A Case Study of OpenCL on an Android Mobile GPU

James A. Ross*, David A. Richie†, Song J. Park‡, Dale R. Shires‡, and Lori L. Pollock§
*Engility Corporation, Chantilly, VA
james.ross@engilitycorp.com
†Brown Deer Technology, Forest Hill, MD
drichie@browndeertechnology.com
‡U.S. Army Research Laboratory, APG, MD
{song.j.park.civ,dale.r.shires.civ}@mail.mil
§University of Delaware, Newark, DE
pollock@cis.udel.edu

***Abstract*—An observation in supercomputing in the past decade illustrates the transition of pervasive commodity products being integrated with the world's fastest system. Given today's exploding popularity of mobile devices, we investigate the possibilities for high performance mobile computing. Because parallel processing on mobile devices will be the key element in developing a mobile and computationally powerful system, this study was designed to assess the computational capability of a GPU on a low-power, ARM-based mobile device. The methodology for executing computationally intensive benchmarks on a handheld mobile GPU is presented, including the practical aspects of working with the existing Android-based software stack and leveraging the OpenCL-based parallel programming model. The empirical results provide the performance of an OpenCL N-body benchmark and an auto-tuning kernel parameterization strategy. The achieved computational performance of the low-power mobile Adreno GPU is compared with a quad-core ARM, an x86 Intel processor, and a discrete AMD GPU.**

***Keywords*-handheld GPU; OpenCL; Android; N-body;**

I. Introduction

Mobile graphics processing units (GPUs) paired with ARM processors on low-power system-on-a-chip (SoC) devices have been available for several years in smartphones and tablets. Despite widespread adoption and deployment of discrete GPUs as accelerators, there remains no clear path for programming heterogeneous systems with co-processors. The development of application programming interfaces (APIs) and programming methodologies for mixed architectures are a topic of active research and development. A major challenge is the lack of availability and maturity in software technologies that should, in principle, allow for programmability and efficiency across the coupled disparate cores in these devices.

Handheld mobile GPU architectures now possess general-purpose compute capabilities and a software stack sufficient to begin exploring these devices as computational accelerators. This development in the technology mirrors that of desktop, workstation, and large-scale high performance computing (HPC) platforms where GPUs are increasingly used to offload parallel algorithms for improved application performance. Leveraging the compute capability of modern mobile GPUs through Android platforms may allow computationally demanding applications to achieve higher performance. However, the programmability of these heterogeneous systems for effective HPC remains an open question.

The main contributions of this paper are the following:

- A case study in porting an OpenCL parallel benchmark to the mobile GPU on an Android device using a higher-level programming API that leverages OpenCL.
- Empirical results from tuning an OpenCL N-body benchmark on a Qualcomm Adreno 320 mobile GPU in comparison to other common architectures.
- An evaluation of mobile Android platforms directed towards gathering an initial picture of the GPU capability in terms of architecture, hardware features, usability, and performance.

II. Motivation

Processing technologies continue to decrease in size, thus increasing the mobility of compute resources. The history of computing suggests that computational power trickles down from stationary systems to mobile devices. As a case in point, consider modern smartphones that now rival past supercomputer performance. Many individuals are walking around with compute-capable devices, which provides mobility of sophisticated computing.

With the end of the frequency scaling single-core era [1], [2], parallel computing technologies are ubiquitous and can be found across a wide range of devices such as smartphones, laptops, and supercomputers. The economics of computing and market forces favor low-cost, high-volume, and general-purpose processors rather than slowly evolving, performance-centric, specialized products [3], [4]. The rise of clusters using commodity personal computers is an example of how the popularity of the everyday PC transformed the supercomputing community. Modern GPUs appear to satisfy this economic observation where video cards are reasonably priced, widely available, and support general-purpose computing. An execution of floating-point intensive simulation on a low-power em-

bedded GPU technology provides a comparative performance case study for potential future-generation HPC systems.

