Investigation of Parallel Programmability and Performance of a Calxeda ARM Server Using OpenCL

David Richie[1], James Ross[2], Jordan Ruloff[2], Song Park[3], Lori Pollock[4], and Dale Shires[3]

[1] Brown Deer Technology, Forest Hill, MD
drichie@browndeertechnology.com
[2] Dynamics Resources Corporation (DRC), Andover, MA
jaross,jruloff@drc.com
[3] U.S. Army Research Laboratory, Aberdeen, MD
song.j.park.civ,dale.r.shires.civ@mail.mil
[4] University of Delaware, Newark, DE
pollock@udel.edu

Abstract. This paper explores the parallel programmability, performance, and energy efficiency of a recently available Calxeda ARM-based server as a potential energy-efficient platform for computationally-intensive applications. A novel OpenCL-based parallel programming model for the Calxeda ARM server is achieved via the use of a higher level STandarD Compute Layer (STDCL) application programming interface and a remote procedure call implementation. Empirical measurements of the performance of the platform are obtained and presented using an N-body code executed in various configurations. Furthermore, an auto-tuning technique was developed and analyzed for optimization of the N-body algorithm on a specific architecture.

Keywords: High Performance Computing, ARM server, OpenCL

1 Introduction

Power consumption has arguably become the single most critical factor impacting High Performance Computing (HPC) for architectures ranging from large-scale supercomputers to data centers to mobile computing platforms [8, 2, 5]. It is within this context that ARM processors, which dominate the mobile smartphone and tablet market, are being re-purposed as an alternative to the x86 processors that have dominated HPC, desktops, and workstations, for well over a decade. Whereas a modern x86 processor is based on a CISC architecture with substantial per-core capability, ARM processors are based on a RISC architecture designed for low-power operation [7].

The newly developed ARM-based servers such as those from Calxeda offer low-energy usage, high-memory density, high-core density, and high-storage density – all qualities that motivate using these servers for high performance computing and big data applications. However, the relative core efficiency as compared

with x86 CPUs in terms of $\frac{performance}{power}$ and $\frac{performance}{price}$ metrics remains an open question. Currently, an additional limitation exists where the ARM processors do not provide native support for 64-bit operations, hindering their applicability to many HPC applications. The Calxeda ARM servers [4] represent the most direct attempt to date aimed at bringing ARM processors into competition with traditional x86 servers. These servers form the basis for the Hewlett-Packard Moonshot project that aims to develop and deploy energy-efficient servers, initially targeting web server applications. Calxeda servers represent a shift from high-power, high-capability computing platforms to collections of lower-power processors using technology derived from the mobile smartphone market.

This paper reports exploration into the parallel programmability, performance, and energy efficiency of a Calxeda ARM server using an OpenCL-based approach. OpenCL provides an explicit low-level programming model for co-processor architectures that may also be used as a convenient means of exploiting the parallelism of systems with multi-core CPUs treated as abstract OpenCL compute devices.

The main contributions of this paper are:

- The demonstration of an OpenCL-based parallel programming model for targeting the Calxeda ARM server for computation-intensive applications.
- The extension of a parameterized auto-tuning OpenCL N-body benchmark that can be used to optimize performance across different architectures.
- An empirical evaluation of performance and power efficiency of a Calxeda ARM server using an N-body benchmark representative of other simulations.

Section 2 describes the Calxeda server system used for evaluation. Section 3 investigates the use of existing software packages that leverage OpenCL to provide a novel parallel programming model for platforms like the Calxeda ARM servers. Section 4 discusses the use of an N-body benchmark, including auto-tuning parameterizations, to measure the performance of the Calxeda ARM server. The N-body algorithm has been shown to be useful for benchmarking other platforms for scientific simulations as the N^2 problem requires memory access patterns that are representative of many simulations [11]. Evaluation, results, analysis, and discussion are provided in Section 5. Section 6 presents related work on OpenCL performances. Finally, Section 7 summarizes with conclusions and future work.

2 The Calxeda ARM-based Server

The Calxeda ARM server used for this study was provided by the Exxact Corporation and accessible from a remote desktop allowing direct access to a login node connected to the Calxeda system. It was configured with two EnergyCards, each containing four quad-core ARM system on chip (SoC) devices acting as independent nodes (device nodes), thus, a potential total core count of $2 \times 4 \times 4 = 32$ cores. However, the server that was provided had five device nodes, or 20 cores,

configured and accessible for use. Based on documentation, the ARM cores operate at a clock frequency in the range of 1.1 GHz to 1.4 GHz.

