A Systematic Approach to Composing and Optimizing Application Workflows

Ewa Deelman, Aram Galstyan, Yolanda Gil, Mary Hall, Kristina Lerman (University of Southern California /Information Sciences Institute)
Aiichiro Nakano, Priya Vashista (University of Southern California)
Joel Saltz (Ohio State University)

I. Introduction

Scientists are conducting data analysis of unprecedented complexity and scale. Many scientific applications are being built not as monolithic entities designed by a single individual, but rather by combining models and analysis routines contributed by many scientists, specializing in a particular area 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. Some of these components require very specialized, high-performance resources to achieve reasonable performance.

In conventional computing, complex software systems are often composed from layering on top of an existing code base, taking advantage of existing libraries and component technology to express aspects of the composition of library elements. But in conventional computing, performance is often sacrificed in order to build robust software environments. In developing high-end computing applications, performance is an additional critical dimension that must be addressed. The performance properties of the workflow elements must be represented and used to both to select the workflow components and perform the mapping of the application to hardware.

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 up to high-level application composition logic. 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. Further, the components must be composed in such a way that the solution performs well globally and makes productive use of valuable computing resources. The efficiency of high-end computing applications depends on a number of factors, such as utilization of the memory hierarchy and the individual processor, effective parallelization, communication costs as well as overlap of communication with useful work, i/o costs as well as overlap of i/o with useful work, etc. In turn, these performance metrics depend on a variety of application-level features and the set of transformations applied by the compiler.

The shear difficulty of developing scalable applications of such complexity has made the process only approachable by small teams of highly skilled individuals. The work is slow and tedious. Program development methodology is often sacrificed for performance. From the standpoint of advancing technology, it is important to broaden the appeal and approachability of high-end computing so that it will be more widely used. Also, due to limited resources, we need to establish a simpler, more systematic approach to developing high-end workflows. High-end platforms today have unprecedented resources that could be utilized to do some of the work that humans do today.

In this paper, we propose 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.

2. Motivating Application and Approach

Our goal is a general strategy towards systematic workflow optimization, but it will be developed in the context of a specific class of applications, molecular dynamics (MD) simulations. A common MD problem is to derive the phase-space trajectories of the system in terms of the positions and velocities of all particles at all times [10]. Force laws describe the interactions among particles as a function of their relative positions over time. In past years, we have developed several algorithms to reduce the computational cost of molecular dynamics simulations. We have been developing scalable MD simulation algorithms with $O(N)$ complexity using a linked-cell-list approach [11], as well as a divide-and-conquer algorithm based on hierarchical clustering of particles[6]. For long-range particle interactions, we have developed a parallel algorithm based on a topology-preserving computational spatial decomposition scheme to minimize latency through structured message passing and load-imbalance/communication costs through a novel wavelet-based load-balancing scheme [1,2]. This algorithm has been used successfully to simulate multibillion particles on thousands of processors [10].

We have executed these MD algorithms on a variety of systems of up to 1024 processors. Their performance depends on computational parameters such as the cell size in the linked-list method, the tree level in the hierarchical clustering approach, and the number of time steps to skip long-range force calculations. We have gained significant expertise in how to hand-tune these parameters for efficient and scalable performance, though their optimal values are largely unknown. Capturing this domain knowledge as properties of workflow components or "patterns" provides a foundation for an MD environment.

The MD 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.

3. System Overview

To address these challenges, we propose a suite of productivity tools that intelligently exploit the vast machine resources of such platforms (processing, memory, disk) to automate application mapping. Through high-level linguistic mechanisms, generalized compiler and run-time technology, workflow management, machine learning and knowledge representation techniques, we provide a systematic solution to what is now a tedious, trial-and-error process to derive scalable applications. This technology will be developed in the context of an important application domain that has already been scaled to several hundreds of processors, particle simulation based on molecular dynamics (MD) and related simulation methods used in broad areas such as materials science, biology, ecology, fluid dynamics, robotics and computer graphics. The proposed approach will facilitate initial development of applications in this domain, as well as porting to new parallel platforms with different performance properties, and will provide applications whose performances is robust in response to new input data set characteristics, algorithm changes, etc.

