Research Statement

Thomas R. W. Scogland
tom.scogland@vt.edu

1 Motivation

Over the past several years, the average computer has shifted from a serial machine with a single processor into a highly complex parallel system. The rise in intra-node complexity is an unavoidable consequence of the industry-wide trend toward parallelism as the main driver of performance in new processor and system designs. While we classically expect our supercomputers and high performance workstations to have complex architectures, the heterogeneity is now increasing throughout the computing ecosystem. Devices from supercomputer nodes to cell phones, CAD workstations to smart watches now carry multiple CPU cores. Beyond the expansion to parallelism, computational coprocessors with separate instruction sets, architectures and programming models are now being employed for general purpose computation. Coprocessors power the fastest ranked supercomputer in the world as of this writing. They also accelerate operating system functions of the laptop this document was written on, and the smartphone nearby.

Heterogeneity has become a fact of life in modern computing, and it only serves to complicate the already daunting task of bringing parallel computing into the mainstream. A modern application must employ parallelism in order to attain reasonable performance. Even then, it must adapt to a myriad of possible hardware targets and further to the shifting capabilities of each piece of hardware at runtime. If it does not, then when run on systems with different capabilities, different numbers or types of GPUs or coprocessors or CPUs for example, one or more of the systems will be underutilized, and prevent the application from being performance portable.

Automatically and transparently adapting applications to system resources is now critically important. My research focuses on adapting application behavior through affinity mapping [9], scheduling and work-sharing [10, 14] across potentially heterogeneous systems. In support of these goals, I also perform research in architecture specific optimization [1–5], benchmarking [7], concurrent data structures and energy efficiency [6, 8, 11, 13–15]. While each of these is a separate component, they all support the goal of creating an efficient, autonomous runtime to provide performance portability and programmability to high performance computing. In the following sections I will describe my work up to this point, my current work, and what I plan to pursue going forward.

2 Process Affinity in MPI

In some cases, the heterogeneity of a system is not inherent in its hardware, but caused by imbalances in the operating system or locality. The beginning of my investigation into runtime adaptation systems began with a homogeneous system that did not act like one. Repeated runs of a simple point-to-point network bandwidth test on this physically homogeneous system would yield results varying by as much as 30%. Despite the physical homogeneity of the machines being tested, the core used to run the bandwidth test on each side was enough to produce significant variance. In this first case, the operating system's handling of a high volume of network interrupts caused one core to have extremely high bandwidth, one low, and two in the middle along with similar effects on the compute performance on each core as well. With the cause found for this case, the question then became, can these imbalances be quantified, detected and mitigated automatically?

From that question we developed the Systems Mapping Manager (SyMMer) [9] runtime as part of MPICH2. SyMMer is designed to detect imbalances caused by a high speed network interface, as well as natural imbalances in the compute phases of the application being run and frequent passing of messages between local ranks that may benefit from locality. Based on a set of heuristics keyed on performance counters, MPI buffering behavior and time spent in communication and computation phases, SyMMer remaps ranks across cores in a node to achieve consistently high performance. In addition, it automatically resolves certain issues caused by imbalance, even problems that could only be detected across multiple nodes in a distributed system. The overall effect was highly positive for our test benchmarks including GROMACS achieving 10-15% better performance, LAMPPS reducing communication time by 50% and the MPI implementation of the FFTW library running 3-5% faster overall.

3 Workload Scheduling Across Heterogeneous Systems

Physically heterogeneous systems present a different kind of challenge. Rather than hiding the heterogeneity from the user, they require a user to explicitly implement their algorithm for each type of device in order to use it. In the best case, as with programming models like Accelerated OpenMP, a user can annotate a single source implementation to offload a portion of the workload to another device. Even in accelerated OpenMP however, there is no mechanism to work-share a parallel region across all available resources, CPUs, GPUs, and coprocessors alike, losing one of the greatest features of OpenMP on the CPU, the ability to transparently adapt to nearly any number of available cores.

In fact, current programming models offer neither worksharing across CPUs and GPUs nor work-sharing across multiple GPUs. This is a result of the fact that their work-sharing constructs do not cross address space boundaries. Some systems, such as StarPU and OmpSs, offer automatic load balancing of user-specified discrete tasks, but not work-sharing in the style of OpenMP. If such is desired, users must implement it themselves. This can be a very daunting task, and generally results in static scheduling if it is done at all.

