On-Chip Interconnect Network Communication Management for Multi-Core Design

He Zhou, Mariya Bhopalwala, and Janet Roveda

***Abstract*—Modern On-chip Multi-core design will continue Moore's law and facilitate platforms for wired and wireless communications. It has been predicted that the future computing platform will have a tightly integrated complex system that can process "big data" with swift speed and high quality. However, it is not clear how the current multi-core systems would react to large volume of data and how the data volume would impact the interconnect network design and architecture of the future computing platform. The goal of this paper is to raise these questions and provide some answers. In particular, this paper provides a series of cost models and a new optimization scheme "Interconnect Communication Cost Minimization" (ICCM) to manage tasks and their data. Task flows and their partitions are considered with data amount created and consumed. The consequent partitions are mapping virtually to the multi-core system through a data and task scheduling optimizer. Through experimental results, we demonstrated that an average of 50% reduction in the communication cost, an average of 23.1% of throughput improvement and 35% of dynamic power reduction.**

***Index Terms*—Multi-core, data-centric design, interconnect communication cost minimization, accuracy adaptive adder.**

I. Introduction

Modern on-chip multi-core design will continue Moore's law and facilitate platforms for wired and wireless communications with "big data". It has been predicted that these multi-core architectures will be the future computing platforms with swift speed and high quality. Previous success of IBM, Larrabee, and Intel just demonstrated 32 cores, 64 cores, and 80 cores examples. The recent success of Tilera on a 100 core system [1], [2] has just confirmed the future trend in exploring thousands of cores on a single die. Nevertheless, putting a lot of cores on a single chip is not simple. The most challenging part is not about how many cores we can pack on a single die. It is how to keep these cores being supplied with resources: these cores need to be powered and be supplied with data streams. Otherwise, even if we have thousands of cores on a single die, we can only activate a small portion of them.

A number of researchers [1]-[3] have identified that the success of today's Tilera lies in routing. By replacing long wires in multi-core system with routed networks, with distributing the gigantic 50-ported register file, 16-way ALU clump, and gigantic 50-ported mongo cache, the multi-core system now becomes a tiled structure (Fig. 2) [4]. While power hunger is still a potential challenge, the efficiently routed network among cores is the key reason that improves core to core communication and reduces energy per bit. To support fast communications among cores, large caches, wide bandwidth for memory associated with buses and other interconnection mechanisms are necessary (i.e. optical interconnects and switches) in multi-core designs [1], [3]. The management of these data communication related resources also becomes an important issue.

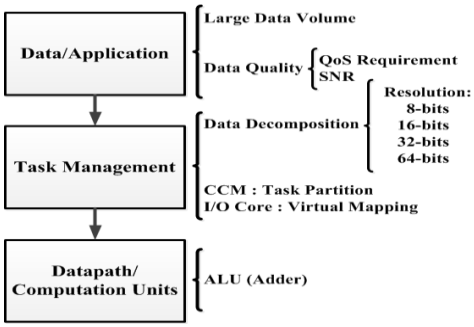


Fig. 1. Flow diagram of the proposed methodology.

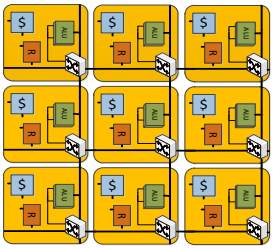


Fig. 2. 3×3 grid of tiled architecture (the similar figure can be found in [4]).

In this paper, we intend to discuss how interconnect communication design can effectively help manage data and its related tasks and resources. While a lot of publications discussed management of multi-core systems, it is not clear how the current multi-core systems would react to large volume of data and how the data volume would impact the design and architecture of the future computing platform. The goal of this paper is to raise these questions and provide some answers. In particular, this paper provides a series of cost models and a new optimization scheme "Interconnect Communication Cost Minimization" (ICCM) to manage tasks and their data. The previous works of task partition only worked on tasks, they did not consider the data volume and data quality in the tasks. Here, our new data-centric method considers task flows and their partitions with various quality levels of data created and consumed. Then it maps the consequent partitions virtually to the multi-core system through a data and task scheduling optimizer and core-index (CI) that monitors core performance and status. Fig. 1 shows a general flow of the

Manuscript received March 20, 2013; revised April 28, 2013.

