Transitioning HPC Software to Exascale Heterogeneous Computing

Wen-Mei Hwu, Li-Wen Chang, Hee-Seok Kim, Abdul Dakkak, and Izzat El Hajj

Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, IL 61801, USA
w-hwu@illinois.edu

Abstract— **An increasing portion of the top supercomputers in the world, including Blue Waters, have heterogeneous CPU-GPU computational units. As we move towards exascale, we can expect even more pervasive deployment of heterogeneous computational units. While a handful of science teams can already use heterogeneous computational units in their production applications, there is still significant room for the growing use. This paper presents the current state and projected path for transitioning software into this new paradigm. We first summarize the currently practical languages such as OpenCL, OpenACC, and C++AMP, in increasing levels of productivity, highlighting their recent advancements in supporting performance portability and maintainability. We will then give a brief overview of some emerging programming systems such as TANGRAM and Troilet that are designed to further enhance developer productivity for heterogeneous computing.**

I. Introduction

It has been almost five years since the top supercomputers in the world started to employ GPU computing. In Blue Waters, deployed in March 2013 at 12.4 GFLOPS, there are two types of computing nodes. The first type (XE6) consists of two AMD Interlogos CPUs with a total of 16-core modules, 0.313 double precision teraflops, 64 GB of DRAM, and 102 GB/s DRAM bandwidth. The second node type (XK7) consists of one AMD Interlogos CPU and one NVIDIA Kepler GPU with a total of 1.56 double precision teraflops, 38 GB of DRAM, 241 GB/s DRAM bandwidth. Even though the second node type has 5X peak compute rate and 2.4X DRAM bandwidth, the power consumption ratings of the two node types are comparable.

In 2013, a team of Illinois researchers, NVIDIA engineers, and scientific developers made a collaborative effort to use GPU computing in Blue Waters for three important science applications that received significant portion of the Blue Waters time allocation: NAMD, Chroma, and QMCPACK. For each application, the execution time is measured from application launch to exit, including all I/O times. The execution time of each application is measured when running on all XE6 nodes and then measured when running on all XK7 nodes.

For NAMD, a test run is performed for a 100-million-atom benchmark with Langevin dynamics and particle mesh Ewald (PME) once every four time steps, using 768 nodes. The total execution time using XE6 nodes (2 CPUs) is 1.8 times longer than using XK7 nodes (1 CPU + 1 GPU). The main part executed by the GPUs is the remote force calculation. While the computing-performance and power-efficiency benefit of using GPUs is very clear for this application, it is actually limited by the PME global communication. We can expect that the benefit of GPU computing will be even better as the NAMD team develop more efficient global communication schemes.

For Chroma, a test is performed with lattice quantum chromo dynamics (QCD) grid size of $48^3 \times 512$ running at the physical values of the quark masses using 768 nodes. The total execution time using XE6 nodes is 2.4 times longer than that of using XK7 nodes. The GPU execution is enabled by calling the GPU-enabled USQCD library, which is a production library used in multiple QCD applications including Chroma.

For QMCPACK, the experiment is done with full run of Graphite $4 \times 4 \times 1$ (256 electrons), with the QMC step followed by the VMC step, using 700 nodes. The total execution time using XE6 nodes is 2.7 longer than that of using XK7 nodes. The benefit of using GPUs is even better than Chroma.

An important aspect of the experiment is that the GPU execution is done in the production code. This ensures that the science teams can see the same benefit from the GPUs in their production runs. All three applications remain as the top users of the GPU XK7 nodes in Blue Waters to date.

These results show that heterogeneous computing systems can indeed offer superior power efficiency for future computing systems. Indeed, heterogeneity is becoming ubiquitous in modern computing systems, ranging from low-power mobile devices to high-performance supercomputers. The biggest challenge, or "elephant in the room," is software cost. Much of the software cost comes with the difficulty in developing and maintaining the customised versions of application code that can deliver the performance-power benefit of heterogeneous computing systems.

Ideally, applications running in heterogeneous systems should exhibit performance portability. That is, they should be able to achieve high performance on different existing and future devices without significant software redevelopment. Performance portability is challenging because portability requires architecture-neutral program representation while performance requires architecture-specific customization and tuning. Balancing these two conflicting goals is the dilemma of heterogeneous programming systems. As a result, current programming solutions for the heterogeneous computing systems require customized code redevelopment for each type of hardware being targeted.

