Project Report on DOE Young Investigator Grant (Contract No. DE-FG02-02ER25525)

Dynamic Scheduling and Fusion of Irregular Computation (August 15, 2002 to August 14, 2005)

Chen Ding
Department of Computer Science
University of Rochester
Rochester, NY

August 16, 2005

A Executive Summary

Computer simulation has become increasingly important in many scientific disciplines, but its performance and scalability are severely limited by the memory throughput on today's computer systems. With the support of this grant, we first designed training-based prediction, which accurately predicts the memory performance of large applications before their execution. Then we developed optimization techniques using dynamic computation fusion and large-scale data transformation. A major part of the project is the development of software tools. We have built a web-based 3D visualization tool for others to examine program locality. We have built a test environment to measure the potential of large-scale transformations. We have collaborated with the IBM production compiler group and implemented data reorganization in the IBM compiler. Directly or indirectly, this research has led to integer factor performance improvements in a variety of computational paradigms, including dynamic mesh simulation, hydrodynamics, molecular dynamics, multigrid methods, matrix algebra, and combinatorial search. Through this research, we have established direct relationship with research groups at many DoE laboratories and development groups at IBM and Intel. The result of this research has been presented at invited colloquia at dozens of conferences, universities, national and industry laboratories.

The research work has three major components. The first is modeling and prediction of cache behavior. We have developed a new technique, which uses reuse distance information from training inputs then extracts a parameterized model of the program's cache miss rates for any input size and for any size of fully associative cache. We have demonstrated that the predictions are accurate across a wide range of data sets that vary in size by many orders of magnitude. The technique provides good approximations even for caches of limited associativity. Using the model we have built a web-based tool using three dimensional visualization. The new model can help to build cost-effective computer systems, design better benchmark suites, and improve task scheduling on heterogeneous systems.

The second component is global computation for improving cache performance. We have developed an algorithm for dynamic data partitioning using sampling theory and probability distribution. The new algorithm outperforms existing sorting methods for both sequential and parallel sorting. For full size applications, we have studied a general method called computation fusion. Recent work from a number of groups show that manual or semi-manual computation fusion has significant benefits in physical, mechanical, and biological simulations as well as information retrieval and machine verification. We have developed an automatic tool that measures the potential of computation fusion. The tool allows for aggressive yet correct reordering by respecting the exact data and control dependences and applying complete data renaming. The new system can be used by high-performance application programmers to estimate the potential of locality improvement for a program before trying complex transformations for a specific cache system.

The last component studies models of spatial locality and the problem of data layout. In scientific programs, most data are stored in arrays. Grand challenge problems such as hydrodynamics simulation and data mining may use an enormous number of data elements. The best array layout depends on the program, the input, the hardware, and the stage of the execution. Moreover, the entire layout may need to change if even part of the program is modified. To optimize the layout across multiple arrays, we have developed a formal model called reference affinity. Experimental results show that the new model of reference affinity consistently outperform all other layouts given by the programmer, previous compiler analysis, frequency profiling, or statistical clustering on machines from all major vendors. Reference affinity, defined at the trace level, has a high overhead. We collaborated with the IBM production compiler group and designed an efficient compiler analysis that performs as well as data or code profiling does. Based on these results, the IBM group has filed a patent and is including this technique in their product compiler.

A major part of the project is the development of software tools. We have developed web-based visualization for program locality. The data can be viewed on the Internet from almost all Intel, Sun, SGI, and IBM platforms. The tool is interactive and allows a user to view miss rates in full precision and from all

angles. The user can zoom in, zoom out, rotate, or translate the result. In addition, we have implemented a prototype of array regrouping in the IBM compiler. The full implementation is expected to come out of IBM in the near future and to benefit scientific applications running on IBM supercomputers. We have also developed a test environment for studying the limit of computation fusion. Finally, our work has directly influenced the design of the Intel Itanium compiler.

The project has strengthened the research relation between the PI's group and groups in DoE labs. The PI was an invited speaker at the Center for Applied Scientific Computing Seminar Series at the early stage of the project. The question that the most audience was curious about was the limit of computation fusion, which has been studied in depth in this research. In addition, the seminar directly helped a group at Lawrence Livermore to achieve four times speedup on an important DoE code.

The PI helped to organize a number of high-performance computing forums, including the founding of a workshop on memory system performance (MSP). In the past two years, one fourth of the papers in the workshop came from researchers in Lawrence Livermore, Argonne, Las Alamos, and Lawrence Berkeley national laboratories. The PI lectured frequently on DoE funded research. His recent visits include Cornell, Rice, Washington, UIUC, MIT and research labs at IBM, Intel, and Microsoft. The support from this grant is explicitly acknowledged at all public lectures as well as listed on the PI's web page.

In a broader context, high performance computing is central to America's scientific and economic stature in the world, and addresses many of the most scientifically and socially important problems of our day. This research has improved the programming support for a variety of computational paradigms, including dynamic mesh, hydrodynamics, molecular dynamics, multi-grid methods, matrix algebra, and sequential and parallel sorting. In the process, the PI's group has developed and strengthened relationships with DoE laboratories and major hardware and software vendors.

B Contents

C Scientific Findings

Improving program locality requires understanding cache behavior and reorganizing both the computation and data. The next three sections describe a new, parameterized model of program cache behavior, two techniques for partitioning computation based on probability distribution for finding the limit of computation regrouping, and finally techniques for changing array data layout through array regrouping.

C.1 Whole-Program Locality Prediction

Since the structure of the processor's memory hierarchy is almost always fixed, the overall efficiency depends on how well the program reference pattern matches the given cache structure. The memory reference pattern of the program itself depends on the particular data set used for input. Thus, it is necessary to understand the fundamental memory reference behavior of a program *across all input data sets* to determine how effective the memory hierarchy will be at efficiently supporting the program in general.

Measurements of locality are made in one of three primary ways. One method is to have compilers perform sophisticated analysis of loop nests to estimate memory access patterns statically. A second method employs profilers to run an application with one or more input data sets and directly measure the dynamic memory behavior, but often with significant performance overhead. The third method uses analysis at run-time to *infer* behavior by sub-sampling activity. Limited sampling keeps the measurement overhead to a minimum. The latter two methods, profiling and run-time analysis, reveal behavior relative to a particular input data set. Static compiler analysis can provide behavior details for a range of input data sets, but accurate analysis is difficult if data indirection is used or if the control flow graph is complex. None of these methods adequately provide the capability to predict the memory reference patterns across a broad range of programs and data input sizes.