Each quad-core ARM device node was accessible via Secure Shell (SSH) and presented itself as a standalone node running its own operating system (OS) image, specifically, Ubuntu 12.10. The environment is a typical Ubuntu OS system and included standard development tools, e.g., GCC 4.7 along with most of the supporting tools needed for compiling code.

3 Parallel Programming Approach

This investigation demonstrates how to leverage software packages that are freely available from the open-source community to support an OpenCL-based parallel programming model for the Calxeda ARM server.

OpenCL is an industry standard programming API (application programming interface) for parallel programming of heterogeneous computing platforms [13]. 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 (Compute Unified Device Architecture), 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 directly into a large HPC code base raises the issues of long-term development and maintenance.

In this work, STDCL (STandarD Compute Layer) is used for host-side application programming. STDCL is a simplified programming API [3] that leverages OpenCL, yet supports more natural programming syntax and semantics for HPC application development. Note that STDCL provides more than wrappers for OpenCL calls and includes support for default compute contexts, conventional memory allocation of device-sharable memory, OpenCL-based event management, and a dynamic kernel loader that supports a more traditional compilation model and an offline kernel compiler (clcc).

Extensive use of the CO-PRocessing THReads (COPRTHR) software development kit (SDK), developed by Brown Deer Technology and freely available under an open-source license (LGPLv3) [1], aided Calxeda performance examination. 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. The portability of the SDK is limited only by the availability of a suitable OpenCL implementation.

In the case of the Calxeda ARM server investigated here, no OpenCL implementation was provided by the vendor as part of the standard software stack. Therefore, the COPRTHR OpenCL implementation for multi-core CPUs was used for the quad-core ARM processors. In addition, the COPRTHR SDK provides CLRPC, a remote procedure call implementation of OpenCL that may be used to access networked compute devices using the OpenCL API. CLRPC was used to investigate inter-device parallelism within an OpenCL programming model. For host-side programming, the SDK provides an implementation of the STDCL API. STDCL includes default compute contexts for various common processor groupings, e.g., stdcpu (all CPUs), stdgpu (all GPUs), stdacc (all accelerators), as well as an experimental context stdnpu (all networked devices). The stdnpu compute context leverages CLRPC servers to create a a single-compute context for all networked devices, and has no explicit equivalent in the OpenCL API. Considered together, these components provide a novel parallel programming model for the Calxeda ARM servers based entirely upon OpenCL.

The operation of CLRPC is outside of the OpenCL standard and warrants discussion. The OpenCL compute capability of any networked device can be exported using a CLRPC server (clrpcd) running on the host node of the device. A clrpcd server will export all available OpenCL platforms on a given host node. An application code running elsewhere can access all exported OpenCL platforms through libocl.so by providing a list of the network addresses for known servers in an ocl.conf file that replaces the standard OpenCL installable client driver (ICD) enumeration.

4 Auto-Tuning OpenCL N-Body Benchmark

The N-body algorithm is used to solve Newton's laws of motion for N particles subject to an inter-particle force. The algorithm requires the update of all particle positions and velocities based upon 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, and 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 into a compute bound regime since computation and data movement scale as $\mathrm{O}(N^2)$ and $\mathrm{O}(N)$, 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

Parameter	Description
nmulti	Number of particle positions updated concurrently per thread
nunroll	Inner loop unrolling
nblock	Number of particle positions to cache in local memory
nthread	OpenCL work-group size

Table 1. Auto-tuning N-body benchmark parameters

implemented on a range of architectures and provides a convenient canonical algorithm for comparative benchmarking.

This investigation included developing an auto-tuning OpenCL N-body benchmark that parameterized the kernel in ways that impact performance across different architectures and compilers. The original code used as a starting point was provided with the COPRTHR SDK as an example of a GPU compute benchmark and tuned for high-end GPUs. This code was modified to create the auto-tuning benchmark run by specifying ranges for parameter sweeps and allowed to test different parameter combinations, recording the resulting performance. Here, the parameters are used to auto-generate the kernel source as opposed to passing arguments, and as such the code relies upon the just-in-time (JIT) compilation capability supported by STDCL through OpenCL.

The kernel parameterization is summarized in Table 1. The parameter *nmulti* is the number of particles updated per thread. This outer-loop multiplicity may allow 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.

