A Fault-Tolerant Processor Architecture[1]

Abdelmajid Bouajila, Thomas Sommer, Johannes Zeppenfeld, Walter Stechele, Andreas Herkersdorf,
Institute for Integrated Systems
Technische Universität München, Arcisstr. 21, 80290 München, Germany
Email: a.bouajila@tum.de

Kurzfassung

In diesem Artikel wird eine neue Architektur für den Entwurf fehlertoleranter Prozessoren vorgestellt, die sich aus der DIVA Technik ableitet. In DIVA wird ein Prüfmodul unmittelbar vor der Speicherstufe (commit stage) des Prozessors eingefügt. Das Prüfmodul führt sowohl die Berechnungen als auch die Zugriffe auf den Hauptspeicher oder den Registersatz erneut aus. Das ursprüngliche DIVA Prüfmodul wird selbst als völlig zuverlässig angenommen. Wenn es einen Fehler erkennt, korrigiert es ihn, speichert das Ergebnis (d.h. schreibt in den Hauptspeicher oder den Registersatz), leert die Pipeline des Prozessors und startet sie mit dem nächsten Befehl.
In unserer Modifizierten DIVA Architektur nehmen wir nicht mehr an, dass das Prüfmodul völlig zuverlässig ist. Im Falle eines Fehlers wird die Prozessor Pipeline geleert und mit dem fehlerverursachenden Befehl neu gestartet. Aus diesem Grund ist unsere modifizierte Architektur zuverlässiger. Um die Leistung zu steigern werden die Lesezugriffe auf den Hauptspeicher mit ECC geschützt und müssen nicht vom Prüfmodul getestet werden. Somit konkurrieren Prüfmodul und Prozessor nicht um den Speicherzugriff, wie es in der ursprünglichen DIVA Umsetzung der Fall ist. Des Weiteren haben wir das Anwendungsgebiet der DIVA Technik um auf einen Standard Pipelined RISC Prozessor erweitert (die ursprüngliche DIVA Technik war hauptsächlich für superskalare Architekturen gedacht).
Diese neuartigen Verbesserungen der Architektur werden im Folgenden erläutert, mit der ursprünglichen DIVA Technik verglichen und die Ergebnisse der VHDL Umsetzung gezeigt. Für die Auswertung dieser neuen Technik wurden in VHDL Simulationen Fehler injiziert.

Abstract

This paper presents a new architecture for Fault-Tolerant processor design inspired from the DIVA technique [1]. DIVA consists of inserting a checker unit in front of the processor commit stage. The checker unit re-executes both computation and memory/register file reads. Whenever an error is detected, the original DIVA checker which is assumed to be fully reliable fixes the error, then commits results (i.e. writes them to memory/register file), flushes the processor and restarts it at the next instruction.
In our Modified DIVA architecture, we no longer assume that the checker is fully reliable. In case of error detection, the processor is flushed and restarted at the erroneous instruction. Therefore our modified architecture is more reliable. In order to increase performance, we protect external memory reads with ECC, our checker unit does not re-execute them and therefore the checker and processor are no longer competing for memory accesses as was the case in original DIVA.
We have also extended the application of the DIVA technique to a standard RISC pipelined processor (original DIVA was mainly aimed at Superscalar architectures).
These new architectural improvements in comparison to original DIVA are presented in this paper, and VHDL implementation results are reported. Fault injection in VHDL simulations was used to evaluate this new technique.

[1] This work is supported by BMBF project 01 M 3083 “Autonome Integrierte Systeme."

1 Introduction

CMOS technology evolution leads to dense integrated circuits with ever smaller devices. Transistors in future integrated circuits will contain only few atom dopants, which will result in variability increase in the same chip [2]. Hence transistors within the same chip will have different electrical characteristics. Also, as they contain small amount of charges, they are more sensitive to external perturbations such as neutrons and alpha particles. This will increase the probability of timing and soft error occurrence [2] and will make the use of fault-tolerant design no longer necessary solely in critical applications such as avionics and medical, but also in consumer electronics.

Building reliable consumer electronics has tougher cost constraints than in critical applications. So the question has been how we can build fault-tolerant circuits while keeping costs low.

In this paper, we present a low-cost fault-tolerant architecture for processor protection. Subsection 1.1 will present related work. Section 2 will present the original DIVA architecture [1]. Section 3 will present the architecture of our new technique which is inspired from DIVA. Section 4 presents implementation results and an evaluation. Paragraph 5 concludes the paper.

1.1 Related work

In [9], Ernst et al. demonstrate the protection of a CPU pipeline with Razor, a technique for the detection and correction of timing errors. Whenever an error is detected, a one cycle delay is inserted for correction. This technique does not protect the pipeline against transient errors.