We show this exploration space graphically in Figure 1. The three dimensional space varies cache size on one axis, associativity on another, and program data set size on a third, for a fixed line size. While prior work on cache characterization techniques address how to quickly explore the planar space delineated by the size and associativity axes, our work shows how to extend the exploration along the program data set size axis in an efficient manner. The method utilizes *data reuse signature patterns*, a fundamental quality of program behavior [1, 2]. The paper was first published in a conference [3] and a symposium [4] and will appear as an article in IEEE Transactions on Computer [5].

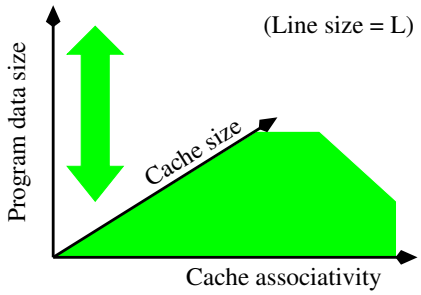
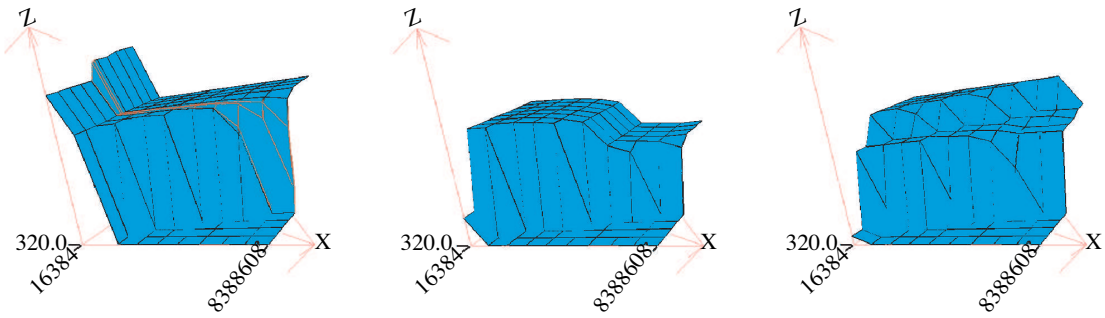


Figure 1: Cache behavior exploration space

The new technique uses reuse distance information from *two* input data sets of different sizes then extracts a parameterized model of the program's fundamental data reuse pattern. Regression with additional training input data sets can further improve the accuracy of the model. Our method converts the data reuse distance information into cache miss rates for any input size and for any size of *fully associative cache*. We have demonstrated that this method accurately depicts memory reference behavior and cache performance. The predictions are accurate across a wide range of data sets that vary in size by many orders of magnitude.



(a) Original Tomcatv (b) Fused Tomcatv (c) FusedGrouped Tomcatv

Figure 2: 3-D plots of unoptimized and optimized version of a vectorized mesh generation program *Tomcatv*

The technique provides good approximations even for caches of limited associativity.

A primary strength of this method is that, unlike static compiler analysis, the method is general so it easily accommodates programs with data indirection or complex dynamic control flow. However, not all programs exhibit predictable access patterns, though a surprising number of applications do. Also, the cache blocking factor is not currently part of the model.

Using the model we have built a visualization tool and used it to evaluate, as an example, two compiler transformations: reuse-based loop fusion and data regrouping [6]. The first three graphs in Figure 2 show the effect on *Tomcatv*. The different shapes of the miss-rate plots show that the improvement from the transformations is not uniform. The fusion reduces miss rates by 20% more when the cache size is larger than 1 MB. The third graph has two sharp jumps along the *Y* axis, one more compared with the first two graphs. This shows that the improvement from the combined transformation varies between three levels depending on the input size. The shapes of the plots show that the improvement is greater for larger data inputs, and the difference is greater for the combined transformation than for loop fusion alone.

By predicting complete program and machine behavior, the tool can help to build cost-effective computer systems, to design better benchmark suites, and to enable cache-sensitive task scheduling.

C.1.1 Cost-Effective Memory Hierarchy Design

Today's computing centers often use thousands of processors to run a few large applications. Rather than having to simulate numerous inputs and extrapolating how the system would perform from those runs, designers could save both time and money by being able to see the change in miss rates as cache size changes. Given the range of data sizes of an application, customers can quickly determine a balance between price and performance when deciding on their memory configurations. The information would also provide the same insight to existing systems that are being used to execute new or larger applications. The customers can check how the miss rate changes with larger data sizes, and whether the current configuration would provide reasonable results. If the system needs an upgrade, the utility could show whether the new investment could actually decrease the miss rate or simply wastes money. Last but not least, the tool can evaluate the cache performance of data sets too large to run on any existing machine. It may show whether the development of larger machines will allow execution of certain large data sets, or whether memory constraints will continue to prevent realistic execution times.

C.1.2 Benchmark Set Design

With accurate estimates of cache performance across numerous inputs, benchmark design may be improved to allow faster evaluation of systems and optimizations. By building benchmarks that run on values directly after knees in the data size versus miss rate plots, the smallest possible data size for a given miss rate could be used in testing. For example in *AppLU* and *Swim*, the new input is a factor of 4 or 12 smaller but its miss rate differs by no more than 0.6% from the larger input. A smaller input saves time in testing. The right input size depends on the program as well as the cache configuration. The prediction tool helps a user to quickly find the smallest input size that yields a particular cache miss rate. Our result may strengthen their method by increasing the coverage. It suggests that many programs have only a few different miss rates across all data inputs, and a wide range of inputs may have the same miss rate. The miss-rate prediction can ensure that a benchmark set includes all miss rates and the smallest program runs for these miss rates by suggesting them as candidates for benchmark selection. Furthermore, the suggestion is parameterized and therefore tailored to any cache configuration being considered.

C.1.3 Cache-Sensitive Task Scheduling

The locality model can be used in scheduling of applications on a heterogeneous environment with different machines installed with a different cache hierarchy. The pattern can be used to predict the miss rate on any number of cache levels of different configurations. A scheduler can then use the prediction for selecting machine-program pairs, in addition to considering the CPU speed and the system load on the machine. Since a scheduled application may take an unknown input, it is useful to know the possible range of the locality behavior.

