Final Report
A Multi-Language Environment For Programmable Code Optimization and Empirical Tuning

Award #:	DE-SC0001770 (ER25946)
Institution:	University of Texas at San Antonio
Principle Investigator:	Qing Yi
Co-PIs:	R. Clint Whaley (University of Texas at San Antonio)
	Apan Qasem (Texas State University)
	Daniel Quinlan (Lawrence Livermore National Laboratory)
Period covered:	09/15/2009 - 09/15/2013
Date of Report:	11/23/2013

1 Introduction

This report summarizes our effort and results of accomplishing the project objective of building an integrated optimization environment, shown in Figure 1, to effectively combine the programmable control and the empirical tuning of source-to-source compiler optimizations within the framework of multiple existing languages, specifically C, C++, and Fortran. The environment contains two main components: the ROSE analysis engine, which is based on the ROSE C/C++/Fortran2003 source-to-source compiler developed by Co-PI Dr.Quinlan et. al at DOE/LLNL, and the POET transformation engine, which is based on an interpreted program transformation language developed by Dr. Yi at University of Texas at San Antonio (UTSA). The ROSE analysis engine performs advanced compiler analysis, identifies profitable code transformations, and then produces output in POET, a language designed to provide programmable control of compiler optimizations to application developers and to support the parameterization of architecture-sensitive optimizations so that their configurations can be empirically tuned later. This POET output can then be ported to different machines together with the user application, where a POET-based search engine empirically reconfigures the parameterized optimizations until satisfactory performance is found. Computational specialists can write POET scripts to directly control the optimization of their code. Application developers can interact with ROSE to obtain

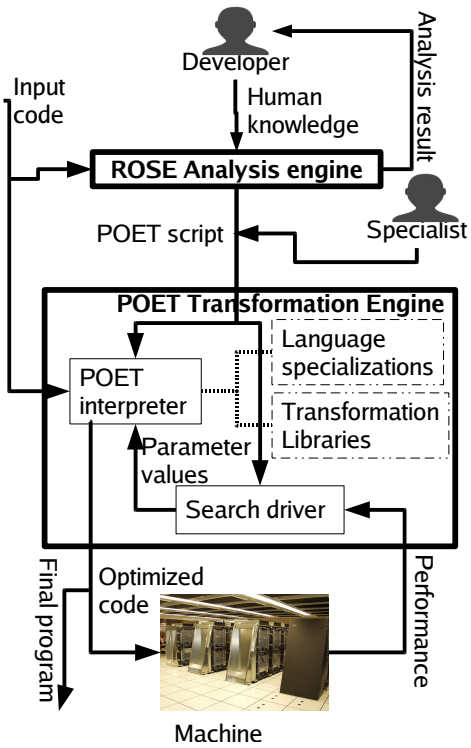

Figure 1: Optimization Environment

optimization feedback as well as provide domain-specific knowledge and high-level optimization strategies. The optimization environment is expected to support different levels of automation and programmer intervention, from fully-automated tuning to semi-automated development and to manual programmable control.

The project development has been funded by DOE from 2009-2013. During this period we have completed the basic infrastructure development, performed research as originally proposed to significantly enhance the effectiveness of compiler optimizations for scientific kernels and applications relevant to DOE, and are working on practical deployment of our infrastructure as a public open-source software development toolset that can fully support the optimization needs of DOE applications as they migrate to the next generation of supercomputers. Our funded research has produced 20 journal/conference publications [1-20] in the areas of high performance computing and compiler optimizations and has partially supported the research development of two Ph.D students (Jichi Guo and Shah F. Rahman) and Two MS students (Akshatha Bhat and M. Ziaul Haque).

2 Infrastructure Development

The basic infrastructure shown in Figure 1 includes four key components that we have focused on developing: (1) the ROSE analysis engine, which performs advanced program analysis to identify profitable compiler transformations; (2) the POET transformation engine, which uses the POET program transformation language developed by Dr. Yi to support the programmable control and parameterization of compiler optimizations; (3) the search driver, which empirically searches the parameter space of each POET script to find satisfactory optimization configurations; and (4) a Graphical User interface (GUI) where developers can conveniently interact with the ROSE analysis engine or the POET transformation engine to devise and control the optimization of their applications.

