Parallelization Strategy for CELL TV

Motohiro Takayama, Ryuji Sakai

Parallelization Strategy for CELL TV

Motohiro Takayama, Ryuji Sakai

Toshiba Corporation

Abstract. Consumer electronics devices are moving forward to utilize multi-core processors. We have developed a series of unique applications for TV based on Cell Broadband Engine™ (*Cell/B.E.*)[1]. This paper introduces such applications realized by the capability of the multi-core processor, and shares the strategy we took to exploit its potential.

1 Introduction

CELL TV ™ [CT] is a real world application utilizing a multi-core processor in the consumer electronics industry. It has unique media processing applications such as simultaneous playback of 8 channels, and 8 channel recording named *Time Shift Machine*. These applications are realized by the processing power and high bandwidth of the heterogeneous multi-core processor Cell/B.E.

In order to provide users with these attractive features, we have created a framework for developing media processing applications, and a parallel programming model for exploiting the parallelism in modules of the applications. We combined the framework with the programming model in a nested fashion. At the coarse level, the modules of an application are scheduled by the framework. The framework manages the concurrency and keeps the resource criteria. To parallelize the module that requires more than single core, the programming model is used. It eases the parallelization and gains the scalability. We also vectorized the serial portion of the module by SIMD instruction to exploit data parallelism.

2 Applications

The computation power and high bandwidth of Cell/B.E. makes the multiple media processing work simultaneously. CELL TV plays 8 [2] channels simultaneously and composite into one display, and records every content of 8 channels within the last 26 hours. With simultaneous playback of 8 channels, users can see all broadcasted channels at once. It replaces the channel zapping. Users just

[1] Cell Broadband Engine and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

[2] The TV supports *8* channels because there are 8 channels for digital terrestrial television in Japan.

select the channel they would like to watch, instead of pressing the next/previous channel button continuously. Time Shift Machine is a feature that records whole contents of every channel within the last 26 hours. That throws the idea "forget to watch" away. Users can watch any contents of any channels In addition, users can watch compute intensive YouTube HD video while recording all channels. This example shows the application is well parallelized to supply both required processing power and high bandwidth.

Those applications are comprised of several media processing modules running on SPEs. Example of those modules are audio/video codec (MPEG-2, H.264/AVC, AAC, etc), stream demultiplexer, and image enhancement filters (Super Resolution [SR], noise reduction, etc). Those modules are connected to compose a single functionality of a complex application.

Fig. 1 shows the connection of modules to play a audio/video stream as an example. Though a module may require more processing power than a single core offers, application developers do not have to care about it. This modularity simplifies complex application development. Fig. 2 depicts how the simultaneous playback of 2 streams works. It is obvious that the multiple stream playback is a simple extension to the single stream playback case.

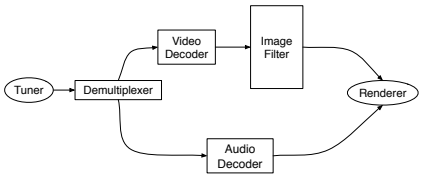

Fig. 1. Stream Decoding

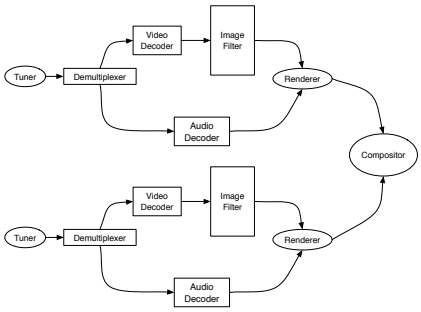

Fig. 2. Multi-Stream Decoding

3 Parallelization Strategies

To make the applications run in real-time, we took a three-level parallelization strategy: inter/inner module parallelization, and SIMD. Fig. 3 describes the overview of the strategy.

The inter/inner module parallelization strategy is analogous to the separation of the OS and the thread library. In ordinary PC, the OS schedules the process to provide the users a view that applications are running concurrently. On the other hand, the thread library has the responsibility to make the application run in parallel, and provides the platform portability.