C.2 Dynamic Fusion of Irregular Computation

C.2.1 Adaptive Data Partitioning

Many types of dynamic data have a total ordering and benefit from partial sorting into sublists. Examples include N-body simulation in physics and biology studies, where particles are partitioned based on their coordinates, and discrete-event simulation in computer networking and economics, where events are ordered by their arrival time. Partial sorting or partition allows these large scale problems to be solved by massively parallel computers. Data partition is also important on machines with a memory hierarchy because it dramatically improves cache performance for in-core data and memory performance for out-of-core data. Therefore, the efficient and balanced data partition is critical to good parallelism and locality.

In early 80s, Janus and Lamagna published a method that first samples the data and estimates its cumulative distribution function (CDF); it then assigns data into equal-size (not necessarily equal-length) buckets through direct calculation. This simple idea achieves balanced data partition in linear time, even for non-uniform distributions. Janus and Lamagna implemented their algorithm using the PL/1 language and measured the performance on an IBM 370 machine. Since then, however, this method seems forgotten and is rarely mentioned by later studies, which either have a high overhead or only apply to uniformly distributed data.

In search for a better sorting method for unbalanced data sets, we independently discovered the idea of using probability distribution. Compared to the method of Janus and Lamagna, our method makes three improvements. The first is a rigorous sampling method that ensures accurate estimate of the probability distribution. Our method guarantees the statistical accuracy for a large class of non-uniform distributions, with the number of samples independent of the size of data and number of categories. We have created an efficient implementation of probability calculations on modern machines. It uses temporary storage to avoid repeated calculations. It uses scalar expansion to improve instruction-level parallelism and hide memory latency, problems that did not exist on earlier machines.

We have applied in sequential and parallel sorting and compared it with the fastest sorting methods in the recent literature. Our *PD-partition* outperforms other partition methods in both efficiency and balance. For example, our sorting implementation outperforms quick-sort by over 10% and outperforms other cache-optimized algorithms by up to 30%. Furthermore, we designed a parallel sorting algorithm using the probability distribution. It achieves 33-50% time saving compared to two techniques popular on modern systems. This work has been published in the algorithm track of the ICPP 2004 conference [7].

C.2.2 The Limit of Computation Fusion

An effective strategy is for improving computation locality is to group computations on the same data so that once the data are loaded into cache, the program performs all their operations before the data are evicted. Well-known techniques such as loop interchange, blocking (tiling) and fusion regroup computations in regular loop nests. Recent studies show that more general forms of computation regrouping have significant benefits in other complex programs such as information retrieval, machine verification, and physical, mechanical, and biological simulations.

Computation regrouping is difficult to apply to large programs because related computations often spread far apart in the program code. Traditional dependence analysis is often not effective because of the complex control flow and indirect data access especially recursive functions and recursive data structures. Until now, the automatic methods are limited to programs written in loop nests, and manual transformation is limited to small fragments of a limited number of applications.

We have conducted a study on the potential of computation regrouping. First, we show that maximizing the locality is different from maximizing the parallelism or maximizing the cache utilization. We show that problem of computation regrouping is NP hard even when not considering program dependences and cache organization. We then develop a simulation tool that measures the potential of computation regrouping. The tool allows for aggressive yet correct reordering by respecting the exact data and control dependences and applying complete data renaming and memory re-allocation. It uses a constrained regrouping heuristic to improve the locality for a given cache size. Finally, we evaluate the new tool on a set of numerical and integer programs. The system and the results have been published in the SC 2004 conference [8].

The new system is useful to programmers, compiler writers, and computer architects. A programmer can use it to estimate the potential of locality improvement for a program before trying complex transformations for a specific cache system. A compiler writer can use it to study the potential improvement over the current techniques and to experiment with new regrouping strategies without a full compiler implementation. A computer architect can use it to tailor the memory system design to the available program locality.

The limit of program locality is vitally important to high-performance computing, where large systems are built to run a few applications. In 1988, Callahan, Cocke, and Kennedy gave a model called *balance* to measure the match between the memory demand of a program and the data bandwidth of a machine. Some programs run well on machines with a memory hierarchy, and some need the high-bandwidth vector memory. The simulation tool can show how the situation changes when program transformations are considered. If the locality of a program cannot be improved by the simulation tool, it is unlikely that the program can utilize cache well even with the best program optimization.

C.3 Array Regrouping in Scientific Programs

The third part of our work studies models of spatial locality and the problem of data layout. All current single-chip processors use cache blocks of at least 64 bytes, making the utilization an important problem. If only one word is useful in each cache block, a cache miss will not serve as a prefetch for other useful data. Furthermore, the program would waste up to 93% of memory transfer bandwidth and 93% of cache space, causing even more memory access.

In scientific programs, most data are in arrays. Grand challenge problems such as data mining or hydrodynamics simulation may use an enormous number of data elements. Which of these elements should be located near each other in memory? To maximize performance, a programmer must choose among an exponential number of possible layout strategies. The correct choice depends on the program, the input, the hardware, and the stage of the execution. Moreover the entire layout may need to change if even part of the program is modified.

To optimize the layout across multiple arrays, we have studied array regrouping. We first developed a formal model called reference affinity and then designed efficient compiler analysis, implemented using the IBM production compiler.

C.3.1 Reference Affinity

We define a relation called *reference affinity*. It measures how close a group of data are accessed *together* in an execution. Unlike most other program analysis, we measure the "togetherness" using the *LRU stack distance*, defined as the amount of data accessed between two memory references in an execution trace [9]. As a notion of locality, stack distance is bounded, even for long-running programs. The long stack distance often reveals long-range data access patterns that may otherwise hide behind complex control flows, indirect data access, or variations in coding and data allocation. We prove that the new definition gives a unique partition of program data for each distance k. When we decrease the value of k, the reference affinity gives a hierarchical decomposition and finds data sub-groups with closer affinity, much in the same way we sharpen the focus by reducing the radius of a circle.

In experiments, our new model of reference affinity consistently outperformed all other layouts given by the programmer, compiler analysis, frequency profiling, or statistical clustering on machines from all major vendors [10, 1, 11]. For example, *Swim* has 14 arrays and over 6 million possible choices, the analysis singles out a layout, which outperforms all others by a wide margin. The affinity hierarchy of *Swim* is a very impressive, large tree, as shown by Figure 3.