Chardonnerau et al. [10] provide both timing and soft-error detection. Their technique is based on Nicolaidis shadow registers [11], and works by adding a second, shadow flip-flop to each main flip-flop that is to be protected. By delaying the clock to the shadow flip-flop, both timing- and soft-errors can be detected. The extra registers are added to the inter-stage registers of the CPU. Error correction was not addressed however. In Bouajila et al. [12], both timing and transient errors are detected. The concept is also based on using Nicolaidis shadow registers [11] as an error monitor, with the addition of a custom micro-rollback implementation [13] for error correction.

2 Original DIVA Architecture

Speculative execution has been used in high performance processors to decrease CPI. For instance, a branch predictor may take wrong decisions but they will be discarded later without register file and memory corruption. DIVA uses such concepts in order to increase processor reliability while keeping overheads low. In DIVA, a hardware checker is inserted in front of the processor commit stage (see Figure 1). The DIVA core (see Figure 1) will compute the instructions and forward in execution order each instruction opcode, its operands and results to the checker, which will re-execute this instruction. In load/store architectures, the instruction results can be either a register update (arithmetic operations, load) or a memory update (store). We say that an instruction has committed its results when it writes them to the register file or memory system.

In case the checker results differ for those forwarded from the DIVA core, an error is reported. **The DIVA checker is simple (no forwarding or prediction logic) so according to [1] it will be assumed reliable,** and its results will be considered as correct. Thus in case of an error detection, the checker results will be committed (written to the register-file/memory).

As stated above, the checker unit receives the instruction opcode, its operands, and its result. We can split the checker verification mechanism into two tasks:

- Verifying the communication part of an instruction: read operands again in RF. Memory loads are re-executed.
- Verifying the computation part of an instruction: arithmetic operations, load/store address generation

In case of error detection, the original DIVA approach will consider that the checker unit is fully reliable and its values will be used to commit the instruction. The DIVA core will be flushed and will be restarted at the next instruction.

High level implementations of DIVA have been reported in [1]. **Main limitations identified by the authors are the fact that the checker competes with the DIVA core on external memory and in register file accesses**. This results in a performance decrease. We should mention that according to our knowledge no full DIVA architecture (integrated core processor and checker unit) register transfer level implementation has been reported so far.

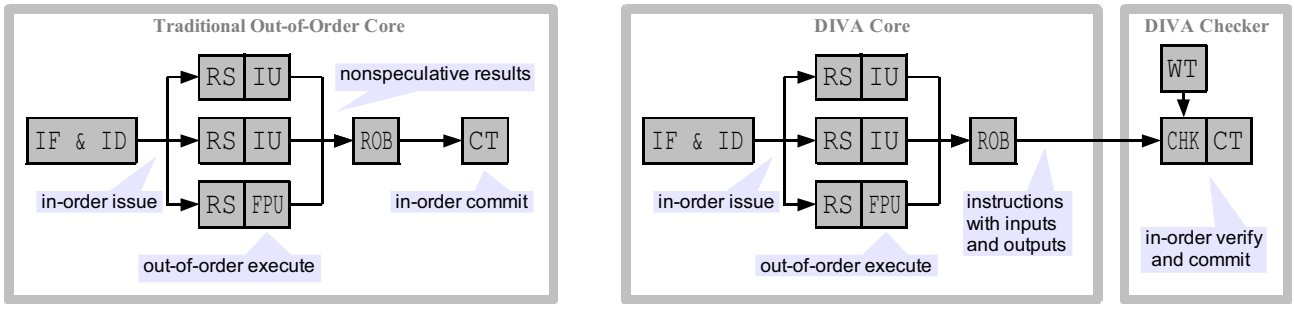

Figure 1: DIVA Concept (DIVA consists of inserting a Checker Unit (CHK) before the commit stage (CT), the processor is unchanged before the re-order buffer (ROB). The processor part till the ROB is unchanged and is called DIVA core (see right figure)

