MINISTERUL EDUCATIEI SI CERCETARII
Universitatea "POLITEHNICA" Bucuresti
Spl. Independentei, 313, Sector 6, Cod 060042, Bucuresti
Tel: +40 21 402 91 00

Facultatea de Automatica si Calculatoare

ANUNȚ

Va facem cunoscut ca în data de 16.06.2005, ora 10 în Sala ED010 a Facultatii de Automatica si Calculatoare din Bucuresti, va avea loc sustinerea publica a tezei de doctorat intitulata:

Cresterea performantei arhitecturilor de calcul cu paralelism la nivelul instructiunilor prin metode predictive

elaborata de domnul inginer
ADRIAN FLOREA

în vederea obținerii titlului de DOCTOR în domeniul STIINTA CALCULATOARELOR

Comisia de doctorat aprobată prin ordinul Rectorului nr. 65/06.05.2005 are următoarea componență:

PRESEDINTE
Prof.univ.dr.ing. DUMITRU POPESCU

CONDUCATOR STIINTIFIC
Prof.univ.dr.ing. MIRCEA PETRESCU

REFERENTI

1. Prof.univ.dr.ing. ADRIAN PETRESCU
Universitatea "POLITEHNICA" Bucuresti

2. Prof.univ.dr.ing. VLADIMIR CRETU
Universitatea "POLITEHNICA" Timisoara

3. Prof.univ.dr.ing. LUCIAN VINTAN
Universitatea "Lucian Blaga" SIBIU

“POLITEHNICA” UNIVERSITY OF BUCHAREST
AUTOMATION AND CONTROL FACULTY

Ph.D. Student:
Adrian FLOREA, Eng., MSc

INCREASING THE PERFORMANCE OF INSTRUCTION LEVEL PARALELISM PROCESSORS THROUGH PREDICTIVE METHODS

Ph.D. Supervisor:
Prof. Mircea PETRESCU, Ph.D.

Table of Contents

Keywords: Instruction level parallel architectures, speculative execution, dynamic branch and value prediction, register value prediction, execution-driven simulation, SPEC benchmarks, indirect branches / function calls, meta-predictors.

Abstract

The computer architecture domain challenges are mainly conceptual, architectural and finally technological. This Ph.D. thesis structured around eight chapters, presents aspects concerning the theoretical and experimental research (complex architectures design and simulation) realized by the author in instruction level parallel architectures. The main aim of this research is to find and develop new techniques and methods for exploiting and increasing the architecture's ILP degree. In this work is approached a relative new speculative technique – *dynamic value prediction*, focalized on different resources (instructions, data, CPUs registers) and used to exploit the instruction and data redundancy existing in common-use programs. The technique was proposed to overcome the fundamental constraints of ILP processors, the declared goal being *to compress the critical path* from programs. The higher degree of value locality exhibited by hardware resources make the value prediction concept feasible to be implemented in silicon. Another problem solved in this Ph.D. thesis tries to implement modern prediction structures for indirect jumps and calls. Also using some simple testing programs (procedural and object-oriented), I showed that hardware and software "hemispheres" are only apparently disjoint, both of them having a common goal: increasing architecture's ILP degree.

The first chapter presents the motivation of the research carried out in the context of the tackled topic, as well as the evolution and the structure of the Ph.D. thesis.

The second chapter entitled "*Fundamental Constraints of ILP Paradigm. Solutions*" starts by presenting the limits of instruction level parallelism paradigm: the producer limit (fetch bottleneck), the consumer limit (issue bottleneck, data-flow bottleneck), causes and solutions. A solution related to fetch bottleneck insists about Trace Cache concept, based on simultaneous prediction of multiple branches, fill units and selection logic. A great part of this chapter is concentrated around two novel techniques: a speculative one – dynamic value prediction and respectively, a non-speculative one – dynamic instruction reuse. Both of them try to reduce the negative effects caused by true data dependences between instructions (issue bottleneck). This section illustrates practically a *state of the art* related to instruction value predictors. These are analyzed in both a qualitative and a quantitative manner, measuring their performance and their limits.