He Zhou is with the University of Arizona, Tucson, AZ 85721 USA (e-mail: hezhou@email.arizona.edu).

Mariya Bhopalwala is with the University of Arizona, Tucson, AZ 85721 USA (e-mail: mariyab@email.arizona.edu).

Janet Roveda is with the University of Arizona, Tucson, AZ 85721 USA (e-mail: wml@ece.arizona.edu).

proposed methodology. Data or applications decide data volume and specify quality of data. Task management allocates both data and tasks depending on the required quality. For example, if the resolution requires 8 bits, the task management would allocate the data and tasks through task flow partitioning. The resolution requirement would pass down to ALU (i.e. adder) to perform 8-bit accuracy computation. At the ALU level, adders and other computation units are accuracy adaptive to save energy per bit. Note that "injection rate" as a key parameter in the existing NoC simulators may not be an explicit way to model the data volume. By definition, injection rate is the number of packets per cycle per node. The parameter shows the core's ability to deal with packets. Given the number of cycles executed for each node, we may find the number of packets the core has processed. On the other hand, the number of packets and flits in each application are not tightly related to the core's ability to handle the data.

When the data volume is high, we are concerned about the energy consumed for processing the data and data related tasks. Here we use data token to present the amount of data [5]. While the task flow with data token models the work load effectively, the real cost still depends on the system or platform that these tasks will be executed. Different platforms would lead to different size of buffers and interconnect bandwidth, so that the cost estimation would be very different. In this paper, we consider tile based multi-core system as in Fig. 2. Each tile can be considered as an independent CPU core, contains its own processor, memory cache, and switch with routers. Instead of bus connection between the tiles, this architecture uses five low-latency physical mesh networks called iMesh. These five networks are Static Network (STN), User Dynamic Network (UDN), I/O Dynamic Network (IDN), Memory Dynamic Network (MDN), and Tiled Dynamic Network (TDN) [6]. Each one of these five networks has link connection to five different directions: up, down, left, right, and one more link connected to the processor of the tile. For each direction, there are two unidirectional links, therefore the link can transmit message in both directions at the same time. In this paper, we care about the tile-to-tile communications, so we concentrate on Tiled Dynamic Network. The TDN on each tile can route operand communications over the entire scalar operand network.

The rest of the paper is organized as follows. Section II introduces data based quality of service metrics. These data properties are separated from the behavior of the systems. Task flows and their data (modeled as data token) are also discussed in this section. Section III discusses a new interconnect communication cost minimization (ICCM) scheme to manage tasks according to their data. Section IV describes virtual mapping of task to cores through Core Index and data QoS comparison. Section V presents our Accuracy Adaptive Adder to illustrate how to deal with data with different accuracy requirements. Finally, Section VI and VII demonstrate experimental results and conclude this paper.

II. Data Decomposition, Task Flow with Data Token

For a high quality video, the data get executed in multi-core system is large. Since the inter-core bandwidth, buffer size, as well as core performance has limitations due to design and performance walls, it is imperative to divide data into several smaller groups and process them parallel on different cores and components. There are several significant advantages of data decomposition. First, after decomposition, these pieces are parallel and independent with each other. Thus, each core can process these data slices independently at the same time. This methodology leads to short execution time and less energy consumption. Secondly, when large quantity of data is split into smaller chunks, the workload assigned to a core is reduced. This can further alleviate each core's workload and anneal the aging effect. Thirdly, when processing one huge quantity of data, there may be several different accuracy requirements. As illustrated by Fig. 3 [7], the original picture is split into four fragments. Fragment 1 and 4 represent the background of the image. So they only require a low accuracy if we do not have much interest in the background i.e. 16 bit resolution. Fragment 2 and 3 represent Lena's portrait in the image, hence we would want these fragments to have high resolution i.e. 64 bit. Thus, a suitable metric for an image QoS is the number of most significant bits (MSB) for the pixels. The difference in accuracy requirement also leads to using of control bits in the flit definition. As discussed above, the data is regrouped based on fragments/slices. Each slice produces data packets. And each packet consists of a set of flits. Fig. 4 represents a 4-flit packet with 32-bit for each flit [8], the accuracy control bits are assigned in the header flit. We assign two control bits which identify the accuracy requirements of the different image segments. For example, '00', '01', and '10' corresponds to 8 bits, 16 bits, and 32 bits data accuracy respectively. By assigning the control bits to the data set we can decide the accuracy mode of the ALUs.