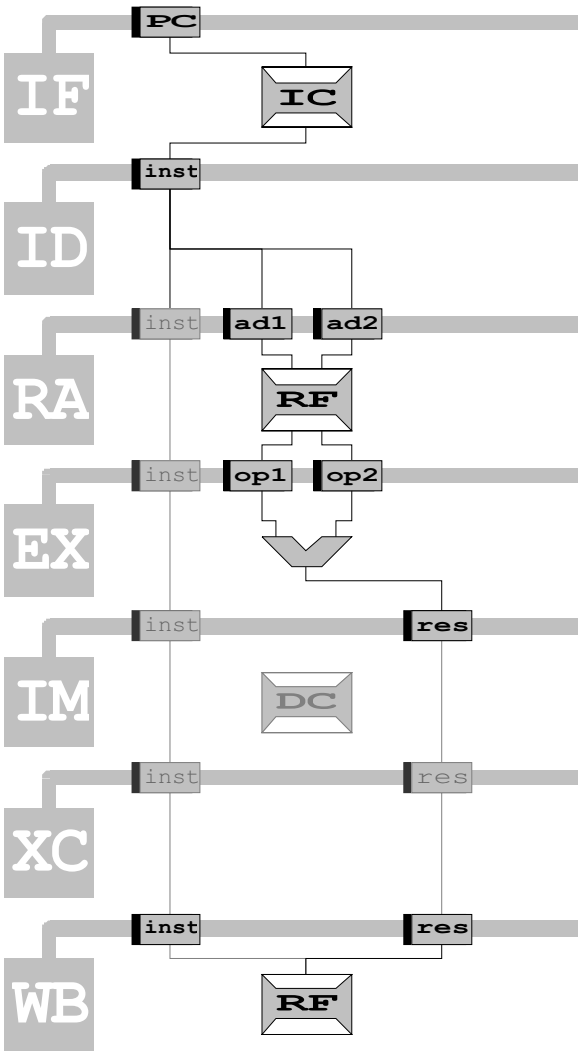

Figure 2: The original 7-stage Leon3 RISC processor.

3 Modified DIVA Architecture

DIVA has good fault coverage (design, soft, timing and permanent faults) while keeping overheads low [5]. We have had ideas to improve both the performance and the reliability of this technique [3]. Also we are presenting architectural investigations that apply the DIVA concept to a standard RISC pipelined processor.

First, we think that it is not realistic that the checker unit is fully reliable as assumed in original DIVA. Data can get corrupted whenever the checker unit is struck by a soft error. We should mention that Austin suggested scaling up checker transistor size to increase their reliability. This is cannot guarantee full checker correctness.

In order to solve this DIVA weakness, we suggest that in case of error detection (DIVA core and checker unit disagreement) both the DIVA core and checker unit must re-execute the errant instruction. This will be implemented by flushing the DIVA core (as done in original DIVA) and re-issuing the errant instruction (in original DIVA, the instruction after the errant one was issued).

Second, in the original DIVA, re-executing communication accesses is done in the checker unit (instruction operands are read from the register file and load transactions are re-executed). **This seems unnecessary as we can guarantee reliable communication with the memory and register file by using techniques such as error correcting codes (ECC). Therefore we suggest that the checker unit re-executes only computation operations and not communication.** This will result in performance increase in our new architecture. In fact, [7] already quantified the performance loss due to communication re-execution in the checker unit.

The DIVA architecture was presented to protect superscalar processors (the overhead of the simple checker is estimated to be a small part of the whole core). Thanks to (modified) DIVA's large fault-coverage [1][7], we think that DIVA is an interesting fault-tolerance technique for RISC pipelined processors. We investigated how to apply modified DIVA to a Leon3 RISC 7-stage pipeline. **The main difficulty was that a RISC pipelined processor has no clear separation between the execution units and the commit stage.** For example in the Leon3 processor, commit stage is implemented in Memory pipeline stage for Memory stores and on the Write-back pipeline stage for register file writes (see Figure 2). A straightforward application of the modified DIVA to Leon3 processor pipeline would mean to postpone stores till the write-back and insert the checker before the write-back.

The re-execution of instructions is achieved in parallel to the exception stage (XC), the comparison of the checker unit's newly computed values with the leon3 values will be done in CP stage (see Figure 2). The re-execution is not done in a separate stage after XC stage, as this would result in the error being detected two cycles after XC stage. This might be too late because an undetected error at XC stage can result on exception occurrence which will change processor state registers and which will be hard to rollback.

4 Implementation

As proof of concept of our modified DIVA architecture, we chose to implement it into a Leon3 processor [4] which is a Sparc V8 compatible RISC pipelined processor. It is a widely used open source soft-core. This will provide a DIVA processor RTL implementation and will demonstrate how DIVA applies to RISC pipelined processors (DIVA was originally suggested for complex superscalar processor protection). Following are the steps needed to implement a (modified) DIVA into an existing processor:

- Identify the commit stage: register file writes and memory writes (stores). Therefore the original processor will be split into commit stage and DIVA core (i.e. a processor without its commit stage)
- Insert a checker unit consisting of re-execution and compare logic
- Implement a rollback mechanism in order to flush the DIVA core and restart it at a certain instruction (errant instruction in case of modified DIVA).

