Optimització del rendiment de processadors superescalars mitjançant la cooperació de mecanismes Software i Hardware

Alex Ramírez i Bellido

May 22, 1999

1 Dades del projecte

Estudiant: Alex Ramírez Bellido

Adreça: Dtr. Brugeroles 17.1, 08980 St. Feliu de Llobregat

Telèfon: 93 666 3337 / 93 401 7187

Director del projecte: Josep Lluis Larriba Pey

2 Resum del projecte

Els processadors superescalars dominen el mercat dels servidors actuals. A mida que apareixen nous avenços en el camp de l'arquitectura de computadors i avancen les tecnologies, la capacitat d'execució paralela d'instruccions d'aquests processadors anirà creixent, en un intent d'explotar més i més paralelisme a nivell d'instrucció (ILP, *Instruction Level Paralelism*).

Però encara ara, el rendiment real d'aquests processadors està molt per sota del seu potencial teòric. Les causes d'aquest desaprofitament son moltes i de naturalesa ben diferent: falles de predicció de salts, falles en els diferents nivells de la jerarquia de memória i bloquejos del processador deguts a dependències de dades son sols alguns exemples.

Hi ha multitud de propostes que pretenen solucionar un o més d'un d'aquests problemes. Però excepte unes poques, totes es limiten a proposar un mecanisme hardware, amb la consequent complicació en el disseny i verificació del processador, o un mecanisme software, gairebé sempre en forma d'una millora del compilador. Gairebé mai es considera que el hardware i el software han de treballar junts. Cal que el hardware s'adeqüi a les característiques de codi que haurà d'executar, i que els compiladors generin un codi que sigui capaç d'explotar al màxim les capacitats del hardware de que disposa.

Nosaltres estudiarem l'efecte conjunt de mecanismes hardware i software, partint de la base que cap mecanisme aïllat pot produir resultats óptims, i éssent conscients que la interacció entre aquests mecansimes no sempre és obvia.

Començarem la nostre recerca examinant un tipus d'aplicació que presenta un rendiment molt baix en els processadors superescalars actuals, com son les bases de dades. A continuació, ens centrarem en un mecanisme aïllat, com es la cache d'instruccions de primer nivell, que presenta un rendiment especialment baix per aquestes aplicacions. A partir d'un estudi del comportament del codi de les bases de dades porposarem un algorisme de reordenació dels blocs bàsics del programa per tal d'optimitzar el rendiment de la cache d'instruccions.

Tot seguit, examinarem l'efecte d'aquesta organització de codi en el conjunt del mecanisme de fetch per a processadors superescalars molt més agresius, i compararem la seva eficiència amb el mecanisme de *trace cache*.

A continuació proposarem mecanismes hardware que treballin conjuntament amb aquest tipus d'organitzacions de codi, i modificarem l'algorisme de reordenació per tal d'aprofitar millor el mecanisme de fetch.

També porposarem modificacions del codi i modificacions al disseny del mecanisme de predicció de salts, així com al *scheduling* d'instruccions en el processador.

Volem implementar totes aquestes modificacions en un post-compilador que sigui capaç de modificar el codi d'un executable ja compilat i enllaçat. L'efecte d'aquestes modificacions software sobre els mecanismes hardware proposats s'avaluarà mitjançant simulació.

3 Objectius

Mecanismes software de reordenació de codi

Com a primer objectiu d'aquest treball vàrem dur a terme un estudi en profunditat del comportament del codi d'una base de dades, PostgreSQL 6.3.2 [18]. El comportament de les babes de dades és diferent del d'altres aplicacions, tant enteres com de càlcul numèric (per exmple els SPEC'95) [2, 17, 19, 21, 30, 35]. En termes generals, cal remarcar el baix rendiment que aquestes aplicacions obtenen en processadors superescalars. Aquest baix rendiment, i la seva gran importància, fa d'aquestes aplicacions un bon camp d'estudi de cara a millorar el rendiment dels processadors superescalars.

