8 CONCLUSIONS

In this thesis we have approached the problem of superscalar processor fetch performance using both software and hardware techniques. We first try to optimize the applications so that they make better use of the available hardware resources, then we adapt the hardware to optimize the interaction with the application optimizations, and finally we design a completely new hardware that exploits the unique characteristics of optimized codes.

The contributions of our work roughly correspond to the different chapters of this document:

The Software Trace Cache is a novel code layout algorithm which targets not only an improvement in the instruction cache performance, but also an increase in the effective fetch width of the fetch engine.

The results obtained with the STC are visible in all three factors of fetch performance. In addition to the performance results, we have also performed a comprehensive analysis of the reasons behind these results

- Instruction cache performance is better than with other code layout optimizations.
  - Optimized applications experience a large increase in spatial locality
  - They execute longer chains of sequential instructions
  - They map only useful instructions to a cache line. Unexecuted code is moved to unused regions of the address space and is rarely loaded into the instruction cache.
  - The increased instruction density allows more opportunity for temporal reuse, increasing the time a line is present in the cache
- Fetch width increases to reach performance similar to that of a trace cache.
  - Optimized codes execute over 16 sequential instructions in average

  - It is much easier to fetch past a not taken branch than it is to fetch both a taken branch and its target
- Branch prediction accuracy improves in simple 2-level branch predictors like gshare, and does not decrease significantly with dealiased predictors like gskew.
  - Optimized codes suffer less negative aliasing because most branches have similar behavior (usually not-taken)
  - The increased ratio of not taken branches has a negative impact in the distribution of data in the 2^{nd} level prediction table

From this part of the work we have obtained several publications:

1. The Software Trace Cache algorithm, performance evaluation, fetch width analysis, synergy with the trace cache mechanism:
   - Alex Ramirez, Josep Ll. Larriba-Pey, Carlos Navarro, Xavi Serrano, Josep Torrellas, and Mateo Valero. **Optimizing instruction fetch for decision support workloads.** 2nd Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-2), Orlando (USA). January 1999.[67]
   - Alex Ramirez, Josep Ll. Larriba-Pey, Carlos Navarro, Josep Torrellas, and Mateo Valero. **Software trace cache.** Proceedings of the 13th Intl. Conference on Supercomputing, Rhodes (Greece). June 1999.[69]
   - Alex Ramirez, Josep Ll. Larriba-Pey, Carlos Navarro, Xavi Serrano, Josep Torrellas, and Mateo Valero. **Optimization of instruction fetch for decision support workloads.** Proceedings of the Intl. Conference on Parallel Processing, Aizu (Japan). September 1999.[68]
2. Impact on the branch prediction mechanism:
   - Alex Ramirez, Josep L. Larriba-Pey, and Mateo Valero. **The effect of code reordering on branch prediction.** Proceedings of the Intl. Conference on Parallel Architectures and Compilation Techniques, pages 189-198, Philadelphia (USA). October 2000.[61]
3. Impact on the instruction memory hierarchy and overall processor performance:
   - Alex Ramirez, Luiz Barroso, Kourosh Gharachorloo, Robert Cohn, Josep L. Larriba-Pey, Geoffrey Lawney, and Mateo Valero. **Code layout optimizations for transaction processing workloads.** Proceedings of the 28th Annual Intl. Symposium on Computer Architecture, Goteborg (Sweden). July 2001.[60]
   - Carlos Navarro, Alex Ramirez, Josep L. Larriba-Pey and Mateo Valero. **Fetch Engines and Databases.** 3rd Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-3), Tolouse (France). January 2000.[50]
   - Carlos Navarro, Alex Ramirez, Josep Ll. Larriba-Pey and Mateo Valero. **On the performance of fetch engines running DSS workloads.** Proceedings of the Intl. Euro-Par Conference, Munich (Germany). August 2000.[51]

Selective Trace Storage is a modification to the fill unit of the trace cache mechanism which avoids the redundancy between the trace cache and the software trace cache, ensuring that traces are present in only one cache (either instruction cache, or trace cache). Our results show that eliminating this redundancy allows a 50–75% reduction in the trace cache size without any performance degradation.

The results for this part of the work were published in:

- Alex Ramirez, Josep Ll. Larriba-Pey, and Mateo Valero. **Trace cache redundancy: Red & blue traces.** Proceedings of the 6th Intl. Conference on High Performance Computer Architecture, Tolouse (France). January 2000.[70]

The *agbias* **branch predictor** is our contribution on the subject of branch prediction using profile data. In addition to enhancing existing dynamic predictors, a complete redesigns of the prediction structures allows an optimum usage of the available profile information. Our agbias predictor proves more accurate than any other dynamic predictor examined.

The description and performance evaluation of the agbias predictor were published in:

- Alex Ramirez, Josep L. Larriba-Pey and Mateo Valero. **Semi-static branch prediction for optimized code layouts.** 3rd Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-3), Tolouse (France). January 2000.[62]
- Alex Ramirez, Josep L. Larriba-Pey, and Mateo Valero. **Branch prediction using profile data** Proceedings of the Intl. Euro-Par Conference, Manchester (UK). August 2001.[64]

The Stream Processor Front-end is our novel fetch architecture. It is designed to take advantage of the special characteristics of optimized applications, and obtains significant performance improvements over the classic BTB architecture, and has a performance close to that of a trace cache while requiring much less cost and complexity.

The introduction of the stream processor, and our development for its fetch architecture can be found in:

- Alex Ramirez, Josep L. Larriba-Pey, and Mateo Valero. **A stream processor front-end.** IEEE TCCA Newsletter, pages 10-13. 2000.[63]
- Alex Ramirez, Josep L. Larriba-Pey, Mateo Valero. **Fetching instruction streams.** Technical Report UPC-DAC-2001-38. November 2001.[65]

In addition to the technical contributions described above, we have also contributed to the field of fetch architectures and code layout optimizations with an invited paper in the special issue on Microprocessor Architecture and Compiler Technology of the Proceedings of the IEEE. The paper describes the state of the art in our field, including the key contributions of our work.

- Alex Ramirez, Josep L. Larriba-Pey, and Mateo Valero. **Instruction fetch architectures and code layout optimizations.** Proceedings of the IEEE (invited paper), 89(11):15881609. November 2001.[66]