The third chapter describes the benchmarks (SPEC suites), the simulation methodology and the standard tools set used. I developed few cycle-accurate executions driven simulator derived from the *sim-outorder* simulator in the *SimpleScalar* tool set (a collection of compilers, assemblers, linkers, simulators, debuggers). The baseline superscalar processor supports out-of-order instruction issue and execution. I modified it to incorporate the indirect jumps' predictors in order to measure target locality, and, respectively to predict targets for indirect jumps and calls. In addition, I developed a value predictor simulator dedicated both for different type of instructions and for processors' registers. I also rebuild the GNU GCC compiler to generate object code for *SimpleScalar* architecture (MIPS compatible) in order to establish and study the qualitative relationship between object-oriented programming paradigm and indirect jumps / calls. To perform my evaluation, I collected results from different versions of SPEC benchmarks: integer and floating point SPEC'95 benchmarks, respectively the CINT SPEC2000 set. I simulated some SPEC'95 benchmarks too in order to compare their behavior with that one involved by more recently SPEC2000. In other words, I intend to discover how these different benchmarks influence the indirect jumps predictors' and value predictors' micro-architectural features.

The main aim of chapter four consists in investigating the two hemispheres, hardware and software, just apparently separated, in which the computer science researchers develop their activity in order to better understand indirect branch behavior and the corresponding prediction processes. Thus, I developed some analysis of C versus C++ languages from execution viewpoint on Instruction Level Parallel Architectures. In addition, I extracted some typical corpus of procedural and object-oriented languages that generate, after compilation, indirect jumps and calls.

The history of processors marks out two paradigms for improving the performance based on software respectively on hardware. Despite their common goal – exploiting and increasing the

Instruction Level Parallelism – the research community is split in two "almost separated" entities for accomplishing it. Whereas the computer designers sewer their efforts for exploiting / optimizing the existing processing techniques through laborious simulations on representative benchmarks in machine code format, without taking account of source code semantic, the compiler writers' issue is to optimize the object code. Sustaining the last idea, D. Knuth himself, after analyzing the static and dynamic behavior of a large collection of Fortran programs, concluded that programmers had poor intuition about what parts of their programs were most time-consuming, and that execution profiles would significantly help programmers improve the performance of their programs.

The idea that processor architecture interacts only accidentally with software domain is completely wrong, between hardware and software existing strong interdependences, unexploited appropriately yet. There are at least two reasons, which prove that the processors and the compilers design processes are made in the same time:

- The simulated benchmarks are compiled for some certain architecture (for example, GNUC Compiler from Linux can generate machine code for Intel or SimpleScalar architecture).
- The compiler should generate code for exploiting the architectural characteristics otherwise this code will be inefficient from execution time point of view.

Object-oriented applications are considered now the biggest challenge both for compilers writers' community and for microarchitectures designers. At compile time the code involving calls to library routines, to procedures defined in separately compiled modules, and to dynamically dispatch "virtual functions" in object-oriented languages, cannot be effectively optimized. Machine code usually has much less semantic information than HLL source code, which makes it much more difficult to discover control flow or data flow information. Control flow analysis of executable files can be difficult because determining the extent of jump tables, and hence the possible targets of the code derived from switch/case statements, can be difficult. In this work, I have analyzed SPEC'95 benchmarks (entirely procedural) as well as some own test programs, two of them being object-oriented. Based on simulation results there are detached some conclusions:

- Indirect jumps occur more frequent in object-oriented programs rather than in procedural programs.
- Late (dynamic) binding realized through polymorphism generates indirect function calls in object-oriented applications (C++, Java, Smalltalk). These languages promote a polymorphic programming style in which late binding of subroutine invocations is the main instrument for modular code design. Virtual function tables, the implementation chosen for most C++ and Java compilers, execute an indirect branch for every polymorphic call. More than that, in Java, instance methods are virtually declared by default. If they are not explicitly declared *final*, they can be overridden in subclasses.
- The presence of indirect jumps in procedural (C) programs is mainly due to the following two aspects:
  - Indirect calls through function pointers (address).
  - The possible targets of the code derived from some special switch/case statements.
  - One of the reasons that favor the indirect jumps existing in programs is the presence of statically or dynamically library function calls (e.g. the qsort function from BorlandC Help and DLLs usage from desktop applications).

In chapter five are illustrated the own contributions to indirect jumps / calls predictions. Starting from the necessity of implementing new performing indirect branch prediction schemes, but taking into account the hardware feasibility desiderate of them, I showed that a modified Target Cache structure, based on confidence mechanism and indexed with extended global correlation information, represents a more simpler and feasible solution that could replace the more complex PPM (prediction by partial matching) predictor. I also determined based on laborious simulations what is the optimum search pattern when different contexts are used. Using profile information, I developed a hybrid predictor with arity-based selection that improves indirect branch prediction

accuracy reaching in average to 93.77% comparable with a more complex multi-stage cascaded predictor.