Fig. 3. An example of data decomposition [7].

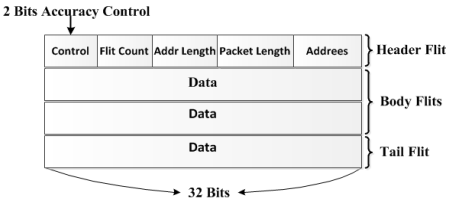


Fig. 4. A 4-bit packet with 32-bit flit.

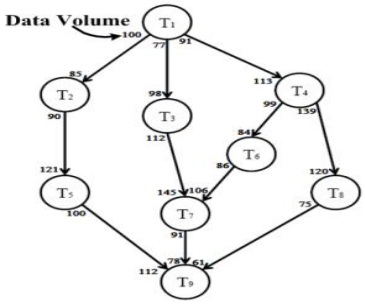


Fig. 5. A task flow example.

Workload is represented by task flows consisting of sets of tasks. And it is also well accepted that tasks from the same flow are highly related while there is not much dependency between tasks from different task flows [9]. We assume the same task flows for multi-core systems. To be more specific, Fig. 5 shows a task flow presented as Directed Acyclic Graph (DAG). $T = \{T_1, T_2, \ldots, T_9\}$ is a set of nodes each representing one task. The computation we apply on each task is addition, so these tasks are defined as fine grain and will be executed by multi-core system. An arc between a pair of nodes indicates precedence relationship of each task. For example, arc (1, 2) between task T_1 and T_2 tells that the execution of T_2 needs the result of T_1's execution. So, before T_2 can start, T_1 has to be first finished. Equation (1) models this precedence relationship as:

$$S_i + C_i^{exec} + C_{(i,j)}^{comm} \le S_j \tag{1}$$

where S_i and S_j are starting time of task T_i and T_j respectively, while C_i^{exex} represents the execution cost of T_i on a particular core, and $C_{(i,j)}^{comm}$ indicates the communication cost between task T_i and T_j.

Different from the traditional task flow, we emphasize the data amount in the task flow. In particular, we consider how much data is executed in the task. For the task which contains a large data amount, it means a large amount of data is consumed and produced. We adopt the data token concept from [5] to present data amount. Here, data token, the number on the arc at each node's input and output, represents the amount of data flits consumed (at the input of task) or produced (at the output of task) by each task [5]. Thus, for the task which contains a large amount of data, we have a large number of data token. The size of flit is dependent on the bus width of router. Once the platform is fixed, number of bits in one flit is fixed. Then the packet number and flit number relies on the data size.

In a task flow, because each task holds a different rate of data flit consumption and production, some data flits need to be stored in the buffer. Therefore, besides the cost to transmit data on the arc, there is another cost to buffer data on the arc. Both types of cost constitute the communication cost between two tasks [10]:

$$C_{comm} = C_{trans} + C_{buff} \tag{2}$$

Since we represent data amount by data tokens, the transmitting cost can be calculated from the division of data volume transmitted on the arc by the router bandwidth:

$$C_{trans} = a_t \times \frac{N_t \times S_f}{B} \tag{3}$$

where N_t is the number of tokens transmitted on the arc, S_f is the size of one flit, B is the bandwidth of router and a_t is the fitting coefficient.

The cost of buffering depends on the number of data tokens. In general, we consider two types of buffering costs: if a task produces more data tokens than its successor can consume, buffering cost is the cost to buffer these redundant data tokens on the arc; if a task produces less data tokens than its successor's needs, the buffering cost is the cost to load data tokens from the arc. No matter which case, the buffering cost can be calculated from the division of data volume buffered/loaded on the arc by the router's buffer speed:

$$C_{buff} = a_b \times \frac{N_b \times S_f}{V_b} \tag{4}$$

where N_b is the number of tokens buffered/loaded on the arc, S_f is the size of one flit, V_b is the buffer speed of router and a_b is the fitting coefficient.