Dedicated or distributed computing over mobile devices, which are becoming increasingly heterogeneous, is an area that pushes this idea of high performance computing to the edge. Indeed, the U.S. military has seized on this trend and has recently approved Android phones for Soldiers [5]. Limited data sharing over *ad hoc* networks, or specialized mobile devices with larger data storage, may alleviate memory storage problems for high resolution databases, but computing efficiency and required time-to-solution remains an issue when high performance servers are not available to handle processing requests. In these situations when mobile devices have to do the heavy lifting in processing, understanding and optimizing efficiency of mobile GPU technology can be critical. This is true for the military and first responders where timely situational awareness is key in avoiding risk in dynamic environments where decisions have to be made rapidly. For example, this might involve optimized way-point planning for entry to or exit from a hostile environment. Line-of-sight (LOS) types of applications will also be paramount in determining threats from explosives or hostile forces. Hazardous chemical leaks into the air will also require extensive processing to determine dispersion based on active weather patterns in the area. In all of these applications, mobile processing can assist as a vital computational resource.

III. Mobile GPUs for General-Purpose Computation

Due to very tight restrictions on energy consumption, mobile devices are not generally used for computationally intensive calculations. The theoretical throughput of mobile ARM processors remains a staggering two orders of magnitude lower than workstation CPUs, although this gap is generally decreasing. Peak FLOPS for ARM Cortex processors are reported in [6] and GFLOPS for Xeon E5-4600 processors are provided in [7]. Mobile GPUs can be used to improve interactivity where previous calculations could not be computed within interactive time frames. The performance data presented here will illuminate that the computational throughput of a mobile GPU goes further in closing the gap in mobile device performance as compared to modern x86 desktops or workstations.

OpenCL provides an industry standard for parallel programming of heterogeneous computing platforms [8], and is designed to meet the requirements of exposing the compute capability of devices such an ARM SoC with a general-purpose GPU. Despite announcements suggesting OpenCL support for mobile ARM devices, as well as public demonstrations at trade shows, vendor support of an OpenCL implementation has been non-existent until very recently. This has changed with the most recent mobile GPUs, namely, the ARM4 Mali-T604 and Qualcomm Adreno 320, both typically found integrated with an ARM Cortex-A15 CPU. The availability of an OpenCL implementation for both GPUs has been confirmed and can be found on consumer products using these devices. This recent development provides a significant opportunity to finally benchmark these devices for computationally expensive applications using OpenCL. Notwithstanding this positive development, the overall software stack and support remains relatively complicated and immature.

Programming with OpenCL on mobile GPUs poses multiple challenges due to several limitations. The Android operating system and software stack are an obstacle for accessing mobile GPU performance with vendor-supplied OpenCL implementations. Since OpenCL is a C-based library, the developer must utilize the Android Native Development Kit (NDK) to cross-compile the native executable. For general-purpose GPU (GPGPU) computation, Google recommends RenderScript, which is a Java-based API primarily used for supplemental graphics effects within an Android App. However, using OpenCL allows the application developer to leverage an existing GPU code rather than porting code to RenderScript. To the best of our knowledge, only a few people have executed OpenCL code on a mobile GPU under Android, as evident in notably few related publications [9], [10].

As in the desktop computing environment, there are multiple vendors with GPU architectures available for integration into low power SoCs. The most notable examples are the Qualcomm Adreno, the ARM Mali, the NVIDIA GeForce ULP, Vivante's ScalarMorphic architecture, and the Imagination Technologies PowerVR architectures. Each of these GPU architectures has been used in a number of mobile Android products, but presently only Adreno and Mali appear to have conformant OpenCL implementations available for current hardware in consumer products.

Qualcomm has shipped an OpenCL implementation with new phones that contain the Adreno 320 GPU. The Qualcomm Snapdragon S4 Pro APQ8064i SoC is in the LG Nexus 4 mobile smartphone. The Snapdragon SoC contains a 1.5 GHz quad-core Krait ARM CPU and Adreno 320 GPU. While hardware details about the Adreno 320 GPU are sparse, existence of the OpenCL library is sufficient to access and benchmark the capabilities of the device.

IV. Auto-Tuning OpenCL N-Body Benchmark

N-body is an algorithm used to solve Newton's laws of motion for N particles subject to an inter-particle force. The basic algorithm requires the update of all particle positions and velocities based on the distance of a given particle to all others. The update is performed by calculating a distance-dependent force and then numerically integrating the equations of motion using a fixed time-step. The algorithm consists of a double loop over all particles, a distance calculation between all particle pairs, and the accumulation of forces acting on each particle.

The N-body algorithm provides an excellent benchmark for the evaluation of a computing platform for several reasons. First, the basic algorithm is representative of many real-world computational kernels such as astrophysics, molecular dynamics, plasma physics, and protein folding [11], [12], hence may serve as a proxy for their expected performance.