Deep pipelines and fast clock rates are necessitating the development of high accuracy branch predictors. The most research in branch prediction is based on two closely related correlation mechanisms (local and global). The global method exploits correlation between the outcome of a branch and the outcome of neighboring branches that are executed immediately prior to the branch. In contrast, the local method depends on the observation that the outcome of a specific instance of a branch is determined not simply by the history of the branch, but also by the previous outcomes of the branch when a particular branch history was observed. Yet branch prediction is a specific example of a far more general time-series prediction problem that occurs in many diverse fields of science. It is therefore surprising that there has not been more cross-fertilization of ideas between different application areas. A notable exception is a paper by Mudge that demonstrates that all two-level adaptive predictors implement special cases of the PPM algorithm that is widely used in data compression. Mudge uses the PPM algorithm to compute a theoretical upper bound on the accuracy of branch prediction, while Steven demonstrates how a two-level predictor can be extended to implement the PPM algorithm with a resultant reduction in the misprediction rate. Other researchers developed some more sophisticated predictors based on neural networks algorithms.

A particularly difficult challenge consists in target prediction for indirect jumps and calls. Because the target of an indirect jump (call) can change with every dynamic instance of that jump, predicting the target of such an instruction is very difficult. From microarchitectural point of view object-oriented programming techniques exercise different aspects of computer architecture to support the object-oriented programming style. In the last decade the importance of indirect branch prediction increased even though, in the computing programs the indirect jumps and calls remain less frequent than the more predictable conditional branches. One of the reasons refers to predicative execution that implies decreasing of conditional branches number. The dimension took by the desktop, visual or object-oriented applications development (C++, Java – characterized by a large amount of indirect branches comparative to procedural programs), and respectively, the portability trend of many of them, as well as the usage of Dynamically-Linked Libraries, represent other reasons which illustrate that indirect branch prediction misses start to dominate the overall missprediction cost. Knowing that the Pentium4 equivalent processor performance degrades by 0.45% per additional branch missprediction cycle and, additional, a very small number of static indirect branches is responsible for more than 90% of dynamic indirect jumps, results that the overall performance of architectures are very sensitive to indirect branch prediction. Since the prediction accuracy generated by classical schemes Branch Target Buffer or Last Value Predictor is less than 75% and the maximum value obtained by a feasible PPM predictor, reported in literature is around 90%, implies the necessity of implementing new performing indirect branch prediction schemes (hybrid or cascaded predictors, or even adapted from value prediction – contextual, PPM predictors).

Next, in a systematic manner, I measured target localities associated with these indirect jumps, in order to estimate how predictable they are. I tried to determine an ultimate context predictability metric of indirect branches. As expected, the conclusion was very optimistic. The value locality concept, first introduced by Lipasti, represents the likelihood of the recurrence of a previously seen value within a storage location. Accordingly, in this case, it will say that an indirect jump (call) target value is local if it belongs to the previous *K* dynamic target instances of that certain jump (call). Obviously, a great target locality degree involves great prediction accuracy, too. In other words, the value locality degree obtained for *K* dynamic target instances represents the maximum achievable prediction accuracy using a context predictor of order *K*. Therefore, my approach establishes an analogy between value prediction and, respectively, indirect jumps target prediction. Statistical results based on simulations have proved that indirect jumps targets are characterized by higher degree of value locality (over 90% for K?4).

Due to higher degree of target localities associated with some indirect jumps, I predicted these indirect jumps and calls using some contextual value predictors, derived from the complete PPM

predictor respectively the Target Cache predictor. The obtained results were better than that reported by other researchers that used more simplified context predictors. PPM predictor seems to be an almost ultimate limit of context target prediction, and, thus, a good frame for further deriving new practical prediction schemes. In my research I find that a complete PPM predictor having an associative indirect jump value prediction table with 256 entries generates average prediction accuracy between 89.33% and 91.58% depending on the context used in simulation: *short history* (the last 32 targets) and a *rich history* (the last 256 targets), the search pattern varying descending from of 4 to 1.