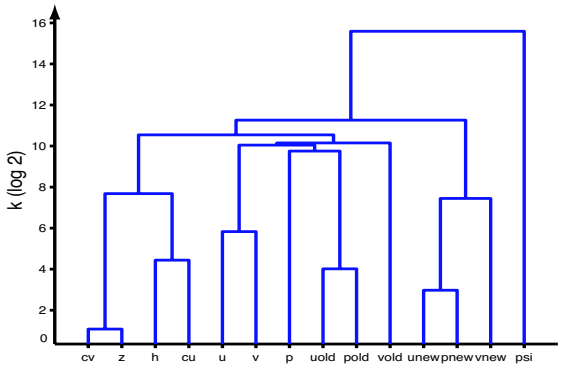


Figure 3: The Reference Affinity Hierarchy among Arrays of *Swim*

From a theoretical perspective, reference affinity has unified and extended previous, specialized results, including the Hilbert curve for N-body and mesh simulation, Morton layout for recursive matrix multiply, Cholesky factorization, and the wavelet transform. We proved highly structured mappings from the temporal pattern of computation to the spatial relationship of data. This is the first trace-based model of hierarchical data locality.

C.3.2 Compiler Analysis

Reference affinity, defined at the trace level using the distribution of reuse distances (reuse signatures), has a high overhead. The slowdown is at least 10 to 100 times. No production compiler is shipped with such

a costly technique. No one would do so before carefully examining whether such a high cost is justified. We collaborated with the IBM production compiler group at IBM Toronto Lab. One of my students spent a semester as a visiting student. Together we have developed lightweight techniques with the IBM group.

The basic idea is to use interprocedural program analysis to measure the access frequency in the presence of array parameters and aliases. To collect the frequency within a loop or a function, we studied two methods. The first is symbolic analysis by a compiler. The second is lightweight profiling. We implemented them in the IBM(r) FORTRAN compiler, evaluated them on SPEC2000 floating-point benchmark programs, and compared them with distance-based methods. Array regrouping improves the performance for the majority of programs tested, including full size applications for simulation of pollutant distribution, computational fluid dynamics, multi-grid solver, shallow water modeling, and molecular dynamics simulation. The pure compiler analysis performs as well as data or code profiling does. The work has been published in the ICS 2005 conference [12]. Based on these results, the IBM group has filed a joint patent and is including this technique in their product compiler.

In addition, we are collaborating with a group at Intel Programming Systems Lab on using reference affinity in improving their program code layout.

D Software Tools

D.1 Web-based Visualization for Program Locality

To use the tool all one needs is access to the Internet, and a browser equipped with the Java 1.4 JVM and Java 3D. *Http://www.cs.rochester.edu/research/locality* gives simple instructions on both how to acquire Java 3D and how to operate the applet. In implementation we leveraged Java3D code for the view controls but wrote custom code in about 50 Java classes to dynamically construct the miss rate model according to the user demand. Our implementation is fully integrated with Java3D. The system currently runs on practically all x86 machines (using Windows or Linux operating systems), Sun, SGI and IBM platforms, although it does not run on Apple platforms.

The tool displays the miss rates of a program across program inputs and cache configurations. For a given program, the tool generates a set of three-element tuples, each containing cache size in the x direction, data size in the y direction, and miss rate in the z direction. It then generates a plane that connects every three adjacent points (along x- and y-directions). The plane represents an approximation (linear interpolation) of the miss rates for the combinations of program inputs and cache sizes covered by the plane. Figure 2 shows examples of the three dimensional graphs.

The tool is interactive and allows a user to view miss rates in full precision and from all angles. To balance speed and accuracy the user can first view the miss rates of a program over a large range of values and subsequently reduce the range of values, effectively zooming in, increasing the number of points in the set around the critical values while avoiding time consuming calculations for estimates of the exploration space. The user can zoom into a single program input and a single cache size and get the exact miss rate.

Once the proper range of values has been determined, a user can zoom in, zoom out, turn the three-dimensional plot in four directions (up, down, left, and right) using the buttons on the bottom left hand side of the window. The user can also manipulate the graph by holding down the left or right mouse buttons and moving the mouse, causing the graph to rotate or translate respectively.

D.2 Implementation of Array Regrouping in the IBM Compiler

This work is implemented in IBM TPO (Toronto Portable Optimizer), which is the core optimization component in IBM C/C++/FORTRAN compilers. It implements both compile-time and link-time methods for intra- and interprocedural optimizations. It also implements profiling feedback optimizations. TPO uses

a common graph structure based on Single Static Assignment form (SSA) to represent the control and data flow within a procedure. Global value numbering and aggressive copy propagation are used to perform symbolic analysis and expression simplifications. It performs pointer analysis and constant propagation. For loop nests, TPO performs data dependence analysis and loop transformations after data flow optimizations.

The reference affinity analysis is implemented at the link step. A software engineering problem is whether to insert it before or after loop transformations. Currently the analysis happens first, so arrays can be transformed at the same compilation pass as loops are. Early analysis does not lead to slower performance in any of the test programs. We are looking at implementation options that may allow a later analysis when the loop access order is fully determined.

We have implemented the analysis that collects the static access-frequency vector and the analysis that measures per-basic-block execution frequency through profiling. We have implemented a compiler flag that triggers either static or profiling-based affinity analysis. The invocation graph is part of the TPO data structure. We are in the process of completing the analysis that includes the complete context sensitivity. The current access-frequency vector takes the union of all contexts. We have implemented the reference affinity graph and the linear-time partitioning. The array transformations are semi-automated as the implementation needs time to fully bond inside the compiler. This is joint work with Yaoqing Gao and Roch Archambault. Roch Archambault is the manager of the group Compiler Technology and Compiler Focus Point for High Performance Computing.

D.3 A Test Environment for The Limit of Locality

We have constructed a simulation tool that measures the potential of computation regrouping in five steps. The first step constructs instruction traces by instrumenting programs to record relevant information. Of course, not all ordering is permissible. The next two steps identify the exact control and data dependences in the trace. The control dependence analysis uses a novel algorithm to handle recursive programs. The data dependence analysis uses complete renaming to avoid false dependences. The fourth step applies constrained computation regrouping to improve the locality for a given cache size. Finally, the last step re-allocates data variables to reduce the memory usage after reordering.

The simulation tool is trace-based and has several limitations. A trace may not represent the program behavior on other inputs, and a trace may be too large to be analyzed. For many programs, earlier work has shown that the temporal locality follows a predictable pattern and the (cache miss) behavior of all program inputs can be predicted by examining medium-size training runs [2, 4, 3]. The simulation uses heuristics and does not find the optimal locality. However, one can use the simulation tool to measure the lower bound of locality improvement in complex programs by trace-level computation reordering.

