Reducing power of functional units in high-performance processors by checking instruction codes and resizing adders

G. Miñana, J.I. Hidalgo, J. Lanchares, J.M. Colmenar, O. Garnica and S. López

Abstract: A hardware technique to reduce static and dynamic power consumption in functional units of 64-bit high-performance processors is presented here. The instructions that require an adder have been studied and it can be concluded that, there is a large percentage of instruction where one of the two source operands is always narrow and does not require a 64-bit adder. Furthermore, by analysing the executed applications, it is feasible to classify their internal operations according to their bit-width requirements and select the appropriate adder type that each instruction requires. This approach is based on substituting some of the 64-bit power-hungry adders with 32-bit ones, which consume much lower power, and modifying the protocol to issue as much instructions as possible to these low power consumption units, while incurring in negligible performance penalties. Five different configurations were tested for the execution units. Results indicate that this technique can save between up to 50% of the power consumed by the adders and up to 21% of the overall power consumption in the execution unit of high-performance architectures. Moreover, the simulations show good results in terms of power efficiency (IPC/W) and it can be affirmed that it could prevent the creation of hot spots in the functional units.

1 Introduction

Nowadays, process technology allows the adaptation of high-performance processors (HPPs) as a promising solution to provide suitable performance/power trade-offs in forthcoming consumer devices. However, these advances (e.g. integration density) are accompanied by increasing static power, dynamic power and hot spots [1, 2], which are critical factors in the implementation of devices and consumer computers.

For this reason, many techniques have been explored to reduce power consumption on HPPs at multiple levels (e.g. microarchitecture, compiler, operating system and application). Some solutions use voltage and frequency scaling [2]. The main disadvantage of these techniques is that they can cause performance degradation.

Brooks and Martonosi proposed using operand values to gate off portions of the execution units [3, 4]. Their scheme detects narrow operand values and exploits them to reduce the amount of power consumed by the execution units by using an aggressive form of clock gating, and disabling the upper bits of the arithmetic logic units (ALUs) when they are not needed.

A technique to exploit critical path information for power reduction was proposed by Seng et al. [5]. In this case, a set of execution units with different power and latency characteristics is provided and the instructions are assigned to these units based on their criticality. Specifically, the instructions predicted as critical for performance are processed by fast, high-power execution units, whereas the instructions predicted as not critical are issued to slow, power-efficient execution units of the same bit-width; thus reducing the overall power consumed by the microprocessors.

A runtime scheme for reducing power consumption in custom logic has been also proposed in [6, 7]. They outline possible ways to reduce power consumption by observing that the bit-widths of certain operands in high-performance superscalar processors are often less than the maximum data width allowed in these architectures. Thus, their techniques divide the functional units into two portions, less significant and the more significant portions. If the data fit within the less significant portion, the more significant portion can be turned off, and their output bits can be computed by a simple sign-extension circuit instead, thus achieving power reduction. However, in some cases this technique degrades processor performance, a handicap for superscalar processors. They also do not reduce static power. Haga et al. [8] present a hardware method for functional unit assignment based on the principle that power consumption in functional units (FUs) can be approximated by the switching activity of its inputs. The basic idea of this method is to reduce the number of bits that switch their stored values between successive computations in the same FU. All these previous methods focus on identifying narrow-width operands in high-performance superscalar processors.

FUs of superscalar processors consume about 20% of the total chip power [9–11]. This power is accounted by the switching activity and by the leakage effects [12, 13]. Although a lot of research exists, not much work has been performed to improve power of functional units in HPPs.
Analysing the use of the FUs of superscalar processors and mixing some of these ideas, avoiding their drawbacks, this study presents a microarchitectural level technique to reduce power consumption of the FUs and to prevent the appearance of hot spots in the processors.

We propose to reduce static and dynamic power consumption by exploiting which instruction is being processed. To this end, we first analyse the use of 64-bit adders in a widely used HPP and evaluate the amount of instructions that could use a 32-bit adder instead (as a large percentage of instructions have a narrow source operand). Next, we present how to combine adders of several lengths (in this case, 32-bit and 64-bit adders) to reduce both static and dynamic power in superscalar architectures. This technique decreases the transistor number in the adders and hence mitigates the hot spot problem in the FUs of the processor. Our experimental results show power reductions without performance penalties.

