Combined Throughput and Energy Optimization for Synthesis of Streaming Applications on Multi-Core Architectures

Po-Kuan Huang Matin Hashemi Soheil Ghiasi

Department of Electrical and Computer Engineering
University of California, Davis
{pohuang, hashemi, soheil}@ece.ucdavis.edu

Abstract

We present a methodology for synthesizing streaming applications, modeled as task graphs, on multi-core architectures. We develop a task graph extraction and characterization framework that accurately determines the structure, computation and communication characteristics of application task graph using its specification in C. Furthermore, we develop a provably effective algorithm that jointly balances the workload assigned to each core, and minimizes inter-core communication traffic. Consequently, both throughput and energy dissipation of implemented applications, that are of prime significance in target application domain, are simultaneously optimized. Experiment results show that our method improves both throughput and energy efficiency of streaming applications significantly.

1. Introduction

An important class of applications, referred to as streaming applications, have to process virtually-infinite steady streams of input data. Example arise in many different application domains ranging from multimedia codecs in embedded applications to packet classification in network switches. These application domains impose increasingly-stringent throughput and energy constraints, especially in portable consumer electronics marketplace.

In this paper, we present a methodology for efficient synthesis of streaming applications for pipelined execution on multi-core architectures. We develop a simulation framework through which, application task graphs are extracted and characterized for task latency, task energy dissipation and inter-task communication traffic. The simulation framework is cycle-accurate and hence, the captured information are very precise at system-level. Furthermore, we develop a provably-efficient partitioning algorithm that jointly optimizes the computation load assigned to cores and inter-core communication traffic.

To illustrate the concept of our work, we discuss its impact on throughput and energy dissipation of an application using a simple example. Figure 1 shows an example task graph. The motivation of our design is to improve the throughput of the system by settling for pipelined execution of streaming application. Once we can construct the task graph for the application and its control and data dependencies are only in one direction, then we can efficiently pipeline the input data stream and communication channel between two processors.

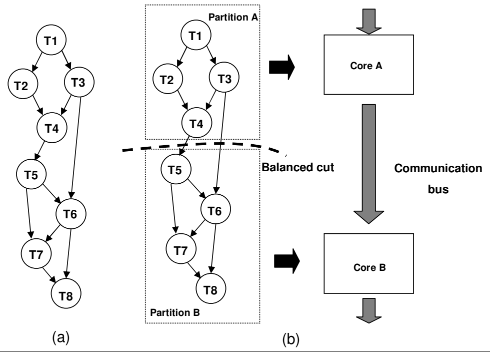

Figure 1. Pipelined execution of an example application

A balanced cut separates the application graph into two partitions, which can be executed on different processing units of the pipelined high throughput system. Since there are only one-direction data/control dependencies from the up-partition to the down-partition, we can pipeline the input data to the proposed system architecture. According to our experiment result, our method improved throughput by 19.87% and 18.15%, respectively. By trading performance to energy saving with voltage scaling mechanism, our method enhances energy saving over the min-communication algorithm and energy aware scheduling by 27.64% and 7.51%, while delivering the same throughput.

In the following sections, we will introduce how our framework constructs the direct acyclic task graph for the application at first. Secondly, our optimization technique will be briefly introduced. Then our proposed cascade multi-processor architecture will be illustrated. Finally, the experiment environment and result will be presented.

2. Task Graph Construction

2.1 Task Graph Model

In this section, We present a task graph model for the streaming application. Our task graph can capture all necessary information of the streaming application such as cycle-accurate latency number for each task, byte-accurate inter-task communication amount, and data/control dependency for each task.

Figure 2 illustrates our task graph model. We adopt the widely used task graph model to represent the computation and communication involved in an application. Formally, an application is repre-

LCTES'07 June 13–16, 2007, San Diego, California, USA.
Copyright © 2007 ACM 978-1-59593-632-5/07/0006. . . $5.00

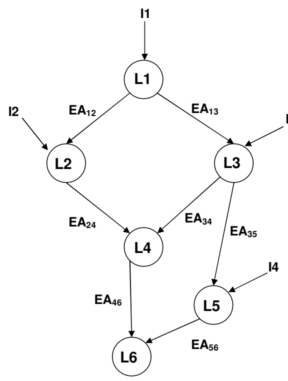

Figure 2. Task Graph Model

sented as a directed-acyclic graph (DAG) G. Vertices of G denote the tasks or constituting computations of the application, and edges of G represent inter-task communication. Each task can be fired (executed on a processing resource) if all of its input data are available. We assume that input data from the environment are ready at the beginning of the execution cycle.

Let L_i denote the latency of task i in cycles. We use the terms latency, delay, computation load or workload of a node interchangeably. Also, we use EA_{ij} to refer to the amount of data that goes from task i to task j over edge e_{ij} in bytes. I_i represents the input data received by i in bytes when executing task i.

Task graphs are typically used to model realtime applications for which, latency is the primary design concern. Note that we are using the model for streaming applications that demand high throughput, and can tolerate large latencies.

