

High Performance Computing on Mobile Devices

電気通信大学 大学院 情報・通信工学専攻 成見研究室

1331098 MARTINEZ NORIEGA Edgar Josafat

1. Introduction

GPGPU (General Purpose Computing on Graphics Processors Units) has become one of the best paradigms to accelerate scientific applications and built supercomputers. Nowadays 2 / 10 supercomputers utilizes GPU's to achieve TFlops (floating operation per second) [1]. GPU's are designed with massively programmable parallel processors, different memory hierarchy and many core chips, thus are really attractive due to its performance/cost benefit. NVIDIA was one of the pioneers to offer an easy way to program and develop for GPU's through CUDA (Compute Unified Device Architecture), released in 2006 [2]. Mobile devices are becoming another interesting way to interact with computers due to its specific and particular capabilities: mobility, portability, low power consumption, touchscreen, connectivity, between other capabilities. In this sense, applications are changing, and also the way to interact and visualize data. Mobile devices are dotted with GPU's as well, however these ones cannot be programed easily to achieve more performance than only using the mobile processor e.g. ARM.

To merge high performance computing and mobile devices, we propose the usage of Distributed Shared CUDA (DS-CUDA). This new technology allows use one or many NVIDIA's GPU from our local network. We implement a molecular dynamics simulation on Tablet device using an external GPU (from our local network) to accelerate its performance.

2. CUDA

NVIDIA's CUDA was released on 2006 with the introduction of GeForce 8800 GTX. This new parallel

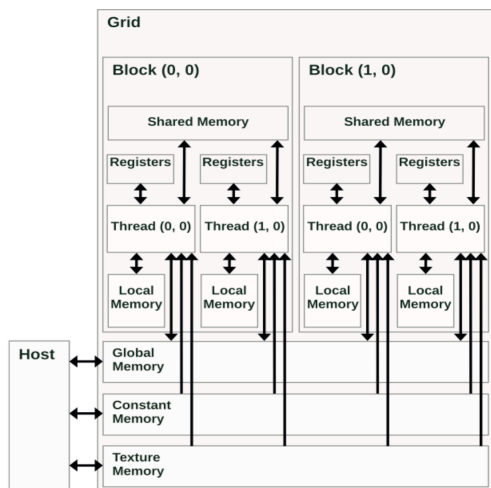


Fig 1: Hierarchy of CUDA memory.

computing platform and programming model enables dramatic increases in computing performance by harnessing the power of GPU. Easily using C /C++ , Fortran, Java or Python (between the most popular languages that support CUDA) we can access to the different memory spaces of GPU fig 1 and manage thousands of threads to achieve great performance in our applications.

3. DS-CUDA

DS-CUDA is a middleware that allows you to manage NVIDIA's GPU's 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 GPU's 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 GPU's 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 TPC socket. A more detailed description is on another paper [3].

4. Proposal System Architecture

On Figure 2, we show the proposed system. The mobile device, client node, is a Nexus 7 tablet running Android 4.3. A PC (server node) with 2 GeForce 460 GTX GPU's and DS-CUDA is the server node. The PC and the tablet are connected in the same LAN through wireless network. Table 1 summarizes

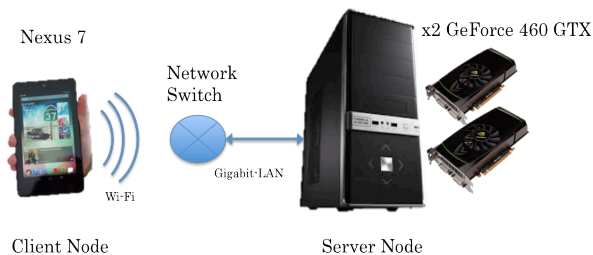


Fig 2: Proposal system architecture.

Client Node Tablet Nexus 7	GPU	ARM-Tegra 3, Quad-core, 1.7 GHz.
	Memory	1GiB
	GPU	N/A
	OS	Android 4.3
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 Gbit/sec (theoretical throughput)
	Wireless	39 Mbit/sec 802.11n@2. 4GHz

Table 1. System components specifications.

5. Molecular Dynamics NaCl Simulation.

We performed a molecular dynamics simulation on Android tablet. A particle conglomerate of NaCl is shown and its behavior under vacuum level. Tosi-Fumi potential [4] is used to describe the interaction between atoms. This potential describes a Coulomb term, a repulsion term, a dipole-dipole term and a dipole-quadruple term, where q_i and q_j are electric charges and r is the distance between them. Initially the system is equilibrated at 300 o K.

$$\phi_{ij} = \frac{q_i q_j}{r} + A_{ij} B \exp\left[-\frac{(\sigma_i + \sigma_j - r)}{\rho}\right] - \frac{C_{ij}}{r^6} - \frac{D_{ij}}{r^8} \dots (1)$$

	A (10^{-19} J)	$\sigma_i + \sigma_j$ (Å)	C (10^{-79} Jm ⁶)	D (10^{-99} Jm ⁸)
++	0.4225	2.34	1.68	0.80
+-	0.3380	2.75	11.20	13.90
--	0.2535	3.17	116.00	233.00

Table 2: Parameters of eq (1) B = 3.15A-1.

In order to compute eq. (1) for all bodies in the system, we allocate all parameters mentioned in table 1 inside of constant memory and send a fraction of each q_i ion to shared memory. Thus, we update the force for each q_j within each block of threads, keeping this result in shared memory as well. Finally we apply reduction sum in each thread block to obtain the complete force for each particle.

We used OpenGL ES 1.1 to render all particles in the system. Using JNI (Java Native Interface) [6], all the main code for the n-body simulation is written on C language. On this way, we can use DS-CUDA libraries and take advantage of faster computation rather than over Java code.

6. Results

Table 3 shows the performance of the molecular dynamics simulation between NaCl particles.

	8 Particles	1728 Particles	5832 Particles
Nexus 7 100Base Ethernet	0.003 Gflops	29.306 Gflops	99.887 Gflops
Nexus 7, DS-CUDA wireless (1m)	0.001 Gflops	9.86 Gflops	34.879 Gflops
Nexus 7, CPU, ARM-Tegra 3	0.052 Gflops	0.054 Gflops	0.054 Gflops
PC, Native CUDA, GeForce 460 GTX.	0.1 Gflops	391.3 Gflops	438.7 Gflops

Table 3: NaCl Molecular Dinamycs Force Performance

For a low amount of particles, CPU shows better performance than GPU, due to sending and receiving memory for a few particles. However, for a bunch amount of particles, latency and bottleneck of memory through the net is hide. With DS-CUDA we achieved almost a quarter of performance between using DS-CUDA with Tablet and native CUDA. We outrange the original Tablet CPU performance for more than 1 800 times faster calculations on NaCl molecular dynamics with 5832 particles.

7. Conclusions

We merge performance computing with mobile devices showing a better performance of molecular dynamics simulation on tablet using DS-CUDA. For future work we intend to make a thorough analysis of memory transfer. A model will be presented to compare theoretical and practical approaches.

References

- [1] <http://www.top500.or/> TOP500 supercomputers website [retrieved: June 2014 List]
- [2] The NVIDIA website. <https://developer.nvidia.com/category/zone/cuda-zone> [retrieved: march, 2014]
- [3] 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.
- [4] M.P. Tosi, F.G Fumi, "J. Phys. Chem. Solids", 25, 1964, 45.