D.4 Implementation of Reuse-base Loop Fusion in the Intel Compiler

In 2001, I gave a talk at a joint seminar between several Intel groups, when I described reuse-based loop fusion by Ken Kennedy and I. In 2003, a group at Intel Compilation Laboratory published a paper in 12th International Conference on Parallel Architectures and Compilation Techniques (PACT'03). The paper is titled "Inter-Procedural Loop Fusion, Array Contraction and Rotation." The authors, John Ng, Dattatraya Kulkarni, Wei Li, Robert Cox, and Scott Bobholz, cited our work as an enabling technology for array contraction and rotation. They reported 12% average speed improvement for SPEC 2000 floating-point benchmarks on Intel Itanium2.

E Service to the High Performance Computing Community

In July 2003, I spoke at the Center for Applied Scientific Computing Seminar Series, hosted by Bronis de Supinski and Daniel Quinlan. I talked about global (including reuse-based loop fusion) and dynamic cache optimization. A number of researchers in the audience asked the question how much room were there for locality improvement. This was exactly the question we proposed to study (Section F in the proposal). Working with first Dan Williams and then Maksim Orlovich, we developed techniques and tools to analyze the limit of computation reordering. The work is mentioned in Section **??**. The technical paper is published in SC 2004 [8]. Williams and Orlovich were undergraduate students at the time and have joined the Ph.D. program at Cornell University.

Another outcome of the seminar is the help to their project on optimizing object-oriented mesh-based code. In 2005 in the International Conference on Supercomputing, they published a paper titled "Improving the Computational Intensity of Unstructured Mesh Applications." The authors are Brian White and Sally McKee at Cornell and Bronis de Supinski, Brian Miller, Daniel Quinlan, and Martin Schulz from Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. The paper cited the study by Ken Kennedy and I [13] for a technique (consecutive packing) that "increases the number of useful prefetches by 30%" and "a 7% performance improvement." Overall they achieved 4.1 times speedup in execution time.

Our work has cited in a number of other studies, including dynamic code and data reordering for scientific simulation code (Strout et al. first as a student at UCSD, in ACM PLDI 2003, and then a postdoc at Argonne, in MSP 2004), the optimization of electronic structure calculations (Cociorva et al. from Ohio State University, ACM PLDI 2002) and the use in tiling and array contraction for scientific programs (Geoff Pike et al. from Berkeley, SC 2002).

I have helped to organize several high-performance computing conferences. I am a program committee member for the following conferences the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe, New Mexico, April 2004; the 11th International Conference on High Performance Computing (HiPC), Bangalore, India, December 2004; and the 32nd International Conference on Parallel Processing (ICPP 2003), 6-9 October 2003, Kaohsiung, Taiwan. In 2004, I served as the vice program chair for the 33rd International Conference on Parallel Processing (ICPP 2003). In addition, my student presented a paper in the 4th Annual Symposium of the Las Alamos Computer Science Institute in 2003.

A group of us started the workshop on memory system performance, which has held three times in 2002, 2004 and 2005. I serve in the steering committee from the beginning, in the program committee in 2002, and as the general chair in 2004. The workshop has become a valuable forum for publication and discussion for memory issues in general purpose and high performance systems. In fact, three out of seven papers in MSP 2004 had authors in one of the DoE labs. Here is a list of the papers published in the past workshops on high performance computing.

- "Automatic Blocking Of QR and LU Factorizations for Locality," Qing Yi (Lawrence Livermore National Laboratory), Ken Kennedy (Rice University), Haihang You, Keith Symour, Jack Dongarra (University of Tennessee), MSP 2004.
- "Metrics and Models for Reordering Transformations," Michelle Strout, Paul Hovland (Argonne National Laboratory and University of Chicago), MSP 2004.
- "An Empirical Performance Analysis of Commodity Memories in Commodity Servers," Darren Kerbyson, Mike Lang (Los Alamos National Laboratory), Gene Patino, Hossein Amidi (Smart Modular Technologies Inc), MSP 2004.
- "Application Analysis Using Memory Pressure," Kartik Sudeep, Ahmed Gheith (IBM Research, Austin), MSP 2005.

- "Impact of Modern Memory Subsystems on Cache Optimizations for Stencil Computations," Shoaib Kamil, Parry Husbands, John Shalf, Leonid Oliker, Kathy Yelick (Lawrence Berkeley National Labs), MSP 2005.

In addition, I lectured on my research at many universities and companies. See my C.V. for a complete list. I presented a High-Performance Computing seminar at IBM T. J. Watson in August 2004. When I gave a seminar at UC Berkeley in October 2004, David Bailey and two other researchers from Lawrence Berkeley Lab came to my talk. In May 2005, I took a two-week tour and gave seminars at six universities and two industry labs: Cornell, University of Minnesota, Microsoft Research, University of Washington, Intel Programming System Lab, University of Arizona, Rice University, and Georgia Institute of Technology. In every lecture, I acknowledge the support of this grant explicitly on my last slide.

F References

[1] Y. Zhong, C. Ding, and K. Kennedy. Reuse distance analysis for scientific programs. In *Proceedings of Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, Washington DC, March 2002.

[2] C. Ding and Y. Zhong. Predicting whole-program locality with reuse distance analysis. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, CA, June 2003.

[3] Y. Zhong, S. G. Dropsho, and C. Ding. Miss rate prediction across all program inputs. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, New Orleans, Louisiana, September 2003.

[4] X. Shen, Y. Zhong, and C. Ding. Regression-based multi-model prediction of data reuse signature. In *Proceedings of the 4th Annual Symposium of the Las Alamos Computer Science Institute*, Sante Fe, New Mexico, November 2003.

[5] Y. Zhong, S. Dropsho, X. Shen, A. Stude r, and C. Ding. Miss rate prediction across all program inputs. *IEEE Transactions on Computers*. to appear.

[6] C. Ding and K. Kennedy. Improving effective bandwidth through compiler enhancement of global cache reuse. In *Proceedings of International Parallel and Distributed Processing Symposium*, San Francisco, CA, April 2001. http://www.ipdps.org.

[7] X. Shen and C. Ding. Adaptive data partition for sorting using probability distribution. In *Proceedings of International Conference on Parallel Processing*, Montreal, Canada, August 2004.

[8] C. Ding and M. Orlovich. The potential of computation regrouping for improving locality. In *Proceedings of SC2004 High Performance Computing, Networking, and Storage Conference*, Pittsburgh, PA, November 2004.

[9] R. L. Mattson, J. Gecsei, D. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM System Journal*, 9(2):78–117, 1970.