A partir d'aquest estudi, vàrem desenvolupar un algorisme de reordenació de codi amb l'objectiu de millorar el rendiment de la cache d'instruccions de primer nivell [27]. Aquesta ordenació de codi incrementa al mateix temps la seqüencialitat del codi, donant peu a una millora en el rendiment global de la unitat de fetch de futurs processadors superescalars.

En aquesta mateixa linia, vàrem estudiar el rendiment d'una unitat de fetch bàsica en vers la d'una unitat de fetch coplementada amb una *trace cache* [31], tant per al codi original com per al codi reordenat usant el nostre algorisme. D'aquest estudi es conclou que ambdos mecanismes, la reordenació en temps de compilació, i la reordenació en tems d'execució, eren complementaris [29].

Implementació en un post-compilador

Seguint aquesta linia d'investigació, pretenem desenvolupar una eina de reordenació de codi que permeti implementar l'algorisme abans mencionat. Per tal de poder dur això a terme, caldrà solucionar tots els problemes que suposa reordenar els blocs bàsics d'un programa, un cop aquest ja ha estat compilat i enllaçat. També pretenem que aquesta eina sigui prou flexible com per permetre la implementació d'altres algorismes de reordenació que puguin millorar els resultats del que nosaltres proposem en [29] i permeti la implementació d'altres tècniques de modificació de codi, com ara *trace scheduling* [8, 13, 22].

Proposta de mecanismes hardware que n'aprofitin la reodenació

Tanmateix, hem volgut aprofundir més en la relació que existeix entre la ordenació del codi i un mecanisme de fetch complementat amb una trace cache. Estem estudiant una sèrie de modificacions al mecanisme bàsic de la trace cache que permeten un major aprofitament dels recursos hardware, especialment en presència d'un codi degudament reordenat.

Mecanismes software i hardware de predicció de salts

L'algorisme que proposem per reordenar els blocs bàsics d'un programa està basat en un *profile* del codi, és a dir, en informació obtinguda a partir d'execucions de prova d'aquest codi. A partir d'aquesta informació s'en pot extreure molt més que una ordenació óptima dels blocs bàsics del programa. També podem aprofitar per observar el comportament dels salts condicionals del programa, i aprofitar-nos d'aquest coneixement per tal d'usar millor els recursos hardware de que disposem, o bé per disenyar mecanismes nous que facin us d'aquest coneixement.

Mecanismes software de reordenació d'instruccions

De la mateixa manera, també és possible trobar una ordenació óptima de les instruccions dins un bloc bàsic, o dins un grup de blocs bàsics, basant-nos en aquesta informació de *profile*. També volem incorporar optimitzacions d'aquest tipus en el post-compilador desenvolupat, extenent el rang d'optimització més enllà del mecanisme de fetch.

4 Estat de l'art

Comportament de les bases de dades

Pel que fa a l'estudi del comportament de les bases de dades, aquest ha estat un camp d'estudi força novedós, i molt popular en els darrers anys [2, 7, 17, 19, 21, 30, 35]. Les aplicacions comercials representen una part molt important del mercat de servidors, i tenen tendència a creixer cada cop més.

El seu comportament és molt diferent del d'altres aplicacions. Les bases de dades implementen complexes mecanismes de *lock* i sincronització, tenen els seus propis mecanismes de manegament de memória, es comuniquen directament amb els dispositius d'entrada/sortida, fan servir complexes estructures de dades, tenen una component molt significativa d'entrada/sortida i un gran paralelisme.

En aquests moments, el coneixement que tenim del comportament d'aquestes aplicacions en processadors superescalars (tant amb execució en ordre, com fora d'ordre) i processadors multithreading és prou bo, però tot i això, s'han proposat poques millores al disseny del processador dirigides a millorar el rendiment d'aquestes aplicacions.

