Intelligent Optimization of Parallel and Distributed Applications

Bhupesh Bansal[2], Umit Catalyurek[3], Jacqueline Chame[1], Chun Chen[1], Ewa Deelman[1], Yolanda Gil[1], Mary Hall[1], Vijay Kumar[3], Tahsin Kurc,[3] Kristina Lerman[1], Aiichiro Nakano[2], Yoon-ju Lee Nelson[1], Joel Saltz[3], Ashish Sharma[3], Priya Vashishta[2]

[1]University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
{jchame,chunchen,deelman,gil,mhall,lerman}@isi.edu

[2] University of Southern California
Department of Physics and Astronomy
Vivian Hall of Engineering, 610
Los Angeles, CA 90089-0242
{bansal,anakano,priyav}@usc.edu

Department of Biomedical Informatics
The Ohio State University
333 West 10th Avenue
Columbus, OH 43210
{umit,vijayskumar,kurc,saltz,ashish}@bmi.osu.edu

Abstract

This paper describes a new project that systematically addresses the enormous complexity of mapping applications to current and future parallel platforms. By integrating the system layers – domain-specific environment, application program, compiler, run-time environment, performance models and simulation, and workflow manager – and through a systematic strategy for application mapping, our approach will exploit the vast machine resources available in such parallel platforms to dramatically increase the productivity of application programmers. This project brings together computer scientists in the areas represented by the system layers (i.e., language extensions, compilers, run-time systems, workflows) together with expertise in knowledge representation and machine learning. With expert domain scientists in molecular dynamics (MD) simulation, we are developing our approach in the context of a specific application class which already targets environments consisting of several hundreds of processors. In this way, we gain valuable insight into a generalizable strategy, while simultaneously producing performance benefits for existing and important applications.

1 Introduction

Scientists are conducting data analysis of unprecedented complexity and scale. Many scientific applications are being built not as monolithic entities, but rather by combining models and analysis routines contributed by many scientists, specializing in different areas of the problem. Resulting applications can be defined as workflows composed of hundreds or thousands of components to be executed in coordination on a variety of resources. The application components may have different performance characteristics and resource requirements and some require very specialized, high-performance resources to achieve reasonable performance.

Efficient, robust execution of application workflows in heterogeneous, distributed environments is composed of a set of problems at different scales-from low-level architecture-specific optimizations to utilize the memory hierarchy and individual processor, effective parallelization, on up to high-level application composition. In the proposed research, we view application optimization as hierarchical, consisting of optimization of individual components, and the composition of components into a workflow. Each component comprising a workflow must be able to execute efficiently on the target architecture(s), and under a variety of execution environment conditions, such as resource constraints and data set characteristics. In turn, these performance metrics depend on a variety of application-level features and the set of transformations applied by the compiler. Further, the components must be composed in such a way that the solution performs well globally and makes productive use of valuable computing resources.

In this paper, we describe a systematic solution for performance optimization and adaptive application mapping – a large step towards automating a process that is currently performed in an ad hoc way by programmers and compilers – so that it is feasible to obtain scalable performance on parallel and distributed systems consisting of tens of thousands of processing nodes. The application components will be viewed as dynamically adaptive algorithms for which there exist a set of *variants* and *parameters* that can be chosen to develop an optimized implementation. A variant describes a distinct implementation of a code segment, perhaps even a different algorithm. A parameter is an unbound variable that affects application performance. Variants and parameters are specified by users or derived by the compiler. An instance of the application can be viewed as a workflow where the nodes represent the application components and dependences between the nodes represent execution ordering constraints. By encoding an application in this way, we can capture a large set of possible application mappings with a very compact representation. The application programmer relies on the system layers to explore the large space of possible implementations to derive the most appropriate solution. Because the space of mappings is prohibitively large, the system captures and utilizes domain knowledge from the domain scientists and designers of the compiler, run-time and performance models to prune most of the possible implementations. Knowledge representation and machine learning techniques utilize this domain knowledge and past experience to navigate the search space efficiently.