Let us now examine the problem of constructing workflows for MD simulation. Concretely, the selection among ***algorithm variants*** in simulation of charged particles (*e.g.*, [3,4,5,6,7,8] can be very sensitive to number of particles N, their distribution (uniform/non-uniform), and boundary conditions,

while the parallel performance of algorithms will depend strongly on the granularity N/P (P is the number of processors) [9,10]. Furthermore, within a given algorithm, there are a number of ***computational parameters*** that may affect the performance. For example, in molecular dynamics (MD) simulation [11] and visualization [12] of particles, a collection of atoms is often abstracted as a rectangular "cell" that encloses the particles, to reduce the computational complexity. Though the computational results are independent of the size of the cell, n (i.e., the average number of particles in each cell), the performance of these cell-based programs is highly sensitive to n. From the perspective of a compiler or run-time system, optimization decisions can also be viewed as selecting among a set of ***compiler variants*** and determining values of specific ***compiler parameters.*** For example, when a compiler optimizes for the memory hierarchy, examples of transformation variants include selecting the appropriate ordering of loops in a nest, modifying the layout of data in the address space, or deciding which data structures should be prefetched into cache.

Beyond optimization of individual application components, challenges arise in composing these algorithms into simulations tailored to specific problems. In materials research, MD simulations are used to discover atomistic mechanisms underlying macroscopic material properties such as hardness and fracture toughness. Macroscopic materials properties result from complex spatio-temporal interactions among various atomistic attributes (e.g., atomic species, 3D coordinate, 3D velocity, stress-tensor components, and a list of adjacent atoms that are chemically bonded to the atom) and other attributes synthesized from these elementary attributes. A graph data structure can be superimposed on the data in each frame, by identifying atoms as nodes and atomic bonds as edges. The node degree is a simple indicator of certain crystalline structures, and more nonlocal information, such as the shortest-path circuits, can be used to automatically detect topological defects (*e.g.*, dislocations) in the bond network [13,14]. Simulation research thus involves calculations of various attributes (or data filters) concurrently with simulation itself. To explain atomistic mechanisms, the outputs from these filters—which show, *e.g.*, shock wave fronts, phase transformation fronts, and damages—need to be cross-correlated/collocated, a complex workflow problem.

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.