També s'han dut a terme estudis del codi font [23, 32] que ens han aportat coneixements sobre el model d'execució de consultes de PostgreSQL. Aquest coneixement del codi font ens ha servit per explicar els resultats obtinguts en l'estudi del comportament d'aquestes aplicacions en processadors actuals.

La conclusió més immediata que s'extreu d'aquests treballs, és que la jerarquia de memória dels processadors actuals no s'adequa a les necessitats d'aquestes aplicacions, especialment pel que fa a la cache d'instruccions de primer nivell.

La cache d'instruccions de primer nivell

Els processadors actuals han estat dissenyats i avaluats tenint en compte un conjunt d'aplicacions determinades (per exemple els SPEC benchmarks). Aquestes aplicacions acostumen a ser de càlcul intensiu, amb tamanys de dades mitjans o petits, i amb codis molt reduits. Donat que el codi d'aquestes aplicacions és prou petit com per encabir-se en caches de 8-32 Kbytes, no es considerava que aquest hagués de ser un problema. Però el codi d'una base de dades és molt més gran, i les seqüències d'instruccions executades son prou llargues com per omplir una cache d'instruccions petita. El nombre d'instruccions executades entre dos usos consecutius d'una mateixa rutina és tan gran, que quan aquesta es vol reutilitzar, ja ha estat desplaçada de la cache, i es produeixen continues falles [23, 27].

Donat que el mapeig d'instruccions en la cache de primer nivell ve fixat per l'adreça lógica d'aquesta instrucció, i que l'adreça d'una instrucció la fixa el compilador, és possible generar un codi que produeixi menys falles de cache que d'altres. La interacció entre el codi generat pel compilador i la cache d'instruccions va ser estudiada en detall per Chen *et al* en [4].

Hi ha multitud de treballs en el camp de la reordenació de codi per millorar el rendiment de la cache d'instrucccions. Aquests treballs proposen un ordre més óptim per al codi d'un programa basant-se en informació de *profile* del codi, és a dir, es tracta d'informació obtinguda *a posteriori*, i que no era disponible en el moment de compilar el programa.

Hwu i Chang [14] proposen una reordenació dels blocs bàsics del programa atenent a la seva seqüencialitat, posant junts aquells blocs bàsics que s'executen de forma consecutiva, sense parar atenció a quin procediment pertanyen els blocs bàsics. Pettis i Hansen [26] proposen un algorisme que determina l'ordre òptim per als procediments d'un programa, i també reordena els blocs bàsics dins el cos del procediment. Cap dels dos treballs té en compte quina es la organització de cache del processador en que s'ha d'executar el programa.

Hashemi, Kalamaitianos *et al* p[roposen un mapeig dels procediments del programa en linies de cache usant un algorisme heredat del colorejat de registres i informació sobre la geometria de la cache d'instruccions. Gloy et al [11] porposenuna extensió de l'algorisme proposat per Pettis i Hansen per tal que també consideri la relació temporal entre els procediments, no tan sols quins procediments criden a quins altres.

Torrellas *et al* [34] extenen l'algorisme de Hwu i Chang per tal de considerar la geometria de la cache. Proposen destinar una zona de la cache per als blocs bàsics més frequentement utilitzats, de manera que cap altre codi sigui mapejat en posicions de memòria que puguin desplaçar aquests blocs bàsics de la cache.

Nosaltres extenem l'algorisme de Torrellas *et al* per tal que consideri seqüències de blocs bàsics, en lloc de blocs bàsics individuals, explotant millor la localitat espaial del programa. En el nostre primer treball [28] comparem el rendiment del nostre algorisme en vers els proposats per Pettis i Hansen [26] i Torrellas *et al* [34], així com amb altres tècniques hardware, com caches associatives i caches de victimes [15].