2 Preliminary considerations about using 64-bit adders on HPPs

Here, we provide a study that shows that in 64-bit processors, a large amount of additions do not require 64-bit adders. We have focused on the Instruction Set Architecture (ISA) of the Alpha architecture [14], one of the most advanced superscalar processors announced to date [15]. The most widely used simulator in HPPs research is the SimpleScalar simulator [16] including a validated model of the Alpha architecture. This advanced architecture uses several FUs and using the Watch tool [9], (an extension of SimpleScalar for power simulations), we are able to evaluate our proposal on a real architecture.

Table 1 summarises the ISA subset of the Alpha architecture that requires an adder. As it is shown, there are seven different types of instructions that need an adder. Looking at the third column of Table 1 we can see the length of the operand involved.

From this table we can extract two conclusions:

- ARITH_LONG instruction operands use the long word data type, so they could be executed with a 32-bit adder.
- Many memory, branch and integer arithmetic instructions where one of the source operands is either a shift or a literal constant do not require a 64-bit adder; they could also be processed in a 32-bit adder.

Initially, all these instructions require a 64-bit adder because the Program Counter (PC) and source registers are 64-bit width. However, in many of the shown instructions, the other source operand is very narrow (e.g. 21-bit width for the BRA instruction type, 13-bit width for the JMP instruction type and so on). Therefore, only in a very limited number of occasions the operation does require a full 64-bit adder. In fact, these instructions require 64-bit adders only if the two following conditions are fulfilled:

1. The operation of adding the value of the PC (or Register) and the value of shift (or immediate value) produces a carry on the most significant bit of the narrow operand source. For instance, if there is a carry produced on the twentieth bit for the BRA instruction (see Fig. 1 for more details).
2. If a carry occurs, this carry has to propagate through the whole 32-bit word. This situation always happens if the narrow operand is negative, or if it is positive and the more significant bits of the first word of the PC (or used register, depending on the operation code) are ‘1’. The number of significant bits depends on the operation code. For example, in the case of the BRA instructions, these bits are placed between the twenty-first bit and the thirty-first bits, both included (Fig. 1).

For illustration purposes, Fig. 1 shows when the BRA, L/S, LDA, JMP and ARIT_IMM instructions need a 64-bit adder, namely, when their operands meet the two previous conditions.

According to the aforementioned analysis, we can estimate theoretically (i.e. by computing probabilistic values) how many special case instructions (or Fig. 1-like cases) could appear during the execution of a program. However, we have performed an experimental study by simulating

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Instruction format</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA</td>
<td>PC + Ext Sig(Shift(21 bits))</td>
<td>63 32 31 21 20 0</td>
</tr>
<tr>
<td>JMP</td>
<td>PC + Ext Sig(Shift(13 bits))</td>
<td>63 32 31 13 12 0</td>
</tr>
<tr>
<td>L/S</td>
<td>Rb + Ext Sig(Shift(16 bits))</td>
<td>63 32 31 8 7 0</td>
</tr>
<tr>
<td>LDA</td>
<td>Ra + Rb</td>
<td>63 32 31 21 20 0</td>
</tr>
<tr>
<td>ARIT</td>
<td>Ra(32 bits) + Rb(32 bits)</td>
<td>63 32 31 21 20 0</td>
</tr>
<tr>
<td>ARIT_LONG</td>
<td>Ra + Immediate(8 bits)</td>
<td>63 32 31 21 20 0</td>
</tr>
</tbody>
</table>

Table 1: Alpha processor instructions that imply the use of an adder.

The third column shows the semantics of the operation. Numbers between parentheses are the length of the operands.
different benchmarks included in SPEC2000 [17] using the SimpleScalar [18] simulator. This analysis shows that the percentage of this kind of special cases of additions is 15%.

The experimental results show also that around 78% of the instructions that require an adder could use a 32-bit adder instead of a 64-bit one. In addition, in order to estimate the number of concurrently needed adders we have analysed the number of cycles in which there is more than one instruction that requires a 64-bit adder. Taking into account the total number of cycles for each benchmark, the averaged percentage is 6.2%.

Using this information, it becomes clear that for a significant number of benchmarks it is feasible to substitute part of the 64-bit adders by 32-bit adder without incurring processor performance penalties. This assumption is validated in a latter section, where we compute performance penalties for all our benchmarks.

