Execution speed up using speculation techniques in computer clusters

Authors: Joan Puiggali, Teo Jové, Salvador Salanova, Josep Lluís Marzo
Institute of Informatics and Applications (IIiA), University of Girona
Campus Montilivi, 17071 Girona, Spain
Phone: +34 972 418889
Fax: +34 972 418098
{joan.puiggali, teodor.jove, salvador.salanova, joseluis.marzo}@udg.es

1. Introduction

Speculation techniques in the design of superscalar processors and multiprocessors [9] [17] [23] [33] [31] [5] [30] [29] [19] have evolved enough to be considered as mature technologies. These techniques allow processors to divide automatically and dynamically the execution of a program into several thread executions, and permit to push further the degree of parallelism of a program thanks to speculation on data and control dependences. With this speculation, an increasing number of functional units can be simultaneously used, extracting better performance from the processor than the performance one could extract using only the program's parallelism.

Cluster based systems like Mosix [20], OpenSSI [22], Kerrighed [12], amongst others, are systems oriented to distributed execution. These systems allow the balancing of the load of the node of the system through migration and distribution of the processes across the system. These systems are not explicitly oriented to the execution of parallel applications, although they can be executed. Cluster based systems can be low cost tools that allow to increase the processing power because they scale easily. These systems also make possible the reuse of sets of networked workstations, and use them in otherwise idle periods.

Packages that allow the execution of parallel programs also exist, such as PVM [25] o MPI [21]. These packages need explicit parallel programming, and have provision for neither automatic parallelization nor for speculation.

Moore's law (processing power doubles each 18 months) and Gilder's law (bandwidth triples each 12 months) show that the costs of information transmission and synchronization between workstations decreases with time lower when compared to processing speed. For example in a period of 15 years, ethernet technology has gone from 10 Mbps to 10 Gbps today, an increment factor of 1000. During this period, clock frequencies have stepped from 25 Mhz to above 2.5 Ghz, an increment factor of 100. These tendencies seem that will continue in the future, so the gap between network speeds and processor speeds will continue to reduce. These premises make possible the idea of transporting speculation techniques to a distributed environment of low cost workstations.

This new idea of extraction of parallelism through the use of speculation in distributed environments has to take as input a non parallel program and execute it in a cluster of workstations. To do this we should be able to divide the programs into blocks and build a graph of these instruction blocks, where for each block its input and output data is known. The parallel execution should provide better performance than the performance provided by the execution on a single workstation, because of the existing parallelism, or because speculation techniques have been applied.

This proposal does not intend to compete with multiprocessors or superscalar processors; its objective is to describe a methodology for extracting and exploiting parallelism in clusters. We want to minimize the programs return time without concerning the global throughput of the distributed system.

Graph algorithms, like Dijkstra’s algorithm [36] or “travelling salesman problem” [37], are examples of algorithms with a high computation cost due their complexity. These algorithms are wide used in networking management tasks like routing tasks (e.g. MIRA algorithm [38]). Our proposal could reduce their execution time. Also this speed up allows to considerer to use algorithms that provide optimal solutions instead of suboptimal or heuristic algorithms.

This paper analyses the viability of the described proposal. In order to do so, we want to exploit the advantages of aggressive speculation.. In section 2, related techniques and the technological evolution that allow this proposal to be possible are analysed. Section 3 describes the proposed architecture and the associated simulator. In section 4, an example and its simulation results are analysed. Finally some conclusions are presented.