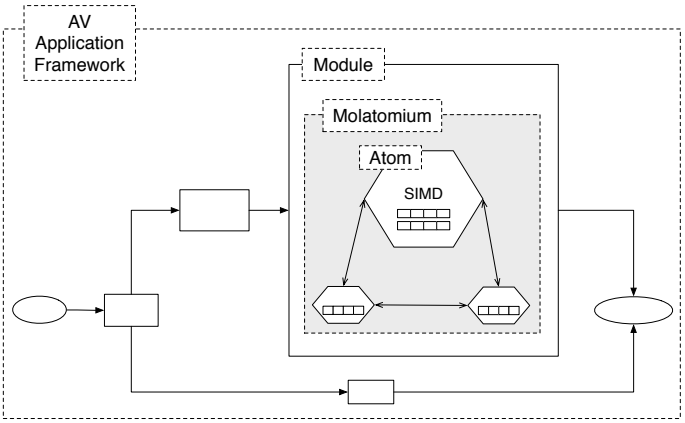

Fig. 3. Hierarchical Parallelization Strategies

3.1 Inter Module Parallelization

AV Application framework [SM] has a responsibility for concurrency among the modules. The framework provides the resource reservation facility and real-time task scheduling. The connection of modules is defined in an XML file with resource attributes. The attributes specify such resources as how long the module uses the processing core, how many cores the module uses, how many memories are necessary for I/O, etc. The framework harmonizes the modules running concurrently on SPEs with satisfying real-time criteria. It is just like a real-time OS for SPEs.

3.2 Inner Module Parallelization and SIMD

Although the framework manages the concurrency and schedules resources appropriately, there still remains a challenge in parallelizing compute intensive modules to achieve the required performance.

We have created a programming model named *Molatomium* [MT] to ease the parallelization. Programming in Molatomium is composed of two part: *Mol* and *Atom*. Mol is a coordinate language that describes the parallelism among Atoms. The code written in Mol is compiled into portable byte code and executed by virtual machines residing in SPEs. Mol is suitable for writing a parallel algorithm in task-parallel manner. On the other hand, Atom is a platform native code written in C/C++ like a kernel in CUDA [CU] and OpenCL [CL]. It is a parallel execution unit of Molatomium, and is optimized with platform dependent facility like SIMD instructions.

Although media processing applications are well suitable for using SIMD type data parallelization, some algorithms like Super Resolution do not really fit into this category. Molatomium can be applied to such case because it can not only describe data parallelism in Atom but also express task parallelism in Mol.

4 Discussion

The parallelization strategy for CELL TV proved the hierarchical approach works well on the Cell/B.E. architecture, which has a programmable memory hierarchy between the local storage and the main memory. It suggests that the hierarchy parallelization strategy would be effective in the coming NUCA (Non Uniform Cache Architecture) platform as well as the processor core count increases.

Though the modular design of the application framework provides the independency and increases the reusability, the task assignment attributes used by the scheduler are static. For instance, while the calculation time the module given is suitable for one content, other contents may require less computation time. We are working on improving Molatomium capability so that it can describe a whole application. Though it lacks the ability to assert real-time criteria so far, It could improve core utilization by its dynamic load balancing. In contrast to AV Application framework which is designed especially for the requirements in developing Cell/B.E. platform, Molatomium provides platform portability. In order to expand the applications created for CELL TV to other platforms, the whole application would be described by Molatomium.

Though the framework and the model work fine, we still tend to get lost in debugging bandwidth issue. That issue will increase in the near future as the many-core processors come to main stream. To solve this, the hardware and the development tool should provide the way to optimize the bandwidth consumption with ease.

5 Conclusion

CELL TV is our latest challenge to apply the multi-core processor to the consumer electronics device. In order to meet the required performance, we parallelized the applications using three-level parallelization strategy: AV application framework for concurrency, Molatomium for parallelizing the modules, and SIMD for exploiting the data parallelism.

References

[CT] CELL TV. `http://tacp.toshiba.com/promopage/celltvbk2A.html`.

[SM] Seiji M, et.al.: A Real-Time Software Platform for the Cell Processor. IEEE Micro. 25. 5. (2005) 20–29

[MT] Motohiro T, et.al.: Molatomium: Parallel Programming Model in Practice. HotPar '10. (to appear)

[SR] Takashi, I, et.al.: Reconstruction-based Super-resolution Using Self-congruency of Images. IEICE technical report. Image engineering. 107. 380. (2007) 135–140

[CL] Aaftab Munshi.: OpenCL: Parallel computing on the GPU and CPU. ACM SIGGRAPH, Tutorial. (2008)

[CU] NVIDIA.: NVIDIA CUDA Compute Unified Device Architecture Programming Guide. `http://www.nvidia.com/cuda`. (2007)