3 Proposed power reduction technique

Here, we present a hardware technique to reduce the power consumption in FUs of current high-performance superscalar processors based on the previous conclusions. As mentioned earlier, a huge amount of the instructions that require an adder to be executed could use a 32-bit adder instead of a 64-bit one. Our approach exploits this fact, and proposes to substitute some of the power-hungry 64-bit adders of current superscalar processors by lower power 32-bit adders.

To this end, we have defined a simple logic to issue an instruction to an adder. The selection logic (arbiter), located in the issue queue, dynamically issues instructions to the available adders. In order to find out the adder type that each instruction requires, this arbiter concurrently checks the opcode, and a few additional bits of the operands to account for the special cases explained on previous sections. If a 32-bit adder is available, then the instruction is issued to it. Otherwise, the instruction is issued to an available 64-bit adder to preserve system performance. The algorithm used by our arbiter is:

1. IF (The instruction can be executed in a 32-bit adder)
2. AND (there are 32-bit adders available)
3. ELSE IF (there are 64-bit adders available)
4. ELSE
5. Instruction waits in the queue

Moreover, we have observed that it is possible to find out the type of adder that each instruction requires by simply checking the opcode and following the next algorithm:

1. If the instruction is ARIT (Table 1), it must be issued to 64-bit adder.
2. If the instruction is ARIT_LONG, it can be issued to a 32-bit adder, because the operand size is 32 bits (Table 1).
3. For the remaining instructions (i.e. BRA, L/S, LDA, JMP and ARIT_IMM), we must detect the special cases outlined in Fig. 1 (checking the necessary bits). To this end, we have defined a carry detection process by checking only the most significant bit of the narrow operand, and its counterpart in the other operand. This test finds more cases than those defined in Fig. 1, but it is a simple mechanism that can be implemented without significantly increasing the logic.

Following this logic arbiter and using the SPEC 2000 benchmark set, we have counted the percentage of instructions which need an adder, but can use a 32-bit adder. These results indicate that only 22% of the instructions on average demand a 64-bit adder.

4 Experimental framework

Our simulations have been performed with a modified version of the Wattch simulator [9], a framework for analysing and optimising microprocessor power dissipation at the microarchitecture level. We have extended this simulator to achieve more accurate power figures for the used FUs.

4.1 Modelling functional units

We have chosen a ‘cluster-based’ model for the FUs, similar to the classical Alpha organisation. Therefore, the ALU is composed of an integer execution unit and a floating point (FP) execution unit. The integer execution unit used is a four-path unit implemented as two FU ‘clusters’. Each cluster includes a copy of an 80-entry, physical-register file and two ‘sub-clusters’ or sub-paths. Additionally, a cross-cluster bus exists for moving integer result values between clusters when necessary. The integer unit contains four 64-bit adders, with one on each path, which calculate results for integer-add instructions, two adders to generate the effective virtual address for load and store instructions, four logic units, two barrel shifters and associated byte logic, two sets of conditional branch logic, two copies of an 80-entry register file, one pipelined multiplier and a fully pipelined unit that executes special instructions.

4.2 Power modelling and simulation environment

In order to be able to simulate the aforementioned architecture of FUs in a cycle-accurate way and with a complete power model, we have modified and extended the SimpleScalar simulator (sim-out-order mode) and the Wattch power modelling framework. These two tools together create a complete simulation framework that can accurately quantify potential power savings. In fact, this framework includes four different simulation models depending on the clock gating strategy of the microarchitecture of the processor. Among them, we have chosen cc3, because it performs the most realistically for current high-performance Deep Sub Micron (DSM) processors. This model computes 100% of the power of a block when it is accessed and 10% otherwise, to estimate static power contributions.

In this work, we have improved the relative accuracy of Wattch for FUs in the following ways:

- Wattch, in terms of power consumption, just considers the integer execution unit as an adder. In order to analyse a real system, we have included more types of FUs. As a result, the new version models the power estimation of the integer execution unit as the addition of the power consumption of an adder, a shifter unit, a specific unit for logic operations and a multiplier.
- Similarly, the power consumption of floating point execution unit is now modelled as the addition of the power consumed by an FP adder and an FP multiplier.
- We have separated power consumption into two components: $P_{\text{dynamic}}$ and $P_{\text{static}}$. The ratio between static power and dynamic power is set to 33% in our experiments, according to the predicted estimations for the 90-nm technology node in the Semiconductor Technology Roadmap [19].
- We have extended the clock gating scheme to be able to turn on/off each FU separately. For example, if a logic
instruction is being executed, the logic unit is active and therefore consumes static and dynamic power, whereas the integer and FP units are inactive and therefore only contribute to static power.