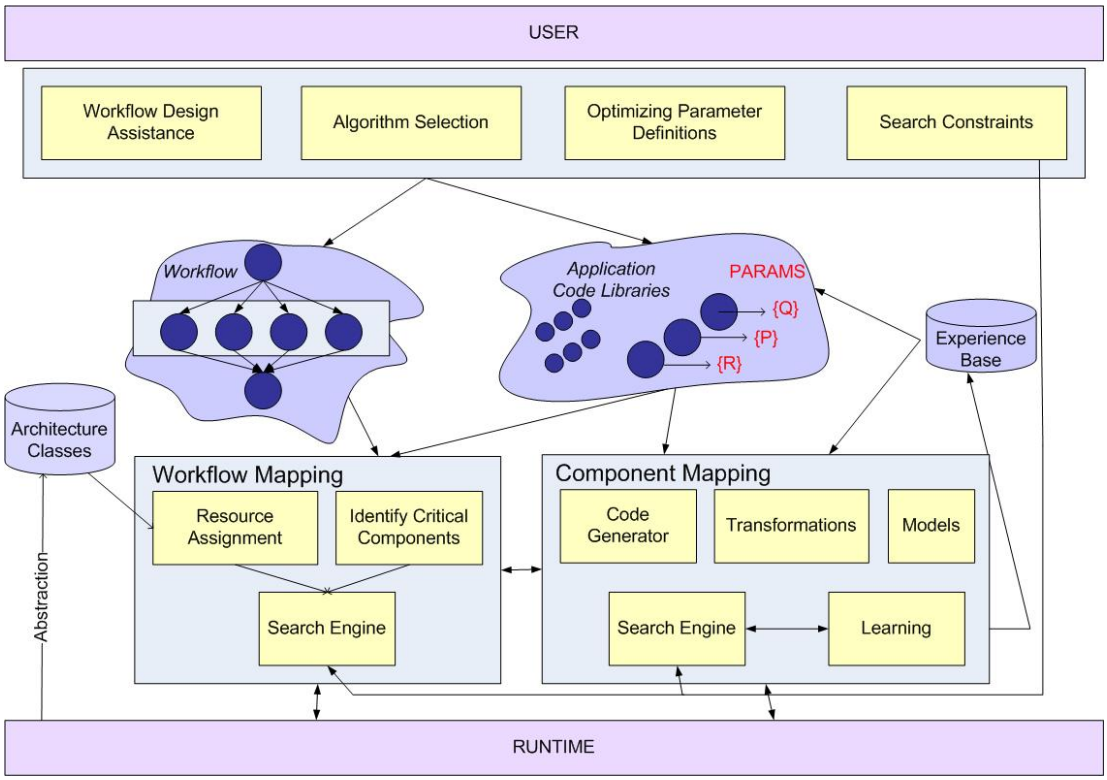


Figure 1. Overview of System

Now let us consider a systematic approach to constructing workflows for MD simulation, using the system shown in Figure 1. The key innovation in the system 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 using ML techniques 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 the available resources, guided by heuristics that aim to perform local optimizations. Randomized mappings are inserted to avoid solutions that are not globally optimal. The mappings are evaluated and the mappings which result in the smallest execution time for the overall workflow are chosen. The 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.

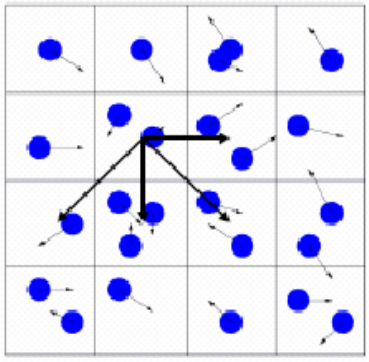
Such a systematic approach to workflow optimization is essential to productive utilization of the vast machine resources of large-scale parallel platforms with thousands of processors. However, a number of considerable technical challenges must be addressed in such a system, as follows.

Workflow design and mapping techniques that capture optimization decisions and choices: A crucial new direction in our research is to assist users to design efficient workflows. Today, users define workflows by hand with no particular methodology or optimization concerns. Workflow mapping tools have no way to undo bad decisions made in the original design of the workflow. In a sense, workflows are similar in nature to task graphs that have been widely studied in high performance computing and parallel compilers. In our work, the task graphs are annotated with performance information that enables the other components in the system to optimize the workflows for efficiency. Mapping workflows onto the available resources with Pegasus [53] then involves searching through a set of possible workflow

component assignments and evaluating the quality of the mapping in terms of the performance (runtime) of the entire workflow. The aim of the mapping is to minimize the *makespan* of the workflow, its runtime measured from the start of the first task to the completion of the last task. The search performs a number of iterations to find the best possible mapping of jobs to resources for a given workflow. During the search many alternative whole workflow allocations are created and compared before the final workflow is chosen.

High-level linguistic mechanisms to specify algorithm variants and computational parameters. Linguistic mechanisms permit the application developer to indicate what the parameters are and their possible value ranges to explore, as well as to offer variants of a particular component. We focus on identifying what the programmer must be able to express related to performance tuning, rather than any specific syntax. These mechanisms can be viewed as either simple extensions to existing languages, or perhaps in the context of domain-specific telescoping languages tools [15].

Capturing the optimization parameters and workflow properties for the MD algorithms. The appropriate linguistic mechanisms must be at a high level, and not impose a heavy burden on the programmer. An example is shown in the abstract code segment below for the force calculation of a molecular dynamics. The simulation space is partitioned into cells, and the atoms are assigned to cells according to their positions. The force calculation for an atom involves atoms in the same cell and atoms in its neighboring cells, as depicted in the figure. The force calculation relies on a linked list to capture the relationship between atoms.



```
/* Calculate forces for N atoms */
OptimizationParameter cellsize,
Range=[InteractionRange,BoxLength]
ncells = (INT(BoxLength/cellsize))^3
head(1:ncells) = 0
DO i = 1, N
   icell= Calculate_Cell_Id(INT(x(i:i + 2) / cellsize)
   lklist(i) = head(icell)
   head(icell) = i
ENDDO
```

On a single processor, the cell size directly affects performance; larger cells lead to greater data reuse in cache, but increase the amount of computation. The optimal cell size finds a balance between these two optimization goals. The linguistic mechanism for expressing to the system that cell size is a parameter to be searched is shown in the first line. The programmer must also provide a range of values for the parameter, which could be calculated by a function rather than with the constant values shown in this example. While not shown here, multiple code variants can be expressed using the selector approach [16].