2 Motivating Example: MD Simulation

Our goal is a general strategy towards systematic workflow optimization. To focus our efforts, we are developing this strategy in the context of a specific class of applications, namely molecular dynamics (MD) simulations. MD simulations involve deriving the phase-space trajectories of the system in terms of the positions and velocities of all particles at all times [15]. Different algorithm *variants* are used, depending on the number of particles N, their distribution (uniform/nonuniform), and boundary conditions, while the parallel performance of algorithms will depend strongly on the granularity N/P, for processors P [16, 14]. Furthermore, within a given algorithm, there are a number of *parameters* that may affect performance, such as the size of a "cell" or collection of particles.

Currently, algorithm variants, computational parameters and workflow mappings for MD are derived empirically through expert knowledge and limited sets of performance measurements. Compiler variants and parameters are selected based on compiler models, which are often derived statically and based on conservative approximations. As the optimization space grows rapidly on scalable systems, such ad hoc approaches to application development and optimization are becoming increasingly inadequate. Further, there is a need for application programmers, compilers and run-time systems to work in collaboration, rather than independently.

The main objective of this project is to develop a systematic framework and set of tools for automatic, optimized mappings of MD simulation workflows. From experience with MD simulation, we will develop a generalizable framework to optimize workflows for large-scale parallel systems. Because we are basing this framework on existing tools in wide use, this system will lead to a research vehicle for building general productivity tools for parallel systems.

3 System Overview

An overview of the proposed system is shown in Figure 1. The key innovation is the incorporation of a pair of search engines that search a set of alternative mappings of the application to the hardware to select the implementation that has the best overall performance for a specific problem instance. A search engine collaborates with the compiler and run-time environment to derive the best implementation of specific application components for a particular target environment, using both models and empirical data. Another search engine considers the space of possible mappings of workflow components to available resources, guided by heuristics for local optimizations, to derive a mapping that results in the smallest execution time for the overall workflow. Domain knowledge of the programmer is captured through the specification of algorithm variants, computational parameters and expected resource requirements. The compiler processes these specifications and provides compiler variants and parameters for a set of target architectures. The workflow mapping composes the optimized components into a final implementation.

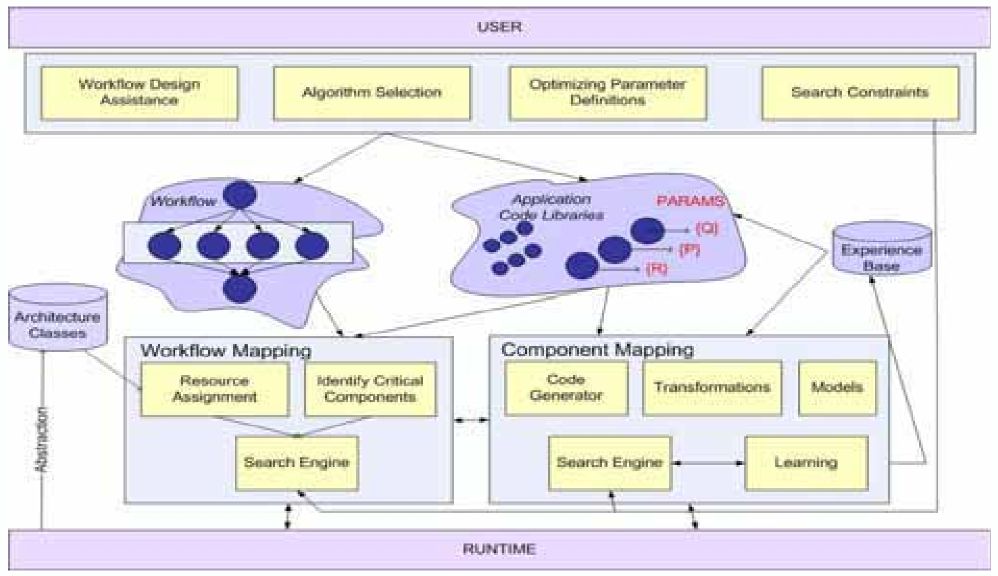

Figure 1. Overview of System.