4.3 Estimation of power consumed by adders

In order to model the effect of using two different kinds of adders (32-bit and 64-bit adders), the 64-bit adder is assigned a baseline power consumption value (i.e. $P_{A64}$) and the 32-bit adder power consumption ($P_{A32}$) is scaled down by a certain ratio. We have included two different ratios ($P_{A32}/P_{A64}$) in our experiments, namely 0.5 and 0.33. These values are based on the fact that 32-bit adders approximately dissipate half the energy of a 64-bit one, because of the fact that 32-bit adder have half the amount of switching, which reduces the dynamic power consumption, and half the amount of transistors which reduces the static power consumption. Therefore, we can use a ratio equal to 0.5.

Also, 32-bit adders are faster than 64-bit adders [20–22]. However, we do not need to use the fastest 32-bit adder. In order to further reduce power consumption, we can choose an adder which has a similar critical path to the 64-bit adder. This allows us to reduce timing requirements in the 32-bit adder synthesis. Moreover, it provides a margin in its synthesis to reduce power consumption by applying two techniques. First, we can reduce the area, and obviously, the number of gates and transistors in the design. Second, we can also resize the transistors and/or replace some of its gates by others with low-driving capabilities. Thus, the power consumption per gate transition is reduced [1, 23, 24].

Applying these techniques, we have estimated that a 32-bit adder could consume approximately 33% of the 64-bit adder power consumption [25]. So, we can use a ratio equal to 0.33.

4.4 Processor model

As mentioned, we used a real architecture model. The processor model simulated (Table 2) can fetch four instructions and issue six instructions (four integer, two FP, by cycle and out-of-order). The execution unit has four integer ALUs, one integer multiplier, one FP adder and one FP multiplier. As Fig. 2 indicates, the baseline configuration has four integer ALUs, each of them has a 64-bit adder, a shifter unit and a specific unit for logic operations. In addition, there is a pipelined integer multiplier, a floating point adder and a floating point pipelined multiplier.

We have tested four additional execution unit configurations in order to test our proposal in terms of power consumption. In the rest of the configurations, we have only changed the number and type of integer adders. In fact, all tested configurations have at least the same number of adders as the baseline configuration in order to not to modify the performance of the processor. The different tested configurations are the following:

- Baseline: four 64-bit adders.
- Configuration 1 (Conf-1): three 64-bit and one 32-bit adders.
- Configuration 2 (Conf-2): two 64-bit and two 32-bit adders.
- Configuration 3 (Conf-3): one 64-bit and three 32-bit adders.
- Configuration 4 (Conf-4): one 64-bit and four 32-bit adders.

Table 2: The Baseline configuration of simulated processor

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor core</td>
<td></td>
</tr>
<tr>
<td>ROB</td>
<td>80</td>
</tr>
<tr>
<td>Int queue size</td>
<td>40</td>
</tr>
<tr>
<td>F.P queue size</td>
<td>30</td>
</tr>
<tr>
<td>LSO (ld/store queue)</td>
<td>Size 32</td>
</tr>
<tr>
<td>Fetch queue size</td>
<td>Four instructions</td>
</tr>
<tr>
<td>Fetch width</td>
<td>Four instructions/cycle</td>
</tr>
<tr>
<td>Decode width</td>
<td>Four instructions/cycle</td>
</tr>
<tr>
<td>Issue width</td>
<td>Six instructions/cycle (out-of-order)</td>
</tr>
<tr>
<td>Commit width</td>
<td>Four integer, two floating point</td>
</tr>
<tr>
<td>Functional units</td>
<td>Four Int ALU, one Int Multiplier, one FP adder and one FP Multiplier</td>
</tr>
<tr>
<td>Branch prediction</td>
<td>Bimodal predictor: table size 4K combining two-level predictor l1 size 1K; l21 size 1K; hist-size 3 meta_table_size 4 K</td>
</tr>
<tr>
<td>BTB</td>
<td>num_sets 512; associativity 4</td>
</tr>
<tr>
<td>Return-address stack</td>
<td>32-entry</td>
</tr>
<tr>
<td>Mispredict penalty</td>
<td>7 cycles</td>
</tr>
<tr>
<td>Memory hierarchy</td>
<td></td>
</tr>
<tr>
<td>L1 data-cache</td>
<td>dl1:512:64:2:L; lat 3</td>
</tr>
<tr>
<td>L1 instruction-cache</td>
<td>l1:512:64:2:L; lat 2</td>
</tr>
<tr>
<td>L2 Unified</td>
<td>u2: 32768:64:1:L; dl2lat 6; il2lat 12</td>
</tr>
<tr>
<td>TLBs</td>
<td>dtlb:1:8192:128:l lat 50</td>
</tr>
</tbody>
</table>