III. Task Flow Partitioning and Task Scheduling Considering Interconnect Communication Cost

As we discussed earlier in this paper, a task flow is composed of several dependent tasks with specified precedence relationships. Here we introduce task level (L_i) to partition a task flow. Note that dividing a task flow into different levels could decide the distribution of tasks. For instance, the task flow in Fig. 5 can be divided into five levels:

L_1: T_1
L_2: T_2, T_3 and T_4
L_3: T_5, T_6 and T_8
L_4: T_7
L_5: T_9

Here tasks at the same level have the same precedence priority, and will be executed at the same cycle. We also define the critical level (L_c) as a level containing the most tasks. This is the level where most parallelism can occur. In this example, it is L_2 or L_3. Because we want to minimize execution time T_s of the whole task flow, we process tasks with the same precedence priority at the same time. In the current paper, we divide task flows into partitions according to the number of tasks in L_c. For instance, in the above example, the number of partitions will be three as we have three tasks in the critical level L_c.

Now it's time to regroup tasks in the other levels. It is important to pay attention that tasks directly connected by arc may have potential to create interconnect communication cost. If a pair of directly connected tasks is distributed to a same group, there is no inter-core communication cost. However if two directly connected tasks are distributed to two different groups, the inter-core communication cost can be computed by (2), (3), and (4). The goal of our work is to repartition tasks with a minimized total communication cost, for this purpose, we need a Global Connectivity Table (GCT). Table I shows the GCT of the task flow in Fig. 5. Where '1' represents direct connection between two tasks and '0' indicates there is no direct connection. The optimization model for the communication cost minimized scheduling problem can be described as follows:

minimize $\sum C_{comm}$ total communication cost
subject to $S_i + C_i^{exec} + C_{(i,j)}^{comm} \le S_j$ timing constraints
$p_i \le p_i^{upper}$ power constraints

for all cores with i as core index.

Algorithm 1: Interconnect Communication Cost Minimized Task Distribution

Input: A task flow to be distributed;
Data flits consumed and produced by each task are represented by number of data tokens
Output: The optimal distribution result;
Total Interconnect communication cost

1: { *L*1, *L*2, …, *Lm*} ⇐ Identify_Task_Level ( )
2: *Lc* ⇐ Find_Critical_Level ( )
3: Group number ⇐ Task number in *Lc*
4: **for** *Lc-k* = *Lc*-1 to *L*1/*Lc*+*k* = *Lc*+1 to *Lm*, *k* = *k*+1
5: **for** task *Ti* in *Lc-k*/*Lc*+*k*
6: **for** Task *Tj* has been distributed to a group already
7: **if** GCT (*i*, *j*) = 1
8: Comm_Cost*i, j* ⇐ Compute_Cost (*i*, *j*)
9: **end if**
10: **end for**
11: Comm_Cost*i* ⇐ sum (Comm_Cost*i, j*)
12: **end for**
13: Distribute tasks in *Lc-k*/*Lc+k* to groups in one case of combination, each group can get at most one task
14: **for** task *Ti* in *Lc-k*/*Lc+k*
15: **for** Task *Tj* distributed to the same group
16: **if** GCT (*i*, *j*) = 1
17: Inner_Cost*i, j* ⇐ Compute_Cost (*i*, *j*)
18: **end if**
19: **end for**
20: Inner_Cost*i* ⇐ sum (Inner_Cost*i, j*)
21: Comm_Cost*i* ⇐ Comm_Cost*i*- Inner_Cost*i*
22: **end for**
23: Comm_Cost ⇐ sum (Comm_Cost*i*)
24: Repeat 16-23 for all cases of combination
35: Comm_Cost ⇐ min (Comm_Cost)
26: **end for**
27: Comm_Cost_Total ⇐ sum (Comm_Cost)
28: **Return** Distribution Result, Comm_Cost_Total