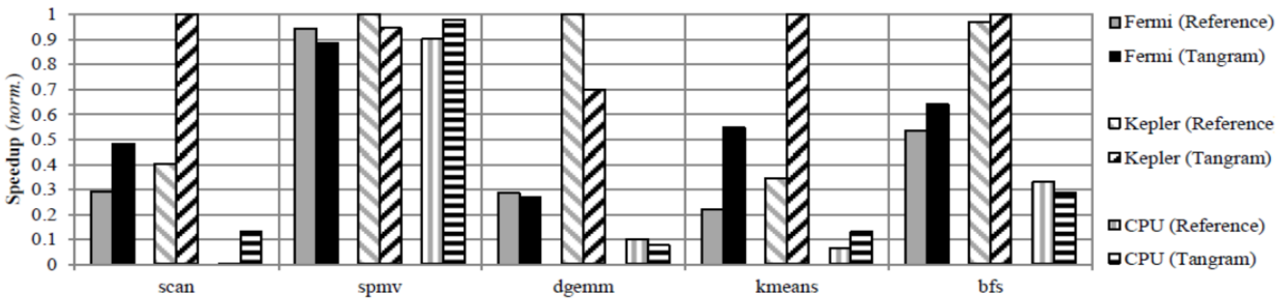

Fig. 1. Comparison of TANGRAM-synthesized code with vendor libraries and hand-optimized code (reference).

II. Near-Term Solutions

In the near future, the industry will have three levels of solutions to the programming challenges of heterogeneous computing systems. OpenCL is geared towards library developers who would like to have detailed program control of hardware execution resources. The OpenCL programming interface is supported for a wide range of CPU, GPU, and FPGA devices. Previously, functions written in OpenCL tend to perform poorly on CPUs due to scheduling approaches that result in poor cache performance. The recent work by the MxPA team [1] has significantly improved the CPU cache and total performance for many OpenCL functions.

A major deficiency of the OpenCL programming interface is that it requires tedious changes to array indexing and explicit code for data transfers. This deficiency is addressed by the OpenACC programming interfaces, where developers annotate each loop to be executed by GPU as OpenACC parallel loop. This way, the developers do not need to rewrite their loops for GPU execution. The compiler translates original indices into the GPU indices and generates the detailed data transfers.

One common deficiency of both OpenCL and OpenACC is that they are not compatible with the modern object-oriented programming style. Furthermore, the annotation approach in OpenACC is hard to debug since the annotation is not part of the C language. This deficiency is corrected in the Kalmar C++AMP programming interface, where the C++ lambda mechanism used to automatically extract a parallel loop from a C++ class method body. All data transfers are also automatically handled by compiler code generation. This will further reduce the software development and maintenance code of heterogeneous computing.

III. Long-Term Outlook

All the solutions so far have a common deficiency: the algorithm and data arrangement schemes are baked into the code. This intrinsically limits the performance portability of many types of computation. A new library development system called TANGRAM [2] is designed to address this problem. In TANGRAM, a codelet is a small piece of code that implements a particular computation. A spectrum is a computation associated with a collection of functionally equivalent codelets. Codelets in the same spectrum have the same name and function signature, but have different implementations. They are implemented either by using different algorithms or by using the same algorithm with different optimization techniques.

There are different types of codelets. Codelets can be atomic or compound. Atomic codelets are self-contained while compound codelets invoke other spectrums. Compound codeletes can be recursive if they invoke their own spectrum. Alternative compound codelets implement different data tiling strategies and recursively call the compound or atomic codelets. Codelets can be scalar or vector. Both scalar and vector codelets operate on scalar values and both are vectorizable. The difference is that scalar codelets are oblivious to other elements of the same vector, whereas vector codelets can communicate with the other elements. In other words, vector codelets may perform actions unique to vector execution and are not meant to be scalarized.

The TANGRAM compiler heuristically searches a space of all possible combinations and recursions of the codelets to identify the code designs that are the best match for each given hardware type. In some cases, there may be multiple designs that can potentially have good match with a hardware type, sometimes the best choice may depend on the shape of an input data. The TANGRAM runtime makes the final test and choice during the actual execution. Figure 1 shows the total execution time of TANGRAM-synthesized library code against vendor library code or hand-optimizied code on GPUs and CPUs. In most cases, TANGRAM code either exceeds or comes close to the speed of the hand-optimized code. For any new hardware type or future generations of hardware, the TANGRAM code will likely significantly outperform existing hand-optimized code.

IV. Conclusion

Advances in programming systems will continue to reduce the cost of software development and maintenance for heterogeneous computing. In the short term, products like MxPAOpenCL and Kalmar C++ will improve the development climate. In the long term, code synthesis approaches like TANGRAM will prevail in practice.

References

[1] H.-S. Kim, I. El Hajj, J. A. Stratton, S. S. Lumetta, and W.-M. Hwu, “Locality-centric thread scheduling for bulk-synchronous programming models on CPU architectures,” *IEEE/ACM Int. Symp. on Code Generation and Optimizaion (CGO)*, San Francisco, CA, 2015.

[2] L.-W. Chang, A. Dakkak, C. I. Rodrigues, and W.-M. Hwu, “Tangram: a high-level language for performance portable code synthesis,” *hiPEAC MULTIPROG Workshop*, Amsterdam, The Netherlands, 2015.