Another goal of my research is to show that a modified Target Cache structure based on confidence mechanism and indexed with extended global correlation information represents a more simple and feasible solution that could replace the more complex PPM predictor. The Target Cache (TC) improves the prediction accuracy for indirect branches by choosing its prediction from the last *N* targets of the indirect branch that have already been encountered. When an indirect jump is fetched, both the PC and the *globalHR* (a history register that retains the behavior of last HRgLength conditional branches) are used to access the TC for predicting the target address. As the program executes, the TC records the target for each indirect jump target encountered. The proposed scheme, for set selection, uses the least significant bits of the word obtained by hashing (XOR) the indirect jump's address (PC) and the *globalHR*. The most significant bits of the obtained output form the *Tag*. In the case of a hit in TC the predicted target consists in the corresponding address belonging to that TC set. In case of a misspredictionafter the indirect branch is resolved, the Target Cache entry is updated with its real target address. It is implemented a LRU replacement algorithm. I have implemented and simulated a *P*-way set associative TC, where *P*=1, 2, 4, 8. In the case of a miss in TC the prediction is considered wrong, it doesn't propose any value and it is added a new entry in the respective set updating with the proper *tag* and the proper *target*, accordingly with the specified replacement algorithm.

A common criticism for all the present two-level adaptive branch prediction schemes (applied to either conditional branches or indirect jumps) consists in the fact that they used insufficient global correlation information. There are situations when for the same static indirect branch and in the same *globalHR* context pattern it is possible to find different targets. If each bit belonging to *globalHR* will be associated during the prediction process with its corresponding PC the current indirect branch's context becomes more precisely and therefore its prediction accuracy could be better. Next, I developed a *path-based predictor*, through extending the correlation information according to the above idea. Thus, the first level of history of Target Cache predictor records the path (the conditional branches' addresses) leading to the current indirect jump. Extending the correlation information in this way suggests that at different occurrence contexts of a certain indirect jump it will access different sets from TC structure, reducing a significant amount of interferences and increasing the prediction accuracy. Compression of this complex information is possible and even necessary, taking into account the request of reasonable costs for these schemes.

The next contribution follows up to improve indirect branch prediction accuracy by selectively ignoring some predictions. Therefore, I attached a confidence counter (degree of reliability) at every Target Cache location, together with a replacement mechanism based on *LRU*, *confidence* and, respectively, both of them. A confidence mechanism performs speculation control by limiting the predictions to those that are likely to be correct. A high confidence degree represents *the continuous correct predictability in a given history* of an indirect jump while the LRU field means its *activity degree* (how many times this branch was accessed). Both *Confidence* field and *LRU* field were implemented as saturating counter being represented on different number of bits. For exploiting this subsection results I used some new metrics, previously proposed in the literature but also I still introduced other. In this sense I use the ***prediction accuracy* (*Ap*)** metric as the probability that prediction generated by a high confident state to be correct. ***Usage*** represents the prediction performed degree, practically the number of cases when the automaton was in a high confident state, reported to the total number of indirect jumps. I introduced the ***predictors' overall performance* (*P*)** metric as the product from **Ap** and **Usage**. I determined how are influenced this

metrics by a parameterized threshold. The predictors' overall performance generated by the new scheme proposed is only with 0.4% under the accuracy provided by a PPM predictor, but the scheme proposed is much simpler and more feasible to be implemented in hardware.

Taking into account PPM's complexity, I tried to implement a simplified PPM or a confidence-base hybrid predictor having as components two contextual predictor of different order. Knowing that only a small amount of the path information leading up to a branch is needed for prediction I tried to find what the optimum search pattern is when different contexts are used. Thus, I obtained that the markovian predictor has the best behavior in two different contexts: using a *short history* (the last 32 targets and a search pattern of 3) and a *rich history* (last 256 targets and a search pattern of 6). The values obtained (orders of Markov predictors) I used in developing a hybrid predictor with arity-based selection. I classified branches according to a dynamic measure, the number of different targets encountered in a running program, or branch's *arity*. This is determined in a profiling run. The profile information table, completely associative retains the arity of every indirect jump and helps to select which component predictor will predict at every moment. The component predictors are a *LastValue* predictor for monomorphic (1 target) or duomorphic (2 targets) indirect jumps, and respectively, the best contextual predictor, previously determined, for polymorphic branches (more than 2 targets). Both LastValue and contextual predictor are completely associative and indexed in the instruction fetch stage with indirect branch address (PC). The hybrid predictor with arity-based selection generated the best result. The new scheme improves indirect branch prediction accuracy reaching in average to 93.77%, comparable with a more complex multi-stage cascaded predictor.