[10] C. Ding and Y. Zhong. Compiler-directed run-time monitoring of program data access. In *Proceedings of the first ACM SIGPLAN Workshop on Memory System Performance*, Berlin, Germany, June 2002.

[11] Y. Zhong, M. Orlovich, X. Shen, and C. Ding. Array regrouping and structure splitting using whole-program reference affinity. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2004.

[12] X. Shen, Y. Gao, C. Ding, and R. Archambault. Lightweight reference affinity analysis. In *Proceedings of the 19th ACM International Conference on Supercomputing*, Cambridge, MA, June 2005.

[13] C. Ding and K. Kennedy. Improving cache performance in dynamic applications through data and computation reorganization at run time. In *Proceedings of the SIGPLAN '99 Conference on Programming Language Design and Implementation*, Atlanta, GA, May 1999.

G Curriculum Vitae: Chen Ding

Computer Science Department, University of Rochester, Rochester, NY 14627-0226.
(716) 275-1373, cding@cs.rochester.edu

Professional Preparation

Beijing (Peking) University	Computer Science	B.S.	July 1994
Michigan Technological University	Computer Science	M.S.	June 1996
Rice University	Computer Science	Ph.D.	January 2000

Appointments

University of Rochester	Assistant Professor	August 2000 to present
Rice University	Postdoc and Lecturer	January to July 2000

Awards

Faculty Early Career Development (CAREER) Award, National Science Foundation, 2002

Early Career Principal Investigator Award, Office of Science, US Department of Energy, 2001

Best paper award, International Parallel and Distributed Processing Symposium, San Francisco, California, April 2001

Honorable Mention, Best Dissertation Competition, Rice University–Texas Medical Center Sigma Xi competition in the Physical Sciences/Engineering, May 2000

Journals

[JPDC'04] "Improving Effective Bandwidth through Compiler Enhancement of Global Cache Reuse," Chen Ding and Ken Kennedy, *Journal of Parallel and Distributed Computing*, Volume 64, Issue 1, January 2004, Elsevier Press, pages 108–134. Introduced reuse-based loop fusion, which helped the Intel compiler group to obtain a 12% speed improvement for SPECfp2K benchmarks on the Intel Itanium2 (see Ng et al., Proceedings of PACT 2003).

[TOC] "Miss Rate Prediction across All Program Inputs," Yutao Zhong, Steve Dropsho, Xipeng Shen, Ahren Studer, and Chen Ding, *IEEE Transactions on Computers*, *accepted in June 2005*.

[TOPLAS] "Distance-Based Locality Analysis and Prediction," Chen Ding, Yutao Zhong, and Xipeng Shen, *ACM Transactions on Programming Languages and Systems*, *accepted with major revision in June 2005*.

Refereed Conferences

[ICS'05] "Lightweight Reference Affinity Analysis," Xipeng Shen, Yaoqing Gao, Roch Archambault, and Chen Ding, in *Proceedings of the International Conference on Supercomputing*, Boston, Massachusetts, June 2005, pages 131–140.

[ASPLOS'04] "Locality Phase Prediction," Xipeng Shen, Yutao Zhong, and Chen Ding, in *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems*, Boston, Massachusetts, October 2004, pages 165–176.

[PLDI'04] "Array Regrouping and Structure Splitting using Whole-Program Reference Affinity," Yutao Zhong, Maksim Orlovich, Xipeng Shen, and Chen Ding, in *Proceedings of the ACM*

SIGPLAN Conference on Programming Language Design and Implementation, Washington DC, June 2004, pages 255–266.

[PACT'04] "The Energy Impact of Aggressive Loop Fusion," Yongkang Zhu, Grigorios Magklis, Michael L. Scott, Chen Ding, and David Albonesi, in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Antibes Juan-les-Pins, France, October 2004, pages 153–164.

[SC'04] "The Potential of Computation Regrouping in Improving Locality," Chen Ding and Maksim Orlovich, in *Proceedings of the SC2004 High Performance Computing, Networking, and Storage Conference*, Pittsburgh, PA, November 2004.

[ICPP'04] "Adaptive Data Partition for Sorting using Probability Distribution," Xipeng Shen and Chen Ding, in *Proceedings of the International Conference on Parallel Processing (algorithm track)*, Montreal, Canada, August, 2004, pages 250–258.

[PLDI'03] "Predicting Whole-Program Locality through Reuse Distance Analysis," Chen Ding and Yutao Zhong, in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, California, June 2003, pages 245–257.

[PACT'03] "Miss Rate Prediction across All Program Inputs," Yutao Zhong, Steven G. Dropsho, and Chen Ding, in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, New Orleans, Louisiana, September 2003, pages 48–63.

[IPDPS'01] "Improving Effective Bandwidth through Compiler Enhancement of Global Cache Reuse," Chen Ding and Ken Kennedy, in *Proceedings of 2001 the International Parallel and Distributed Processing Symposium*, San Francisco, California, April 2001. **(Best Paper Award)**

[IPDPS'00] "Memory Bandwidth Bottleneck and its Amelioration by a Compiler," Chen Ding and Ken Kennedy, in *Proceedings of the International Parallel and Distributed Processing Symposium*, Cancun, Mexico, May 2000, pages 181–190.

[PLDI'99] "Improving Cache Performance in Dynamic Applications through Data and Computation Reorganization at Run Time," Chen Ding and Ken Kennedy, in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, Atlanta, Georgia, May 1999, pages 229–241.

[PDCS'99] "Bandwidth-Based Performance Tuning and Prediction," Chen Ding and Ken Kennedy, in *Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, Cambridge, Massachusetts, November 1999, pages 693–699.

[Euro-Par'97] "Modulo Scheduling with Cache Reuse Information," Chen Ding, Steve Carr and Phil Sweany, in *Proceedings of the International Euro-Par Parallel Processing Conference*, Passau, Germany, August 1997. Springer-Verlag Lecture Notes in Computer Science 1300 (C. Lengauer, M. Griebl and S. Gorlatch, editors), pages 1079–1083.

[HICSS'96] "Improving Software Pipelining with Unroll-and-Jam," Steve Carr, Chen Ding and Phil Sweany, in *Proceedings of the Hawaii International Conference on System Sciences*, Maui, Hawaii, January 1996, pages 183–192.

Refereed Workshops

[MSP'05] "Gated Memory Control for Memory Monitoring, Leak Detection and Garbage Collection," Chen Ding, Shengliang Zhang, and Mitsu Ogihara, in *Proceedings of the Workshop on Memory Performance*, Chicago, June 2005.