2. Related work

a) Technological evolution

So far, technological evolution follows Moore's law and Gilder's law. Expectations are that these laws are going to hold true for the next years. Following these laws, the gap between processor speed and network speed is going to reduce. Today we can find processors that are able to deliver 4 Gips and networks that can transmit 10 Gbps. In five years time, these speeds will 64 Gips and 7290 Gbps respectively. So, instead of executing 8 instructions per transmitted byte of today, 0.07 instructions are going to be executed for each transmitted byte.

b) Instruction Level Parallelism (ILP), Thread Level Parallelism (TLP) and Speculation

The increase in the integration scale of the processors has allowed the increment in the number of functional units. The increase of performance in today's architectures is a result of trying to use simultaneously the maximum possible of functional units. To achieve this, techniques that exploit the parallelism of the applications have been developed. The exploitation of instruction level parallelism (ILP) has been the first step in this direction [11]. The instruction window and control and data dependences limit the maximum number of instructions that can be simultaneously executed [9]. In order to increase the number of independent instructions that can be issued in a given moment, architectures have exploited the simultaneous execution of different threads from the same process or from different processes (TLP) [17] [5] [31]. Beside these techniques, speculation techniques that use the prediction of future values are also used in order to break control and data dependences. In this way one can increase the inherent parallelism of an algorithm [3].

From these techniques, we want to remark those based on the help of the compiler. These techniques modify the programs in such a way that processors can extract the maximum performance from the parallel and speculative execution of the threads. We can divide these techniques into 2 groups: The first group [2] [3] [26] [34] uses support threads that try to predict values or jump decisions. The second group [7] [1] [15] divide the program into pieces that can be executed in parallel. In [16] [24] the different partitions and their effects in the final performance are analysed.

c) Cluster and Grid computing

A cluster [20] [22] [12] is formed by a group of more or less coupled computers that work for a common goal. We can find clusters oriented to increase accessibility, or balance the load, or increase performance. In this last case, they are specialised in the execution of parallel programs. Packages like PVM [25], MPI [21], JAVA-RMI [10], or CORBA [4] allow the development of distributed applications under the distributed memory model. But many applications have been developed under the sequential programming paradigm. In this case manual parallelisation consumes a lot of effort. In the next subsection, mechanisms for automatic parallelisation of applications are analysed.

d) Automatic parallelisation

Tools for automatic parallelisation of sequential programs generate code suitable for multiprocessors or clusters. These tools analyse the program and generate code that is able to exploit the inherent parallelism of that program. The majority of these tools exploit a coarse-grain parallelism and leave to processors the fine-grain parallelism. One example of such tools is the SUIF compiler [13], which is able to parallelise a sequential program with minimal programmer intervention. Another way is the approach taken by works like [6] where the parallelisation of JAVA code is done based on the distribution of the Java Virtual Machine. Works like [8] [35] can be also classified here, because all of then are tools that generate code for parallel execution from a sequential machine code.

3. System architecture

As said before, this article analyses the viability of using clusters to run sequential programs, programs that have been automatically parallelised with the goal of minimizing their return time. To achieve this we propose to exploit the advantages of an aggressive speculation, which is neither limited by the number of functional units nor by their kind.

In order to achieve this proposal we can see a computer clusters as a superscalar computer. Each node can be considered as one functional unit. Some of them execute control tasks and others execute threads, which play the roll of the superscalar instructions. The advantages of a system like this are the big number of functional units that it can have and their versatility. On the contrary, some of the disadvantages in front of superscalar processors or a multiprocessors are: a) a reduced bandwidth between “functional units”; b) an internal parallelism of each “functional unit”, which is minor than the one that we can find in the functional units of a processor; c) and last, the lack on a shared memory mechanism, or a distributed virtual memory mechanism.

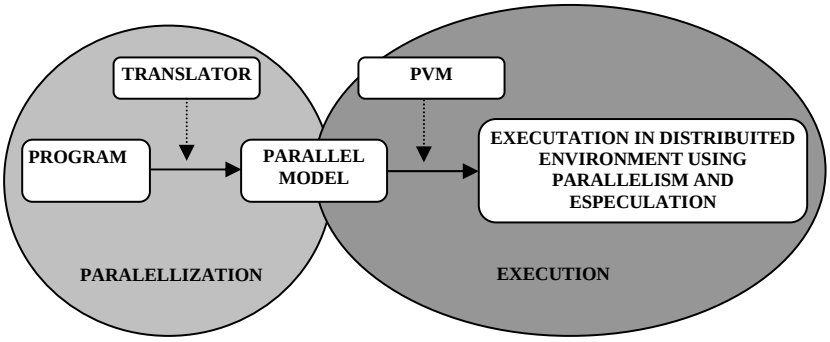


Figure 1: System architecture

The complete design of our system (*Figure 1*) is build from 2 elements:

- A parallelizing subsystem in charge of transforming the original program into the format of the execution environment.
- An execution subsystem on a cluster that allows to execute the parallelized applications applying speculation.