El mecanisme de fetch, la trace cache

Acte seguit vàrem examinar l'efecte d'aquestes organitzacions de codi en el conjunt de la unitat de fetch. Les unitats de fetch actuals són similars a la proposada per Yeh i Patt [38], que incorpora un mecanisme de predicció de salts que permet a la unitat de fetch treballar desacoplada de la unitat d'execució del processador, tot seguint camins especulatius (veure la Figura 1).

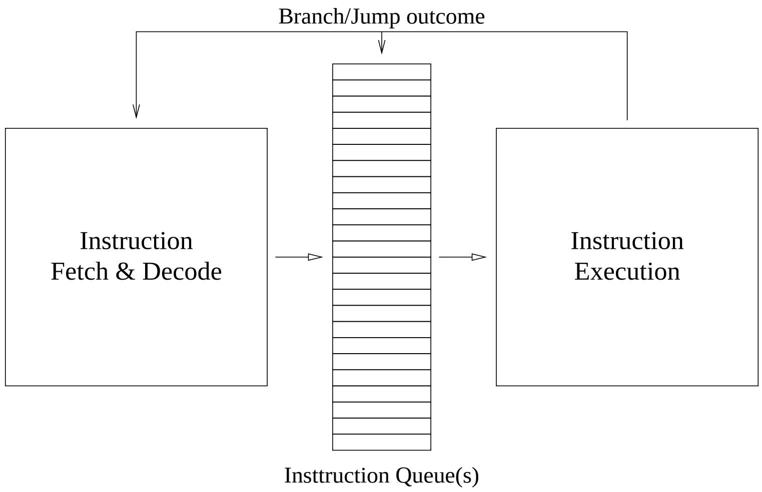

Figure 1: Model desacoplat d'un processador superescalar, amb una unitat de fetch i una unitat d'execució que es comuniquen a través d'una cua d'instruccions.

Però per als processadors futurs, que pretenen executar més de 4 instruccions per cicle, cal proporcionar-los instruccions de més d'un bloc bàsic per cicle. Una extensió natural d'aquesta unitat de fetch permet llegir instruccions de múltiples blocs bàsics consecutius (Figura 2). Però per una organització bàsica del codi, els programes enters executen menys de dos blocs basics consecutius per terme mitjà.

Per tal de solucionar aquest problema, s'han proposat diversos mecanismes hardware. Tots aquests mecanismes llegeixen instruccions de diferents linies de cache, i reorganitzen el seu contingut de manera que es pot proporcionar un seguit d'instruccions a la unitat d'execució com si aquestes fossin consecutives. El *collapsing buffer* [6] i la *branch address cache* [36] fan aquesta reordenació en el moment de llegir les instruccions, introduint un possible retard en l'etapa de fetch. La *trace cache* [9, 25, 31] fa aquesta reordenació un cop les instruccions han estat executades, i guarda la seqüència resultant en una cache de propòsit específic (veure Figura 2), en espera que es torni a produir aquesta mateixa seqüència, si és així, la traça es pot proporcionar de forma inmediata sense cap procés ni retard adicional.

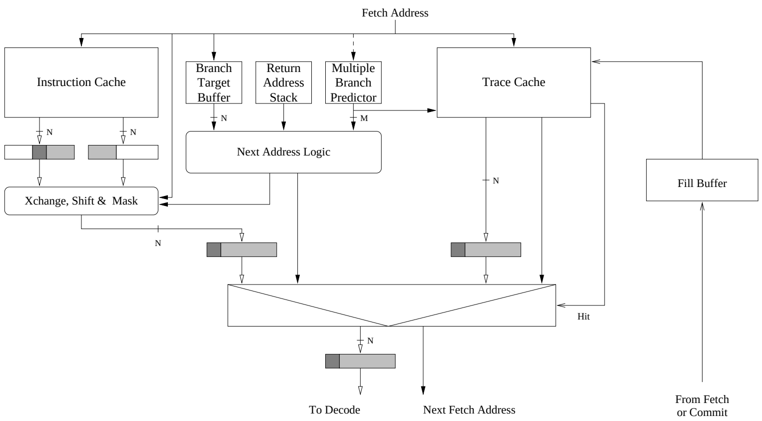

Figure 2: Unitat de fetch capaç de llegir instruccions de múltiples blocs bàsics per cicle, complementada amb una trace cache.