The excellent results obtained impose introducing and exploiting the hybrid and cascaded predictor in other computer architecture issues to increase the instruction and thread level parallelism: conditional branch prediction, instruction value prediction. As a further work, I will try to replace the arity-based selection hybrid predictor with a simple neural network that will select dynamically between ordinary component predictors (Last Value, Target Cache, contextual, hybrid). In addition, it could be studied the feasibility of an indirect branch predictor correlated and decision trees based. Another solution could appear from development of some "*semantic predictors*", based on High Level Language applications' information that I prove being important related to indirect jumps generation (polymorphism, indirect function calls, etc.). This might be a completely new approach in branch prediction domain, where HLL semantics are often hidden. As far as the architecture designers are concerned, their proposed schemes could be more efficient if not only the object code from benchmarks ("*wear off by any semantic information*") is analyzed but they will also look "*higher*" towards high-level sources of simulated programs.

In chapter six, I made some research and bring contributions in dynamic instructions value prediction domain. I implemented the value predictors described in chapter two for measuring the value locality degree and prediction accuracy exhibit by SPEC benchmarks. I also proposed an algorithm to quantify the speed-up obtained by a speculative microarchitecture that implements value prediction technique. As an original contribution, I extend the dynamic value prediction by introducing the concept of register centric prediction instead of instruction centric prediction.

Value Prediction (VP) is a relatively new technique that increases performance by eliminating true data dependencies constraints. Value prediction architectures allow data dependent instructions to issue and execute speculatively using the predicted values. This technique is built on the concept of value locality, which describes the likelihood of a previously seen value's recurrence within a storage location. Obviously, the utility of value prediction techniques is emphasized only in the case of a correct prediction otherwise, it determines structural hazards and a higher instruction's execution latency. In this work, I applied dynamic value prediction to different types of instructions (arithmetical, logical, load and indirect jumps). The value localities obtained on some registers of MIPS architecture were quite remarkable leading to conclusion that value prediction might be successfully applied, at least on these favorable registers. The idea of attaching a value predictor for the processor's favorable registers is original and might involve new architectural techniques for improving performance and reducing the hardware cost of speculative microarchitectures. The

register value prediction technique consists in predicting registers' next values based on the previously seen values. It executes the subsequent data dependent instructions using the predicted values and the speculative execution will be validated when the correct values are known. If the value was correctly predicted the critical path is reduced, otherwise the instructions executed with wrong entries must be executed again.

I examine different favorable register selections and different basic value predictors in order to capture certain type of value predictabilities from SPEC benchmarks ('95 and 2000) to obtain higher prediction accuracies. The "*last value*" predictors predict the new value as the same with the last value stored in the corresponding register. Exploiting the correlation between registers names and the values stored within will decrease the instructions' latencies. The stride predictors make a prediction by computing some function (algorithms) of previous values. The context-based predictors predict the value that will be stored in a register based on the last values stored in that register. A context is a finite sequence of values with repeated apparition like in a Markov chain. The predictors that implement the PPM algorithm represent an important class of context-based predictors. The predicted value is the value that followed the context with the highest frequency. A longer context frequently drives to higher prediction accuracy but sometimes it can behave as noise. A complete PPM predictor contains N simple *Markov* predictors, from 0^{th} order to $(N-1)^{th}$ order. If the $(N-1)^{th}$ *Markov* predictor produces a prediction (the context is matched in the sequence) the process is finished, otherwise the $(N-2)^{th}$ order *Markov* predictor will be activated, and so on. A context-based predictor and respectively a stride predictor, working together, compose the hybrid predictor used. The context-based predictor had always priority; in this way, the value generated by the stride predictor was used only if the context-based predictor cannot generate a prediction.

The simulations results show that there is a time-correlation between the names of the destination registers and the values stored in these registers. The simulations exhibit that the hybrid predictor optimally exploits this correlation with an average prediction accuracy of 85.44%, quite remarkable (on some benchmarks with values over 96%). Considering an 8-issue out-of-order superscalar processor, I showed that register centric value prediction produce average speedups of 17.30% for the SPECint95 benchmarks, respectively of 13.58% for the SPECint2000 benchmarks.