In the baseline configuration, all instructions that require an adder are executed in a 64-bit adder. The rest of the tested configurations exploit our proposal for those instructions that can be executed in a 32-bit adder instead of a 64-bit one. The number of instructions that use 32-bit adders depends on the amount of available 32-bit adders, and the benchmark. The primary metric utilised here is ‘time-averaged’ power. This metric represents a single value for power consumption, obtained by integrating power on a cycle-by-cycle basis over a number of clock cycles. Then, average power is used to calculate battery life or junction temperature; thus, it is the most widely

![Fig. 2 Baseline configuration of simulated processor](image-url)
used way of estimating power \[5\] and is used in our experiments here. In fact, it is similar to the measurement total power, but instead of adding all the values, it is obtained by averaging power over time.

5 Experimental results

As explained, five different configurations of the execution unit have been tested, in order to find which one obtains a higher reduction of power consumption while always preserving the performance of the processor. We have simulated the SPEC CPU2000 benchmarks set with the objective of evaluating our approach. In order to reduce the duration of the simulations we have simulated sets of 100M instructions called ‘Simulation points’ [26]. Simulation points are a small set of execution samples that when combined represent the complete execution of the program.

Fig. 3 shows the percentage of instructions that are executed in a 32-bit adder for every benchmark and for Conf-1, Conf-2, Conf-3 and Conf-4. It indicates that the number of accesses to 32-bit adders increases with each configuration, and Conf-4 shows a number of accesses to 32-bit adders very similar to the predicted figures. In fact, in this configuration all instructions that can use a 32-bit adder can find one available as the issue width is four instructions per clock cycle and this configuration has four 32-bit adders.

On the other hand, in order to accept any change on the execution unit configuration, it is necessary to assure that the performance does not degrade. We have measured performance (instructions per cycle or IPC) for all benchmarks with the five different proposed configurations. We observed that our technique does not affect the processor performance in Conf-1 and Conf-2. However, some benchmarks experience performance loss if only one 64-bit adder exists (i.e. Conf-3 and Conf-4). Overall, the averaged IPC fall lies between 0% and 1.9% (0.9% on average), depending on the configuration and benchmark used.

We have also measured static power consumed in the adders for all configurations, using different ratios \(P_{S32}/P_{S64} = 0.5\) and 0.33. Obviously, results indicate that the amount of static power consumed does not depend on the benchmark; it only depends on the amount and type of adders. All configurations have less static power consumption than the baseline configuration \(P_{static} = 1.16\) because we have reduced the number of 64-bit adders. For example, we have reduced the number of 64-bit adders by half in Conf-2, and by a fourth in Conf-3 and -4. As a conclusion, regarding static power, in all cases Conf-3 is the best configuration, as it has the minimum number of 64-bit adders and the minimum number of other adders.

Let us now analyse dynamic power consumed by the adders (in Watts) for every configuration and benchmark, using different ratios. Experimental results show that the baseline configuration has the highest dynamic power consumption, because all instructions that require an adder are executed in a 64-bit adder. In the rest of the configurations, dynamic power consumption is lower than the baseline configuration. This is because of the fact that some instructions are executed in a 32-bit adder instead of a 64-bit one.

![Fig. 3 Percentage of instructions that are executed in a 32-bit adder for every benchmark and for Conf-1, Conf-2, Conf-3 and Conf-4](image)

![Fig. 4 Total power consumption in the adders (W) for all configurations in every benchmark, using the ratio between 64-bit and 32-bit adders equal to 0.33](image)
Conf-4 is the best configuration in terms of dynamic power consumption. The reason is that all instructions that can use a 32-bit adder can find an available one, as already mentioned. However, a detailed inspection reveals that configurations Conf-3 and Conf-4 have very similar results. This is because of the number of instructions using a 32-bit adder being only slightly different in both configurations.