[LCR'04] "Memory Access Analysis and Optimization Approaches on Splay Trees," Wei Jiang, Chen Ding, and Roland Cheng, in *Proceedings of the Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, Houston, Texas, October 2004.

[TDM'04] "Finding Reference Affinity Groups in Traces Using Sampling Method," Chengliang Zhang, Yutao Zhong, Chen Ding and Mitsu Ogihara, presented in the Workshop on Mining Temporal and Sequential Data, in conjunction with the ACM SIGKDD conference, Seattle, Washington, August 2004.

[LCPC'04] "Phase-Based Miss Rate Prediction," Xipeng Shen, Yutao Zhong, and Chen Ding, in *Proceedings of the International Workshop on Languages and Compilers for Parallel Computing*, West Lafayette, Indiana, September 2004, to appear in Springer-Verlag Lecture Notes in Computer Science.

[LCPC'03] "A Hierarchical Model of Reference Affinity," Yutao Zhong, Xipeng Shen, and Chen Ding, in *Proceedings of the International Workshop on Languages and Compilers for Parallel Computing*, College Station, Texas, October 2003. Springer-Verlag Lecture Notes in Computer Science 2958 (L. Rauchwerger, editor), pages 48–63.

[LACSI'03] "Regression-Based Multi-Model Prediction of Data Reuse Signature," Xipeng Shen, Yutao Zhong, and Chen Ding, in *Proceedings of the Annual Symposium of the Los Alamos Computer Science Institute*, Sante Fe, New Mexico, October 2003.

[MSP'02] "Compiler-Directed Run-Time Monitoring of Program Data Access," Chen Ding and Yutao Zhong, in *Proceedings of the ACM SIGPLAN Workshop on Memory System Performance*, Berlin, Germany, June 2002. ACM SIGPLAN Notices, March 2003, pages 1–12.

[LCR'02] "Reuse Distance Analysis for Scientific Programs," Yutao Zhong, Chen Ding, and Ken Kennedy, in *Proceedings of the Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, Washington D.C., March 2002.

[LCPC'01] "Instruction Balance and its Relation to Program Energy Consumption," Tao Li and Chen Ding, in *Proceedings of the International Workshop on Languages and Compilers for Parallel Computing*, Cumberland Falls, Kentucky, August 2001. Springer-Verlag Lecture Notes in Computer Science 2624 (H. G. Dietz, editor), pages 71–85.

[LCPC'99] "Inter-array Data Regrouping," Chen Ding and Ken Kennedy, in *Proceedings of the International Workshop on Languages and Compilers for Parallel Computing*, San Diego, California, August 1999. Springer-Verlag Lecture Notes in Computer Science 1863 (J. Ferrante and L. Carter, editors), pages 149–163.

Currently Under Review

"Predicting Locality Phases for Dynamic Memory Optimization," Xipeng Shen, Yutao Zhong, and Chen Ding, submitted to a journal, August 2005.

"A Hierarchical Model of Data Locality," Chengliang Zhang, Yutao Zhong, Mitsunori Ogihara, and Chen Ding, submitted to a conference, July 2005.

"Behavior-based Parallelization," Xipeng Shen and Chen Ding, submitted to a workshop, July 2005.

"Measuring Temporal Locality Variation Across Program Inputs," Roland Cheng and Chen Ding, submitted to a symposium, July 2005.

Patent

"Temporal affinity analysis using reuse signatures," Chen Ding, Yutao Zhong, and University of Rochester, filed at the US Patent and Trademark office in January 2004.

Professional Service

Organized the tutorial "Program Locality Models and Their Use in Memory Performance Optimization", which was the most attended among eight tutorials at ACM ASPLOS (sponsored by SIGARCH, SIGPLAN, and SIGOPS) conference in Boston, MA, October 2004. The tutorial will happen again in PACT in St. Louis in September 2005.

Co-organized the first and chaired the second (General Chair) ACM SIGPLAN Workshop on Memory System Performance (MSP) in 2002 and 2004.

Vice Program Chair (compilers and programming languages track), International Conference on Parallel Processing, Montreal, Canada, 2004.

Program Committee member, PACT 2006, PLDI 2005, CGO 2005, IPDPS 2004, HiPC 2004, NPC 2004, EUC 2004, ICPP 2004, ICPP 2003, CDP 2004, CDP 2004, and MSP 2002.

Steering Committee member for Workshop on Memory System Performance (MSP), sponsored by ACM Special Interest Group for Programming Languages (SIGPLAN), from 2002; Compiler-Directed Performance Workshop (CDP), sponsored by IBM and Canadian Research Council, from 2004.

Organizing Committee member for ASPLOS 2006, PPoPP 2005, and PACT 2003.

Member, NSF Panel for the CCF regular program in 2005 and 2004, the OCR regular program in 2003, and the medium-size ITR program in 2001.

Reviewer for Cambridge University Press; Morgan Kaufmann Publishers; Journal of Parallel and Distributed Computing (JPDC); ACM Transactions on Programming Languages and Systems (TOPLAS); ACM Transactions on Embedded Computing Systems (TECS); IEEE Transactions on Parallel and Distributed Systems (TPDS); IEEE Transactions on Computers (TOC); Journal of Functional Programming.

Reviewer for ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI); International Conference on Parallel Architecture and Compilation Techniques (PACT); International Conference on High Performance Computing and Communications (SCxy); ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES); ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS); International Symposium on Computer Architecture (ISCA); International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES); IEEE Conference on High Performance Computer Architecture (HPCA); IEEE Conference on Application-specific Systems, Architectures, and Processors (ASAP); ACM Java Grande Conference; Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR).

Grants

PI, "Program phase detection and exploitation," NSF Computer System Research, 5/05–5/09, $520K.

PI, "Compiler-assisted data adaptation," NSF Faculty Early Career Development (CAREER) award, 5/03–4/08, $400K.

PI, "Dynamic scheduling and fusion of irregular computation," DoE Early Career Investigator program, 9/02–8/05, $294K.

Co-PI, "ITR: Dynamically tunable clustered multithreaded architectures," NSF ITR Small Grants program, 9/02–8/04, $250K.

Co-PI, "Energy-efficient computation and communication in wireless devices," DARPA, 3/02–12/02.