2.1 The ROSE Analysis Engine

In 2009-2010, collaborating with the ROSE compiler group led by Co-PI Dr. Quinlan from DOE/LLNL, Dr. Yi and her group successfully extended the ROSE compiler to automatically perform loop optimization analysis and then produce parameterized POET scripts as output, which can then be ported together with the user application to different computing platforms and empirically tuned. The work has been published in CGO'11 [9]. The analysis engine currently supports OpenMP parallelization, loop blocking, loop unroll-and-jam, array copying, scalar replacement, strength reduction, and loop unrolling. Using ROSE to produce POET output was a milestone that we set for the first year of our project management plan. Accomplishing this milestone indicates the initial completion of our infrastructure for supporting programmable control of compiler optimizations. Additional development work was performed collaboratively by Dr. Yi and Dr. Quinlan's groups to enhance the program analysis capabilities, e.g., pointer/array and data-flow analysis, within the ROSE compiler, and to enhance the compiler's ability to effectively parameterize varying loop optimizations by building a closer connection between ROSE and POET.

In 2011-2013, Dr. Yi worked with her Ph.D student (Jichi Guo) and Dr. Jiayuan Meng from DOE/ANL on extending the ROSE compiler to analytically model the holistic execution flow of large scientific applications and then employ parameterized hardware performance models to project application performance on conceptual and future architectures. This work is being submitted for publication [22]. Dr. Yi is currently working with Jichi Guo, while collaborating with Dr. Jiayuan Meng and Dr. Pavan Balaji from DOE/ANL, on extending this performance modeling framework to guide the optimization of large MPI applications to enhance their performance portabilities.

2.2 The POET Transformation Engine

The POET language was initially designed by Dr. Yi in 2007 to support flexible parameterization, programmable control, and empirical tuning of compiler optimizations as well as the code generation needs of ad-hoc domain-specific optimizations for scientific computing applications. The first implementation of a preliminary design of the POET language was published by Yi et. al in 2007 [20]. The language implementation was first released open-source in 2008. Over the years, many new releases of the package has been posted as both the language itself and its optimization library evolve to become increasingly more sophisticated. The language has been used as an optional project implementation language for the compiler construction classes taught by Dr. Yi since 2009. The publicly released POET optimization library currently includes OpenMP parallelization, loop blocking, loop unroll-and-jam, array copying, scalar replacement, strength reduction, loop unrolling, loop interchange, loop distribution/fusion, SSE vectorization, among others. Additionally, it has been used to support a number of domain-specific languages including automatic tester/timer generation [1], stencil code optimization [10], and automatically translating a finite-state-machine-based behavior modeling language to C++/Java code [14].

In 2012-2013, Dr. Yi worked with Dr. Yuquan Zhang and his Ph.D student, Qian Wang, from the Institute of Software, Chinese Academy of Sciences, on using the POET transformation engine to build a template-based optimization framework, AUGEM, to support the programmable optimization of dense linear algebra (DLA) kernels by automatically generating fully optimized assembly code that can surpass the performance of Intel MKL and AMD ACML BLAS libraries on both Intel Sandy Bridge and AMD Piledriver processors. The work is being published in Supercomputing'2013 [20]. Qian Wang is currently working as a visiting scholar (Oct 2013-Sep 2014) supervised by Dr. Yi . Dr. Yi is currently collaborating with Qian Wang and Dr. Huimin Cui from the Institute of Computing, Chinese Academy of Sciences, on using POET to support specialized pattern-based compiler optimizations for dense matrix computations.

2.3 Empirical Tuning Of Optimizations

In 2009-2010, Dr. Yi and Dr. Whaley collaboratively worked with Joshua Magee, a full-time developer hired to work on the project, on using the POET language to automatically produce testing and timing drivers that accurately test the correctness and measure the performance of individual routines taken from whole scientific applications. Published in SMART'10 [1], this work enables us to automatically support the tuning of large scientific applications by replicating the timing variations of their routines and providing accurate performance feedback when these routines are optimized with different configurations.

To efficiently explore the configuration space when using POET to optimize scientific computations, in 2009-2010, Dr. Qasem worked on extending the capabilities of the PSEAT search engine developed at Texas State University with two search algorithms: a genetic algorithm and a steepest descent method that uses ant colony heuristics. In 2010-2011, Dr. Qasem made several additional enhancements to PSEAT, including (1) a Particle Swarm Optimization (PSO) search technique, which allowed exploration of the search space in parallel and is particularly suitable for leveraging today's parallel architectures; and (2) a pareto-normal search algorithm which can be used to optimize multiple objective functions simultaneously and thus support optimization tuning for both power efficiency and performance on scalable multicore platforms.

