Prototyping Methodology with Motion Estimation Algorithm

Jinglin Zhang[1], Jian Shang[1], and Cong Bai[2(✉)]

[1] School of Atmospheric Science,
Nanjing University of Information Science and Technology, Nanjing, China
jinglin.zhang@nuist.edu.cn
[2] The College of Computer Science, Zhejiang University of Technology,
Hangzhou, China
congbai@zjut.edu.cn

Abstract. With CPU, GPU and other hardware accelerators, heterogeneous systems can increase the computing performance in many domains of general purpose computing. Open Computing Language (OpenCL) is the first open and free standard for heterogeneous computing on multi hardware platforms. In this paper, a parallelized *Full Search* Motion Estimation (FSME) approach exploits the parallelism available in OpenCL-supported devices and algorithm. Different from existing GPU-based ME approach, the proposed approach is implemented on the heterogeneous computing system which contains CPU and GPU. In the meantime, we propose the prototyping framework directly generates the executable code for target hardware from the high level description of applications, and balances the workload distribution in the heterogeneous system. It greatly reduces the development period of parallel programming and easily access the parallel computing without concentrating on the complex kernel code.

1 Introduction

General-Purpose computing on Graphics Processing Units (GPGPU) is the technique of using a GPU, which typically handles computation only for computer graphics, to perform computation across a variety of applications traditionally handled by the CPU. Compute Unified Device Architecture (CUDA) is a parallel computing platform and programming model created by NVIDIA and implemented by the GPUs that they produce. Open Computing Language (OpenCL) is a framework executing programs on heterogeneous platforms consisting of CPUs, GPUs and other dedicated processors. It is also a good candidate in comparison with the CUDA approach specifically developed for GPU platforms of NVIDIA.

Generally speaking, programming with GPU is one complex, error-prone and time-consuming procedure comparing with sequential dataflow and programming. For computer vision algorithms, there are various programming environments and languages of different platforms. So it is very hard to make one rapid

E. Chen et al. (Eds.): PCM 2016, Part I, LNCS 9916, pp. 338–344, 2016.
DOI: 10.1007/978-3-319-48890-5_33

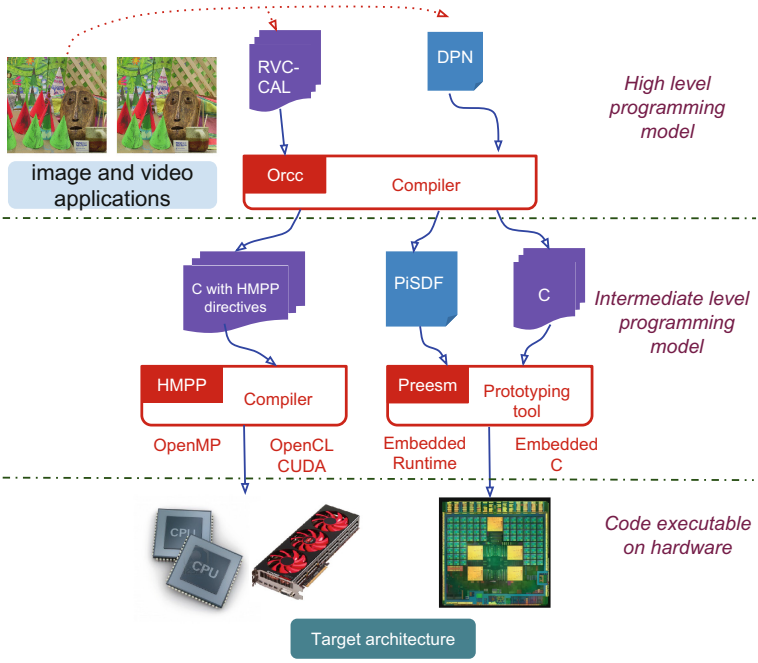

Fig. 1. The flowchart of proposed prototyping framework

development on various hardware accelerators like GPU. In general, there are two ways to perform the code porting from sequential code to the code executable on GPU. One is to manually write the kernel code of CUDA or OpenCL, which redesigns and compiles the kernel code with the specified compiler. Another way is to automatically generate code executable for hardware accelerators from the high-level description of application with the aid of the prototyping framework. Proposed prototyping framework consists of such three tools: Open Reconfigurable Video Coding Compiler (Orcc) [1], Parallel and Real-time Embedded Executives Scheduling Method (Preesm) [2], and Hybrid Multicore Parallel Programming (HMPP) [3] as shown in Fig. 1. As the high level programming model, Orcc contains an Reconfigurable Video Coding(RVC)-CAL textual editor, a compilation infrastructure, a simulator and a debugger which is proposed by IETR of INSA-Rennes. Lots of works have been proposed with Orcc and many backends are supported in Orcc like *C, C++, VHDL, HMPP* and so on [6–8,11]. As the intermediate level programming model, Preesm offers a fast prototyping tool for parallel implementations used in many applications like LTE RACH-PD algorithm. HMPP directive-based programming model is such a tool that offers a powerful syntax to efficiently offload computations on hardware accelerators and keeps the original C/C++ or Fortran codes [3–5,9,10].