Invited Presentations

Presented "Locality and phases: dynamic structures in complex program behavior" at

- Compiler and computer architecture groups, Department of Computer Science, University of Illinois, Urbana-Champion, July 2005, hosted by David Padua
- colloquium, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, June 2005, hosted by Martin Rihnard
- colloquium, Center for Experimental Research in Computer Systems, Georgia Institute of Technology, May 2005, hosted by Santosh Pande
- colloquium, Center for High Performance Software, Rice University, May 2005, hosted by Ken Kennedy
- colloquium, Department of Computer Science, University of Arizona, Tucson, May 2005, hosted by Rajiv Gupta
- Intel Programming Systems Lab, Santa Clara, May 2005, hosted by Youfeng Wu
- colloquium, Department of Computer Science, University of Washington, Seattle, May 2005, hosted by Craig Chamber
- Microsoft Research, Seattle, May 2005, hosted by Trishul Chilimbi
- Department of Computer Science Colloquium, University of Minnesota, Minneapolis, May 2005, hosted by Pen Yew, department chair
- Computer System Lab, Cornell University, May 2005, hosted by Keshav Pingali

Presented "Distance-based locality analysis and optimization" at

- EECS Department, University of California at Berkeley, October 2004, hosted by Alan Smith
- Computer Group, Department of Electrical Engineering, University of Toronto, October 2004, hosted by Tarek Abdelrahman
- High-Performance Computing Seminar, IBM T. J. Watson, New York, August 2004, hosted by Peng Wu

Presented "Understanding and Improving Program Memory Behavior" at

- Department of Electronics and Information Systems, University of Ghent, Ghent, Belgium, May 2004, hosted by Eric D'Hollander
- Compiler group, Institute of Computing Technology, Chinese Academy of Sciences, January 2004, hosted by Zhang Zhaoqing
- Compiler group, Intel Beijing Lab, January 2004, hosted by Lueh Guei-Yuan
- Compiler groups, IBM Toronto Lab, October 2003, hosted by Kevin Stoodley and Rock Archembault
- Center for High Performance Software, Rice University, October 2003, hosted by Ken Kennedy

– Department of Computer Science, University of Texas at Austin, September 2003, hosted by Kathryn McKinley

Presented "Whole-program reference affinity" at Compiler groups, IBM Toronto Lab, March 2004, hosted by Kevin Stoodley and Rock Archembault

Presented "Compiler Management of Dynamic Data Reuse" at Intel Programming Systems Lab at Santa Clara, June 2003, hosted by Shih-Wei Liao

Presented "Compiler Management of Global and Dynamic Data Reuse" at

– Center for Applied Scientific Computing Seminar Series, DoE Lawrence Livermore National Lab, June 2003, hosted by Bronis de Supinski and Dan Quinlan
– System research groups, CS and ECE departments, Cornell University, August 2002, hosted by Keshav Pingali
– Advanced Computer Systems Laboratory, Purdue University, March 2002, hosted by Rudolf Eigenmann
– Computer Architecture and Parallel System Laboratory, University of Delaware, August 2001, hosted by Guang Gao
– Compiler group, Texas Instruments Inc., Houston, August 2001, hosted by Reid Tatge and Brad Huber
– Production and research compiler groups, Intel Microprocessor Research Lab, Santa Clara, April 2001, hosted by Shih-Wei Liao and Roy Ju

Courses

CSC 255/455, "Advanced programming systems," Fall 2004, Fall 2003, Fall 2002. I created the course to teach advanced compilation, dependence theory, and programming language semantics to graduate students and senior undergraduate students.

CSC 252, "Computer organization," Spring 2004, Spring 2003.

CSC 200, "Undergraduate problem seminar," Spring 2002.

CSC 573, "Advanced compilation seminar," Spring 2001.

CSC 254/454, "Programming language design and implementation," Fall 2001 and Fall 2000.

Thesis Committees

Thesis advisor for Yutao Zhong, Xipeng Shen, and Kirk Kelsey, Computer Science. Dr. Zhong is my first student and will soon join George Mason University as an assistant professor. She is the first female graduate of the systems group in the history of the department. She was the only female speaker in the 2004 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).

Committee member for William Scherer III, a Ph.D. candidate in Computer Science; Grigoris Maglis, a Ph.D. candidate in Computer Science, currently at Intel Lab at Barcelona, Spain; Rajeev Balasubramonian, Ph.D. in Computer Science in 2003, now an Assistant Professor in the Computer Science Department, University of Utah.

External committee member for Brandon Jasonowski and Michelle Lay, M.S. Computer Engineering in 2004; Lei Chen, Ph.D. in Computer Engineering in 2004, now at IBM Austin; Ahmet Becene, Ph.D. in Mechanical Engineering in 2003, first employment at Delphi Co; Wenjie Xie, Ph.D. in Biomedical Engineering in 2002, first employment at Ansys Co; Yongkang Zhu, a Ph.D. candidate in Computer Engineering.

Undergraduate Research

2004: Maksim Orlovich, Roland Cheng and Vlad Vanyukov.

2003: Maksim Orlovich and Ahren Studer.

2002-2003: Grant Farmer, Dan Williams, Leo Boyarsky, Matthew Nettleton, Habib Chaudhary, and Eli Kim.

Ph.D. program placement after graduation: Maksim Orlovich (2004), Cornell University; Ahren Studer (2004), Carnegie Mellon University; Dan Williams (2003), Cornell University.

Departmental Service

Faculty recruiting committee member, 2004–05.

Colloquium committee member, 2004–05.

College Faculty Council representative, 2001–04.

Curriculum committee member, 2002–04.

Admission committee member, 2000–2002.

Student Summer Intern Placements

Tongxin Bai, Intel Beijing Lab, Summer 2005

Kirk Kelsey, Lawrence Livermore National Lab, Summer 2004

Xipeng Shen, visiting student, IBM Toronto Lab, spring 2005; Microsoft Research, Seattle, Summer 2004

Yutao Zhong, Intel Beijing Lab, Summer 2004; Microsoft Research, Seattle, Summer 2002

Yongkang Zhu, Intel Beijing Lab, Summer 2003

H Selected Publications

We attach in this report a copy of the upcoming article in IEEE Transactions on Computers, the paper in the International Conference on Supercomputing in 2005, the SC 2004 High Performance Computing, Networking, and Storage Conference, the International Conference on Parallel Processing in 2004, the ACM SIGPLAN Conference on Programming Language Design and Implementation in 2004, and the 4th Annual Symposium of the Las Alamos Computer Science Institute in 2004. The complete list and a pdf copy of all papers are available from the web at *http://www.cs.rochester.edu/ cding/publications.html*.