In 2010-2011, Dr. Yi and Dr. Qasem collaboratively worked on connecting the POET transformation engine and the PSEAT search engine with a generic interface so that PSEAT can automatically interpret the optimization space of POET scripts automatically generated by our ROSE

analysis engine or manually developed for optimizing dense matrix computations. Additionally, a transformation aware search algorithm, published in iWapt'11 [11], was developed by Dr Yi and her students to better utilize the knowledge of the compiler optimizations at hand. Both the PSEAT search engine and the transformation-aware search algorithm have been used to explore the combined search spaces of data locality and parallelization optimizations, for both performance and power efficiency. The work has been published in NPC'10 [7], HIPEAC'11 [8], and iWapt'11 [11].

2.4 Graphical User Interface

In 2010, working with a Ph.D student (Jichi Guo) funded by this grant, Dr. Yi started extending our programmable compiler optimization infrastructure with a Graphical User Interface (GUI) implemented using C++ so that developers can more easily build their own optimizers. Since Jan 2012, working with several undergraduate students and another Ph.D student partially funded by this grant, Dr. Yi worked on building a web GUI to support programmable compiler optimizations using a combination of Javascript and Php, with a similar GUI using C++ to be added later which can be downloaded for the same ROSE analysis and POET transformation engines. The objective is to integrate the program analysis and optimization work we have developed in a unified environment with a convenient graphical interface that allows easy access and programmable control of compiler optimizations. The plan is to deploy the web-based unified environment in the Cloud for any potential users to try out our optimization infrastructure online. Then, if desired, a C++ based open-source GUI environment with a similar interface can be downloaded for local use.

Currently we have completed a working prototype of the web-based interactive compiler optimization environment. Specifically we have completed the development work of providing a graphical interface for our existing compiler optimizations and are working on integrating everything in a consistent fashion and testing the robustness of the environment by using it to optimize a reasonably large number of important scientific applications, while enhancing the capabilities of the environment in the process. The environment is expected to be deployed in the public Cloud in the near future, with a downloadable local environment distributed open-source.

3 Dissemination of Results and Application Engagement

Since Fall 2009, the POET language has been distributed online open source at our project web page initially at

`bigbend.cs.utsa.edu/POET`

and later moved to

`http://www.cs.uccs.edu/~qyi/poet/index.php`

after Dr. Yi moved from UTSA to the University of Colorado at Colorado Springs in Fall 2012. A comprehensive documentation of the POET language manual and a number of tutorials are posted at the project page together with the software releases. A summary of the language and its use cases have been published as a journal paper in Software Practice And Experience [14]. Two tutorials on how to use POET to support programmable compiler optimizations were given by Dr. Yi at ICS'11: International Conference on Supercomputing, on May 31st, 2011 and at HiPEAC'2012: High-Performance and Embedded Architectures and Compilers, on Jan 23, 2012.

Led by Co-PI Dr. Quinlan, the ROSE compiler has been released open-source at http://rosecompiler.org

The ROSE compiler precedes the development of POET and has been used by an increasingly large number of research projects within USA to develop source-to-source and binary level compiler optimizations. Since 2012, POET has additionally been integrated within ROSE as a third-party library/project and has been alternatively released together with ROSE.

Twelve conference and journal publications [1,7-12,14,15,17,19,20] were produced on the subject of developing and using our optimization infrastructure to support the optimization and tuning needs of various scientific applications, eight of which in first/second tier venues such as ICPP (2), HiPEAC(1), Computing Frontier(2), TACO(1), CGO(1), and Supercomputing (1). Nine additional publications [2-6,13,16,18] were produced on general subjects of high performance computing and compiler optimizations, four of which in first/second tier Computer Science venues such as PPOPP(1), IEEE Cluster(1), IPDPS(1), and PACT(1). In particular, our paper in SC'13 demonstrates that using POET, high performance computing specialists can automatically attain the highest level of performance (exceeding those attained by the vendor-supplied libraries such as MKL and ACML) for dense matrix computations. This result is expected to draw significant interest from the HPC community in using our toolsets to enhance the performance portability of their libraries and applications.