Also a simulator subsystem is also developed in order to evaluate the technological evolution impact, or bigger computer clusters.

a) Parallelising subsystem

The parallelising subsystem transforms the sequential program, which has been written in C, into the format that understands the execution environment. This format divides the program into blocks that can be executed in parallel. For each block, the set of input variables and the set of output variables is identified. In this experimental prototype, we have chosen to divide into blocks the basic structures, like loops or conditions, also functions and procedures [24]. The most important criterion that we have used for choosing these blocks has been the number of instructions that will be executed. Each block can end with a branch instruction that indicates the next block to be executed and that define the dependence graph.

As a result of the translation process, 2 programs are generated: a master and a slave. The master manages the parallelism and the speculation of the system. The slave will execute in each of the nodes, and runs the code of the blocks into which the sequential program has been divided.

Today this subsystem is still in development. The parallelisation has been handmade on synthetic simple programs written in C.

b) Execution subsystem

As we said before, the execution environment is structured in 2 elements: the master program and the slave program:

The master program has a fixed part which is always the same for any program, which manages the execution: identification of blocks ready to be executed (input variables ready or speculated), node assignation, data transmission, result collection, verification of speculated values, and finally, if necessary, deletion of wrong executions.

And a second part, which is program dependent. This last part is related to the definition of blocks into which the program has been divided by the translator, and its dependence graph.

The execution environment behaves like a superscalar processor. In this case the instructions of the superscalar processor are the blocks into which the sequential program has been divided. The nodes that execute the slave program are like the functional units that execute instructions (blocks). The control unit is the node that executes the master program. The master program allows the out of order execution (start and finish). To allow this, WAR and WAW dependences are broken in the same manner that a superscalar architecture would do. True dependences RAW are the only ones that have to be preserved, but they can be relaxed with data value speculation techniques. We have implemented the following speculation mechanisms: Data Value Speculation [14]; Last Value Predictor [14]; Stride Predictor [28] and Context-based Value Predictor [27]. Control dependencies are managed with branch prediction techniques based on a BTB with 2 bit history [32].

In case of data or control misspredictión only blocks that have been executed with wrong speculated values or a branch misspredictión are discarded and its execution is restarted from the last stable point.

In order to reduce the master bottleneck, the master can distributed tasks to others nodes that work as a new master, that we call master-slave nodes. This design can improve the return time of some structures like nested loops.

Today's prototype can be executed in a Linux cluster with PVM; it can also be executed in a single workstation that acts as master and slaves at the same time. The implementation is designed to allow the simultaneous execution of more than one branch of the dependence graph, the simultaneous execution of a single block with different input values.

c) Simulation tool

As we have said before, we have a simulator that is able to run in a single workstation or in a cluster of PCs. The analysis of the proposal, with more nodes or different technological restrictions (present or future) has taken us to develop a simulator which is event based and trace driven.