Algorithm 1 presents the pseudocode for the interconnect communication cost minimized method. When a task flow is sent into the multi-core system, this algorithm first repartitions the task flow into different precedence levels ('Identify_Task_Level' operator). The 'Find_Critical_Level' operator searches for the level with the most tasks. In the pseudocode, tasks in the critical level are first divided into different groups. Then the algorithm regroups tasks in the previous level and next level, layer by layer. The 'Compute_Cost(*i*, *j*)' operator computes communication cost between two directly connected tasks T_i and T_j. Inner_Cost$_{i,j}$ indicates that T_i and T_j are assigned to the same core, so there is no inter-core communication cost among these two tasks. The algorithm goes through all combinations of distribution cases in each level and finds the distribution combination with minimum communication cost. This distribution pseudocode is repeated until all tasks in every level are repartitioned to each group.

TABLE I: GLOBAL INTERCONNECT CONNECTIVITY TABLE

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
T_1	-	1	1	1	0	0	0	0	0
T_2	1	-	0	0	1	0	0	0	0
T_3	1	0	-	0	0	0	1	0	0
T_4	1	0	0	-	0	1	0	1	0
T_5	0	1	0	0	-	0	0	0	1
T_6	0	0	0	1	0	-	1	0	0
T_7	0	0	1	0	0	1	-	0	1
T_8	0	0	0	1	0	0	0	-	1
T_9	0	0	0	0	1	0	1	1	-

After the repartition of a task flow, these groups will be assigned to the tiled architecture, each tile handles one group. The TDN router on each tile is responsible for routing the operand communication between tiles through the whole network [4]. Fig. 6 shows scheduling result of the task flow in Fig. 5. Target task flow is repartitioned into 3 groups and assigned to Tile A, B, and C. The solid arrows between each tile represent the operand communication over scalar operand network.

IV. I/O CORE CONTROLLED TASK SCHEDULING

A concept of competitive index (CI) has been established to evaluate the performance ability of each core in the multi-core system [11]. CI reflects the performance and operation conditions of the device. The core with a high CI level would be assigned with more tasks to work at a high utilization level; conversely the core with a low CI level would be assigned with fewer tasks to anneal the aging effect.

In the tiled architecture, each tile is a single core with heterogonous supporting components. Although these cores are homogeneous cores, after working for a while, they tend to develop varying operation conditions and performance reliabilities. Hence, when scheduling tasks with data into the network, we need to consider each tile's CI value. The proposed data-centric method in the previous sections handles cores in a virtual manner. Only when the data with required QoS matches CI, we assign the tasks to physical cores. The task management core, also called 'I/O core', manages the connectivity of all the tiles.

I/O Core has access to all of the tiles. It can access the real-time operation conditions and performance reliabilities of each tile and compute each tile's current CI value. When a task flow comes into the multi-core system the core management module analyzes the task flow and repartitions it. Then it schedules the regrouped tasks into the tiles for execution according to each tile's CI value. Each tile can operate the Self-Evolving Algorithm [11] to schedule its operating voltage and frequency to achieve lower power consumption with higher reliability. The I/O core makes connections between cores according to core management module. The error rate is a function of supply voltage and operating frequency. In Data-Centric design, when the I/O Core schedules tasks to tiles, it will assign task with high accuracy requirement to a tile with high CI value, to make sure there is a sufficient space for scheduling core's operating condition without violating the error rate limitation. The task with low accuracy requirement will be assigned to relatively low CI tile, working at a low operation condition.

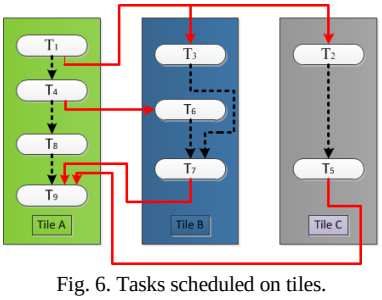


Fig. 6. Tasks scheduled on tiles.

