CUDA Enabled for Android Tablets through DS-CUDA

EDGAR JOSAFAT MARTINEZ-NORIEGA†, ATSUSHI KAWAI ‡, KAZUYUKI YOSHIKAWA†, KENJI YASUOKA‡ AND TETSU NARUMI†

1. Introduction

The usage of GPUs (Graphics Processor Units) for parallel computing, GPGPU (General-Purpose Computing on GPU), has become a natural way to increase dramatically the performance of computer applications and simulations. Some of the top 10 super computers are already equipped with many GPU accelerators [1]. Since its first appearance in 2006, NVIDIA's CUDA [2] (Compute Unified Device Architecture) has become one of the most complete and common frameworks for parallel computing using GPUs. CUDA has already successfully accelerated many applications in an enormous variety of fields [3-4].

On the other hand, the way we interact with computers has been changed since the appearance of smartphones and tablets. These relative new devices offer a new way to represent data and interact (i.e. touch displays) with it taking input from different sources (i.e. gyro, accelerometer, camera etc.). However their computing power is still very limited compared with PCs.

In this paper, we propose to use DS-CUDA (Distributed-Shared CUDA), on Android tablet in order to merge these two worlds, mobility and high performance computing.

2. DS-CUDA Overview

DS-CUDA is a middleware that allows you to manage NVIDIA's GPUs on a distributed network. A single client node and various server nodes compose one DS-CUDA system. The server nodes have one or more CUDA capable GPUs that are handled by server processes. An application on the client side can use these parallel devices to process data without having a physical GPU. The program sees all GPUs contained into a cluster as if they were actually attached to the client node. Therefore, DS-CUDA is a kind of GPU-virtualization tool at source code level. When the client calls native CUDA API, the DS-CUDA preprocessor handles the correct wrapper function, which communicates with the server nodes through an InfiniBand (IB-Verb) or RPC (Remote Procedure Call) socket. A more detailed description is on another paper [5].

3. System Architecture

On Figure 1, we show the proposed system. The mobile device, client node, is a Nexus 7 tablet running Android 4.2.2. A PC (server node) with 2 GeForce 460 GTX GPUs and DS-CUDA is the server node. The PC and the tablet are connected in the same LAN through wireless network. Table 1 summarizes some important specifications of the system.

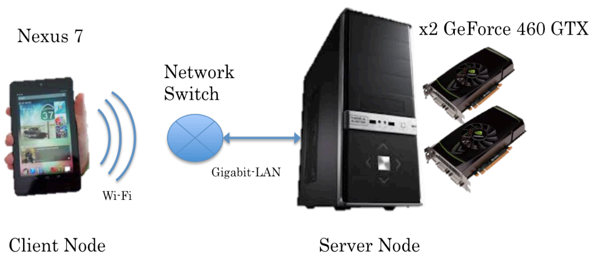

Figure 1. Proposed system.

† Faculty of Informatics and Engineering, The University of Electro-Communications
‡ Faculty of Science and Technology, Keio University

Client Node Tablet Nexus 7	GPU	ARM-Tegra 3, Quad-core, 1.7 GHz.
	Memory	1GiB
	GPU	N/A
	OS	Android 4.2.2
Server Node PC	CPU	Intel Core i7 920, 2.7GHz, 8 cores
	Memory	6 GiB
	GPUx2	NVIDIA GeForce 460 GTX, 7 Multiprocessors, 1.3 GHz 336 CUDA cores Global memory 1024MB
	OS	Ubuntu 12.04 x86 Linux
	CUDA	Driver 310.32, toolkit 4.1, SDK 4.1
Interconnect	Ethernet	1 Gibit/sec (theoretical throughput)
	Wireless 802.11n @2.4GHz	39 Mbit/sec

Table 1. Specifications of components of a system.

4. Implementation

We implemented a matrix multiplication program using the algorithm within the NVIDIA's CUDA-SDK "mutrixMul" [6]. The product of matrices A and B is calculated into matrix C. The contents of each matrix are floating numbers generated randomly.

First, DS-CUDA compiler processes the CUDA code of the matrix program, then two files are generated: source C code of a main program and .ptx file (low level code for GPU). These two files are mounted manually into the Android SD memory. We used a console environment to run the executable file in Android as in a normal linux environment.

5. Results

Figure 2 shows the time spent to produce matrix C. X-axis is the total amount of operations generated during the process. Total number of FLOPS (Floating-Point Operations per second) is represented on Y-axis.

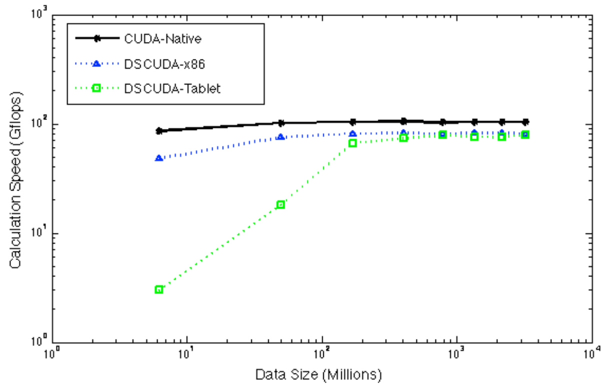

Figure 2. Calculation speed of matrix multiplication.

On 'CUDA Native', we performed the algorithm via CUDA Driver API, on 'DS-CUDA x86' we used the node server also as a client to generate the C Matrix. 'DS-CUDA Tablet' uses Android's Nexus 7 to perform the calculations.

6. Conclusion

In this paper we propose to use DS-CUDA as a linkage between a mobile device and GPGPU computing. DS-CUDA virtually enables Tablet to use GPUs in a server mode. Current work is focused on the detailed analysis of the communication time between server and client. How this time affects the performance of the application. Besides the comparison between the native CUDA API and CUBLAS (CUDA Basic Linear Algebra Subroutines) [7] will be also tested. The usage of DS-CUDA with Java trough JNI (Java Native Interface) [8], in order to run native application on Android´s Tablet, will be also a future topic. Running N-Body or Dynamics Molecular Simulations on Android's tablet is one of the primary targets of future research.

References

[1] http://www.top500.or/ TOP500 supercomputers website [retrieved: November 2012 List]

[2] The NVIDIA website. https://developer.nvidia.com/category/zone/cuda-zone [retrieved: march, 2012]

[3] Wen-Mei, W. Hwu, "*GPU Computing Gems, Emerald Edition*" , book, Morgan Kaufmann Publishers, 2011.

[4] http://gpgpu.org. General-Purpose Computation on Graphics Hardware site. [retrieved: march, 2012]

[5] Atsushi Kawai, Kenji Yasuoka, Kazuyuki Yoshikawa, and Tetsu Narumi, "Distributed-Shared CUDA: Virtualization of Large-Scale GPU Systems for Programability and Reliability", The Fourth International Conference on Future Computational Technologies and Applications, Nice, France, 2012.

[6] https://developer.nvidia.com/cuda-downloads Nvidias's Developer Kit site. [retrieved: March 2013]

[7] www.developer.nvidia.com/cublas Nvidias's developer site [retrieved: March 2013].

[8] www.developer.android.com/tools Android's developer site [retrieved: March 2013].