5 Evaluation Study

5.1 Methodology

The OpenCL-based parallel programming model described previously was tested, and the performance and energy usage under different configurations were measured and reported in this section.

Parallel Programmability The software stack described in Section 3 had never been tested on Calxeda ARM server architecture. Therefore, the various components of the OpenCL-based parallel programming model were tested for correct operation. The simplest use cases involved the execution of code using the ARM OpenCL implementation directly (OpenCL direct).

The CLRPC implementation uses a client-server model to export the OpenCL compute device to another node. A CLRPC server (clrpcd) is run on

one of the device nodes enabling a connection over the network. A STDCL host program is then executed from another device node, which provided access to the remote compute device through the CLRPC client OpenCL implementation (CLRPC remote). A variation of this configuration is to execute the clrpcd server and STDCL host program on the same device node thereby allowing access to the local compute device indirectly through CLRPC (CLRPC local).

The STDCL API provides a default compute context, stdnpu, for all networked devices available via CLRPC servers. Whereas CLRPC provides access to networked compute devices as additional platforms, this default STDCL compute context allows a host application to access all devices in a manner no different than a single node containing multiple devices. The exposed parallelism is more transparent than anything available using OpenCL directly.

Performance Performance measurements used an OpenCL N-body benchmark based on default code originally tuned for high-end GPUs. This code was modified to develop the auto-tuning benchmark described in Section 4. This benchmark was used with different configurations and parameters to explore the Calxeda ARM server platform and the OpenCL-based parallel programming model described in Section 3.

The default benchmark was first used on a single device node with the conventional OpenCL configuration involving a STDCL host program using the OpenCL platform implementation directly (OpenCL direct). The auto-tuning version of the benchmark was then used to identify the optimum parameters for the parameterized kernel, anticipating that the search would improve the measured performance of the algorithm.

Performance measurements were obtained using CLRPC which provides a key component for expanding the parallel programmability of the platform beyond a single quad-core device. To address the issue of networking overhead, measurements were taken by running the CLRPC server and STDCL host program on the same device node (CLRPC local) and then repeated by running the CLRPC server on a different device node (CLRPC remote). These tests used the same OpenCL ARM implementation, indirectly, as that used in the previously described OpenCL direct testing.

Power The platform included a baseboard management controller (BMC) that allowed power measurements to be obtained for each ARM device node. Power is measured for more than the quad-core ARM processor itself and includes other elements of the SoC like the network fabric, global memory, etc. Power was measured at idle and under full load while running an N-body benchmark with 32,768 particles so that the device would be driven well into the compute regime and under full utilization.

5.2 Results

The default N-body benchmark was executed with 16,384 particles and nthread=16 using the OpenCL direct configuration and exhibited 983 MFLOPS

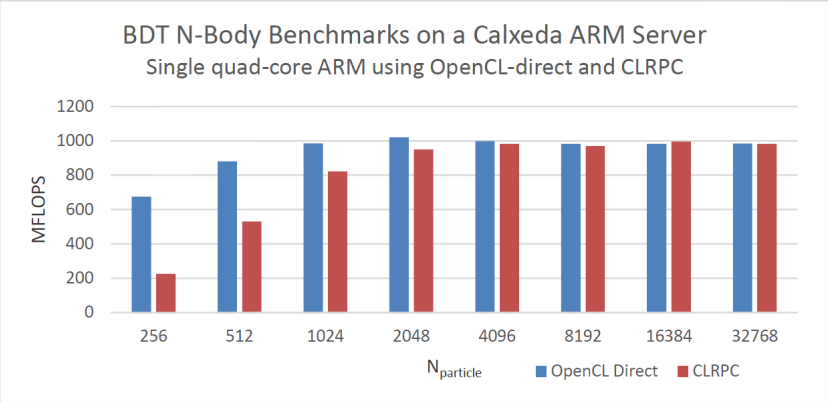

Fig. 1. Performance results for BDT N-body on a single quad-core ARM device on a Calxeda server: OpenCL directly versus Networked device with CLRPC

(million floating point operations per second). With this number of particles, the simulation is driven well into a compute regime on this architecture, as discussed more thoroughly later. Examining CPU utilization using the Linux top command showed nearly 400% utilization, indicating that all four ARM cores were utilized at full load. The auto-tuning benchmark was run using the same number of particles and identified an optimized kernel that exhibited 1094 MFLOPS, a gain of 11.3% over the default benchmark.