In order to fill this void, I have created a runtime known as the Splitter library in [12] and renamed to CoreTSAR in my continued work in [10], that is compatible with any CUDA/C code or accelerated OpenMP implementation that emits CUDA compatible code. This implementation operates on the principle that each iteration in an accelerated loop construct may be treated as separate, though related, task and scheduled on GPUs or CPUs at will, assuming its inputs and outputs are made available.

The main components of the system are a memory manager, with an interface designed to associate data to computation, and a scheduling/load-balancing system that models a device's performance on a particular kernel and determines how to divide work across devices. Since the input and output data are specified in terms of the computation, the data required by a subset of that computation can be computed by the runtime and transferred to and from any memory space in the system at will. Combining that expressive power with the scheduling system, which implements a set of predictive and work-queue-based designs, allows work to be efficiently distributed across CPUs and GPUs. Given the work to be run on each device, the system automatically distributes the appropriate portion of memory to each device, and combines the results afterward as though all of the work were done in a single memory space.

To evaluate the prototype, I implemented the necessary modifications in five benchmarks; GEM, an n-body molecular modeling application; cg, the conjugate gradient benchmark from NPB; kmeans, the classic feature clustering algorithm; Helmholtz, a discrete finite difference code implementing the Helmholtz equation in a very GPU-unfriendly way; and CORR, an upper-triangular matrix transformation from the polybench-GPU suite. Full results are available in the paper [10], but CoreTSAR achieves nearly linear speedups adding in further devices in amenable benchmarks, and demonstrates a capability to back off of GPU devices if they are determined to harm performance. Perhaps the biggest performance advantage of this approach is the small runtime scheduling overhead. Since a parallel region is treated as a single block of work consisting of many related tasks, as opposed to a set of distinct tasks to be scheduled independently, many of the costs associated with task scheduling are reduced or removed.

4 Current Research: Affinity Mapping in Heterogeneous Systems

In my work to date, I have shown that heterogeneous scheduling can improve the performance and programmability of software by adapting the application to the system's capabilities at runtime. Each of my previous systems targeted a particular issue and approach. SyMMer improves the locality of an application by re-mapping MPI processes for improved affinity. CoreTSAR improves load balance and system utilization by controlling the amount of work assigned to each resource. These are complementary goals, which should be addressed together. Unfortunately SyMMer, operating on MPI processes (one per core), and CoreTSAR, operating in OpenMP, are fundamentally incompatible.

I am currently working on combining these approaches to produce a system offering the benefits of both together. Rather than controlling locality by process re-mapping, as SyMMer does, AffinityTSAR *re-maps and transforms memory and workload*. Adding the ability to arbitrarily and automatically transform the memory operated on by a parallel region, without breaking the target compute code, allows for automatic replication, data packing, blocking (even if the input data is not itself blocked) and a variety of other transformations. This allows the runtime to address all the issues SyMMer and CoreTSAR targeted and also far greater capacity for automated optimization of memory management. Preliminary results with the design, using CPU cores only in a NUMA system, recently found a nearly 50% increase in performance for a generalized matrix multiply by remapping data that was allocated on inappropriate memory nodes without any modification to the compute code.

5 Future Research

In the coming years, I am interested in further exploring the potential for automated management of heterogeneity. Up until now, all of my work has focused on a model of coprocessors, generally devices like GPUs, that assumes all work must be "pushed" to them, or explicitly sent to them by the CPU. Recent enhancements in the hardware from major GPU vendors, and through slightly different means on coprocessors like Intel Xeon Phi or Texas Instruments DSPs as well, allow atomic operations and synchronization across *all devices in a system.* Given these, concurrent data structures can be created that are accessible by all devices simultaneously and safely. This in turn allows the creation of a common work-list or graph that all threads, all devices in general in a system, could pull work from. The potential benefits are many, including increased programmability, decreased device downtime, and decreased dependence of computational coprocessors on direct CPU control while allowing for far finer-grained load-balancing to take place. More generally, I intend to investigate the potential for creating a common model for expressing computation and data movement to be consumed by the entire system.

Just as my work has focused on a push-based model of work distribution, it has also focused specifically on mitigating the effects of heterogeneity *within a node* and at a single level of heterogeneity. The problem of scheduling and load-balancing is really a hierarchical one, how much work should be distributed to a given rack, or node, or processor in that node are all really independent problems. A hierarchical scheduler, designed to nest either inside itself or with others, could allow each level to be treated independently and generate a balanced plan at multiple levels. It would likely begin with scheduling across sockets in a single system, then applying a nested scheduler inside to determine what work to provide each CPU core and coprocessor, but could grow to larger-scale systems as well, such as heterogeneous clusters, clouds or grids.