4.1 Leon3 Processor

The Leon3 is a 7-stage pipelined processor. It is similar to the Hennessy and Patterson RISC pipeline [8], but with a decode stage divided into two stages (decode and register access stages). A new stage called the exception stage is added as well, in which traps and interrupts are handled.
The functionality of the resulting 7 stages is as follows [4]:
1 IF (Instruction Fetch): Instructions are transferred from the instruction cache.
2 DE (Decode): Operand addresses are extracted from the instruction word.
3 RA (Register Access): Operands are read from the register file.
4 EX (Execute): ALU operations are performed (including base-offset calculations for memory addresses)
5 IM (Memory): Data is read from or written to the data cache.
6 XC (Exception): Traps and interrupts are resolved.
7 WR (Writeback): The result of the operation is written to the register file.
Most of Leon3 instructions take 1 cycle for excecution. But stores take 2 cycles. Although loads take 1 cycle for execution, they communicate with memory controller during 2 cycles (EX and IM).

4.2 Commit Stage Identification

The commit stage includes the memory stores which are implemented in the Leon3 memory stage (it should be mentioned that the Leon3 pipeline communicates with the memory controller during the execute and the memory).
The commit stage also includes the register file writes (write-back stage).

4.3 Checker Unit Insertion

According to the DIVA design, the checker logic comprises a two-stage pipeline: one stage to redo the instruction (will be referenced as Redo in this paper) and one to compare the checker results to that of the core (Compare stage: CP). The lack of a reorder-buffer in the LEON3 requires fitting the checker stages into the original 7-stage pipeline. We chose the following design:

- An additional stage was inserted between the exception and commit stages. It hosts the compare logic.
- The redo-logic is placed in the exception stage and processed in parallel to the exception logic.

The entire pipeline is thus expanded to 8 stages (see Figure 3). Latency is not affected, as bypasses are provided for register file writes and memory writes were excluded from the commit stage. Error induced exceptions are canceled by the checker and have no visible effect.
The checker requires the intermediate results of the core, i.e. the inputs and outputs of all checkable stages. These values have to be passed on to the checker through the pipeline.

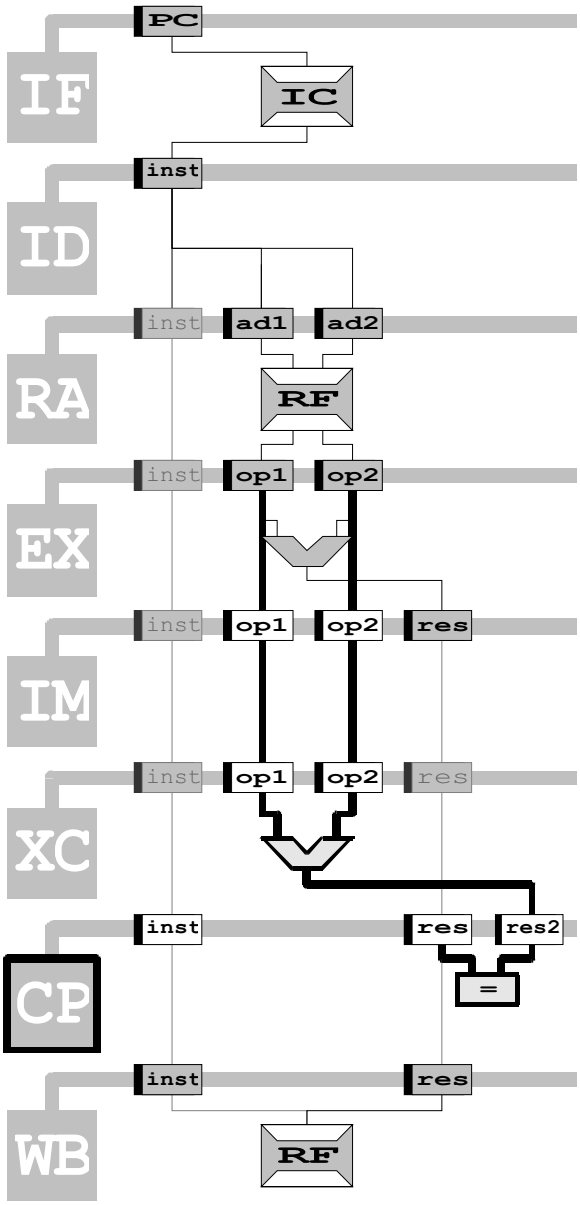

Figure 3: Modified DIVA applied to the Leon3 RISC (Redo is done in parallel with exception stage (XC), and comparison is done in the newly inserted stage CP)

4.4 DIVA Core Rollback

When an error is detected, a rollback procedure is activated. It includes two major steps:

- The errant instruction is not committed, i.e. its results will not be written to register file (or memory in the general case). Likewise, all following instructions in the pipeline will be canceled and prevented from committing. The cancel ensures that all bypasses are deactivated. This procedure is very similar to the pipeline flushing employed when a trap is encountered, so already existing resources in the LEON3 can be reused.
- The pipeline is restarted at the errant instruction by providing its address to the fetch stage. It will traverse every stage of the core and checker a second time.

