Running CUDA on Android through GPU Virtualization

**Introduction**

The way we interact with computers has been modified since tablets and smartphones have become integral parts of daily life. New smart and interactive devices offer a new way to represent data and interact (i.e., touch display) with it, taking input from different sources (i.e., gyro, accelerometer, camera, etc.). Nowadays, they are equipped with multicore processors on GPUs and software to accelerate GPU, which are both designed for low power consumption. However, they are still not capable of multicore computing such as CUDA (Compute Unified Device Architecture) or OpenCL. Since its appearance in 2006, NVIDIA’s CUDA has become one of the most complete and common frameworks to use GPGPU (General-Purpose Computing on GPU) such that, some of the top 10 supercomputers are already equipped with many NVIDIA accelerators [1]. As well, CUDA has been successful to accelerate various applications in many fields [3-4]. Nevertheless, with the arising of cloud computing, taking resources from the network is also important, such as remote GPGPU [2,7]. We propose the usage of DS-CUDA (Distributed-Shared CUDA) in order to bring one of the most popular libraries of multicore computing, CUDA, to Android ecosystem.

**DS-CUDA**

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 TCP socket. More detailed description is on [this paper] [2].

**Molecular Dynamics Implementation**

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-quadrupole term, where $q_i$ and $q_j$ are electric charges and $r_{ij}$ is the distance between them. Initially the system is equilibrated at 300 °K.

$$g(r)=\frac{6\pi\varepsilon_0}{r} + A\text{Exp}\left(\frac{\alpha r_{ij} - 1}{\beta}\right) + \frac{C}{r^6} + \frac{D}{r^{12}} \ldots (1)$$

$\varepsilon$ (Nanometer) $\alpha$, $\beta$, $C$, and $D$ are the parameters (19).

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$C$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4225</td>
<td>3.34</td>
<td>1.08</td>
<td>0.00</td>
</tr>
<tr>
<td>0.5368</td>
<td>2.75</td>
<td>11.20</td>
<td>13.00</td>
</tr>
<tr>
<td>0.2535</td>
<td>2.34</td>
<td>1.68</td>
<td>0.80</td>
</tr>
</tbody>
</table>

Table 1: Parameters of $g(r)$ (1) $B = 3.15A^5$.

**System Architecture**

Below, a system proposed is shown. The client node (mobile device) is a Nexus 7 tablet running Android 4.2.3. The server node (PC) has 2 GeForce 460 GTX GPUs. The PC and the tablet are connected in the same LAN network through wireless 802.11n and 100Base Ethernet using an USB 2.0 adapter.

**Results**

Data transfer speed performance using different networks connections are shown below. Latency on these mediums is also included. We also added a Gigabit PC test connection, just for reference purposes. H2D means “Host to Device”, while D2H refers to “Device to Host” data transfer.

The amount of floating operations per second (FLOPS) to compute force between all bodies was measured. For low amount of particles, DS-CUDA does not show good performance due to bottleneck communication between client and server. However for more than 64 molecules, parallel computation on GPU hide this latency problem. For more than 5 000 molecules on the simulation, DS-CUDA accelerate the app 1000x times more than the CPU implementation.

**Conclusion**

We were able to run CUDA code on Android device using DS-CUDA framework. Through GPU virtualization, we implemented the force computation of a molecular dynamics simulation. We reach up to 1000 times speed up using DS-CUDA over 100 Base Ethernet compare with the tablet CPU. And almost a quarter of performance compared with the native CUDA running on a PC.

---

**References:**

[1] NVIDIA GeForce ULP (CUDA no enabled)
[3] EDGAR JOSAFAT MARTINEZ-NORIEGA†, ATSUSHI KAWAI†, KAZUYUKI YOSHIKAWA†, KENJI YASUOKA† AND TETSU NARUMI†

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

---

**Figures and Tables:**

- **Figure 1:** Proposed system.
- **Figure 2:** Performance of cudaMemcpy ( ).
- **Figure 3:** Force Computation Performance.
- **Table 1:** Parameters of $g(r)$ (1) $B = 3.15A^5$.