is unique in the sense that it targets performance and energy, but is also a product of countless effort by researchers over the years that has resulted in better models of computation, languages and compiling technology for dataflow applications on multi – processing and chip multi – processing platforms. One such tool [2] has been enhanced by our work to take dynamic speed and power scaling into account since this is a feature that is common to recent hardware developments. We focus on using synchronous data flow (SDF) [3] models to describe the application, given the fact that SDFs are very suitable for modeling signal flows, especially since the schedule is determined in advance of the run time environment. We aim 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 make up the tool; the graph based intermediate representation (IR) which maps the modeled application to the target machine, the enhancements made to target processors with speed and voltage scaling capabilities and the nature of the abstract interpretation that is used to provide a feedback to the application developer (could also provide a feedback to an auto tuner). We also present the power model which gives the abstract interpreter the ability to provide energy estimates based a particular mapping on a per core basis.

It is important to note that the tool does not aim at providing a circle 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 that application.

RELATED WORK

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 [4] developed an algorithm for mapping dataflow task graphs onto a linear array of processors while Sih and Lee [8] 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] have used a two-step strategy which is 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 to include both performance and energy as the basis for iteratively improving the schedule.

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 and application for a deeper understanding of the assumptions that have been made in order to make this work manageable and realizable.

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. This makes it easier to predict communication timing and notice its effects on end to end latency constraints.

Leena and Luna International, **Oyama, Japan.**

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].

Just as in [2], each actor is represented by a tuple $< r_p, r_m, R_s, R_r >$

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.
- $R_s = [R_{s1}, R_{s2}, R_{s3}, \ldots\ldots, R_{sn}]$

It is a sequence where R_{si} is the number of words produced on channel i each firing.

- $R_r = [R_{r1}, R_{r2}, R_{r3}, \ldots\ldots, R_{rn}]$

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 machine model as described in [2] is given by;

$M = < (x, y), p, b_g, g_w, g_r, o, s_o, s_l, c, h_l, r_l, r_o >$

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.
- 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.

In order to model the concept of local timing for the cores, we have included more parameters;

- sf stands for scaling factor, and is used to determine the speed of the core with reference to the global operating frequency of the machine. Sf_i ε SF 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.

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:

$$t_p (r_p, p) = [r_p/p]$$

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

$$t_s (R_s, o, s_o) = [P_s/framesize] \times o + P_s \times s_o$$

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 however; we use a setting of 8.

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

$$t_r(R_r, o, r_o) = [R_r/framesize] \times o + R_r \times r_o$$

The receive overhead is calculated in a similar way as the send overhead, except that parameters of the receiving core replace the parameters of the sending core.

Network Propagation Time: This is expressed as

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

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 $|x_s - x_d| + |y_s - y_d|$. 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 the propagation time when streaming data from the global memory to a receiving core. It is expressed as

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

Streamed Global Memory Write: This the propagation time when streaming data to the global memory from a sending core. It is expressed as

$$t_{gw} = r_l + d(x_s,y_s,x_d,y_d) \times h_l + n_{turns}$$

ENERGY MODEL

Power is the rate at which a system performs its work and energy is the work performed over a period of time. 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.

$$P = W/T$$

$$E = P * T$$

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

$$Power = P_{dyn} + P_{short} + P_{Leakage}$$

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

Therefore,

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

Core Power Consumption

$$P_{dyn} \sim bC_{Eff} V^2 f$$

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.

$$P_{Leakage} = VI_{Leakage}$$

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,

$$\text{Power} \sim bC_{Eff} V^2 f + VI_{Leakage}$$

INTERCONNECTION NETWORK

J. Hu et al. [9] 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.

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

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. , through their work, in determining the exact amount of energy consumed when a bit goes through a router and wires. This is given by,

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

For packet switched networks and

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

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.

ENERGY FUNCTIONS

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

The resources of an abstract cored architecture can be described using three tuples, M, F and E. while M is the set of machine parameters that describe the resources of the machine, F is the set of abstract performance functions that can be derived from the machine parameters and E is the set of abstract energy functions describing the energy consumed by atomic operations of the machine as a function of M. 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).

The complete set of E are described as follows:

$$E(M) = \langle e_p, e_s, e_r, e_c, e_{gw} e_{gr} \rangle$$

- 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 given as:

$$e_p(r_p,p,sf,b, CEff,V,f) = [b \times C_{Eff} (sfV^2f) + VI_{Leakage}] \times (r_p/p)$$

Where sf is the speed scaling factor, V represents the operating voltage, f represents the core operating frequency, p ε M represents the core processing power, and b represents the activity factor which is always 1 when the system is active and 0 when it is not switching. $I_{Leakage}$ ε 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

$$e_s = [bC_{Eff} (sfV^2f) + VI_{Leakage}] \times t_s$$

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

$$e_r = [bC_{Eff} (sfV^2f) + VI_{Leakage}] \times t_r$$

Network Propagation Energy: This is expressed as

$$e_c = 0.98 \times d(x_s,y_s,x_d,y_d) + [(0.39 + 0.12 \times w_l) \times [d(x_s,y_s,x_d,y_d) - 1]]$$

where w_l is the average wire length between cores

INTERMEDIATE REPRESENTATION (IR)

The IR is a graph representing an SDF application graph, after it has been partitioned and mapped to a specific many-core target. We can use the IR as input to a code generator, in order to configure each core as well as the interconnection network and plan global memory usage. It is also ideal for studying optimizations that can be applied to the stream graph in later stages of the tool. We can then use the IR as input to an abstract interpreter for evaluating the dynamic behavior of the application when executed on the machine.

The IR graph $G_M^A(V, C)$ is a description of the application A that has been mapped to the abstract machine, M (See [2] for detailed description). V is the set of Vertices and C represents the set of communication channel. During scheduling, the each SDF sub-graph is assigned a core in M. After constructing the IR, each v ε V and each e ε E 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.

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

CONCLUSION

In the last four years, 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 only come with 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 stream application development on a class of many - core processors; Two dimensional processor arrays. Three sets of models have been developed; 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. An abstract interpreter 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 two – dimensional processor array, ranking them according to specified end – to – end latency and energy constraints.

REFERENCES

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

2. J. Bengtsson, "A set of Models for Manycore Performance Evaluation Through Abstract Interpretation of Timed Configuration Graphs," School of IDE, Tech. Rep. IDE0856, 2008.

3. E. A. Lee and D. G. Messershmitt, "Static Scheduling of Synchronous Data Flow Programs for Signal Processing", IEEE Transactions on Computers, Vol. C – 36, No. 1, January 1987.

4. S. H. Bokhari, Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers*, Jan. 1988.

5. S. Banerjee et al. Macro pipelining based scheduling on high performance heterogeneous multiprocessor systems. *IEEE transactions on signal processing,* June1995.

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

7. Aiken and A. Nicolau. Optimal loop parallelization. In *Proceedings of the*

ACM Conference on Programming language Design and Implementation, 1988.

8. J. and Svensson, B., "A Configurable Framework for Stream Programming Exploration in Baseband Applications," In Proc. Of 11th 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.

9. J. Hu and R. Marculescu, "Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures", *Proc. IEEE DATE,* pp. 688–693, 2003.