Second, the manner in which the simulation is performed is relatively clean without superfluous computations that would complicate the interpretation of the performance benchmarks. Third, the algorithm provides a simple mechanism of sweeping a single parameter, the number of particles, to drive the system into compute bound regimes since computation and data movement scale as $O(N^2)$ and $O(N)$, respectively. This is critical when studying co-processor architectures since it is widely observed that the cost of data transfer can overtake the benefit of additional co-processing compute capability in a given computational problem. Finally, the simulation is commonly implemented on a range of architectures and provides a convenient canonical algorithm for comparative benchmarking.

In this investigation, we use an in-house auto-tuning OpenCL N-body code (which we call the N-body benchmark) that parameterizes the kernel in ways that impact performance across different architectures and compilers. The host code uses the STDCL API (see Section V-A) that simplifies the use of OpenCL for HPC applications. The auto-tuning capability of the benchmark may be used to optimize performance on a given architecture by specifying ranges for parameter sweeps, allowing the code to test many parameter combinations and record the resulting performance.

The kernel is parameterized by the following parameters. The parameter *nmulti* is the number of particles updated per thread. The outer-loop multiplicity allows a compiler to automatically vectorize the computation. The parameter *nunroll* defines the explicit unrolling within the inner loop over particle pair interactions. The parameter *nblock* is the number of particle positions cooperatively cached by a work-group in local memory for calculating particle pair interactions. This essentially replaces the inner loop over particles with a double loop over blocks of *nblock* particles and a nested loop over the cached particle positions. The parameter *nthread* determines the OpenCL work-group size. Additionally, there is a Boolean parameter *fma* that enables the use of the built-in fused multiply-add. A partial code listing of the kernel generation scheme based on some of the parameters appears in Fig. 1 and makes heavy use of pre-processing.

Within the context of computing on GPUs, an increase in any of these parameters generally increases the register and local memory usage, which does not necessarily harm performance. Increasing the number of registers and local memory does, however, begin to indirectly limit the number of thread teams ("wavefronts" in AMD terminology or "warps" in NVIDIA terminology) that can execute concurrently on a GPU compute unit. This limits the total number of threads in flight, which decreases the GPU's capability to hide memory latencies and hence can cause performance degradation. Since the register usage is determined by the OpenCL JIT compiler and not the programmer, the programmer cannot possibly know *a priori* which kernel parameter configuration will result in the highest number of thread teams in flight while fully utilizing the hardware. Since mobile GPU architecture details are limited and evolving so rapidly, parameterized auto-tuning

```
#pragma for I,0,NMULTI-1
T4 p@I@ = pbuf[NMULTI*get_global_id(0) + @I@];
T4 a@I@ = (T4)(0.0,0.0,0.0,0.0);
#pragma endfor
for(uint i = 0; i < n; i += NMULTI) {
   T4 p,dp;
   T inv;
#pragma for J,0,NMULTI-1
   p = pbuf[i+@J@];
#pragma for I,0,NUNROLL-1
   dp = p - p@I@;
   invr = rsqrt(
      FMA(dp.x,dp.x,
         FMA(dp.y,dp.y,
            FMA(dp.z,dp.z,eps))));
   a@I@ = FMA(p.w,invr*invr*invr*dp,a@I@);
#pragma endfor
#pragma endfor
}
```

Fig. 1. In this kernel snippet example, the code is doubly unrolled by the pre-processor definitions *NMULTI* and *NBLOCK*. *FMA* is a macro that either uses the built-in fused multiply-add routine or basic math operations, while *T* and *T4* are the data type and vector data type, respectively. Our pre-processor replaces the *@I@* and *@J@* values directly with the loop index inside *#pragma for* statements. This pre-processor code generation scheme improves upon other OpenCL kernel string generation schemes by making the code more readable.

OpenCL kernels may serve as an effective means to achieve high performance across devices and evolving software stacks as long as care is taken to validate correct operation for arbitrary parameter values.

V. Programming Mobile GPUs on Android Platforms

A. OpenCL-based Software Stack

Although OpenCL is commonly viewed as a GPU programming language, its applicability is more general and specifically includes multi-core CPUs. OpenCL provides a portable vendor- and device-independent API for targeting parallel processors. OpenCL is an explicit API that enables precise coding of host-side scheduling, data movement, and algorithm implementation for compute devices.