In this paper, we evaluate the prototyping framework with the previous *Full Search* ME method. We not only implement the ME method on the

heterogeneous system with one CPU and one GPU, but also find the balance to distribute the workload in heterogeneous computing system. With the prototyping framework, we can generate code automatically and evaluate the performance rapidly.

2 Full Search Motion Estimation Algorithm

Full Search motion estimation is to search the best candidate of macroblock (MB) in the reference frame for the original macroblock in the current frame. The detailed illustration is introduced in our previous research work [4]. The following two subsection with Pseudo code: Algorithms 1 and 2 simply summarize the procedure of the proposed ME algorithm, which is divided into two OpenCL kernels. One is SADs computation; the other is SADs comparison for the best SAD candidate.

SAD Computation. When an OpenCL program invokes a kernel, N work-groups are enumerated and distributed as thread blocks to the multiprocessors with available Compute Units of CPU and GPU. In *kernel_compute*, all pixels of current MB are transferred into local memory (*local*[256]) by the 256 work-items in one work-group. One novelty of this paper is as follows. Until all these work-items in the same work-group reach the synchronous point using *barrier()* function, all the 256 work-items continue transferring the 2304 pixels of search region concerned of reference image into local memory (*local_ref*[2304]). This differentiates our approach from previous approaches of [12,13] and obtains better time efficiency (speed-up). In their work, current MB is stored in local memory, but the search region of reference frame still locates in the global memory, which results in inevitable re-fetching from global memory with performance loss. At the end, we adopt *Full Search* strategies to calculate the 1024 SAD candidates in local memory, in order to avoid re-fetching from the global memory as presented in Algorithm 1. All these 1024 SAD candidates are transferred back to global memory (*cost*[1024]) for SAD comparison.

SAD Comparison. In *kernel_compare*, we search the best candidate with the minimum SAD from *cost*[1024] using 256 work-items as presented in Algorithm 2. First, we transfer the *cost*[1024] into the local memory. Then, each work-item compares 4 candidates with a *stride* to find the minimum value. We employ the parallel reduction method [14] which adopts x times iterations ($2^x = 256, x = 8$) to find the candidate with the minimum SAD value from the remaining 256 candidates, also to obtain the final MV.

2.1 Experiments Discussion

We evaluate the performance of the proposed ME algorithm with manual OpenCL kernels in such hardware HASEE environment: Intel I7 2630qm

Algorithm 1. *kernel_compute()*

input : Current and reference frames
output: SAD costs candidates with offset in the x and y axis
Initialize the local memory space for macroblocks and search window;
for $n = 0; n <$ *number of macroblocks/number of CUs*$; n++$ **do**
  for $m = 0; m < 4; m++$ **do**
    256 work-items in one work-group calculate 256 SAD candidates simultaneously;
  end
end
return (SAD, MV)

Algorithm 2. *kernel_compare()*

input : SAD costs candidates with offset in the x and y axis
output: motion vector with the minimum SAD cost
Initialize the local memory for 1024 costs candidates of each blocks;
Each work-items compares 4 SAD candidates and return the minimum;
for $n = 0; n < \log_2 256; n++$ **do**
  Parallel reduction for the minimum SAD, half number of work-items compare the adjacent data and return the minimum data to next iteration;
end
return MV with the minimum SAD

(2.8 GHz), NVIDIA GT540m. We compare the performance of GPU-based FSME implementation and available state-of-the-art fast ME algorithms. To the best of our knowledge, there are two criterions of evaluation: *time efficiency, and matching accuracy*. Our experimental results mainly focus on PSNR.

The PSNR is defined as:

$$psnr = 10 \cdot log10(width \times height \times 255 \times 255/sse) \quad (1)$$

$$sse = \sum_{n=0}^{sum} \sum_{i=0}^{blocksize} \sum_{j=0}^{blocksize} (current_{frame}(i,j) - ref_{frame}(i+dx, j+dy))^2 \quad (2)$$

where sse is the error sum of square, *sum* is the total number of macroblocks in one frame, and (dx, dy) is the calculated motion vector. Proposed approach has faster speed and better accuracy than the state-of-the-art fast ME methods.