Three Ph.D students, Jichi Guo (UTSA & UCCS), Shah F. Rahman (UTSA & UCCS), and Rui Zhao(UCCS); two MS students, Akshatha Bhat (UTSA) and M. Ziaul Haque (UTSA); and three undergraduate students, Ansley Skillern (UTSA), Jesse Leija (UTSA), and Brandon Nesterenko (UCCS), have been partially supported by this grant while working towards getting their degrees. The POET language has been used as an optional project development language in the Compiler Construction courses that Dr. Yi taught since 2010, in both the University of Texas at San Antonio (2010-2012) and the University of Colorado at Colorado Springs (Spring 2013).

From May, 2011 through Aug, 2011, both of the Ph.D students, R. Faizur Rahman and Jichi Guo, worked at DOE/LLNL as summer interns to establish closer collaboration between the ROSE project at LLNL and the POET project at UTSA. Dr. Yi visited DOE/LLNL in June, 2011 to look into applying the optimization and tuning infrastructure funded by this grant to improve the performance of DOE lab applications at LLNL. In 2012, two of Dr. Yi's students, Jichi Guo (Ph.D) and M. Ziaul Haque (MS), worked as Summer interns at the DOE Argonne National Lab, supervised by Dr. Pavan Balaji and Dr. Jiayuan Meng respectively, on using POET to enhance the performance portability of MPI applications and on using the ROSE compiler to analytically model the performance of large DOE scientific applications. Currently Dr. Yi is actively collaborating with Co-PI Dr. Quinlan's group on building a tighter connection between the POET transformation language and the ROSE compiler so that they can be used collectively in optimizing large DOE applications for many-core architectures.

4 Research Activities And Results

While developing our basic infrastructure, including the ROSE analysis engine and the POET transformation and tuning engine, our research activities have focused on the following areas.

- High performance computing: investigating and developing new optimization techniques to attain a highest level of performance for important scientific computing kernels.
- Programmable compiler optimization: devising new advanced formulations and algorithms to automate the optimization strategies from existing HPC research, while integrating domain-specific knowledge and programmable control of the compiler optimizations.

- Tuning and modeling the performance and energy efficiency of scientific applications: establishing insightful understanding of application performance and power consumption behaviors to enhance the effectiveness of their optimizations.

The following subsections detail our key research activities and results in these areas.

4.1 High Performance Computing

In year 2009-2010, Co-PI Dr.Whaley investigated how to better optimize bus-bound threaded dense linear algebra codes, which typically get little or no speedup from parallelization, using both new algorithmic and systems approaches. The study revealed the importance of dynamic scheduling and its interaction with cache efficiency, which has become increasingly important on today's variable and heterogeneous architectures. In particular, the use of co-processors such as GPUs appear to place very asymmetrical loads on the CPUs of a processor (one processor runs at essentially half speed using CUDA, for instance), which in turn can kill parallel performance if static scheduling is used. However, in operations with cache reuse, fully dynamic scheduling can result in loss of cache reuse, leading to decreased parallel performance. Working with his students, Dr. Whaley investigated ways to apply a specialized blocking which allows superlinear speedup of out-of-cache operations which previously achieved little or no speedup for out-of-cache performance. The key idea is to enable cache reuse by using extremely low-overhead parallel synchronizations (leveraging cache coherence to run at hardware speed), and then introducing parallel overheads into the *innermost* loop in order to enable parallel cache reuse, which in turn provides superlinear speedups in operations. This body of work leads to better optimizations for modern heterogeneous architectures and has been published in PPOPP'10 [2].

In year 2010-2011, Dr.Whaley worked on extending the Parallel Cache Assignment (PCA) work for parallelizing bus-bound operations to the 2-sided factorizations such as Hessenberg factorization, which is one of the most time consuming steps in finding the eigenvalues, and is almost completely bus bound during normal operations. This work achieved super-linear speedup for small to moderate-sized problems and commandingly surpassed the state of the art for this range of problem sizes. This body of work has been published to IEEE Cluster 2011 [13].