One drawback of OpenCL is that the API is better suited as a lower-level middleware layer than an API for HPC application development. In comparison to its closest vendor supported competitor, CUDA, the direct use of OpenCL is significantly more tedious and complicated. The strength of OpenCL is in its portability as a compute device API and not in its syntax or semantics for HPC application developers. The introduction of OpenCL into a large HPC code can make long-term development and maintenance more difficult.

To mitigate the complexity of host-side application programming with OpenCL (API 1.1), STDCL (STandarD Compute Layer) (version 1.5) was used in this work. STDCL is a simplified programming API [13] that supports OpenCL with higher abstraction application development. Note that the concepts behind STDCL is not simply to provide a wrapper for OpenCL calls. STDCL supports default compute contexts, conventional memory allocation of device-sharable

memory, OpenCL-based event management, and a dynamic kernel loader (reflecting a more traditional compilation model and an offline kernel compiler [clcc]).

This case study depended on extensive use of the COPRocessing THReads (COPRTHR) SDK developed by Brown Deer Technology that is freely available under an open-source license (LGPLv3) [14]. The COPRTHR SDK provides libraries and tools that leverage OpenCL for portability and supports the development of applications that exploit the multi-threaded parallelism of modern multi-core and many-core processors. STDCL, mentioned previously, is a predominate component of COPRTHR. The portability of the SDK is limited only by the availability of a suitable OpenCL implementation, and includes support for processors from AMD, Intel, and NVIDIA. Furthermore, the ability to build the SDK from source allows portability to new or experimental platforms.

B. OpenCL-based Program on an Android Device

The C libraries required to support OpenCL-based code are presently inaccessible directly from the Android SDK, and instead require the use of the Android NDK. The NDK workflow is based on the use of cross-compilation techniques and requires the device to be paired with a workstation running Linux on which the actual software development will take place. Compiling software using the Android NDK is complicated, and required resolving numerous issues including missing libraries and functionality in the Android operating system as compared with a standard Linux distribution.

The NDK (release 8e in this work) does not include an OpenCL library or implementation. In order to cross-compile an application that links to the OpenCL library, one must copy the library (libOpenCL.so) from the target device to the host machine performing the cross-compilation link. Resolving all issues with the Android NDK provided the ability to cross-compile the COPRTHR SDK from source and then cross-compile the auto-tuning N-body benchmark code that relies upon the SDK.

Development process proved that executing native applications under Android is complicated, cumbersome, and inefficient when compared to a conventional software development and testing workflow. The process requires using either an Android App to kickstart the binary or running the binary over an SSH-like shell called ADB Shell. Alternatively, Android Terminal is an application available from the Google Play Store that gives the user a terminal with access to the file system and limited set of UNIX commands. However, Android Terminal cannot generally execute binaries anywhere within the file system due to Android permission policies, whereby a virtual user is associated with each Android application, constraining execution to within the application's dedicated directory. In this case, the Android Terminal application only has permission to run binaries under its own directory. Furthermore, application directories are usually hidden and the paths are unknown.

Fortunately, Android Terminal openly published their application directory. Using the Android Terminal, one can change directory to this location and use the UNIX "cat" command to create a copy of the binary program file (directly copying does not work due to permission issues). Finally, the permissions of the newly created file can be changed to 755, designating it as an executable. This recipe is complicated and clearly not designed to allow the easy execution of user binaries, but it nevertheless provides a work-around.

Another strategy uses the ADB Shell to create a directory under /data/local/, runs "adb push [filename]" to copy the executable program to that directory, and changes permissions to 6755 to make the binary executable. Subsequently, the program can be executed as normal under the ADB shell.

VI. Evaluation Study

A. Methodology

The Android NDK cross-compiling environment (Ubuntu 12.10 x86, 64-bit, and gcc 4.7) was used to compile the COPRTHR SDK and the OpenCL N-body benchmark code for the Adreno 320 mobile GPU on the Nexus 4 smartphone. The benchmark was then used to empirically measure the performance of the mobile GPU. The auto-tuning of the N-body benchmark adjusted parameters for the target architecture. Performance measurements were collected for a range of values of the number of particles in the simulation. Sweeping this value allows the benchmark to be systematically driven into a compute-bound regime. Performance from the auto-tuning benchmark is compared with that of the default kernel tuned for high-end discrete GPUs. The overall results are compared to similar empirical measurements for selected CPU and discrete GPU processor architectures.