Developing a systematic approach to workflow optimization is essential to productive utilization of the vast machine resources of large-scale parallel platforms with thousands of processors, but deriving a simultaneously effective and practical approach is quite challenging. Even with thousands of processors, empirical search of a large number of alternative implementations of a computation is infeasible. To be successful, this research program must address the following considerable technical challenges:

- Workflow design and mapping techniques that capture optimization decisions and component choices.
- A set of linguistic mechanisms that allow the application programmer to specify algorithm variants and computational parameters at a very high level.
- Capturing the optimization parameters and workflow properties for the MD algorithms.
- Formulation and representation of component optimization as a machine-learning problem.
- Identification of search algorithms to efficiently navigate the search space of various optimizations.
- Compiler technology to decouple analyses and code transformations from optimization.
- Appropriate models to guide search algorithms and prune the search space.
- An efficient approach to dynamically generate or select component implementations

In the remainder of the paper, we will discuss the technology that is used in our system, and the progress in the early months of the project on addressing these challenges.

4 Technology Development

We are basing this framework on existing tools, some of which are already in wide use. This section briefly describes the existing tools.

Describing and Scheduling Workflows Workflows are described using Wings (Workflow INstance Generation and Selection) [10]. Wings allows for the specification of a workflow template (a skeleton of the analysis) that identifies all the application components and their dependencies. Then it enables the user to select the appropriate input data products and generate a workflow instance that uniquely describes the overall analysis in a resource-independent way. Wings relies on semantic descriptions of the workflow components to make sure that only semantically-compatible components are composed in the workflow template. When the workflow template is instantiated, the semantic types of data that is used to populate the template are validated against the specifications in the template. Wings also maintains information about the metadata of the data (both intermediate and final) generated by the workflow. The workflow instance is then given to Pegasus [7], a workflow mapping system that maps the resource-independent workflow description onto the available resources. Pegasus queries: 1) the environment for the available resources and their characteristics, 2) the transformation catalog to find where the executables for the workflow components are located and what their environment requirements are, 3) data registries to find the location of the input data. Together, Wings and Pegasus aim to optimize the workflow based on the overall workflow runtime by selecting the implementations for the workflow components, and/or the parameters for the components that are most appropriate given the available resources.

Managing Data in DataCutter. The system incorporates DataCutter [1], a component-based middleware framework. The DataCutter framework provides a coarse-grained data-flow system and allows combined use of task- and data-parallelism. In DataCutter, application processing structure is implemented as a set of components, referred to as *filters*, that exchange data through a *stream* abstraction. A stream denotes a unidirectional data flow from one filter (i.e., the producer) to another (i.e., the consumer). The DataCutter runtime system supports execution of filters on heterogeneous collections of storage and compute clusters in a distributed environment. Multiple copies of a filter can be created and executed, resulting in data parallelism. The runtime system performs all steps necessary to instantiate filters on the desired hosts, to connect all logical endpoints, and to invoke the filter's processing function. Each filter executes within a separate thread, allowing for CPU, I/O and communication overlap. Data exchange between twolters on the same host is carried out by pointer hand-off operations for languages that support pointers (C++), while message passing is used for communication between filters on different hosts.

Model-Guided Empirical Optimization in ECO. The Empirical Compilation and Optimization (ECO) compiler uses an approach to compiler-directed code optimization called *model-guided empirical optimization* [3]. This approach simultaneously optimizes across multiple levels of the memory hierarchy for dense-matrix computations by combining compiler models and heuristics with guided empirical search to take advantage of their complementary strengths. The models and heuristics limit the search to a small number of candidate implementations, and the empirical results provide the most accurate information to the compiler to select among candidates and tune optimization parameter values. Results on matrix multiply have demonstrated results that are comparable and in some cases outperform hand-tuned libraries with a compiler-based approach. While the initial framework used a heuristic-based based on compiler domain knowledge, a subsequent paper examined how to formulate the search systematically and considered how to use AI search techniques [4]. In our system, components for which this completely compiler-based optimization strategy is applicable are tuned automatically for multiple levels of the memory hierarchy.

5 Recent Progress

Workflow Optimization. Many workflows today are not only compute-intensive, but also data-intensive. As the execution environment is often shared among many communities and applications, the amount of resources available to a particular workflow may be limited. Although workflow performance may suffer, if the compute resources are not adequate, the application may fail due to lack of data storage resources. In our current work, we are studying mechanisms for minimizing the amount of disk space needed by a particular workflow. One approach requires adding tasks to the workflow to explicitly remove the data when they are no longer needed. This approach involves examining the use of data files within the workflow and identifying when the data have been staged out to storage or when they have been consumed by a workflow component and no longer needed. In our simulations of the execution of a LIGO workflow (a gravitational-wave physics analysis) on 4 distributed execution sites, we were able to reduce the maximum amount of space needed by the workflow by approximately 50%. The largest workflow we simulated contained about 40,000 tasks [8].

