CURRICULUM VITAE

Jean Christophe BEYLER

Born the 6th of august 1980 in Seoul, South Corea
Nationality : French

Address

personal : 4A, rue de Buehl
67120 Avolsheim
France

professional : Laboratoire des Sciences de l'Image,
de l'Informatique et de la Télédétection UMR 7005 ULP-CNRS
Pôle API, Bd Sébastien Brant, 67400 Illkirch
France

Tel : (33) 3 90 24 45 52   Fax : (33) 3 90 24 45 47
Email : *beyler@icps.u-strasbg.fr*
Webpage : http ://icps.u-strasbg.fr/∼beyler

Education

Since 2004	PhD Candidate in Computer Science at the *ICPS* (Image et Calcul Parallèle Scientifique) Team, part of the *LSIIT* (Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection) Laboratory (UMR 7005 ULP-CNRS), Strasbourg, France. **Title of the PhD thesis :** Dynamic Memory Access Optimization. **PhD Advisor :** Pr. Philippe Clauss.
2002-2004	Master in Computer Science with honors (Rank : 1st). University of Louis Pasteur, Strasbourg.
1999-2002	Bachelor in Computer Science with honors (Rank : 1st). University of Louis Pasteur, Strasbourg.
1998-1999	First year of Medical School University of Louis Pasteur, Strasbourg.
1998	High School Diploma, Scientific specialty International High School of Hong Kong.

Internships

2005	Worked on dynamic optimization for the Itanium processor as a part of my thesis with Professor Gao. CAPSL, University of Delaware, Newark, DE, USA (4 months).
2004	Graduate intern, *Esodyp : an Entirely Software and Dynamic Data Prefetcher based on a Markov Model.* ICPS Team/LSIIT Laboratory, Strasbourg (5 months).

Teaching activities

2004-2007	Worked as a teaching assistant at the Computer Science Department of the University of Louis Pasteur, Strasbourg. Subjects taught : Introduction to Computer Science, Imperative Programming, Shellscripts and Compilation Theory.

Oral Presentations

August 2007	Presentation of the paper *Esodyp+ : Prefetching in the Jackal Software DSM* at the International Conference Euro-Par 2007, Rennes, France.
June 2007	Presentation of the paper *Performance Driven Data Cache Prefetching in a Dynamic Software Optimization System*, at the 21st ACM International Conference on Supercomputing, ICS'07, Seattle, USA.
June 2007	Talk on *Dynamic optimization systems for the Itanium processor* at Reservoir Labs, New York City, NY.
January 2006	Presentation of the paper *ESODYP : An Entirely Software and Dynamic Data Prefetcher based on a Markov Model* at the 12th Workshop on Compilers for Parallel Computers, CPC'06, A Coruna, Spain.

International conferences

[1] Michael Klemm, Jean Christophe Beyler, Ronny T. Lampert, Michael Philippsen and Philippe Clauss, *Esodyp+ : Prefetching in the Jackal Software DSM*, Euro-Par 2007, Rennes, France, August 2007.

[2] Jean Christophe Beyler and Philippe Clauss, *Performance Driven Data Cache Prefetching in a Dynamic Software Optimization System*, 21st ACM International Conference on Supercomputing, ICS'07, Seattle, WA, USA, June 2007.

[3] Jean Christophe Beyler and Philippe Clauss, *ESODYP : An Entirely Software and Dynamic Data Prefetcher based on a Markov Model*, Proceedings of the 12th Workshop on Compilers for Parallel Computers, CPC 2006, University of A Coruna, pages 118-132,ISBN :54-609-8459-1, A Coruna, Spain, January 2006.

[4] Ph. Clauss, B. Kenmei and J.C. Beyler, *The Periodic-Linear Model of Program Behavior Capture*, Euro-Par 2005, LNCS 3648, pages 325-335, Springer, august-september, 2005, Lisboa, Portugal.

Submitted papers

[5] Jean Christophe Beyler and Philippe Clauss, *Lightweight Profiling for Memory Access Optimizations*, submitted to an international journal.

[6] Jean Christophe Beyler and Philippe Clauss, *Advanced Instrumentation Code Abstraction for Dynamic Data Cache Prefetching*, submitted to an international conference.

[7] Jean Christophe Beyler, Michael Klemm, Michael Philippsen and Philippe Clauss *Markov-based Prefetching with Binary Code Rewriting in Object-based DSMs*, submitted to an international conference.

Research activity

Introduction

When a program accesses data, the latency is generally hidden by data cache memories. A cache miss occurs when the data is not in the cache and the program needs to stall, waiting for the data to arrive from memory. Data prefetching is an efficient solution to hide cache miss latencies since it limits the number of stalls by requesting the data in advance. If done correctly, the data will be directly in the cache when the program is accessing it. However, in order to be efficient, the different optimization schemes should only consider loads whose latencies are higher than the cache access latency, also called *delinquent loads*. Once the delinquent loads are selected, a process calculating what to predict and also when to prefetch the data must be implemented. Prefetching too late means that the data will not be present in the cache when the program will require it, prefetching too early risks a data eviction before the program needs the data. The right distance must be used in order for the data to be present at the right time.