V. Accuracy Adaptive Arithmatic Logic Unit

In this section, we present an Accuracy Adaptive Adder (AAA) to illustrate how to deal with data with different accuracy requirements. Previous adders in [12]-[14] are designed to give reduced accuracy but improved power consumption. These adders are application specific. They can be used in applications which require reduced accuracy. There are certain applications which cannot compromise on accuracy. Thus the need arises for an adder which can compute in both accurate and inaccurate mode. The adder in [15] has error detection and correction circuit and the accuracy of the adder can change during runtime. Our motivation is to design a circuit in which we can have the flexibility to choose between variant accuracy levels before the addition is performed. The block diagram shown in Fig. 7 depicts the general working of a 32-bit AAA. For the 32-bit adder two bits are assigned to the control bits. The control bits at the input of the Logic Circuit for Disabling Gates (LCDG) decides the accuracy level required for the computation for a given set of input data.

Fig. 8 shows our proposed accuracy adaptive adder for a two input 1-bit adder. This design incorporates both the inaccurate [12] and the accurate two input 1-bit adder on the same circuit. The majority of power consumed by a combinational circuit is because of dynamic switching. Our rationale behind this design is to disable the gates which are not required in the inaccurate mode. In this implementation we use a single control bit to decide the accuracy mode of the adder. There are two levels of accuracy. For high accuracy the control bit is set to 1 and for low accuracy the control bit is set to 0. The truth table for the design is shown in Table II. In Table II, X represents the control bit. A, B are the input bits. S1, S0 are the output bits. Blocks marked in color are the inaccurate results. The LCDG consists of two, 2:1 multiplexers, one AND gate and one OR gate. The gates comprising the LCDG are marked in color. The EXOR gate which is labeled as XOR-2 in Fig. 8 and AND-3 can be disabled to reduce the accuracy of the circuit [12]. We achieve this by multiplexing one of its inputs with the control bit labeled X in Fig. 8. The control bit depends on the accuracy requirement of the application. By disabling these gates, the number of gates used in the circuit is dropped from 14 to 9, based on the assumption that each EXOR gate comprises of 4 NAND gates.

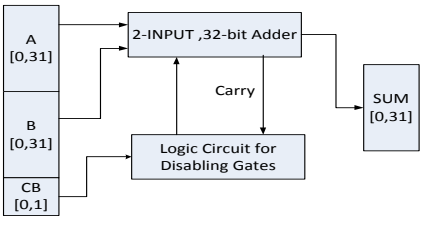


Fig. 7. Accuracy adaptive adder

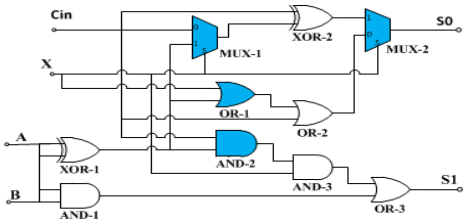


Fig. 8. Proposed AAA- two input 1-bit architecture.

TABLE II: Truth Table

X	A	B	Cin	S1	S0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	**1**	**0**	**0**	**0**	**1**
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

VI. Experimental Results

Experimental results of the proposed Interconnect Communication Cost Minimization methodology and Data Decomposition methodology are presented in this section. We generate all task graphs by profiling realistic applications from several benchmark suites, for example MediaBench, MiBench and NetBench. While the task flow with data token, models the work load effectively, the real cost still depends on the system or platform that these tasks will be executed. Different platforms would lead to different size of buffers and interconnect bandwidth, so that the cost estimation would be very different. In this paper, we consider tile based multi-core system. By replacing the long wires in multi-core system with routed networks, distributing the gigantic 50-portd register file, 16-way ALU clump, and gigantic 50-portd mongo cache, the multi-core system now becomes like a tiled structure (Fig. 2) [4]. Each tile can be considered as an independent CPU core, contains its own processor, memory and switch with routers. The TDN router on each tile can route operand communications over the entire scalar operand network.

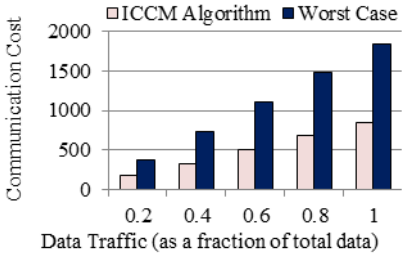


Fig. 9. Comparison of communication cost between ICCM Algorithm and the worst case scenario.