Wings and DataCutter As an initial step toward integrating Wings and Data-Cutter, we have developed a Wings representation for an electron microscopy workflow, based on the algorithms and optimizations described in [12]. This workflow resembles the MD simulation computations in its pipeline structure and the need for out-of-core data management. Since it builds on extensive evaluation using Data-Cutter with this application, this application provides a good starting point for integrating these tools.

Evaluating MD visualization parameters. We have developed an implementation of the visualization phase of MD simulation, in which application-level parameters and a range of values for these parameters are provided to the system. We have focused on two application-level parameters: (1) *cell size*, the grouping of particles into an aggregate object; and, (2) *cache size*, an additional grouping of neighboring cells to be cached during a computation to manage locality. Varying both of these parameters, and for several data sets and numbers of processors, we have analyzed the performance relationship between these two parameters. Our current work focuses on how to automatically search for the appropriate combinations of these parameters for a given data set.

Model-guided empirical optimization. We have extended model-guided empirical optimization to architectures with multimedia extensions. This is accomplished by combining our model-guided empirical optimizations to the cache and TLB levels of the memory hierarchy with optimizations targeting superword-level paralelism (SLP) at the register level. Our compiler-based approach yields performance on matrix multiply that outperforms the hand-coded Intel MKL library, and achieves results within 4% of the ATLAS self-tuning library on an Intel Pentium M. To expand the scope of applications to which this approach is applicable, we have developed a new transformation framework handles imperfect loop nests and non-rectangular loop bounds. Compiler-generated LU decomposition achieves performance within 7% of the hand-tuned Intel MKL library for the Pentium M and within 18% of the hand-tuned SCSL library on the SGI R10K.

6 Related Work

Performance tuning of large parallel applications, in practice, involves a tedious manual process of evaluating a set of alternative implementations, coded and debugged by the programmer, examining their performance properties, and repeating until either satisfactory performance has been obtained, or the programmer gives up on further improvement. This ad hoc approach is used in MD simulation, and a similar strategy has been used in LS-DYNA, a widely used engineering crash code, which we have automated using a specialized code generator [13]. Since the performance of an application is largely architecture dependent, this process must be repeated for each port to a new parallel architecture.

A wealth of high-level language, compiler and run-time technology has been developed to support this process, and while this technology has achieved some degree of success at the small to moderate scale of parallel system (on the order of 64 or fewer processors), it has not been generally effective in scaling up to larger platforms. Today, most applications developed for hundreds to thousands of processors have been written in MPI, and have often bypassed high-level compiler transformations in favor of programmer-controlled optimization, as in the LS-DYNA example above.

A recent suite of self-tuning libraries and domain-specific tools have been developed that use specialized code generators and a search engine to systematize performance tuning and porting to new architectures, largely focusing on memory hierarchy optimization. These systems have been hugely successful at portable high performance, largely for the single processor. As examples, ATLAS [22] and PhiPAC [2] focus on linear algebra, while FFTW [9] and SPIRAL [18], are for the signal processing domain. Most use brute force search, applying some domain knowledge to prune the search space. In the case of SPIRAL, genetic tree algorithms are used [19]. In general, the search technology in these systems is used only at library installation time.

Recently, several research groups have been applying machine learning techniques to compiler optimization problems, in specific ways. The current work in this area largely focuses on low-level optimizations, such as determining optimization order [6, 11], instruction scheduling [20] and code size for embedded systems [5]. High-level optimizations considered focus only on loop tiling for the memory hierarchy [17, 21]. The types of search algorithms include simulated annealing [6, 17], hill climbing [6], and genetic algorithms [5, 21, 20]. These types of search algorithms incorporate little or no domain knowledge from the decades of compiler optimization research. Thus, the searches are prohibitively expensive and are of necessity performed off line. Where domain knowledge has been used, the practicality of using these techniques has greatly increased [11].

7 Conclusion

In this paper we described a systematic approach to composing and optimizing large-scale complex application workflows on high-performance architectures, using molecular dynamics simulation as an initial design point for the system. We draw upon techniques developed in a variety of computer science fields such as compilers, workflow optimization, AI and others to develop system components that can work together to better represent the problem space and support an efficient search for solutions.