Tots aquests mecanismes tenen com a objectiu enfrontar-se a la falta de seqüencialitat del codi, mentre que la nostre organització de codi intenta augmentar aquesta seqüencialitat. En aquest sentit, es tracta de mecanismes complementaris, com també es pot comprovar en els resultats obtinguts simulant el comportament d'una unitat de fetch complementada o no amb una trace cache, per al codi original i per al codi reordenat segons la nostre proposta [29].

Un cop hem pogut comprovar la relació existent entre el macanisme hardware de la *trace cache* i la nostre organització de codi, es pretén augmentar-la modificant la part de la trace cache que fa la reordenació dinàmica de les instruccions, la *fill unit*, per tal que la trace cache no repeteixi la feina que ja hem fet nosaltres en temps de compilació. Tan sols es té coneixement d'un altre treball que proposi modificacions de la *fill unit*. Friendly *et al* [10] proposen extendre la *fill unit* per tal que faci optimitzacions sobre el codi al mateix temps que el reordena. Aquesta modificació de la *fill unit* és aplicable tant al model clàssic de la *trace cache*, com a la *block-based trace cache* proposada per Black *et al* [3].

Predicció de salts

Una altra part del mecanisme de fetch és el de predicció de la següent adreça de fetch, és a dir: el predictor de direccions de salt (BP, *branch predictor*), i el predictor de destinacions de salt (BTB, *branch target buffer*).

Com més acurades siguin les prediccions, menys instruccions pertanyents a un camí d'execució incorrecte seran passades cap a la unitat d'execució, estalviant cicles de feina incorrecte.

Els predictors de direccions (també anomenats simplement predictors de salts) han avançat molt darrerament, començant per la introducció dels predictors de dos nivells [37], que tenen en compte no tan sols de quin salt estem parlant, sinó el que ha passat amb els salts anteriors, i arribant als predictors de salts amb una longitud d'história adaptable [16], on la quantitat de salts anteriors considerats s'asapta automàticament segons siguin els requeriments del codi.

També ha calgut modificar aquests predictors de salts per tal que puguin oferir varies prediccions en un sol cicle [9, 31, 33], modificació necessària si volem ser capaços de llegir múltiples blocs bàsics per cicle.

També hi ha treballs en la predicció de salts en temps de compilació [1, 12] i fins i tot un article relacionant la predicció estàtica de salts amb la trace cache [24]. Si bé els resultats obtinguts en predicció estàtica de salts no son tan bons com els obtinguts amb predictors dinàmics, també és cert que la precisió d'aquests últims baixa a mida que fem més i més prediccions per cicle, de manera que poder podrem aprofitar la predicció de salts estàtica per reduir el nombre de prediccions necessàries i augmentar la precisió global.

Instruction scheduling

L'ordre de les instruccions en un programa també aftecta al rendiment del processador. L'ordre en que aquestes s'envien a executar determina el nivell de paralelisme a nivell d'instrucció que es pot explotar, aprofitant el temps que una instrucció ha d'esperar a que els seus operands estiguin calculats per executar altres instruccions independents. Aquest problema ha estat molt estudiat en el camp dels processadors VLIW (*Very Long Instruction Word*), on el paralelisme a nivell d'instrucció és explícit.

Totes aquestes tècniques, com *trace scheduling* [5, 8], *superblock* [13] i *hyperblock* [20] són també aplicables als processadors superescalars amb execució en ordre, com l'alpha 21164 (ev5).