B. Results

Fig. 2 depicts the results of the performance measurement on the Adreno 320 mobile GPU using the OpenCL N-body benchmark. The number of particles in the simulation was increased from 512 to 8,192. Auto-tuning parameters for simulations greater than 4,096 had to be manually set due to failure at slower configurations (most likely a watchdog timer issue with the platform).

Running simulations using more than 8,192 particles failed to execute properly. This type of limitation could be the result of a real constraint of the architecture or an artificial constraint such as a watchdog timer preventing the device from executing a compute kernel beyond a certain amount of a time. A similar artificial constraint has been observed on NVIDIA discrete GPUs used concurrently to drive a display in order to maintain interactivity [15].

The performance as a function of the number of particles shows that the simulation is driven into a compute-bound regime at around 4,096 particles. The lower number of particles shows a decrease in the measured performance indicating that other factors, such as latency and data movement costs, are impacting the results, whereas using more particles shows no significant increase in performance. This general behavior has

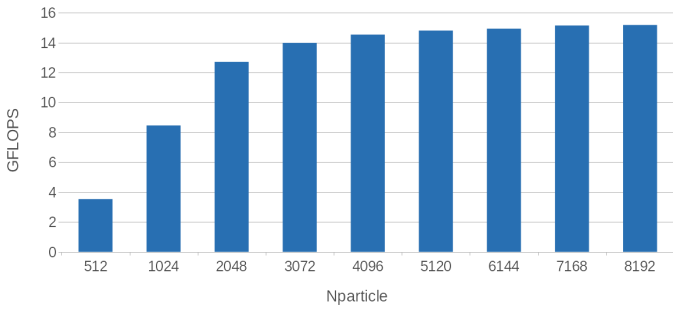

Fig. 2. Performance of an OpenCL N-body benchmark for a range of particle system sizes on an Adreno 320 mobile GPU of an Android smartphone. The simulation is driven into a compute-bound regime at around 4,096 particles, beyond which increasing the number of particles in the simulation shows negligible increase in measured performance.

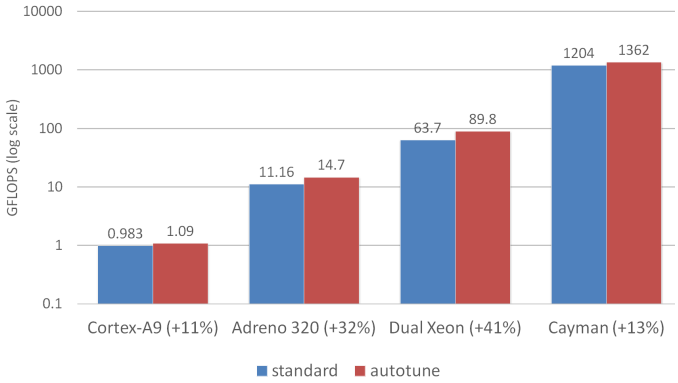

Fig. 3. Performance results for the standard and auto-tuned N-body algorithm executed on all cores of a quad-core ARM Cortex A9 (1.1-1.4 GHz), the Qualcomm Adreno 320 mobile GPU, dual Intel Xeon X5650 (twelve x86 cores at 2.67 GHz), and the AMD Radeon HD 6970 GPU (Cayman). The performance in parentheses is the improvement in performance of the auto-tuning benchmark compared to the default kernel. The performance is calculated using a standard measure of 20 floating-point operations per particle-particle interaction, which includes square root and division operations known on many architectures to require more cycles than a multiply-add.

been observed for all processor architectures tested including discrete GPUs and CPUs, and is a general characteristic of the algorithm itself. The peak measured performance for the Adreno 320 is found to be approximately 15.2 GFLOPS with 8,192 particles.

Fig. 3 compares the measured performance of Adreno GPU to ARM, Xeon CPU, and AMD GPU. All implementations are parallelized for optimal utilization of the processing units. Xeon CPU result is based on using both the Xeon CPUs in a dual socket, reflecting the performance of combined CPU chips. The difference in performance between Adreno 320 and two hexa-core Xeon was 6.1x for the autotuned comparison. Considering that Qualcomm Adreno is powered by a battery, observed computational performance is admirable.

Although the LG Nexus 4 OpenCL implementation reported an ARM CPU back end, OpenCL kernels failed to execute properly on the device. Another ARM-based platform with a valid OpenCL implementation is reported here for comparison [16]. The illustration of results consists of the default OpenCL kernel, originally tuned for a high-end AMD Cypress GPU, as well as the peak performance found with the auto-tuning benchmark presenting the effect of tuning OpenCL kernel parameters. When comparing Cortex-A9 versus Adreno 320 inside the smartphone, an order magnitude higher value is evidenced by the Qualcomm mobile GPU.