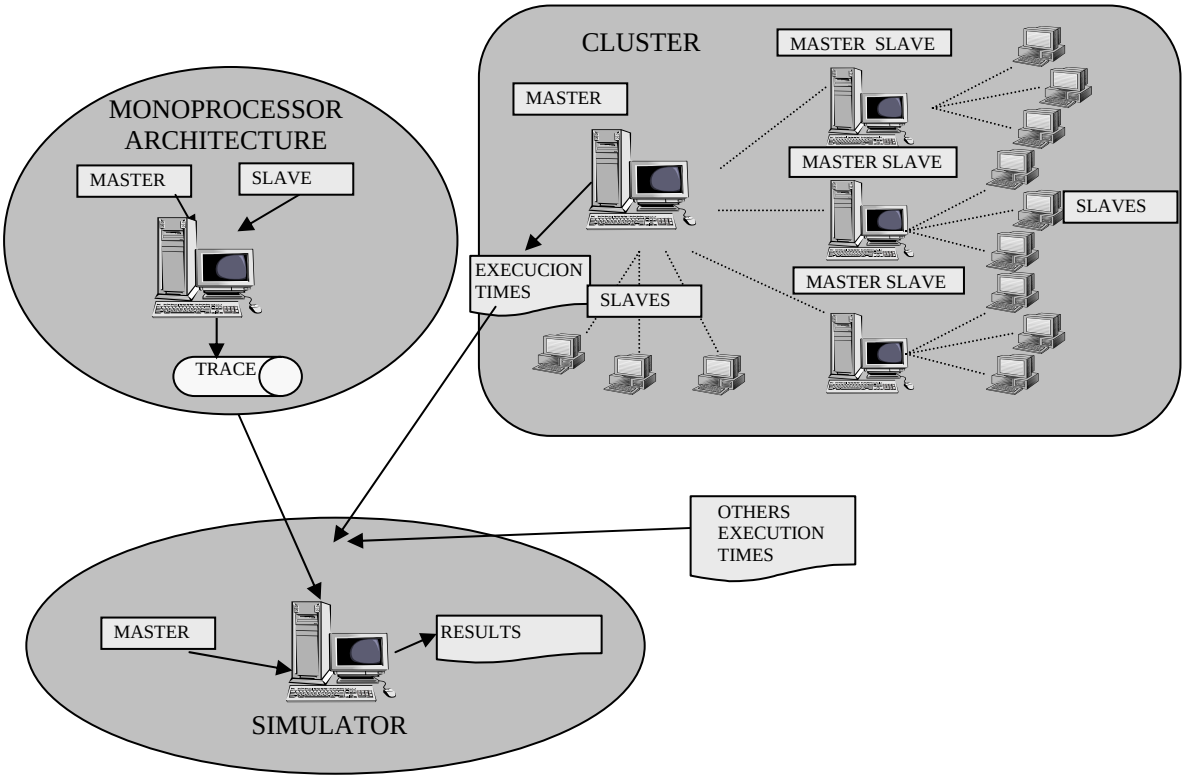


Figure 2: Execution environment.

For the design of the simulator (figure 2) we have divided the master's code in order to determine the block execution order, the possible speculations and the detection and correction of wrong speculations. The execution cost of the different blocks and the cost of the master's different tasks are determined by the input values of the simulator. These values are obtained whether from real executions in the cluster or from work suppositions for situations different than our current cluster. The trace of the program is extracted through the execution of the sequential program in a non parallel environment. The block execution is simulated analyzing the traces and their running times.

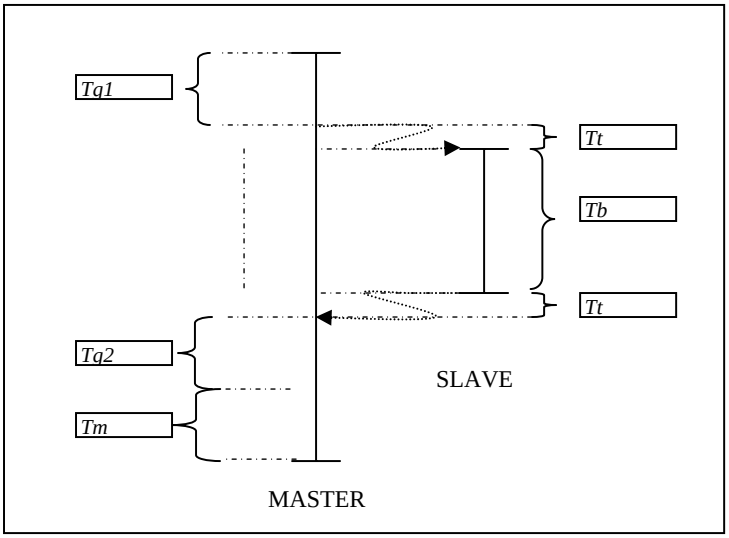


Figure3: Running times for master and slaves

Considered times are (Figure 3)