Amb l'aparició dels processadors superescalars amb execució fora d'ordre, la recerca en aquest camp ha decaigut molt, donant pas a altres técniques com ara el reús d'instruccions o la predicció de resultats, amb l'objectiu d'evitar les dependències de dades. Amb l'anunci per part d'Intel i Hewlett-Packard del nou IA-64

(Merced), l'arquitectura VLIW ha tornat a guanyar força, i pot ser interessant tornar enrera i fer un cop d'ull a totes aquestes tècniques.

5 Plà de treball

Estiu 1999

Desenvolupament d'una eina de recompilació de programes executables en col.laboració amb Compaq-Western Research Lab. Aquest post-compilador serà capaç de modificar el codi d'un executable per tal de reordenar els blocs bàsics del programa, i fins i tot modificar-ne el contingut, sense que el resultat del programa sigui diferent. Inicialment s'implementaran les optimitzacions del mecanisme de fetch i del rendiment de la cache d'instruccions de primer nivell.

Hivern 1999

Desenvolupament de nous algorismes de reordenació de codi amb l'objectiu de millorar el rendiment de la unitat de fetch. Estudi de mecanismes de fetch alternatius que aprofitin la nova organització del codi, cerca d'un mecanisme óptim en rediment i en cost d'implementació.

Any 2000

Recerca en altres optimitzacions software per altres aspectes del processador superescalar (com ara *trace scheduling* o predicció de salts en temps de compilació) i incorporació de les mateixes en el post-compilador.

Un cop tot el paquet d'optimitzacions estigui incorporat en el post-compilador i s'hagin avaluat els seus efectes en els processadors actuals i futurs, es donarà per conclós el treball de tesi.

6 Publicacions

Code Reordering of Decision Support Systems for optimized Instruction Fetch. Alex Ramírez, Josep Ll. Larriba-Pey, Carlos Navarro, Xavi Serrano, Josep Torrellas and Mateo Valero. Technical Report UPC-DAC-1998-56, Universitat Politecnica de Catalunya, December 1998.

Optimization of Instruction Fetch for Decision Support Workloads. Alex Ramírez, Josep Ll. Larriba-Pey, Carlos Navarro, Xavi Serrano, Josep Torrellas and Mateo Valero. 2nd Workshop on Computer Architecture Evaluation using Commercial Workloads, HPCA-5, January 1999.

Software Trace Cache. Alex Ramírez, Josep Ll. Larriba-Pey, Carlos Navarro, Josep Torrellas and Mateo Valero. Technical Report UPC-DAC-1999-5, Universitat Politecnica de Catalunya, January 1998.

Software Trace Cache. Alex Ramírez, Josep Ll. Larriba-Pey, Carlos Navarro, Josep Torrellas and Mateo Valero. 13th ACM-SIGARCH International Conference on Supercomputing, Rhodas (Grècia), Juny 1999.

Code Reordering of Decision Support Systems for optimized Instruction Fetch. Alex Ramírez, Josep Ll. Larriba-Pey, Carlos Navarro, Xavi Serrano, Josep Torrellas and Mateo Valero. Intl. Conference on Parallel Processing, Tokyo (Japó), Setembre 1999.

References

[1] Thomas Ball and James R. Larus. Branch prediction for free. *Proc. ACM SIGPLAN'93 Conf. on Programming Language Design and Implementation*, pages 300–313, June 1993.

[2] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. *Proceedings of the 16th Annual Intl. Symposium on Computer Architecture*, pages 3–14, June 1998.

[3] Bryan Black, Bohuslav Rychlik, and John Paul Shen. The block-based trace cache. *Proceedings of the 13th Intl. Conference on Supercomputing*, June 1999.

[4] William Y. Chen, Pohua P. Chung, Thomas M. Conte, and Wen-Mei Hwu. The effect of code expanding optimizations on instruction cache design. *IEEE Transactions on Computers*, 42(9):1045–1057, September 1993.