C. Analysis and Discussion

At the time of this work, the theoretical throughput and thermal design power (TDP) of the Adreno 320 were unknown, as well as many other details about the processor architecture itself. Publicly available information remains sparse on this architecture design. A reasonable estimate based on the observed performance and the constraints of the platform in which the device is integrated suggests that these metrics are on the order of a few tens of GFLOPS in theoretical peak performance and a few watts for the TDP. Without precise data from the manufacturer, devising an accurate power efficiency in comparison to other relevant processor architectures is challenging due to the difficulty in identifying the correct pins to probe for the Qualcomm Adreno 320 chip.

Considering that Adreno 320 is packaged inside a mobile phone, the embedded GPU displayed a commanding N-body performance. Comparing Adreno to Intel Xeon, the power consumption is drastically different between a battery-powered device and a wall-plugged workstation. Yet, dual Intel Xeon X5650 reached 89.8 GFLOPS and Adreno was observed achieving 14.7 GFLOPS. It should be noted that the measured Xeon GFLOPS number is for two Xeon X5650 processors, because of the dual CPU socket configuration in the test bed workstation. The N-body results show that the Nexus 4 mobile devices that many of us carry are beginning to approach the single-precision floating-point capabilities of the Xeon X5650.

Based on the results of the two OpenCL applications under Android, some broad generalizations can be derived about the Qualcomm Adreno 320 and underlying OpenCL implementation. The implementation appears to be relatively immature as exhibited by the stability issues, compiled kernel performance variation, and kernel compiler speed. However, these kinds of issues have appeared on other OpenCL platforms in the past and have generally improved with time as the software matures. As the hardware advances and OpenCL implementations appear, software developers will leverage the OpenCL capabilities on smartphones and the new competition between vendors will pave the way to improvements in the software stack.

VII. Related Work

Several research studies have been conducted on using mobile handheld devices for general-purpose computation [17], [18], [19], [20]; however, our work is distinguished by executing an OpenCL framework that extends to diverse heterogeneous resources. Accordingly, a single code base was compared to various computing elements consisting of an ARM, X86, and discrete GPU architectures.

The Mont-Blanc project is an European initiative to build HPC from mobile phone processors [21]. Stanisic et al. present performance evaluation on low-power embedded platforms to address energy-efficiency in reaching exascale computing. An experimental HPC cluster named Tibidabo was constructed with NVIDIA Tegra2 chips [22]; however, CUDA or OpenCL is not supported on the GPU in Tegra2. The Mont-Blanc project argues for the potential of embedded devices in future HPC systems. Albeit single device, our research adopted the broader view of OpenCL framework and employed STDCL abstraction.

Additionally, an auto-tuning method of parameterized kernel to identify an optimal kernel was applied for our case study. In comparative performance studies between CUDA and OpenCL versions of the same applications, Fang et al. [23] and Komatsu et al. [24] state that performance can be comparable if the kernels are optimized by hand or by compiler optimizations. They also concluded that automatic parameter tuning is essential to enable a single OpenCL code to run efficiently on various GPUs, motivating the need for auto-tuning for each system and comparative performance studies. Yao et al. [25] studied the performance portability of OpenCL across diverse architectures including NVIDIA GPU, Intel Ivy Bridge CPU, and AMD Fusion APU, using three OpenCL benchmarks — SGEMM, SpMV, and FFT. They found that performance portability requires tuning threads-data mapping, data layout, tiling size, data caching, and operation-specific factors. Our parameterized kernel approach starts to address the challenges of performance portability.

VIII. Conclusions and Future Work

This case study demonstrated a promising approach to port an OpenCL-based software stack to an Android platform by compiling and executing a representative benchmark for computationally expensive scientific applications for an Adreno 320 mobile GPU on a Nexus 4 smartphone. Empirical performance measurements for an auto-tuning OpenCL N-body benchmark portrayed that the smartphone embedded GPU reached 15.2 GFLOPS in a compute-bound regime, which can be on par with Intel desktop processors. The investigation revealed that the use of the mobile GPU showed some limitations with large simulations or using certain kernel parameters, hence technical challenges exist in using the device for HPC tasks. Nevertheless, the results show a competitive edge for offloading computationally intensive tasks to a mobile GPU when compared to the capabilities of an ARM processor.