Acknowledgements. This research has been supported by the National Science Foundation under awards ACI-0204040, CSR-0509517 and CSR-0615412, and by a gift from Intel Corporation.

References

[1] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with datacutter. *Parallel Computing*, 27(11):1457–1478, Oct. 2001.

[2] J. Bilmes, K. Asanović, C.-W. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology. In *Proceedings of the 1997 ACM International Conference on Supercomputing*, June 1997.

[3] C. Chen, J. Chame, and M. W. Hall. Combining models and guided empirical search to optimize for multiple levels of the memory hierarchy. In *Proceedings of the International Symposium on Code Generation and Optimization*, Mar. 2005.

[4] C. Chen, J. Chame, and M. W. Hall. A systematic approach to model-guided empirical search for memory hiearchy optimization. In *Proceedings of the 18th International Workshop on Languages and Compilers for Parallel Computing*, Oct. 2005.

[5] K. D. Cooper, P. J. Schielke, and D. Subramanian. Optimizing for reduced code space using genetic algorithms. In *Proceedings of ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'99)*, May 1999.

[6] K. D. Cooper, D. Subramanian, and L. Torczon. Adaptive optimizing compilers for the 21st century. *The Journal of Supercomputing*, 23(1):7–22, Aug. 2002.

[7] E. Deelman and et al. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13:219–237, 2005.

[8] E. Deelman, A. Ramakrishnan, R. Sakellariou, G. Singh, K. Vahi1, H. Zhao, K. Blackburn, D. Meyers, and M. Samidi. Scheduling data-intensive workflows onto storage-constrained distributed resources. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, May 2007.

[9] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE: Special Issue on Program Generation, Optimization, and Platform Adaptation*, 93(2):216–231, Feb. 2005.

[10] Y. Gil, V. Ratnakar, E. Deelman, M. Sparagen, and J. Kim. Wings for pegasus: A semantic approach to creating very large scientific workflows. In *OWL: Experiences and Directions 2006, OWL-2006*, 2006.

[11] P. Kulkarni, S. Hines, J. Hiser, D. Whalley, J. Davidson, and D. Jones. Fast searches for effective optimization phase sequences. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2004.

[12] V. S. Kumar, B. Rutty, T. Kurc, U. Catalyurek, S. Chow, S. Lamont, and J. Saltz. Large image correction and warping in a cluster environment. In *Proceedings of SC'06*, Nov. 2006.

[13] Y. Lee, P. Diniz, M. Hall, and R. Lucas. Empirical optimization for a sparse linear solver: A case study. *International Journal of Parallel Programming*, 33, 2005.

[14] A. Nakano, R. K. Kalia, P. Vashishta, T. J. Campbell, S. Ogata, F. Shimojo, and S. Saini. Scalable atomistic simulation algorithms for materials research. *Scientific Programming*, 10:263), Year = 2002.

[15] S. Ogata, T. Campbell, R. Kalia, A. Nakano, P. Vashishta, and S. Vemparala. Scalable and portable implementation of the fast multipole method on parallel computers. *Computer Physics Communications*, 153:445–461, 2003.

[16] J. C. Phillips, G. Zheng, S. Kumar, , and L. V. Kale. Namd: biomolecular simulation on thousands of processors. In *Proc. Supercomputing '02*, Nov. 2002.

[17] G. Pike and P. N. Hilfinger. Better tiling and array contraction for compiling scientific programs. In *Proceedings of Supercomputing '02*, Nov. 2002.

[18] M. Püschel, J. M. F. Moura, J. R. Johnson, D. Padua, M. M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE: Special Issue on Program Generation, Optimization, and Platform Adaptation*, 93(2):232–275, Feb. 2005.

[19] B. Singer and M. Veloso. Stochastic search for signal processing algorithm optimization. In *Proceedings of Supercomputing '01*, Nov. 2001.

[20] M. Stephenson, S. Amarasinghe, M. Martin, and U.-M. O'Reilly. Meta optimization: Improving compiler heuristics with machine learning. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2003.

[21] X. Vera, J. Abella, A. González, and J. Llosa. Optimizing program locality through CMEs and GAs. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2003.

[22] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, Jan. 2001.