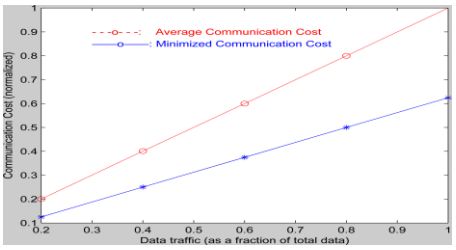


Fig. 10. Comparison between minimized communication cost and average communication cost.

To demonstrate the algorithm's performance in different data traffic, we first generate a task flow with a full data

traffic, then reduce it by 20%, 40%, 60% and 80%. Fig. 9 shows the total inter-core communication cost at different levels of data traffic. 'ICCM' represents task scheduling results by minimized communication cost algorithm and 'Worst case' considers task scheduling results in the worst communication cost scenario.

The comparison of minimized and average communication cost (normalized) is illustrated in Fig. 10. The results show that by applying the algorithm, multi-core system achieves a 37% inter-core communication cost reduction on average.

As we known, when total data amount contained by a particular task flow is low, the total time for a multi-core system to process this flow is mostly depend on the execution time of each task in each core. However, as the total data amount increasing, impact of the inter-core communication time becomes more and more significant. When total data amount contained by a particular task flow reaches a high level, the inter-core communication time could dominate the total processing time. Fig. 11 illustrates throughput improvement after using the proposed methodology. We generate a task flow with a full traffic of data level that has execution time which is the same as the inter-core communication time. A graceful improvement of throughput after applying ICCM can be observed from Fig. 11. As the data traffic increases, we can see an increasing throughput improvement. When the data traffic reaches 1 (a full data traffic), a significant difference of 23.1% can be observed between the two scenarios.

The power constraints of the cores can be reduced further by employing the AAA unit. Depending on the requirement of the application the proposed AAA can switch its accuracy mode. In the inaccurate mode, the dynamic power consumption can be reduced up to 35% based on the assumption that each logic gate consumes equal power. Fig. 12 shows the variation in the number of gates with respect to the accuracy mode selected and the number of output bits of the adder. We have considered two accuracy modes, but there is future scope of increasing the accuracy mode depending on the value assigned to the control bits in the header.

Fig. 11. Throughput improvement after applying ICCM

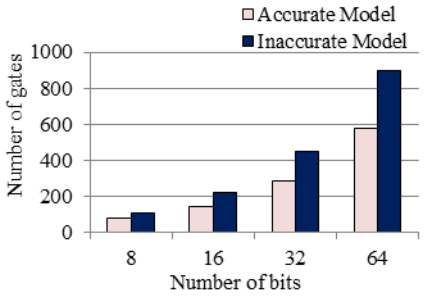


Fig. 12. Gate optimization in AAA.

VII. Conclusion

This paper proposes a new framework for multi-core system considering Data-Centric design. This new methodology starts from the task flow structure and tiled architecture, provides a new task scheduling model based on a minimized communication cost algorithm and virtual mapping design. Experimental results demonstrate that this new methodology reduces the inter-core communication cost while scheduling task flow to multi-core system and the data decomposition method achieves trade-offs between power reduction and graceful accuracy degradation.

References

[1] C. Demerjian. (October 29, 2009). A look at a 100-core Tilera Core Gx. *Efficiency, Microprocessor and Servers*. [Online]. Available: http://semiaccurate.com/2009/10/29/look-100-core-tilera-gx/

[2] R. Schooler, *Tile Processors: Many-Core for Embedded and Cloud Computing*, MIT, Cambridge, MA, 2011.

[3] R. Joshi. (March 26, 2011). Data-Centric Architecture: A Model for the Era of Big Data. *Design and Architecture.* [Online]. Available: http://www.drdobbs.com/architecture-and-design/229301018

[4] A. Agarwal, *Tiled Multicore Processors: The Four Stages of Reality*, MIT and Tilera, Cambridge, MA.

[5] E. A. Lee, and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, issue. 9, pp. 1235-1245, 1987.