Using the CLRPC local configuration, where the clrpcd server and STDCL host program are run on the same device node, the identical default benchmark exhibited 840 MFLOPS. Examining CPU utilization showed that the clrpcd server was running at approximately 350% with the host program using the balance of approximately 50%. With this configuration, the clrpcd server executes the computational load, and the host program drives the computation through RPC client calls. The results provide an indication of the host program overhead incurred when using CLRPC. The reduction in performance (15%) is consistent with the reduction in utilization (13%).

Using the CLRPC remote configuration, where the clrpcd server and STDCL host program are run on different device nodes, the identical benchmark exhibited 980 MFLOPS and the clrpcd server showed nearly 400% utilization. This demonstrates that the resource contention found with the CLRPC local configuration can be entirely mitigated when the clrpcd server is run on a separate device node and the CLRPC overhead remains with the host program.

Figure 1 shows results for different numbers of particles. Using the OpenCL direct configuration, the benchmark reaches a pure compute regime at around 1024 particles. For smaller numbers of particles, the performance is negatively impacted by non-compute operations. This reduced performance for small numbers of particles is observed on nearly all architectures and is not surprising.

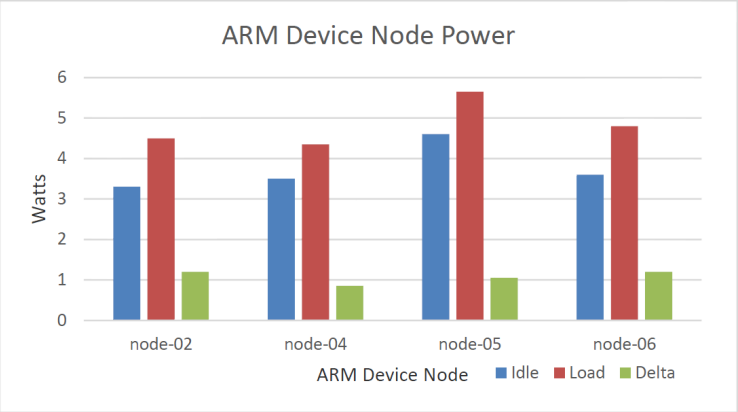

Fig. 2. Calxeda power requirements at idle (blue) and under load (orange) while executing N-body application. The difference between idle and load are indicated by green bar.

Results using the CLRPC remote configuration are also shown in Figure 1, where it can be seen that a greater overhead is incurred for small numbers of particles. Not only does the benchmark show that it reaches a compute regime at a slightly larger number of particles by roughly a factor of two, there is a greater decrease in performance as the number of particles is reduced.

Two factors are likely impacting the performance. First, the relative cost of data movement and compute will decrease linearly with the number of particles, being O(N) and O(N^2), respectively. Second, there is is an overhead associated with the CLRPC implementation that will become more pronounced as the computational load is reduced. Nevertheless, the results show that CLRPC can be used effectively for computationally intensive tasks.

Power measurements are shown in Figure 2 and indicate a power variance between ARM SoCs that is significant, with a minimum (maximum) idle power of 3.3 W (4.6 W). Under full load, the power increase is reasonably consistent with a minimum (maximum) cost of an additional 0.85 W (1.2 W).

The STDCL default compute context stdnpu containing all networked devices was tested by performing a simple calculation using all available quad-core ARM devices on the server. A simple STDCL test program that calculated pi was run across all five ARM processors successfully. The calculation itself is very limited in computational load and does not provide a reliable performance benchmark. However, the test was successful in evaluating the functional capability of using all ARM devices cooperatively in a calculation from a single STDCL compute context. This capability will be explored more in future work.

5.3 Analysis and Discussion

The Calxeda ARM server provides a platform that is functionally equivalent to a small cluster of quad-core processors, each running a standard Linux OS. Using

an OpenCL N-body benchmark, the ARM processor exhibited the computational performance of approximately 1 GFLOP. The ARM SoC device nodes used 3.3 W-4.6 W of power when idle and 4.35 W-5.65 W under full load. Therefore, the power efficiency is approximately 1 GFLOPS/W in terms of processor power and approximately 0.2 GFLOPS/W in terms of total node power.

By comparison, using the auto-tuning N-body benchmark on a system with dual Intel Xeon 5650 CPUs exhibits 83.7 GFLOPS with a combined thermal design power (TDP) of 190 W for a power efficiency of approximately 0.4 GFLOPS/W. Using an AMD Radeon HD 6970 GPU, the benchmark exhibits 1358 GFLOPS with a TDP of 250 W for a power efficiency of 5.4 GFLOPS/W. Based on this data, it remains difficult to assess the relative power efficiency of the quad-core ARM SoC device since the measured power includes components that do not align with the power measurements of other architectures.