- Initialization time (*Ti*): is the time needed to initialize all processes, start up of the slaves and initialization of data structures.
- Start up time (*Tg1*): this is the time needed to check if a process can start and to obtain their input values, speculating them if necessary.
- Message transmission time (*Tt*): this is the time for sending and receiving a message between 2 processes.
- Data update time (*Tg2*): average time that master process needs to read the slave's messages and stores the resulting values, plus the time that is needed to sort data writes and to speculation errors. Tg1 + Tg2 build Tg, which has been described in 3.
- Process deletion time (*Tm*): time which is needed the data from the processes whose speculation is wrong.

4. An example and results

As an example of algorithm running in our system we use an implementation of the well knows "Travelling Salesman Problem" [37]. It founds the shortest hamiltonian circuit in a graph. The problem is NP-hard.

There are several methods based on suboptimal or heuristic algorithms [39] [40] [41]. Figure 4 shows the algorithm that we use. This approach is prepared to be executed in our system in parallel with or without speculation.

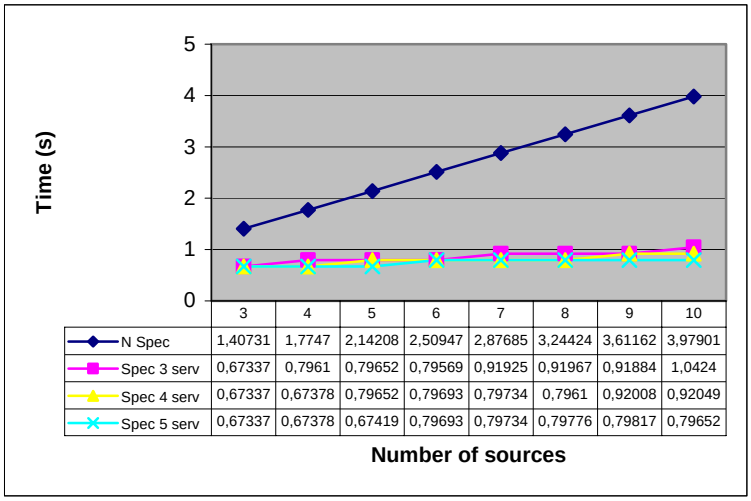
```
For N start nodes
        Create a first route call actual_route
        Repeat
                route_a = Swap some neighbour nodes (actual_route)
                route_b = Swap some neighbour nodes (actual_route)
                route_c = Swap some neighbour nodes (actual_route)
                actual_route = Select the best route (actual_route, route_a, route_b, route_c)
        Until the actual_route is not improved
Select the best N routes
```

Figure4: Travelling Salesman Problem algorithm

It works as follows: At the first it selects randomly the start node of the road. With this start node, a first road is believed adding nodes, one to one, in such way that the distance is minimized with the nodes that are already part of the road. With this first road it tries to improve the obtained solution swapping neighbour nodes. This operation is repeated while an enhancement of the road is obtained. If more nodes are used as a first city of the first route in several executions, best results are produced by the algorithm.

In a parallel execution without speculation, only the code which obtains the three routes: route_a, route_b and route_c, can be run in parallel. Nevertheless, by means of the speculation techniques, all the iterations of all loops could be run simultaneously.

The Figure 5 shows the return time obtained for different number of start cities (from 3 to 10) and for different execution methods: parallel execution without speculation and parallel execution with speculation. In this latter method we use different number of "master-slave" nodes, from 3 to 5. The number of slaves nodes has been considered unlimited. As it can be observed, the speculative execution method is able to reduce the execution time drastically. The data and control dependences limit the maximum parallelism that programs can obtain. In this example the speculation is able to predict the values of the induction variables easily and therefore to increase significantly the parallelism degree of the program. On the other hand the impact of the usage of different master-slave does not offer, in this case, a significant enhancement of the return time.