Certain optimization strategies can be used on programs but, at compile time, it is not always clear what strategy and parameters should be chosen. For example, the compiler can insert instructions to give hints to the processor for prefetching data. However, this is only possible for static data structures. When considering more general and dynamically allocated data structures, the compiler does not have all the needed information, some of it being only available at runtime.

Different techniques are used to detect and analyse delinquent loads. Profiling techniques, requiring a first instrumented execution of the program and then a second compilation phase, offer a good solution when the execution of a program does not differ too much from one execution to the other. During this second compilation, the compiler studies the profile of the program in order to select delinquent loads and calculate, if possible, the memory access behavior. For example, the compiler can then insert prefetch instructions to reduce the cache miss latencies.

However, programs that depend on input files or have a random behavior are difficult to profile correctly. In such cases, a dynamic system can be chosen to profile during the execution and adapt to the current behavior. Dynamic optimizers analyze the binary code of the target program during its execution to detect portions of code that could be modified in order to optimize certain aspects of the program.

A dynamic framework can be hardware and/or sofware and is executed in parallel with the target program. The dynamic system generally has a certain control on the target program and can intervene to modify and optimize its execution. The main difficulty in such a system is to compensate the overhead incurred by the system.

Dedicated hardware solutions exist and are generally less costly in terms of overhead but more costly in their development. However, they are generally limited to simple optimization schemes and are not portable across different architectures. Hybrid solutions are a good compromise but are still dependent of the architecture. Finally, certain solutions use external programs that are dependent of the operating system and yield good results.

Markovian Model [3]

At first, we implemented a totally software and dynamic system based on a Markovian study of the memory strides between consecutive accesses. Generally, a Markovian predictor contains a log of prior memory accesses to accurately predict the next stride. It tries to predict, using statistical information on prior memory behavior, and then prefetch the data into the different caches before the processor needs it. If done correctly, a significant speed-up can be achieved.

Our model, called *Esodyp*, does not handle the actual addresses of the memory accesses but the strides between two consecutive accesses. This is different from other approaches and enables us to consider large memory accesses when the general behavior is constant. This means that the model is often relatively small but is still able to predict accurately the accesses. Furthermore, the model considers multiple prior accesses in order to calculate its predictions, compared to other systems that only consider the last access. Our experiences show that, using our system, we obtain speed-ups ranging from 7% to 109% on different programs from known benchmarks.

Optimizing parallel programs [1,7]

In a parallel program, multiple tasks are executed on different machines, also called *nodes*. Certain Distributed Shared Memory systems, such as *Jackal*, allow a transparent parallel programming solution for the programmer. The programmer does not handle directly the different tasks or the different data transfers required.

For example, the data is automatically divided amongst the different nodes. When a node needs to access a certain data, the system checks if the data is available in the local cache. If not, the system then checks the local memory or sends a message to the node that contains that data. It is then necessary to wait for the request message to arrive and the answer message to come back with the data in order to continue the calculations.

A data predictor could send prefetch messages for data in advance, allowing the system to continue the calculation without having to stall. Furthermore, since the time required to fetch the data in DSMs is far greater than the latency for a simple cache-miss in a sequential program, the potential gains are more impressive. Finally, since the potential is greater, the optimizer can use more advanced and costly techniques to calculate the predictions.

In this work, we implemented the Esodyp framework in the Jackal system. We called the new system Esodyp+ and compared it to two other predictors. This comparison shows that Esodyp+ is a relatively lightweight predictor and remains highly accurate. The accuracy rate of Esodyp+ was in average at 80% and achieved speed-ups ranging from to 1% to 18%.

Transparent, automatic and dynamic optimizations [2,5,6]

The major drawback of the Esodyp/Esodyp+ systems is the necessity to insert calls to the model in order to optimize programs. The calls can be directly inserted by a compiler, using first a program profile or can simply be inserted manually. This is often a long and difficult process that is not always possible.

A more interesting solution is to directly modify the program while it is executing to study the actual behavior of each load instruction. This is different than using code interpretation or emulation techniques. Once delinquent loads are found, the system can insert optimizations that can help the program. The framework we implemented is executed at the start of any compiled program and directly modifies the binary to insert instrumentation code. During the study of the program, interesting loads are sent to an optimizer that will insert a prefetching instruction to lower the number of cache misses.

For implementation reasons, it is often difficult to implement complex optimization solutions. Certain solutions require more registers or are difficult to insert into the control flow without taking certain precautions. This is why our system was integrated directly into the control flow using only 4 registers and without any function call. The accelerations of the different benchmark programs using our system ranged from 2% to 143%.

Conclusion and future work

We have implemented several dynamic optimization systems that are able to prefetch data in ways that are difficult or even impossible to do at compile time. Different predictors and prefetchers have been implemented which showed their usefulness. For example, a transparent monitoring system can be attached to a program and can modify its execution without any knowledge of its existence by the base program.

We are currently working on extending the systems to allow a user to modify, study and optimize the code. Compared to different solutions, our system is not trace based but can handle each load of the program independently. This is a major advantage when continuous optimizations and de-optimizations are required. To achieve this, we need to be able to save the context of the program and restore it once the user has finished his instrumentation and decided if an optimization should be tried.

We are also extending our solutions on different architectures to show that the systems are portable even on embedded systems.