[5] R. P. Colwell, R. P. Nix, J. J. O'Donell, D. B. Papworth, and P. K. Rodman. A vliw architecture for a trace scheduling compiler. *IEEE Transactions on Computers*, 37(9):967–979, August 1988.

[6] T. Conte, K. Menezes, P. Mills, and B. Patell. Optimization of instruction fetch mechanism for high issue rates. *Proceedings of the 22th Annual Intl. Symposium on Computer Architecture*, pages 333–344, June 1995.

[7] Richerd J. Eickenmeyer, Ross E. Johnson, Steven R. Kunkel, Mark S. Squillante, and Shiafun Liu. Evaluation of multithreaded uniprocessors for commercial application environments. *Proceedings of the 23th Annual Intl. Symposium on Computer Architecture*, pages 203–212, 1996.

[8] J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computers*, 30(7):478–490, July 1981.

[9] Daniel Holmes Friendly, Sanjay Jeram Patel, and Yale N. Patt. Alternative fetch and issue techniques from the trace cache mechanism. *Proceedings of the 30th Anual ACM/IEEE Intl. Symposium on Microarchitecture*, December 1997.

[10] Daniel Holmes Friendly, Sanjay Jeram Patel, and Yale N. Patt. Putting the fill unit to work: Dynamic optimization for trace cache microprocessors. *Proceedings of the 30th Anual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 173–181, November 1998.

[11] Nikolas Gloy, Trevor Blackwell, Michael D. Smith, and Brad Calder. Procedure placement using temporal ordering information. *Proceedings of the 30th Anual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 303–313, December 1997.

[12] Amir H. Hashemi, David R. Kaeli, and Brad Calder. Procedure mapping using static call graph estimation. *Proc. Workshop on Interaction between Compiler and Computer Architecture*, February 1997.

[13] W. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Water, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara G. E. Haab, J. G. Hold, and D. M. Lavery. The superblock: An effective technique for vliw and superscalar compilation. *Journal on Supercomputing*, (7):9–50, 1993.

[14] Wen-Mei Hwu and Pohua P. Chang. Achieving high instruction cache performance with an optimizing compiler. *Proceedings of the 16th Annual Intl. Symposium on Computer Architecture*, pages 242–251, June 1989.

[15] N. J. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. *Proceedings of the 17th Annual Intl. Symposium on Computer Architecture*, pages 364–373, June 1990.

[16] T. Juan, S. Sanjeevan, and J. J. Navarro. Dynamic history-length fitting: A thrid level of adaptivity for branch prediction. *Proceedings of the 25th Annual Intl. Symposium on Computer Architecture*, pages 155–166, June 1998.

[17] Kimberly Keeton, David A. Patterson, Yong Quiang He, Roger C. Raphael, and Walter E. Baker. Performance characterization of a quad pentium pro smp using oltp workloads. *Proceedings of the 25th Annual Intl. Symposium on Computer Architecture*, pages 15–26, June 1998.

[18] M. Stonebreakerand G. Kemnitz. The postgres next generation database management system. *Communications of the ACM*, October 1991.

[19] Jack L. Lo, Luiz André Barroso, Susan J. Eggers, Kourosh Gharachorloo, Henry M. Levy, and Sujay S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. *Proceedings of the 25th Annual Intl. Symposium on Computer Architecture*, pages 39–50, June 1998.

[20] Scott A. Mahlke, David C. Lin, William Y. Chen, Richard E. Hank, and Roger A. Bringmann. Effective compiler support for predicated execution using the hyperblock. 1992.

[21] A. M. Maynard, C. M. Donnelly, and B. R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. *Proceedings of the 6th Intl. Conference on Architectural Support for Programming Languajes and Operating Systems*, pages 145–156, October 1994.

[22] S.-M. Moon and K. Ebicioglu. An efficient resource constrained global scheduling technique for superscalar and vliw processors. *Proceedings of the 25th Anual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 55–71, 1992.