2.2 Task Graph Extraction Framework

We use a simulation based framework to extract the task graph from the application source code. Our simulation framework is based on the Intel Xscale simulator, Xtrem[1]. At first, we mark the application source code to different sections and insert the labels to the source code as section boundaries. Each section means a task node in the task graph. Our simulator can then measure and record the executed cycles for each section. In general, we don not assign different sections to one loop statement to avoid cyclic graph.

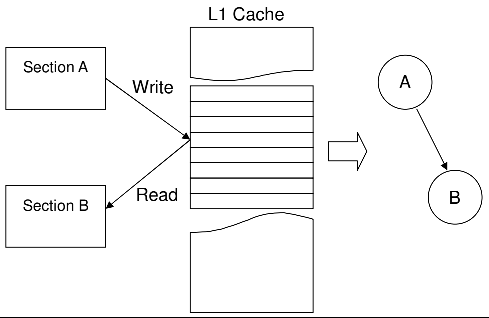

Figure 3. Cache Monitor Mechanism

Control and data dependency can be captured by monitoring the data cache and memory of the processor. We built a data cache and memory monitoring mechanism in our simulator. This mechanism monitors the load/store activities of each physical memory address of data cache and memory. As shown in the figure 3, once a task section B loads a physical memory address which was lastly updated by task A, then task B has dependency on task A. By using this mechanism, the edges in our task graph can be easily constructed.

We also capture the communication volume for edges of the task graph. We monitor the amount of memory traffic for each memory load and store in bytes. Therefore, the communication amount from task i to j can be estimated by aggregating the data amount loaded by task j from the physical addresses lastly updated by task i.

Our extraction framework reflects the parallelism of the original application implementation while it does not intend to exploit any other parallelism from the application. If the extracted task graph is cyclic, we merge the cyclic portion of the graph to one task node. According to our experiment, most of the streaming application kernel are highly parallel. Therefore our extraction framework is suitable for constructing the task graph for streaming application.

3. Combined Throughput and Energy Aware Partitioning

We propose a task-graph bi-partitioning algorithm that jointly considers the maximum computation load assigned to a core, and inter-core communication traffic. From a theoretical point of view, our algorithm is optimal in minimizing the ratio of the communication between the two partitions over the size of the smaller partition for planar directed-acyclic graphs (DAG). Note that to minimize the ratio, inter-partition communication has to be reduced while size of the smaller partition has to be enlarged, i.e., the computation load of the two partitions should be fairly balanced. Details of our algorithm and its correctness proof are due to page limitation.

4. High Throughput Energy Efficient System Architecture

4.1 System Configuration

Figure 4 illustrates our proposed high throughput energy efficient system. It has three pipeline stages: computation stage 1, data transmission stage, and computation stage 2. We use 32-bit computation cores with the frequency/voltage scaling mechanism for each computation stage. A voltage scaling(VS) regulator is used to set the operating frequency and voltage for the computation core. It can perform the frequency/voltage scaling statically.

By trading off the performance, quadratic energy saving can be obtained to achieve high energy efficiency. The frequency/voltage scaling mechanism will be discussed in the following subsection. There is a 32-bit data bus connecting two processing unit and its operating frequency is 100MHz. The behavior of our proposed architecture is described as following. Computation stage 1 execute the up-partitioned tasks of our task graph model. It receives the input data from the system input(SI) buffers.

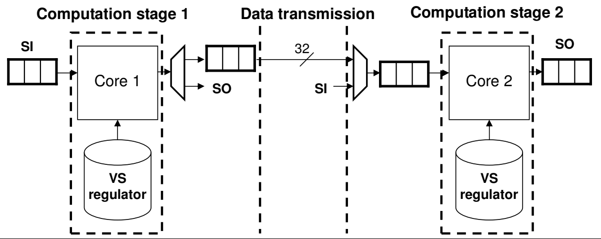

Figure 4. High throughput energy efficient system architecture

After processing the input data, it writes the result to either output buffers of the stage 1 or the system output(SO). As the output buffer of the stage 1 is becoming full, it starts to send the stored data to the computation stage 2 though the data bus. Then computation stage 2 will start to execute the down-partitioned tasks of our task graph model and send the final result to the system output. Our proposed architecture enhances the system performance by using pipelining techniques. The pipeline period of our proposed system is determined by the maximum value of the computation stage 1 latency, data transmission time, and computation stage 2 latency.

Since the enhancement of the system performance, our proposed architecture can finish the whole task in a shorter period of time compared to the traditional system. Thus it is energy efficient even without the voltage scaling technology. While the frequency/voltage scaling technology provides higher energy saving when the performance demanding is lower than the normal situation. We have built a simulator for our proposed architecture. It can simulate the performance and energy consumption of the application. In the energy simulation framework, we add the voltage scaling mechanism to it and consider the energy consumption for processing unit, data transmission line, and buffers.