	3	4	5	6	7	8	9	10
N Spec	1,40731	1,7747	2,14208	2,50947	2,87685	3,24424	3,61162	3,97901
Spec 3 serv	0,67337	0,7961	0,79652	0,79569	0,91925	0,91967	0,91884	1,0424
Spec 4 serv	0,67337	0,67378	0,79652	0,79693	0,79734	0,7961	0,92008	0,92049
Spec 5 serv	0,67337	0,67378	0,67419	0,79693	0,79734	0,79776	0,79817	0,79652

Figure5: Execution times for non speculative and speculative techniques with 3, 4 or 5 master-slave nodes and infinite slaves.

5. Conclusions

To sum up in this paper, we have analysed the viability to use clusters for parallel and speculative execution of processes. We have developed a first prototype able to be executed on Linux/PVM. Also a simulator has been developed to simulate any number of nodes and different technological situations. Finally, a simple example has been developed.

As a result we can see that the speculation techniques can improve meaningfully the return time of some algorithms by means of increasing the parallelism degree of programs.

References

[1] Akkary H. and Driscoll M.A., "A Dynamic Multithreading Processor", in Proc. of the 31st Int. Symp. on Microarchitecture, 1998

[2] Chapel R.S., Stark J., Kim S.P., Reinhanrdt S.K. and Patt Y.N., "Simultaneous Subordinate Microthreading (SSMT)", in Procs. of the 26th Int. Symp. on Computer Architecture, pp. 186-195, 1999

[3] Collins J.D., Wang H., Tullsen D.M., Hughes C., Lee Y-F., Lavery D. and Shen J.P., "Speculative Precomputation: Long Range Prefetching of Delinquent Loads", in Proc. of the 28th Int. Symp. on Computer Architecture, 2001

[4] CORBA, http://www.corba.org

[5] Diekendorff K., ¨Compaq Chooses SMT for Alpha¨, Microprocessor Report, December, 1999

[6] Fang W., Wang C.-L., and Lau F. C. "On the Design of Global Object Space for Efficient Multi-threading Java Computing on Clusters". Parallel Computing, 29:1563–1587, November 2003.

[7] Franklin M. and Sohi G.S., "The Expandable Split Window Paradigm for Exploiting Fine Grain Parallelism", in Proc. of the 19th Int. Symp. on Computer Architecture, 1992

[8] Garcia Quiñones C., Madriles C., Sanchez J., Marcuello P., Gonzalez A. and Tullsen D. M., "Mitosis Compiler: An Infrastructure for Speculative Threading Based on Pre-Computation Slices", in Proc. of the 2005 ACM SIGPLAN conference on Programming language design and implementation, Pages: 269 – 279, 2005

[9] González J., González A., "Limits of Instruction Level Parallelism with Data Speculation", Proc. of 3rd. Int. Meeting on Vector and Parallel Processing, Porto (Portugal), pp. 585-598, June 21-23, 1998