In 2011-2012, PI Dr.Yi collaborated with Dr.Huimin Cui from the Institute of Computing, Chinese Academy of Sciences, on speeding up the reuse distance analysis algorithm, a dynamic program analysis technique widely used to model the memory system behavior of applications. Traditional reuse distance analysis algorithms use tree-based data structures and are hard to parallelize, missing the tremendous computing power of the emerging GPUs. This paper presents a highly-parallel reuse distance analysis algorithm (HP-RDA) to speedup the process using the SPMD execution model of GPUs, by using a hybrid data structure of hash table and local arrays to flatten the traditional tree representation of memory access traces. A probabilistic model is then used to correct any loss of precision from a straightforward parallelization of the original sequential algorithm. Experimental results show that using an NVIDIA GPU, this algorithm achieves a factor of 20 speedup over the traditional sequential algorithm with less than 1% loss in precision. This work has been published in IPDPS'12 [16].

4.2 Programmable Compiler Analysis And Optimization

In 2009-2011, PI Dr.Yi worked on developing the basic optimization framework as we originally proposed to effectively combine programmable control by developers, advanced optimization by compilers, and flexible parameterization of optimizations to achieve portable high performance for scientific applications. The framework extended the ROSE C/C++/Fortran optimizing compiler

from DOE/LLNL to automatically analyze scientific applications and discover optimization opportunities. Instead of directly generating optimized code, our optimizer produces parameterized scripts in the POET program transformation language, so that developers can freely modify the optimization decisions by the compiler and add their own domain-specific optimizations if necessary. The auto-generated POET scripts support additional optimizations beyond those available in the ROSE optimizer. Further, all the optimizations are parameterized at a fine granularity, so the scripts can be ported together with their input code and automatically tuned for different architectures. Our results show that this approach is highly effective, and the code optimized by the auto-generated POET scripts can significantly outperform those optimized using the ROSE optimizer alone. This work has been published in CGO'11 [9].

In 2010-2011, to improve the aggressiveness of program analysis techniques in ROSE, Dr. Yi worked with Jichi Guo (a Ph.D student partially funded by this grant) on developing annotation-based inlining techniques to enable conventional single-procedural optimizations to be applied to whole programs in a modular and incremental fashion. Since Sep, 2010, Yi and Guo collaborated with Dr. Psarris (a full professor at UTSA) on evaluating how the new techniques can extend the applicability of conventional loop dependence analysis techniques and enable them to be applied more effectively. Result from this work has been published at ICPP'11 [12]. We have fully integrated this body of work within the ROSE analysis engine and are working on use the infrastructure to support the performance portability of large-scale MPI applications in C++/Fortran by automatically optimizing their communication operations to reduce the communication overhead and to enable communications better overlapped with independent computations.

In 2011-2012, Dr. Yi collaborated with Dr. Huimin Cui from the Institute of Computing, Chinese Academy of Sciences, on using POET to develop an annotation-based framework to overcome the sensitivity of compiler optimizations to the varying data-layout of dense matrix computations. Our results show that while the efficiency of a computational kernel differ when using different data layouts, the alternative implementations typically benefit from a common set of optimizations on the operations. Therefore separately optimizing the operations and the data layout of a computation could dramatically enhance the effectiveness of compiler optimizations compared with the conventional approaches of using a unified representation. Based on this result, we have developed a data layout oblivious optimization methodology, where by isolating an abstract representation of the computations from complex implementation details of their data, we enable these computations to be much more accurately analyzed and optimized through varying state-of-the-art compiler technologies. This work has been published in TACO'2013 [17] and presented in the HIPEAD'2013 conference in Berlin, Germany.

In 2012-2013, Dr. Yi and Dr. Whaley collaboratively worked with Majedul Haque Sujon, a Ph.D student of UTSA, on enhancing the effectiveness of SIMD vectorization within compilers to improve the performance of floating point intensive scientific kernels on multicore architectures, by overcoming the presence of unknown control flow surrounding partially vectorizable computations. In particular, we have developed a new approach, speculative vectorization, which speculates past dependent branches to aggressively vectorize computational paths that are expected to be taken frequently at runtime, while simply restarting the calculation using scalar instructions when the speculation fails. We have integrated our technique in an iterative optimizing compiler and have employed empirical tuning to select the profitable paths for speculation. Our results show that up to 6.8X speedup for single precision and up to 3.4X speedup for double precision can be attained for 9 floating point benchmarks in AVX through our speculative vectorization optimization, which also vectorized some operations considered not vectorizable by prior techniques. This work has been published in PACT'2013 [18]