Compiler technology to decouple analyses and code transformations from the searches to apply the transformations. In recent work, we have demonstrated the power of empirical search for compiler-directed optimization of the memory hierarchy [17,18]. We have developed an algorithm for simultaneously optimizing across multiple levels of the memory hierarchy (registers, L1 cache, L2 cache and TLB) for dense-matrix computations. Our current approach combines compiler models and heuristics with guided empirical search to take advantage of their complementary strengths. Compiler models and heuristics provide domain knowledge to a search engine of the search space properties of optimizations. This domain knowledge can be used to limit the optimization search to a small number of candidate implementations. The empirical results provide the most accurate information to the compiler to select

among candidates and tune optimization parameter values. Our results to date using this approach on Matrix Multiply on the SGI R10K and Sun UltraSparc II outperform the native compiler, the ATLAS self-tuning library and even the hand-tuned BLAS library for each platform. Results on Jacobi Relaxation also substantially outperform the native compilers [63]. We have used a similar approach to search for the appropriate FPGA implementation of multimedia kernels [19,20].

Appropriate models to guide search algorithms and prune the search space. Since we are willing to consider multiple implementations of a computation, the output of the model-guided optimization is a set of *parameterized code variants* as well as *constraints on parameter values* to facilitate pruning of the search space by the search engine. We advocate the use of models that are very cheap to evaluate, albeit at some loss of precision as compared to the most accurate models available. To the extent that models can be used rather than empirical data, the efficiency of the search engine will be improved. Even inaccurate models can be effective at identifying performance trends and pruning off large uninteresting portions of the search space. Scalable applications can be modeled by producing a stripped-down application designed to exhibit performance characteristics that resemble those displayed by the application. We have developed a framework for accomplishing this, consisting of two components: application emulators [21,22] and a suite of simulators [23]. Application emulators provide a parameterized model of data access and computation patterns of the applications and enable changing of critical application components (data partitioning, data declustering, processing structure, etc.) easily and flexibly.

Parameterized run-time libraries that support general application functions. The use of appropriately parameterized run-time support libraries is important to carry out coordinated data management, communication and partitioning tasks. The use of such libraries greatly reduces the optimization search space as it becomes possible to obtain a parametric performance characterization of the complex run-time support library. The CHAOS and PARTI runtime support libraries have been used as compiler runtime support for Fortran molecular dynamics codes including CHARMM and a Discrete Simulation Monte Carlo code [24, 25]. CHAOS and PARTI have been generalized to create a new framework called Active Proxy G. Active Proxy G functions in cluster, parallel or grid environments and maintains semantically cached or prefetched application level data [116-123]. Active Proxy G supports the indexing and semantic caching of data generated at each timestep. Our approach greatly reduces overheads associated with repartitioning problems having dynamically changing computational characteristics. Active Proxy G also simplifies the process of supporting data management and communication requirements associated with this class of irregular applications. Run-time support will either be embedded by programmers or embedded by the compiler following strategies motivated by the user-level linguistic support described above.

An efficient approach to dynamically generate or select component implementations. Once the optimization parameters have been identified by either the application programmer or compiler, the generation of alternative optimized code implementations is straightforward and mechanical, and can easily be supported by compilation tools. This tool support greatly increases programmer productivity, since manual creation of variants is tedious, and limits the programmer's search. Further, to support empirical search of variants, we have developed a *code isolator* to extract code segments from large applications, generating an executable of this segment with a representative data set, and that sets machine state [26]. A challenging aspect of this work is managing the diversity of code variants, and efficient run-time code selection/generation. To the extent that the run-time optimization involves binding parameter values, the compiler can generate a single version of the code. If the number of variants is small, the compiler can generate all variants and dynamically select which version to execute using a dynamic feedback approach [27]. Looking forward, a critical requirement will be the ability to generate code on-the-fly, exploiting the available machine resources in a large-scale system to generate code in parallel and begin executing the optimized version when it becomes available, similar to what is done in ADAPT [28].

The research challenge is managing and relating the parameters characterized by the AI formulation, to code variants that can be dynamically generated.

Formulation and representation of component optimization as a machine-learning problem. From the machine learning point of view, the performance optimization problem consists of finding a set of features that capture the salient elements of the problem. It is convenient to think of the scalable performance optimization problem as consisting of the following features or elements:

- {**A**}: Set of application-level parameters such as characteristics of the *input data* (e.g., sample size) and the *cell size* parameter used in MD simulations
- {**T**}: Code variants provided by compiler, but possibly suggested by programmer
- {**C**}: Set of compiler-level parameters, e.g., unroll amount, communication
- {**R**}: Architectural features, such as number of registers, cache size/associativity, number of processors, etc. These are independent variables provided by the workflow manager.