3 Heterogeneous Parallel Computing with OpenCL

Besides of manually writing the kernel code of OpenCL, we target to parallel computing with the aid of prototyping framework. Based on our previous research on motion estimation, we describe the proposed motion estimation approach with one high level description (RVC-CAL). There are two branches in our

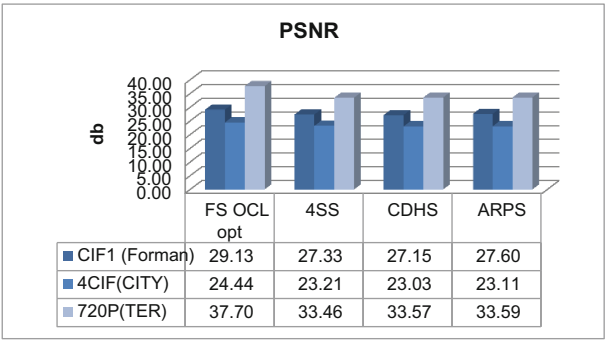

Fig. 2. PSNR comparison of specified state-of-the-art fast ME algorithms

Algorithm 3. *HMPP_Transformation()*

```
1 #pragma hmpp motion_estimation codelet, target=CUDA;
2 Definition of function motion_estimation();
3 ...;
4 Main(argc, argv);
5 #pragma hmpp motion_estimation callsite;
6 Function call of motion_estimation();
```

rapid prototyping framework, one branch is $RVC-CAL \rightarrow Orcc \rightarrow HMPP \rightarrow GPU(OpenCL/CUDA)$, and another one is $RVC-CAL \rightarrow Orcc \rightarrow Preesm \rightarrow DSP/ARM(EmbeddedC)$ as shown in Fig. 1: rapid prototyping framework. In this paper, we choose the first branch to generate and verify C/OpenCL/CUDA implementation on heterogeneous platforms such as multi-core CPU and GPU platforms respectively. With the HMPP-backend of Orcc, we obtain the C code with HMPP directives from the high level description. Then the HMPP compiler will automatically generate the OpenCL/CUDA kernel code. Our prototyping methodology framework can greatly simplify the procedure of implementation target to hardware devices (Fig. 2).

In the aforementioned discussion, there are paired directives of HMPP: *Codelet* and *Callsite*. Algorithm 3 presents the pseudo-code of default HMPP Transformation. Using simple paired directives, HMPP can replace the complex procedure of manually writing the CUDA/OpenCL kernel code. As shown in the above sample code, the *codelet* is the routine implementation for hardware accelerator and the *callsite* is the routine invocation of ME function on hardware. When we compile the sample code with HMPP, we can obtain the .cu or .cl kernel of proposed method which targets to GPUs. Instead of manually designing the CUDA/OpenCL kernel, HMPP greatly reduces the development period with GPU and CPU device.

The Reconfigurable Video Coding (RVC) [15] defines a set of standard coding techniques called Functional Units (FUs). A FU is described with a portable,

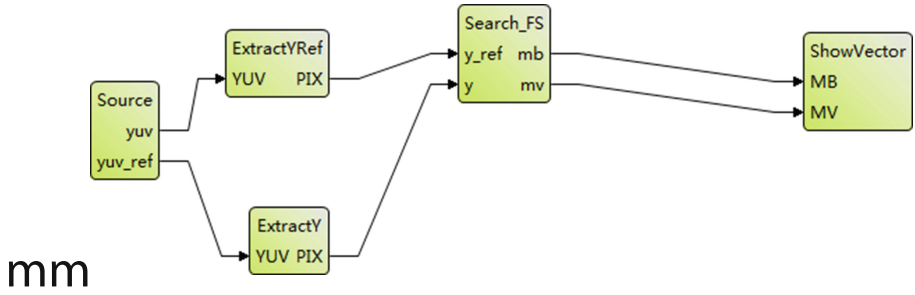

Fig. 3. The dataflow diagram of proposed ME approach

platform-independent language called RVC-CAL. It is one high level language of description which is designed to describe the reconfigure video coding standard. Now RVC-CAL is not only used in video coding, but also in some image and video processing algorithms, like motion estimation and stereo matching [4]. RVC defines a XML-based format called FU Network Language (FNL) that is used for the description of networks, also named the XML Dataflow Format (XDF). The Model of Computation defines the behavior of *Full Search* ME as a dataflow graph shown in Fig. 3.