Obviously, this fixed prioritization is not optimal. Next, in order to increase the prediction accuracy is realized a dynamic prioritization based on some confidences. I introduced several different metaprediction structures, in order to properly select the current best predictor: two non-adaptive metapredictors and an adaptive one, represented by a neural network. The experimental results obtained using metaprediction applied only to the best four favorable registers (having high value locality degrees) show an average prediction accuracy of 91.40%, measured on SPEC benchmarks. The accuracy gain obtained on these registers versus the old hybrid predictor is 2.27%.

Chapter seven presents the most eloquent quantitative results and remarks from qualitative point of view. The results are structured on three levels: the first of them exhibits the indirect jump prediction depending on the predictions structures used (Target Cache, PPM, hybrid). The second class of results analyzes problems related to value locality and value predictions on different resources (instructions, data). The last class of results tries to prove the registers prediction concept feasibility. Simulations were realized partially under Windows operating systems and partially under Linux RedHat, using powerful execution driven simulators (developed by the author), very flexible, portable and extensible.

The eighth chapter illustrates the scientific contributions of this work and the quantitative gain of every novel proposed techniques. Also, there are indicated some directions for future research. The problems approached in this Ph.D. thesis should be continue in order to solve other ILP problems, remained open, very interesting and connected with that presented here.

CURRICULUM VITAE

First Name:	**ADRIAN**
Last Name:	**FLOREA**
Date and place of birth:	June 26th 1974, Sibiu, ROMANIA
Home Address:	No. 25, Ecaterina Varga Street, Sibiu, 550225, ROMANIA
Telephone:	++40-269-23.77.00
Fax:	++40-269-21.27.16
E-mail / web_address:	adrian.florea@ulbsibiu.ro, http://webspace.ulbsibiu.ro/adrian.florea
Teaching position:	Lecturer
Member of scientific associations:	Romanian Society of Control Engineering and Applied Informatics (SRAIT)
Education:	1992 – Graduate of the "Gheorghe Lazar" High school from Sibiu, filled of study: mathematics – physics (Graduate mark –9.32). 1997 – Graduate of the Faculty of Engineering, Computer Science Department, "Lucian Blaga" University of Sibiu (Graduate mark: 9.94) 1998 – Master license (MSc) in specialty "Parallel and Distributed Processing Systems", Faculty of Engineering, Computer Science Department, University "Lucian Blaga" Sibiu, (Dissertation Title: "*Optimization of Instruction Processing in a Superscalar Harvard RISC Architecture*" – Graduate mark: 10.00)
Training and qualifications:	31.05.2000 – 15.07.2000 – I beneficiate of one research scholarship for preparing the Ph.D. in Computer Architecture domain at Hertfordshire University, Computer Science Department, Hatfield Hertfordshire, United Kingdom. The International TEMPUS JEP – AC 13559 Program financed the scholarship.

Teaching activities and professional experience:

- **1997-1999:** System Testing, Diagnosing and Software Engineer, P.I.M. Sibiu, Romania
- **1999-2001:** Local secretary of Joint Tempus Program 13559-98 (Retraining Support for Small and Medium Enterprises), having academicals partners in England, Ireland, Germany and Spain, with a budget about 200,000 Euro (about 70,000 Euro at Sibiu).
- **1999-2005:** full-time employee at "Lucian Blaga" University of Sibiu, Computer Science Department (Topics: *Microarchitectures Organization and Design*, *Simulation and Optimization of Advanced Computer Architectures, Prediction and Speculation in Advanced Architecture*, *Introduction in Computer Science*).
  - 1999-2002, Teaching tutor;
  - 2002-2005, Teaching assistant;
  - 2005 to date, Lecturer.

Scientific Activity:

- I published as a co-author 3 books in Computer Architecture, one in "Lucian Blaga" Publishing House, Sibiu (1999), one in The Technical Publishing House, Bucharest (2000) and another one in "MatrixROM" Publishing House Bucharest (2003).
- In addition, I published over 16 scientifically papers in Computer Architecture in some prestigious journals (ISI-INSPEC) and international conferences (see above my web page address).
- I got 6 National Research Grants as a Project member (ANSTI, CNCSI).

Fields of Professional Interest:

Advanced Computer Architecture, Instruction and Thread Level Parallel Architectures, Microprocessors and Microsystems, Microcontrollers and Applications, Software Scheduling, Operating Systems, Simulators, New Computing Paradigms, History of Computers.