The measure of performance, ***F***, is up to the designer. The standard performance metric is execution time, although in other situations the application designer may care more about other properties, such as throughput, etc. The optimization problem is then to find the set of parameters **{A*, T*, C*}** that for a given architecture **R** and family of problems maximize the specified performance metric ***F***, i.e.,

$$\mathbf{\{A^*, T^*, C^*\}} = \text{argmax } \boldsymbol{F}(\mathbf{A, T, C})$$

This optimization problem may be supplemented by certain constraints specified by the user.

Search algorithms to navigate the search space of the various optimization parameters. The main feature that makes the above optimization problem very hard is that we do not know a priori the functional form of ***F***, i.e., how the performance depends on the parameters. Clearly, a simple search by trial and error will not work due to the exponentially large search space and the non-negligible cost of evaluating ***F*** for a single point in the search space. On the other hand, if we have some insights on the interdependencies of parameters then we can develop appropriate heuristics for pruning the search space and guiding the optimization procedure. A key challenge is *learning* an appropriate model for $\boldsymbol{F}(\mathbf{A, T, C})$.

Given the enormous complexity of the problem we need to make some simplifying assumptions to make the problem tractable. Specifically, hierarchical decomposition can be used to obtain simpler models that nevertheless maintain the main properties of the original problem. The first step in our hierarchical decomposition is to neglect interdependencies between the parameters of different type, so that ***F*** can be represented as $\boldsymbol{F}(\mathbf{A, T, C}) = \boldsymbol{F_1}(\mathbf{A})\boldsymbol{F_2}(\mathbf{T})\boldsymbol{F_3}(\mathbf{C})$. This expression states that the optimization procedure can be carried out separately for each set of parameters. The individual models of performance measure, $\boldsymbol{F_i}$, can be obtained from domain experts, derived empirically, or a combination of the two approaches.

Once an approximate (possibly partial) model for the metric ***F*** is learned, various search algorithms can be used for finding the parameter set that optimizes the performance. Heuristic search algorithms, like A* [29], seem to be especially well suited for our purposes. These algorithms assume there exists an evaluation function that helps decide in what direction to move the search. We believe incorporating domain knowledge about performance metric ***F*** into the evaluation function will efficiently guide the search. In light of resources provided by a large parallel machine, searching among alternative implementations can be executed in parallel from different starting points in the search space.

4. Conclusions

In this paper we described an approach to optimizing large-scale complex applications on high-performance architectures. Using molecular dynamics simulation as an initial design point for the system, we develop a systematic approach to compose high-performance application workflows. 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.

References

[1] A. Nakano and T. J. Campbell, "An adaptive curvilinear-coordinate approach to dynamic load balancing of parallel multiresolution molecular dynamics," *Parallel Comput.*, vol. 23, pp. 1461, 1997.

[2] A. Nakano, "Multiresolution load balancing in curved space: the wavelet representation," *Concurrency: Practice and Experience*, vol. 11, pp. 343, 1999.

[3] T. Darden, D. York, and L. Pederson, "Particle mesh Ewald: an Nlog(N) method for Ewald sums in large systems," *J. Chem. Phys.*, vol. 98, pp. 10089, 1993.

[4] J. Barnes and P. Hut, "A hierarchical O(NlogN) force-calculation algorithm," *Nature*, vol. 324, pp. 446-449, 1986.

[5] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, pp. 325, 1987.

[6] A. Nakano, R. K. Kalia, and P. Vashishta, "Multiresolution molecular-dynamics algorithm for realistic materials modeling on parallel computers," *Computer Physics Communications*, vol. 83, pp. 197-214, 1994.

[7] A. Nakano, "Parallel multilevel preconditioned conjugate-gradient approach to variable-charge molecular dynamics," *Comput. Phys. Commun.*, vol. 104, pp. 59, 1997.

[8] S. Ogata, T. J. Campbell, R. K. Kalia, A. Nakano, P. Vashishta, and S. Vemparala, "Scalable and portable implementation of the fast multipole method on parallel computers," *Computer Physics Communications*, vol. 153, pp. 445-461, 2003.

[9] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kale, "NAMD: biomolecular simulation on thousands of processors," *Proc. Supercomputing '02*, ACM, 2002.

[10] 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*, vol. 10, pp. 263, 2002; the **Best Paper** in *Proc. Supercomputing 2001* (ACM).

[11] L. H. Yang, E. D. Brooks, and J. Belak, "A linked-cell domain decomposition method for molecular dynamics simulation on a scalable multiprocessor," *Sci. Programming*, vol. 1, pp. 1, 1993.