[6] P. Griffin, H. Hoffmann, L. Bao *et al.*, "On-Chip Interconnection Architecture of the Tile Processor," *Micro of the IEEE*, vol. 27 , no. 5, pp. 15-31, 2007.

[7] Wikipedia. File:Lenna.png. [Online]. Available: http://en.wikipedia.org/wiki/File:Lenna.png/

[8] P. P. Pande, H. Zhu, A. Ganguly, and C. Grecu, "Crosstalk-aware Energy Reduction in NoC Communication Fabrics," in *Proc. SOC, IEEE International Conference*, 2006, pp. 225-228.

[9] H. EI-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*, Prentice Hall, 1994.

[10] J. Sun, R. Lysecky, K. Shankar, A. Kodi, A. Louri, and J. M. Wang, "Workload capacity considering NBTI degradation in multi-core systems," in *Proc. ASPDAC*, 2010, pp. 450-455.

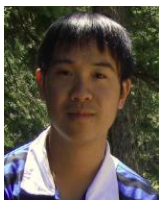
[11] J. Sun, R. Zheng, J. Velamala, Y. Cao, R. Lysecky, K. Shankar, and J. M. Wang, "A Self-Evolving Design Methodology for Power Efficient Multi-core Systems," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010, pp. 264-268.

[12] D. Shin and S. K. Gupta, "A Re-design Technique For Data Path Modules in Error Tolerant Applications," in *Proc. Asian Test Symp*, 2008.

[13] S. L. Lu, "Speeding Up Processing with Approximation Circuits," *IEEE Computer*, vol. 37, no. 3, pp. 281-290, 2004.

[14] N. Zhu, W. Goh, K. Yeo, W. Zhang, and Z. Kong, " Design Of Low-Power High Speed Truncation-Error-Tolerant Adder and its Application in Digital Signal Processing," *IEEE Trans. on VLSI Systems*, vol. 18, no. 8, pp. 1225-1229, 2010.

[15] A. B. Khang and S. Kang, "Accuracy-Configurable Adder for Approximate Arithmetic Design," in *Proc. DAC*, 2012, pp. 820-825.

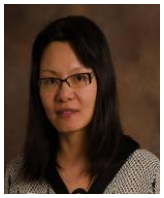


He Zhou was born in Chengdu, Sichuan, China on February 2, 1987. He got his B.S. degree in electrical engineering at the University of Electronic Science and Technology of China, Chengdu, Sichuan China, in 2009. He got his M.S. degree in electrical engineering at the State University of New York at Buffalo, Buffalo, NY USA, in 2011. He is pursuing his Ph.D. degree right now at the University of Arizona in electrcial and computer engineering field. He is working as research assistant and teaching assistant during the Ph.D. study. His current research is focused on data intensive multi-core design. Mr. Zhou has one paper accepted by the 2013 International Conference on Electronic Engineering and Computer Science, and it will be published by IERI Procedia (ISSN: 2212-6678).

Mariya Bhopalwala was born in Mumbai, India 05/18/1984. She got her B.S. degree in electrical engineering at the University of Mumbai, India in 2007.

She is currently pursuing M.S. degree in electrical engineering at the University Of Arizona. Currently working as a teaching assistant with the electrical and computer engineering department and as a research assistant in the VLSI Lab.

Ms. Bhopalwala interests in high speed data computation and multi core design.



Janet M. Wang-Roveda received a B.S. degree in computer science from The East China Institute in 1991, M.S., and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley in 1998 and 2000, respectively.

She is currently an associate professor in electrical and computer engineering department at the University of Arizona. Her primary research interests focus on robust VLSI circuit design, biomedical instrument design, Smart grid, VLSI circuit modeling and analysis, and low power multi-core system design.

Dr. Roveda was a recipient of the NSF career award and the PEACASE award in 2005 and 2006, respectively. She received the best paper award in ISQED 2010 as well as best paper nominations in ASPDAC 2010, ICCAD 2007, and ISQED 2005. She is the recipient of the 2008 R. Newton Graduate Research Project Award from DAC, and the 2007 USS University of Arizona Outstanding Achievement Award.