Fig. 4 illustrates the total power consumption in the adders. Our results show that for every configuration, ratios and benchmarks, our technique reduces the total power consumption of the adders. Conf-3, in terms of total power consumption, is the best because it has the smallest static consumption and its dynamic consumption is very similar to Conf-4.

Average power saved in the adders used are between 14.7% for Conf-1 and 50% for Conf-3. According to our experimental results, Conf-3 and Conf-4 present quite similar results. The difference between these configurations is that Conf-4 has an extra 32-bit adder. This adder contributes more static power consumption than the amount of dynamic power consumption it saves. In fact, as static power consumption contribution is becoming more important, its effects will be even larger [13, 25].

Fig. 5 shows the power saved in the execution unit using our optimisation. These results incorporate power consumed because of the extra logic used to select the adder. It shows power consumption because of the selection logic we have used to select the adder and the power consumption of the original selection logic. We have estimated power consumption of the arbiter using the Wattch model for a similar structure and adding our logic.

Our results show how our technique reduces total power consumption of the whole execution unit. In summary, the energy savings in the execution unit are between 6.1% and 21.1%.

In addition, we have measured the penalty in the performance of the processor (i.e. ‘execution efficiency’) for our particular microprocessor structure. This metric is defined as performance (i.e. IPC) divided by the power consumed by that structure. This does not necessarily correspond to chip-level performance/energy ratios, which are important in a battery-based environment, but they do reflect the usefulness of our technique to improve the performance/power density ratio for a functional block which is a potential hot spot. This metric is appropriate assuming that the power density of the targeted structure is a constraint on the overall design, and is similar to a cache performance study that assumes the cache sets the cycle time of the processor [27]. Therefore, the optimisation achieving the best performance measuring component power ratio, and reducing power density to acceptable levels would represent the best design. We can assert that all configurations (especially Conf-3 and Conf-4) can execute instructions using half of the power of the baseline configuration.

6 Conclusions

This study has presented a hardware technique to reduce the static and dynamic power consumption in FUs of a 64-bit superscalar processor by performing narrow operands detection by relying on the opcode and few bits of one of the operands, instead of exhaustively checking both source operands as previous works suggested. We have proposed an approach that replaces some of the 64-bit power-hungry adders by 32-bit lower power-consumption adders, and modifies the arbiter protocol to issue as many instructions as possible to those low power consumption units.

In our results, we have tested five different configurations for the execution unit in order to find which one obtains a higher reduction on power consumption while preserving the performance of the system. In order to model the effect of using two different kinds of adders (32-bit and 64-bit adders), we have assigned to the 64-bit adder a baseline power consumption value and the 32-bit adder power consumption is scaled down by a ratio. We have used two values for the $P_{A_{32}}/P_{A_{64}}$ ratio: 0.5 and 0.33.

Fig. 6 summarizes results of power consumption in the adders. It shows how our technique reduces total power of the adders in all cases. The Conf-3 option obtains the best
results, reducing the total power consumption in adders between an 14.7% and a 50%. On the other hand, our technique also reduces the power in the execution unit. This reduction is between 6.1% and 21.1% of power consumption in the execution unit.

On the other hand, with Conf-3, we can save up to 2.5% of total power in the processor, averaged over all benchmarks and for the different ratios with our optimisation. Although total power is important by itself, and this value seems to be very small, the hot spots are also important. If too much power is used to achieve a target delay, hot spots may be created. Therefore, we must also bear in mind reduction of power in FUs to avoid hot spots, even if the reduction of total power is not too high. In this line, our technique decreases the transistor number in the adders and hence mitigates the hot spot problem in the functional units of the processor.

We are working on applying these techniques to other parts of the processors and as well as studying other configurations.

7 Acknowledgments

This work has been supported by Spanish Government Grant TIN 2005-5619 and Madrid Government Grant UCM-CAM 910467.

8 References

15 Hennessy, J. and Patterson, D.: ‘Computer architecture, a quantitative approach’ (Morgan Kaufmann, 2003)
19 Available at http://www.intel.com/technology/silicon/itroadmap.htm