Further work is necessary to identify methods for collecting accurate measurements of power consumption in smartphones for power efficiency analysis. Tightly packaged construction of mobile phones makes Adreno GPU isolation a challenge.

References

[1] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobbs Journal*, vol. 30, no. 3, pp. 202–210, 2005.

[2] K. Olukotun and L. Hammond, "The future of microprocessors," *Queue*, vol. 3, no. 7, pp. 26–29, Sep. 2005.

[3] R. Vuduc and K. Czechowski, "What GPU computing means for high-end systems," *Micro, IEEE*, vol. 31, no. 4, pp. 74–78, 2011.

[4] G. Bell, "Bell's law for the birth and death of computer classes," *Commun. ACM*, vol. 51, no. 1, pp. 86–94, Jan. 2008.

[5] (2013, May) U. S. military approves Android phones for soldiers. [Online]. Available: http://www.bbc.co.uk/news/technology-22395602

[6] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with commodity CPUs: Are mobile SoCs ready for HPC?" in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 40:1–40:12.

[7] "Intel xeon processor E5-4600 series," http://download.intel.com/support/processors/xeon/sb/xeon_E5-4600.pdf, May 2012.

[8] *OpenCL The open standard for parallel programming of heterogeneous systems*, Khronos Group Std., 2009.

[9] (2013, January) OpenCL reveals Kindle Fire's true potential. [Online]. Available: http://gearburn.com/2013/01/opencl-reveals-kindle-fires-true-potential/

[10] G. Wang, Y. Xiong, J. Yun, and J. Cavallaro, "Accelerating computer vision algorithms using OpenCL framework on the mobile GPU - a case study," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013, pp. 2629–2633.

[11] G. Blelloch and G. Narlikar, "A practical comparison of n-body algorithms," in *Parallel Algorithms*, ser. Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1997.

[12] L. Nyland, M. Harris, and J. Prins, "Fast n-body simulation with CUDA," *GPU gems*, vol. 3, pp. 677–695, 2007.

[13] *STDCL Reference Manual*, www.browndeertechnology.com/docs/stdcl-reference-rev1.4.html, Brown Deer Technology, 2012, revision 1.4.

[14] "COPRTHR SDK," www.browndeertechnology.com/coprthr.htm, 2012.

[15] "NVIDIA CUDA toolkit v5.0 release notes errata," October 2012.

[16] D. Richie, J. Ross, J. Ruloff, S. Park, L. Pollock, and D. Shires, "Investigation of parallel programmability and performance of a Calxeda ARM server using OpenCL," in *The Sixth Workshop on Unconventional High Performance Computing 2013 (UCHPC 2013)*. IEEE, Aug 2013.

[17] N. Singhal, I. K. Park, and S. Cho, "Implementation and optimization of image processing algorithms on handheld GPU," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, Sept 2010, pp. 4481–4484.

[18] K.-T. Cheng and Y.-C. Wang, "Using mobile GPU for general-purpose computing - a case study of face recognition on smartphones," in *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*, April 2011, pp. 1–4.

[19] M. Bordallo Lpez, H. Nyknen, J. Hannuksela, O. Silvn, and M. Vehvilinen, "Accelerating image recognition on mobile devices using GPGPU," pp. 78 720R–78 720R–10, 2011.

[20] B. Rister, G. Wang, M. Wu, and J. Cavallaro, "A fast and efficient sift detector using the mobile GPU," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013, pp. 2674–2678.

[21] L. Stanisic, B. Videau, J. Cronsioe, A. Degomme, V. Marangozova-Martin, A. Legrand, and J.-F. Mehaut, "Performance analysis of HPC applications on low-power embedded platforms," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 2013, pp. 475–480.

[22] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez, "Tibidabo: Making the case for an ARM-based HPC system," *Future Generation Computer Systems*, vol. 36, no. 0, pp. 322 – 334, 2013.

[23] J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of CUDA and OpenCL," in *Proceedings of the 2011 International Conference on Parallel Processing*, ser. ICPP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 216–225.

[24] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi, "Evaluating performance and portability of OpenCL programs," in *The Fifth International Workshop on Automatic Performance Tuning*, June 2010.

[25] M. S. I. Yao Zhang and A. A. Chien, "Improving performance portability in OpenCL programs," 2013.