This work, along with what I have done to date, will serve as the underpinning for my longer-term research goal. That goal is to develop a method of expressing computation, and the attendant communication, patterns in such a way that the resulting pattern can be exploited to naturally express parallelism and assist in scheduling and synchronization decisions. Much in the way Map-Reduce has become a powerful and popular programming model thanks to providing a commonly used communication pattern that users can fit their computation into, other patterns can be generalized to similarly powerful effect. N-body for example can be represented by a simple abstract pattern, and reasoned about in terms of its computation and communication, as can structured grids, or even dynamic programming or graph traversal applications. The more of these patterns that can be developed the more feasible it would be to have a scientist, or

developer in general, import the pattern like a library and target systems of nearly any size or complexity transparently.

References

[1] R. Anandakrishnan, T. R. W. Scogland, A. T. Fenley, J. C. Gordon, W.-c. Feng, and A. V. Onufriev. Accelerating Electrostatic Surface Potential Calculation with Multi-scale Approximation on Graphics Processing Units. *Journal of Molecular Graphics and Modelling*, 28(8):904–910, June 2010.

[2] J. Archuleta, Y. Cao, T. R. W. Scogland, and W.-c. Feng. Multi-dimensional Characterization of Temporal Data Mining on Graphics Processors. *IPDPS '09: Proceedings of the 2009 IEEE International Parallel and Distributed Processing Symposium*, pages 1–12, 2009.

[3] M. Daga, W.-c. Feng, and T. R. W. Scogland. Towards accelerating molecular modeling via multi-scale approximation on a GPU. In *International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*, pages 75–80, 2011.

[4] M. Daga, T. R. W. Scogland, and W.-c. Feng. Architecture-Aware Mapping and Optimization on a 1600-Core GPU. In *International Conference on Parallel and Distributed Systems*, pages 316–323, Tainan, Taiwan, 2011. IEEE Computer Society.

[5] M. Elteir, H. Lin, W.-c. Feng, and T. R. W. Scogland. StreamMR: An Optimized MapReduce Framework for AMD GPUs. In *International Conference on Parallel and Distributed Systems*, pages 364–371, 2011.

[6] W.-c. Feng, K. Cameron, and T. R. W. Scogland. The Green500 List: A look back to look forward. In J. S. Vetter, editor, *Contemporary High Performance Computing From Petascale toward Exascale*, pages 31–41. Chapman and Hall/CRC, Jan. 2013.

[7] W.-c. Feng, H. Lin, T. R. W. Scogland, and J. Zhang. OpenCL and the 13 Dwarfs: a Work in Progress. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, Boston, MA, Apr. 2012. ACM Request Permissions.

[8] W.-c. Feng and T. R. W. Scogland. The Green500 List: Year One. In *High-Performance, Power-Aware Computing Workshop (IPDPSW: HPPAC)*, Rome, Italy, May 2009. IEEE Computer Society.

[9] T. R. W. Scogland, P. Balaji, W.-c. Feng, and G. Narayanaswamy. Asymmetric Interactions in Symmetric Multi-Core Systems: Analysis, Enhancements and Evaluation. In *ACM/IEEE Supercomputing Conference*, pages 1–12, 2008.

[10] T. R. W. Scogland, W.-c. Feng, B. Rountree, and B. R. de Supinski. CoreTSAR: Task Scheduling for Accelerator-aware Runtimes. Technical Report TR-12-20, 2012.

[11] T. R. W. Scogland, H. Lin, and W.-c. Feng. A First Look at Integrated GPUs for Green High-performance Computing. *Computer Science-Research and Development*, 25(3-4):125–134, Aug. 2010.

[12] T. R. W. Scogland, B. Rountree, W.-c. Feng, and B. R. de Supinski. Heterogeneous Task Scheduling for Accelerated OpenMP. In *International Parallel and Distributed Processing Symposium*, pages 144–155. IEEE Computer Society, May 2012.

[13] T. R. W. Scogland, B. Subramaniam, and W.-c. Feng. Emerging Trends on the Evolving Green500: Year Three. In *High-Performance, Power-Aware Computing Workshop (IPDPSW: HPPAC)*, pages 822–828. IEEE Computer Society, May 2011.

[14] T. R. W. Scogland, B. Subramaniam, and W.-c. Feng. The Green500 List: Escapades to Exascale. *Computer Science-Research and Development*, pages 1–9, 2012.

[15] B. Subramaniam, W. Saunders, T. R. W. Scogland, and W.-c. Feng. Trends in energy-efficient computing: A perspective from the Green500. In *International Green Computing Conference (IGCC)*, pages 1–8, 2013.