5. Experimental Results

We briefly describe our experiment framework in follow. In order to construct the task graph, we do task labeling to the source code for task assignment at first. Then, after cross-compiling the labeled source code, we use our task graph extraction simulator to extract the task graph from the Xscale binary. The cycle-accurate latency and byte-accurate inter-task communication amount can be acquired at this step. If our task graph has cyclic or non-planar part, planarization and acyclic pass will be applied to make the task graph appropriate for our optimization technique.

After forming the appropriate task graph, we apply our partitioning algorithm to find a cut in the planar DAG. This balanced cut separates the application graph into two sections: up-partition and down-partition. The up-partition and down-partition will be executed on different processing units of the pipelined high throughput system. Since there are only one-direction data/control dependencies from the up-partition to the down-partition, we can pipeline the input data to the proposed system architecture. Once the computation capability exceeds the computation requirement, voltage scaling mechanism can scale down the frequency and supply voltage of the processor in order to save energy.

In order to verify the effectiveness of our proposed technique, we compare the throughput and energy consumption of our design to the traditional double-core architecture with two scheduling algorithm: minimal communication scheduling[2] and energy minimization scheduling[5]. We select five well-known streaming application from ALPBench[4] and Mediabench[3] as our testbench.

Figure 5 reports the experiment results for system performance. According to our experiment results, the throughput of our hardware/software co-design scheme outperforms the min-communication scheduling and energy minimization scheduling by 19.87% and 18.15% respectively in average. Since our partitioning algorithm cut the application into two balanced partitions, our pipelined double-core architecture can be fully utilized. While the traditional scheduling algorithm can be significantly limited by structure of the DAG. Therefore, the utilization of the traditional double-core architecture is lower than our proposed architecture.

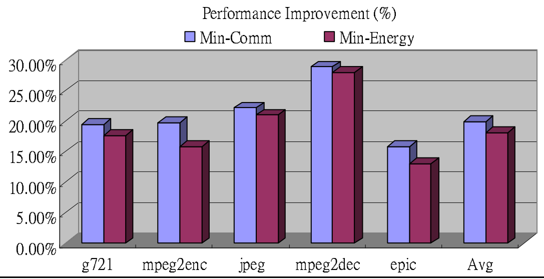

Figure 5. Performance Improvement

Figure 6 shows the results of simulating the energy consumption. Since the proposed scheme has a higher performance, the frequencies/voltages of the two processor units are scaled down to match the performance of other approaches. However, we save more energy because of the voltage scaling mechanism. Our experiment results show that this design outperform the min-communication algorithm and energy aware scheduling by 27.64% and 7.51% respectively in average.

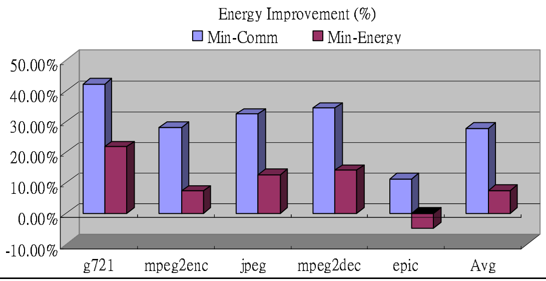

Figure 6. Energy Improvement

According to the experiment result, our proposed scheme can achieve higher performance for all of the testbench. Traditional competitors are designed to minimize the inter-task communication amount and system energy consumption respectively. Although our inter-partition communication is greater than the summation of total inter-task communication of the min-communication scheduling, the performance of our pipeline architecture is not affected significantly since the pipeline period is determined by the maximal value of the stage 1 latency, data transmission time, and stage 2 latency.

6. Conclusion

We present a task graph extraction and partitioning flow for enhancing throughput of streaming applications running on a pipeline of processors. Our main contribution includes the development of the task graph extraction framework, software optimization technique for min-quotient partitioning problem, and specific system architecture design achieving high throughput and energy efficiency and its performance/energy simulator.

Simulation result shows that our design consistently outperform the cooperation of the traditional double-core processor and scheduling algorithm in both system throughput and energy consumption. Future work include extension to generic multiprocessor architectures, and validation of our simulation results on real hardware.

References

[1] G. Contreras, M. Martonosi, J. Peng, R. Ju, and G. Lueh. Xtrem: a power simulator for the intel xscale core. *ACM SIGPLAN Notices*, 39(7):115–125, July 2004.

[2] C. M. Krishna and K. G. Shin. *Real-time Systems*. WCB/McGraw-Hill, 1997.

[3] C. Lee, M. Potkonjak, and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*, pages 34–41, 1997.

[4] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The alpbench benchmark suite for complex multimedia applications. In *Proceedings of the IEEE International Symposium on Workload Characterization*, 2005.

[5] G. Varatkar and R. Marculescu. Communication-aware task scheduling and voltage selection for total systems energy minimization. In *Proc. Intl. Conf. on Computer-Aided Design (ICCAD)*, November 2003.