[23] Carlos Navarro-García. Caracterización de la ejecución de consultas sobre postgres95. *Projecte final de carrera, Facultat d'Informàtica de Barcelona, UPC*, June 1998.

[24] Sanjay Jeram Patel, Marius Evers, and Yale N. Patt. Improving trace cache effectiveness with branch promotion and trace packing. *Proceedings of the 25th Annual Intl. Symposium on Computer Architecture*, pages 262–271, June 1998.

[25] Sanjay Jeram Patel, Daniel Holmes Friendly, and Yale N. Patt. Critical issues regarding the trace cache fetch mechanism. Technical Report CSE-TR-335-97, University of Michigan, May 1997.

[26] Karl Pettis and Robert C. Hansen. Profile guided code positioning. *Proc. ACM SIGPLAN'99 Conf. on Programming Languaje Design and Implementation*, pages 16–27, June 1990.

[27] Alex Ramírez, Josep Ll. Larriba-Pey, Carlos Navarro, Xavi Serrano, Josep Torrellas, and Mateo Valero. Code reordering of decision support systems for optimized instruction fetch. *Proc. of the Intl. Conference on Parallel Processing*, September 1999.

[28] Alex Ramírez, Josep Ll. Larriba-Pey, Carlos Navarro, Xavi Serrano, Josep Torrellas, and Mateo Valero. Code reordering of decision support systems for optimized instruction fetch. Technical Report UPC-DAC-1998-56, Universitat Politecnica de Catalunya, December 1998.

[29] Alex Ramírez, Josep Ll. Larriba-Pey, Carlos Navarro, Josep Torrellas, and Mateo Valero. Software trace cache. *Proceedings of the 13th Intl. Conference on Supercomputing*, page to apperar, June 1999.

[30] Parthasarathy Ranganathan, Kourosh Gharachorloo, Sarita V. Adve, and Luiz André Barroso. Performance of database workloads on shared-memory systems with out of order processors. *Proceedings of the 8th Intl. Conference on Architectural Support for Programming Languajes and Operating Systems*, October 1998.

[31] E. Rottenberg, S. Benett, and J. E. Smith. Trace cache: a low latency aprroach to high bandwith instruction fetching. *Proceedings of the 29th Anual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 24–34, December 1996.

[32] Xavi Serrano-Gispert. Simulació de la jerarquia de memòria en la reordenació del codi de postgres95. *Projecte final de carrera, Facultat d'Informàtica de Barcelona, UPC*, January 1999.

[33] A. Seznec, S. Jourdan, P. Sainrat, and P. Michaud. Multiple-block ahead branch predictors. *Proceedings of the 7th Intl. Conference on Architectural Support for Programming Languajes and Operating Systems*, October 1996.

[34] Josep Torrellas, Chun Xia, and Russell Daigle. Optimizing instruction cache performance for operating system intensive workloads. *Proceedings of the 1st Intl. Conference on High Performance Computer Architecture*, pages 360–369, January 1995.

[35] Pedro Trancoso, Josep Ll. Larriba-Pey, Zheng Zhang, and Josep Torrellas. The memory performance of dss commercial workloads in shared-memory multiprocessors. *Proceedings of the 3rd Intl. Conference on High Performance Computer Architecture*, February 1997.

[36] T. Y. Yeh, D. T. Marr, and Y. N. Patt. Increasing the instruction fetch rate via multiple branch prediction and a branch address cache. *Proceedings of the 7th Intl. Conference on Supercomputing*, pages 67–76, July 1993.

[37] T. Y. Yeh and Y. N. Patt. Two-level adaptive branch prediction. *Proceedings of the 24th Anual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 51–61, 1991.

[38] T.-Y. Yeh and Y. N. Patt. A comprehensive instruction fetch mechanism for a processor supporting speculative execution. *Proceedings of the 25th Anual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 129–139, December 1992.