[10] JAVA-RMI, http://java.sun.com/products/jdk/rmi/
[11] Jouppi N.P., Wall D.W., “Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines”, Proc. of ACM Conf. on Architectureal Support for Programming Languages and Operating Systems, 1989.
[12] Kerrighed, http://www.kerrighed.org
[13] Liao S., Diwan A., Bosch R.P., Ghuloum A., Lam M.S., “SUIF Explorer: An Interactive and Procedural Parallelizer”, 7th SIGPLAN Symp. On Principles & practice of Parallel Programming, 1999.
[14] Lipasti M. H., Wilkerson C. B., and Shen J. P., "Value Locality and Data Speculation," in Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 138--147, October 1996.
[15] Marcuello P, and González A., “Clustered Speculative Multithreaded Processors”, in Proc. of the 13th Int. Conf. on Supercomputing, pp. 365-372, 1999
[16] Marcuello P. and González A., “Thread-Spawning Schemes for Speculative Multithreaded Architectures”, in Proc. of the 8th Int. Symp, on High Performance Computer Architectures, 2002
[17] Marcuello P., González A. and Tubella J., "Speculative Multithreaded Processors", 12th Int. Conf. on Supercomputing, Melbourne, Australia, July 1998, pages 77-84
[18] Marcuello P., Tubella J. and González A., “Value Prediction for Speculative Multithreaded Architectures”, in Proc. of the 32nd. Int. Conf,. on Microarchitecture, pp. 203-236., 1999
[19] Marr T. et al., “Hyper-threading Technology Architecture and Microarchtiecture”, Intel technology Journal, 6(1), 2002
[20] Mosix, http://www.mosix.org
[21] MPI, http://www.csm.ornl.gov/pvm/pvm_home.html
[22] OpenSSI , http://openssi.org
[23] Oplinger J. and Lam M. S., “Enhancing Software Reliability using Speculative Threads”, Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems, October 2002.
[24] Oplinger J. and Lam M. S., “Enhancing Software Reliability with Speculative Threads”, Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems, 2002.
[25] PVM, http://www-unix.mcs.anl.gov/mpi/
[26] Roth and Sohi G.S., “Speculative Data-Driven Multithreading”, in Proc. of the 7th. Int. Symp. On High Performance Computer Architecture, pp. 37-48, 2001
[27] Sazeides Y. and Smith J.E., "The Predictability of Data Values," 30th IEEE/ACM International Symposium on Microarchitecture (MICRO), December 1997.
[28] Sazeides Y., Vassiliadis S., Smith J.E., “The performance potential of data dependence speculation and collapsing”, 9th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-29), 1996
[29] Storino S. and Borkenhagen J., “A Multithreaded 64-bit PowerPC Commercial RISC Processor Design”, in Proc. Of the 11th Int. Conf. on High Performance Chips, 1999
[30] Tremblay M. et al., “The MAJC Architecture, a synthesis of of Parallelism and Scalability”, IEEE Micro, 20(6), 2000
[31] Tullsen D. M., Eggers S.J. and Levy H.M., “Simultaneous Multithreading: Maximizing On-Chip Parallelism”, in Proc. of the 22nd Int. Symp. on Computer Architecture, pp. 392- 403, 1995
[32] Yeh T.Y,, Patt Y.N., “Two-level adaptive branch prediction”, Proceedings of the 24th ACM/IEEE International Symposium on Microarchitecture, 51—61, 1991
[33] Zilles C., “Master/Slave Speculative Parallelization and Approximate Code”, Ph.D. Dissertation, University of Wisconsin-Madison, August 2002.
[34] Zilles C.B. and Sohi G.S., “Execution-Based Prediction Using Speculative Slices”, in Proc. of the 28th Int. Symp. on Computer Architecture, 2001
[35] Zilles C.B. and Sohi G.S., “Master/Slave Speculative Parallelization”, in Proc. of the 35th Int. Symp. on Microarchitecture, 2002
[36] E. W. Dijkstra, “A note on two problems in connexion with graphs”, Numerische Mathematik. 1 (1959), S. pag. 269–271
[37] G. B. Dantzig, R. Fulkerson, and S. M. Johnson, “Solution of a large-scale traveling salesman problem”, Operations Research 2 (1954), pag. 393-410.
[38] K. Kar, M. Kodialam and T. Lakshman, "Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications", IEEE Journal on Selected Areas in Communications (JSAC), volume 18, number 12, pag. 2566-2579, December 2000.
[39] Golden,B.,Bondin,L.,Doyle,T. y Stewart, J.R. (1980).Approximate Travelling Salesman Algorithms. Operation Research 28,694-711.
[40] Christofides,N.(1976).Worst-Case Analisys of a New Heuristic for the Travelling Salesman Problem. Management Sciences Research Report 388.
[41] Ravikumar,C.P. (1992). Parallel techniques for solving large scale travelling salesperson problems. Microprocessors and Microsystems 16(3),149-158.

Acknowledgments

This work was partially supported by the Spanish Science and Technology Ministry under contract TIC2003-05567, and by the Department of Universities, Research and Information Society (DURSI) of the Government of Catalonia under contract 2005-SGR-00296.