In 2012-2013, Dr. Yi worked with Md Ziaul Haque, a MS student of UTSA partially funded by

this grant, while collaborating with Dr. Pavan Balaji from DOE Argonne National Laboratory, on using our POET transformation engine to automatically enhance the performance portability of MPI applications across platforms. In particular, we have developed a framework that enables users to provide hints about communication patterns used their MPI applications. These annotations are then used by an automated program transformation system to leverage different MPI operations that better match each systems capabilities. Our framework currently supports three automated transformations: coalescing of operations in MPI one-sided communications; transformation of blocking communications to nonblocking, which enables communication-computation overlap; and selection of the appropriate communication operators based on the cache-coherence support of the underlying platform. We used our annotation-based approach to optimize several benchmark kernels and demonstrated that the framework is effective at automatically improving performance portability for MPI applications. This work has been published in ICPP'2013 [19].

In 2012-2013, Dr. Yi collaborated with Dr. Yuquan Zhang from the Institute of Software, Chinese Academy of Sciences, while co-advising Qian Wang, a Ph.D student of the institute, on using the POET language developed by Dr. Yi (funded by this project) to build a template-based optimization framework, AUGEM, to automatically generate fully optimized assembly code for dense linear algebra (DLA) kernels, such as GEMM, GEMV, AXPY and DOT, on varying multi-core CPUs without requiring any manual interference from developers. Based on domain-specific knowledge about algorithms of the DLA kernels, the framework uses a collection of parameterized code templates to formulate a number of commonly occurring instruction sequences within the optimized low-level C code of these DLA kernels. Then, it uses a specialized low-level C optimizer (implemented using POET) to identify instruction sequences that match the pre-defined code templates and thereby translates them into extremely efficient SSE/AVX instructions. The DLA kernels generated by this template-based approach surpass the implementations of Intel MKL and AMD ACML BLAS libraries, on both Intel Sandy Bridge and AMD Piledriver processors. The work has been published in Supercomputing'2013 [20].

4.3 Tuning And Modeling Performance and Energy Efficiency

In 2009-2010, PI Dr.Yi worked with Co-PI Dr.Qasem and several graduate students on studying the optimization space of multi-threaded scientific codes, the parameterization of the optimization space for automatic performance tuning, and the impact of performance optimizations on the power consumption of modern architectures. An experimental study was performed on optimizing and tuning three SPEC95 benchmarks collectively using the POET transformation engine developed at UTSA and the PSEAT search engine developed at Texas State University. The POET scripts were manually composed, and the PSEAT search engine was used to explore the optimization space and study the interactions of OpenMP parallelization and cache optimizations. This body of work has been published in NPC'2010 [7].

In 2010-2011, collaborating with Dr. Qasem, Dr.Yi worked with M. Faizur Rahman (a Ph.D student partially funded by this grant) on studying how to effectively optimize stencil codes on multi-core architectures. A variety of different strategies, including loop blocking, time skewing, pipelining, and wavefront parallelization, were implemented in a stencil kernel optimizer, using our POET language, to enhance both the parallelization and memory performance of several stencil kernels. To model the relationship between performance improvements achieved by different optimizations and their efficiency of utilizing various hardware components, we used hardware counters to monitor the efficiency of architectural components and then developed a set of formulas via linear regression analysis to model their overall performance impact in terms of the affected hardware counter numbers. Our experimental results show that a precise formula can be developed

for each kernel to accurately model the overall performance impact of varying optimizations and thereby effectively guide the performance analysis and tuning of these kernels. Our approach can be used to systematically extract meaningful insights from large collections of experimental data. Such insights can then be used by developers to enable more effective optimization tuning of their applications. This work has been published in Computing Frontior'2011 [10].

In 2010-2011, Dr. Yi worked with R. Faizur Rahman and Jichi Guo (both Ph.D students partially funded by this grant) on providing an automated empirical tuning framework that can be configured to optimize for both performance and energy efficiency of multi-threaded scientific codes on modern multi-core architectures. In particular, we extensively parameterized the configuration of a large number of compiler optimizations, including loop parallelization, blocking, unroll-and-jam, array copying, scalar replacement, strength reduction, and loop unrolling. We then used hardware counters combined with elapsed time to estimate both the performance and the power consumption of differently optimized code to automatically discover desirable configurations for these optimizations. We used a power meter to verify our tuning results on two multi-core computers and show that our approach can effectively achieve a balanced performance and energy efficiency on modern SMP architectures. Our experimental evaluation shows that by assigning different priorities when evaluating the overall efficiency of differently optimized code, our fine-grained auto-tuning system can simultaneously attain both high performance and high energy efficiency for several matrix computation kernels. This work has been published in HIPEAC'11 [8].