[12] A. Sharma, A. Nakano, R. K. Kalia, P. Vashishta, S. Kodiyalam, P. Miller, W. Zhao, X. L. Liu, T. J. Campbell, and A. Haas, "Immersive and interactive exploration of billion-atom systems," *Presence: Teleoperators and Virtual Environments*, vol. 12, pp. 85-95, 2003; one of the **Best Papers** in *Proc. Virtual Reality 2002* (IEEE Computer Society).

[13] J. P. Rino, I. Ebbsjö, R. K. Kalia, A. Nakano, and P. Vashishta, "Structure of rings in vitreous SiO_2," Phys. Rev. B **47**, 3053 (1993).

[14] I. Szlufarska, R. K. Kalia, A. Nakano, and P. Vashishta, "Nanoindentation-induced amorphization in silicon carbide," Appl. Phys. Lett. **85**, 378 (2004)

[15] K. Kennedy, B. Broom, K. Cooper, J. Dongarra, R. Fowler, D. Gannon, L. Johnsson, J. Mellor-Crummey, and L. Torczon, "Telescoping Languages: A Strategy for Automatic Generation of Scientic Problem-Solving Systems from Annotated Libraries," July, 2000.

[16] P. Diniz and B. Liu. Selector: An Effective Technique for Adaptive Computing. *Proc. of the Fourteenth Workshop on Languages and Compilers for Parallel Computing (LCPC'02)*, August, 2002.

[17] N. Baradaran, J. Chame, C. Chen, P. Diniz, M. Hall, Y. Lee, B. Liu and R. Lucas, "ECO: an Empirical-based Compilation and Optimization System," *Proceedings of the Workshop on Next Generation Software*, held in conjunction with IPDPS '03, April, 2003.

[18]C. Chen, J. Chame, M. Hall, "Combining Models and Guided Empirical Search to Optimize for Multiple Levels of the Memory Hierarchy," In *Proceedings of the Code Generation and Optimization Conference*, March, 2005..

[19] B. So, M. Hall, and P. Diniz. A compiler approach to fast design space exploration in FPGA-based systems. In Proc. ACM Conf. Programming Languages Design and Implementation, pages 165--176, June 2002.

[20] H. Ziegler, M. Hall and B. So. Search space properties for mapping pipelined FPGA applications. In Proceedings of the Workshop on Languages and Compilers for Parallel Computing. Oct. 2003.

[21] T. Kurc, M. Uysal, H. Eom, J. Hollingsworth, J. Saltz , A. Sussman, "Efficient Performance Prediction for Large-Scale Data-Intensive Applications", *The International Journal of High Performance Computing Applications*, Volume 14, number 3, pages 216-227, 2000.

[22] Uysal, M. Kurc,T., Sussman, A. and Saltz, J., "A Performance Prediction Framework for Data Intensive Applications on Large Scale Parallel Machines," Lecture Notes In Computer Science, 4th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers, Pages: 243 – 258, 1998.

[23] Eom, H., Hollingsworth, J. "Speed vs. Accuracy in Simulation for I/O-Intensive Applications". *IPDPS 2000*: 315-322.

[24] Y.-S. Hwang, R. Das, J. Saltz, M. Hodoscek, and B. Brooks *Parallelizing Molecular Dynamics Programs for Distributed Memory Machines*, **IEEE Computational Science and Engineering**, Volume 2, Number 2, pages 18-29, Summer 1995.

[25] R. Nance, R. Wilmoth, B. Moon, H. Hassan, and J. Saltz, Parallel DSMC Solution of Three-Dimensional Flow over a Finite Flat Plate, **Proceedings of the ASME 6th Joint Thermophysics and Heat Transfer Conference**, AIAA, pages Colorado Springs, Colorado, June 20-23, 1994.

[26] P. Diniz, Y. Lee, M. Hall and R. Lucas, "A Case Study Using Empirical Optimization for a Large, Engineering Application," *Proceedings of the Workshop on Next Generation Software,* held in conjunction with IPDPS '04, April 2004.

[27]P. Diniz and M. Rinard, Dynamic Feedback: An Effective technique for Adaptive Computing, *Proc. of the 1997 ACM/SIGPLAN Conference on Programming Language Design and Implementation (PLDI'97),* 1997.

[28] M. Voss and R. Eigenmann, High-Level Adaptive Program Optimization with ADAPT, Proc. of the ACM SIGPLAN Conference on Principles and Practice of Parallel Programming (PPoPP'01), ACM Press, Jun, 2001.

[29] N. J. Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann, San Francisco, CA, 1998.