The Calxeda system supported the use of CLRPC to allow OpenCL applications to access any networked OpenCL devices. CLRPC performance shows some overhead for workloads with light-weight kernels, and matches the performance of using OpenCL directly for computationally intensive workloads.

6 Related Work

Others have developed test suites that cover combinations of OpenCL operations, element types, and local sizes to test performance and numerical precision of GPU systems in terms of OpenCL operations [9].

In comparative performance studies between CUDA and OpenCL versions of the same applications, Fang et.al [6] and Komatsu et.al. [10] have shown that performance can be comparable if the kernels are optimized by hand or by compiler optimizations. They also showed 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 for comparative performance studies. Yao et.al. [14] 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. Other studies of OpenCL performance include comparing against OpenMP for multi-core CPUs [12].

7 Conclusions and Future Work

To our knowledge, this is the first effort to investigate the use of OpenCL for HPC on a Calxeda ARM server. The initial success of applying a novel OpenCL-based parallel programming model that includes the use of a high-level abstraction for OpenCL and an RPC implementation of OpenCL to access networked compute devices demonstrates the potential of this model for utilizing the large number of ARM cores available on a fully configured Calxeda system. Empirical results

are obtained for executing an OpenCL N-body benchmark using various configurations and for a range of parameters from which the power efficiency of the quad-core ARM processors could be evaluated and compared with other architectures. An auto-tuning benchmark was developed that can be used to optimize the computational kernel for a given architecture. Power efficiency results are not sufficient to make a determination as to the energy efficiency of an ARM-based server compared with competing architectures. At issue are the comparisons between platforms that include or exclude a different mix of components. Thus, the application of an ARM-based server from an energy efficiency standpoint remains an open question.

References

1. COPRTHR SDK. www.browndeertechnology.com/coprthr.htm (2012)
2. Borkar, S.: The exascale challenge. In: VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on. pp. 2–3 (2010)
3. Brown Deer Technology: STDCL: A Simplified C Interface for OpenCL (2012), revision 1.4
4. Calxeda: Calxeda launches the energycore processor; delivers 10 times the performance for the same power. www.calxeda.com (2013)
5. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone. In: Proceedings of the 2010 USENIX conference on USENIX annual technical conference. pp. 21–21. USENIXATC'10, USENIX Association, Berkeley, CA, USA (2010)
6. Fang, J., Varbanescu, A.L., Sips, H.: A comprehensive performance comparison of CUDA and OpenCL. In: Proceedings of the 2011 International Conference on Parallel Processing. pp. 216–225. ICPP '11, IEEE Computer Society, Washington, DC, USA (2011)
7. Furber, S.: ARM System on Chip Architecture. ADDISON WESLEY Publishing Company Incorporated (2000)
8. Gioiosa, R.: Towards sustainable exascale computing. In: VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP. pp. 270–275 (2010)
9. Jurecko, M., Kocisová, J., Jr., J.B., Kasanický, T., Domiter, M., Zvada, M.: Evaluation framework for GPU performance based on OpenCL standard. In: ICNC. pp. 256–261 (2010)
10. Komatsu, K., Sato, K., Arai, Y., Koyama, K., Takizawa, H., Kobayashi, H.: Evaluating performance and portability of OpenCL programs. In: The Fifth International Workshop on Automatic Performance Tuning (June 2010)
11. Playne, D.P., Johnson, M.G.B., Hawick, K.A.: Benchmarking GPU devices with N-body simulations. In: Proc. 2009 International Conference on Computer Design (CDES 09). pp. 150–156. WorldComp, Las Vegas, USA (13-16 July 2009)
12. Shen, J., Fang, J., Sips, H.J., Varbanescu, A.L.: Performance gaps between OpenMP and OpenCL for multi-core CPUs. In: ICPP Workshops. pp. 116–125. IEEE Computer Society (2012)
13. Stone, J.E., Gohara, D., Shi, G.: OpenCL: A parallel programming standard for heterogeneous computing systems. IEEE Des. Test 12(3), 66–73 (May 2010)
14. Yao Zhang, M.S.I., Chien, A.A.: Improving performance portability in OpenCL programs (2013)