In the block diagram of the motion estimation, the block "source" indicates the function of *load_frame*, a FU that loads video frames; the block "ExtractYRef" and "ExtactY" indicate the function of *extractY*, a FU that extracts Y channel from the current and reference frames of YUV format video sequence; the block "Search_FS" indicates the function of *fullSearchME*, a FU that does the *Full Search* motion estimation; the block "ShowVector" indicates the function of *display*, a FU that shows the calculated motion vectors. With the high level description of applications, the prototyping framework can directly and rapidly generate the target code like OpenCL/CUDA for heterogeneous systems, which greatly reduces the development period of parallel programming and easily accesses the parallel computing without concentrating on the complex kernel code.

4 Conclusion

We introduce one prototyping framework to implement the proposed ME method, and evaluate the prototyping methodology. Experimental results show that, our implementation has better performance than other GPU-based FSME implementations. One basic method is introduced to find the balance of workload on the heterogeneous parallel system with OpenCL. Additionally, we have found the accurate method to distribute the workload in video applications based on the heterogeneous system. It is also the first prototyping methodology generating target code for OpenCL-supported device from the high level description (different with other code generator like OpenACC), which greatly reduces the development period of parallel programming and easily accesses the parallel computing without concentrating on the complex kernel code.

Acknowledgements. This work was carried out with the Scientific Research Foundation (s8113055001) of Nanjing University of Information Science & Technology, Scientific Research Foundation (BK20150931) of JiangSu Province and Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the second phase). The work of Cong Bai is funded by Natural Science Foundation of China under Grant No. 61502424, 61402415, U1509207 and 61325019, Zhejiang Provincial Natural Science Foundation of China under Grant No. LY15F020028, LY15F030014, LY16F020033 and Zhejiang University of Technology under Grant No. 2014XZ006. The authors would like to thank the anonymous reviewers and the associate editor.

References

1. IETR, Orcc. http://orcc.sourceforge.net/
2. IETR, Preesm. http://preesm.sourceforge.net/website/
3. CAPS, Hybrid multicore parallel programming (HMPP). http://www.caps-entreprise.com/technology/hmpp/
4. Zhang, J., Nezan, J.-F., Cousin, J.-G.: Implementation of stereo matching using a high level compiler for parallel computing acceleration. In: 27th Image and Vision Computing New Zealand, pp. 279–283 (2012)
5. Grauer-Gray, S., Xu, L., Searles, R., Ayalasomayajula, S., Cavazos, J.: Auto-tuning a high-level language targeted to GPU codes. In: Proceedings of Innovative Parallel Computing, pp. 1–10 (2012)
6. Gorin, J., Wipliez, M., Prêteux, F., Raulet, M.: LLVM-based and scalable MPEG-RVC decoder. J. Real-Time Image Process **6**(1), 59–70 (2011)
7. Gu, R., Janneck, J.W., Bhattacharyya, S.S., Raulet, M., Wipliez, M., Plishker, W.: Exploring the concurrency of an MPEG RVC decoder based on dataflow program analysis. IEEE Trans. Circuits Syst. Video Technol. **19**(11), 1646–1657 (2009)
8. Zhang, J., Bai, C., Nezan, J.F.: Joint motion model for local stereo video-matching method. Opt. Eng. **54**(12), 123108.1–123108.10 (2015)
9. Wang, M., Hong, R., Yuan, X.-T., Yan, S., Chua, T.-S.: Movie2Comics: towards a lively video content presentation. IEEE Trans. Multimedia **14**(3), 858–870 (2012)
10. Wang, M., Hong, R., Li, G., Zha, Z.-J., Yan, S., Chua, T.-S.: Event driven web video summarization by tag localization and key-shot identification. IEEE Trans. Multimedia **14**(4), 975–985 (2012)
11. Janneck, J.W., Miller, I.D., Parlour, D.B., Roquier, G., Wipliez, M., Raulet, M.: Synthesizing hardware from dataflow programs. J. Signal Process. Syst. **63**(2), 241–249 (2011)
12. Lee, C.-Y.: Multi-pass, frame parallel algorithms of motion estimation in H.264/AVC for generic GPU. In: IEEE International Conference on Multimedia and Expo, pp. 1603–1606 (2007)
13. Chen, W.-N.: H.264/AVC motion estimation implementation on compute unified device architecture (CUDA). In: IEEE International Conference on Multimedia and Expo, pp. 697–700 (2008)
14. NVIDIA, OpenCL Programming for the CUDA Architecture, v2.3
15. Mattavelli, M., Amer, I., Raulet, M.: The reconfigurable video coding standard [standards in a nutshell]. IEEE Signal Process. Mag. **27**(3), 159–167 (2010)