In 2011-2012, Dr. Yi worked with 5 Ph.D and MS students in her research group, while collaborating with Co-PI Dr Quinlan's group in DOE/LLNL, on an extensive study of the impact of application-level optimizations on the overall system power variations of two multi-core architectures, an 8-core Intel and a 32-core AMD workstation, by using these machines to execute a wide variety of sequential and multi-threaded benchmarks using varying compiler optimization settings and runtime configurations. Our extensive study provides insights into two questions: 1) what degrees of impact can application level optimizations have on reducing the overall system power consumption of modern CMP architectures; and 2) what strategies can compilers and application developers adopt to achieve a balanced performance and power efficiency. In particular, we found that for most benchmarks, while the variations in performance are much more dramatic than those in power consumption, there seems to be no direct correlation between the performance attained and the power consumption incurred by application-level optimizations. Similar levels of performance can often be attained while incurring dramatically different power consumptions. As a result, collectively optimizing both the performance and power efficiency of applications is not only possible, but also profitable. This work has been published in Computing Frontior'2012 [15].

In 2012-2013, Dr. Yi collaborated with Dr. Jiayuan Meng from DOE/Argonne National Laboratory, while working with Jichi Guo (a Ph.D student partially funded by this grant) on developing an analytical performance modeling framework to gain first-order insights into hardware-dependent application behavior, specifically, the most time-consuming regions of an application and reasons behind performance bottlenecks. Our framework analytically models the holistic execution flow of an application and then employs parameterized hardware performance models to project application performance for varying architecture design configurations. The overall workflow includes three steps. First, a source-to-source application analysis engine, developed using the ROSE compiler, analyzes the input code to generate a structural description of the application in the form of the SKOPE workload modeling language (developed by Dr. Meng) that depicts the applications control flow, data flow, communication, and computation intensity. During this step the analysis engine also automatically profiles the application on a local machine to obtain frequencies of the control-flow branches and incorporates the branch outcome distributions into the code skeleton. Second, based on the resulting code skeleton, our performance analysis engine automatically constructs

a static model for the application execution flow, which is then characterized using a hardware performance model parameterized with the appropriate architecture configuration. Finally, the performance projection for potential hot regions and their performance bottlenecks are projected and reported. The obtained performance insights include hot spots and their run time coverages, how they are reached and connected during the execution, and their limiting performance factors. By capturing the statistical behavior of the application control flow and integrating the estimated characteristics with hardware performance models, our technique is able to statically project and analyze the performance within a few minutes, and the projection time remains invariant to the input data size. We have validated our framework over DOE production codes, including an earthquake simulation application, and showed that the hot spot selection quality averages at 95.8% and is no worse than 80% in all cases. This work has been submitted for publication [22].

5 Publications

Following is a list of our publications produced during the funding period.

1. J. Magee, Q. Yi, and R. C. Whaley. Automated timer generation for empirical tuning. In The 4th Workshop on Statistical and Machine learning approaches to ARchitecture and compilaTion, Pisa,Italy., Jan. 2010.

2. A. M. Castaldo and R. C. Whaley. Scaling LAPACK Panel Operations Using Parallel Cache Assignment. In Proceedings of the 2010 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 223-231, Bangalore, India, January 2010.

3. Santosh Sarangkar and Apan Qasem, Restructuring Parallel Loops to Curb False Sharing on Multicore Architectures, Proceedings of the 11th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, Apr 2010

4. Q. Yi, S. Sarangkar, and A. Qasem. Improving autotuning effciency and portability through feedback diagnostics. In The Fifth International Workshop on Automatic Performance Tuning (position paper), Berkely, CA, June 2010.

5. Apan Qasem, Locality-Conscious Superpaging for Improved TLB Behavior of Stencil Computations, Proceedings of the 2010 International Conference on High Performance Computing Systems (HPCS10), Jun 2010.

6. Hammad Rashid, Clara Novoa, Apan Qasem, An Evaluation of Parallel Knapsack Algorithms on Multicore Architectures, Proceedings of 12th International Conference on Scientific Computing (CSC10), Jun 2010.

7. A. Qasem, J. Guo, F. Rahman, and Q. Yi. Exposing tunable parameters in multi-threaded numerical code. In 7th IFIP International Conference on Network and Parallel Computing, Zhengzhou, China, Sept. 2010.

8. Automated empirical tuning of scientific codes for performance and power consumption. S. F. Rahman, J. Guo, and Q. Yi. In High-Performance and Embedded Architectures and Compilers (HIPEAC'11), Pages 107–116, Heraklion, Greece, Jan 2011.

9. Automated Programmable Control and Parameterization of Compiler Optimizations. Qing Yi. In IEEE International Symposium on Code Generation and Optimization (CGO'11). Apr 02-06, 2011. Chamonix, France.

10. Understanding Stencil Code Performance On MultiCore Architectures. S. Faizur Rahman, Qing Yi, and Apan Qasem. In ACM International Conference on Computing Frontiers (CF'11). May 3-5, 2011. Ischia, Italy.

11. Extensive Parameterization And Tuning of Architecture-Sensitive Optimizations. Qing Yi and Jichi Guo. In The Sixth International Workshop on Automatic Performance Tuning (iWapt'11). Proceedings of the Internaltional Conference on Computational Science (ICCS). Pages 2156-2165. June, 2011.

12. Enhancing the Role of Inlining in Effective Interprocedural Parallelization. Jichi Guo, Mike Stiles, Qing Yi, and Kleanthis Psarris. In International Conference On Parallel Processing (ICPP'11), Sep, 2011. Taipei, Taiwan.

13. Achieving Scalable Parallelization For The Hessenberg Factorization. Anthony M. Castaldo and R. Clint Whaley. In IEEE Cluster 2011, pages 65-73, Austin, TX, September 26-30, 2011.

14. POET: A Scripting Language For Applying Parameterized Source-to-source Program Transformations. Qing Yi. Software Practice & Experience. John Wiley&Sons. Vol 42, issue 6, pages 675-706. May, 2012.

15. Studying The Impact Of Application-level Optimizations On The Power Consumption Of Multi-Core Architectures. S. Faizur Rahman, Jichi Guo, Akshatha Bhat, Carlos Garcia, Majedul H. Sujon, Qing Yi, Chunhua Liao, and Daniel Quinlan. In ACM International Conference on Computing Frontiers (CF'12). May 15-17, 2012. Cagliari, Italy.

16. A Highly Parallel Reuse Distance Analysis Algorithm on GPUs. Huimin Cui, Qing Yi, Jingling Xue, Lei Wang, Yang Yang, and Xiaobing Feng. In 26th IEEE International Parallel & Distributed Processing Symposium (IPDPS'12). May 21-25, 2012. Shanghai, China.

17. Layout-oblivious compiler optimization for matrix computations. Huimin Cui, Qing Yi, Jingling Xue, and Xiaobing Feng. ACM Transactions on Architecture and Code Optimization. Vol 9, No 4, pages 35:1-20. Jan, 2013.

18. Vectorization Past Dependent Branches Through Speculation. Majedul Haque Sujon, R. Clint Whaley, and Qing Yi. In the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT'13). Edinburgh, Scotland. Sep, 2013.

19. Enhancing Performance Portability of MPI Applications Through Annotation-Based Transformations. Md. Ziaul Haque, Qing Yi, James Dinan, and Pavan Balaji. In the 42nd International Conference on Parallel Processing (ICPP'13). Lyon, France. Oct, 2013.

20. AUGEM:Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs. Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. In the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13). Denver, CO. Nov, 2013.

21. POET: Parameterized Optimizations for Empirical Tuning. Qing Yi, Keith Seymour, Haihang You, Richard Vuduc and Dan Quinlan POHLL'07: Workshop on Performance Optimization for High-Level Languages and Libraries. Long Beach, California. Mar,2007.

22. J. Guo, J. Meng, Q. Yi, Vitali Morozov, and Kalyan Kumaran. Analytically Modeling Application Execution Flow For Software-hardware Co-design. Submitted for publication.