It should be mentioned that the current VHDL implementation of modified DIVA does not yet include checking for store and load instructions. This is subject to future work.
In the Leon3 processor a store instruction communicates with the memory cache controller for 3 cycles (during the execute, memory and exception pipeline stages). Therefore isolating the store unit and moving it after the checker unit will result in a large design effort. This is not the case in superscalar architectures, where store units are more easily identified.

4.5 VHDL Implementation and FPGA Testing

The new modified DIVA architecture has been implemented in the Leon3 processor pipeline [4]. We injected some errors using Modelsim commands. Errors were detected, the processor flushed and then correctly restarted at the errant instruction.
Exhaustive error injections in VHDL simulations will be reported in the next section. FPGA synthesis has been done using Xilinx ISE 8.2. Tests on a Virtex4 FPGA board have been done with injected errors, which were properly detected and corrected.

5 Results Evaluation

We added a VHDL module which automatically inserts faults in the DIVA core. Simulation was conducted by running a set of Mibench testbenches [6]. **All injected errors were detected and correctly recovered.**
The test runs displayed a constant execution time overhead of approximately 70 percent for an error rate of 10 percent, i.e. one error every ten cycles. For example, the Mibench crc32 test provided an overhead of 74.1 percent for an error rate of 10.6 percent, and an overhead of 0.13 percent for 0.0183 percent error rate. These measurements match theoretical estimations. A detected error requires flushing the core and restarting the errant instruction. This results in an overhead of at least 7 cycles. The minimum can be reached when no extra cache misses are encountered during rollback, as is usually the case (the data is still present 7 cycles later).
As we have do not have an implementation of the original DIVA on a Leon3, and since the reported results on original DIVA were on a superscalar processor, we will make both qualitative and quantitative comparisons where possible. As the modified DIVA has no memory accesses re-execution, it should have better performance than original DIVA, since memory bandwidth is often a major bottleneck in processor based systems. (In [1], 8 cycles are required for error correction in original DIVA, and with an error rate of 0.1 percent, the slowdown was 2.6 percent. With our technique, for a 0.1 percent error rate we only got 0.7 percent slowdown (for Mibench crc_32, Dijkstra and qsort tests). However, please note that this is a very rough comparison, as no in-depth information is provided in [1], and we are comparing a superscalar vs. pipelined processor.)

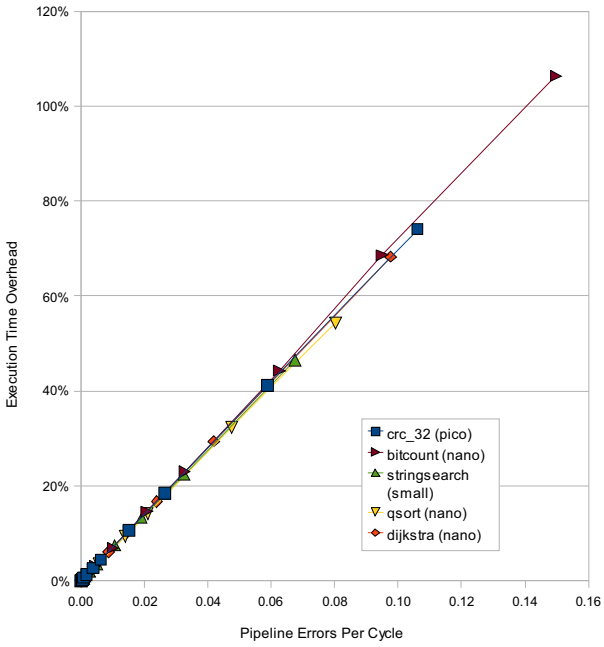

Figure 4: Relative CPI = f (Error rate)

It is worth mentioning that **in error free operation**, Modelsim simulations showed that with all Mibench testbenches, **there are no performance overheads for the modified DIVA processor**. In other words, for a 0 percent error rate, modified DIVA has the same performance as a non-modified Leon3 processor. This is as was expected, since, we have added forwarding paths from the checker unit. **This is better than original DIVA, which has a performance decrease during error free operation** of more than 10% when running the SPEC95 turbo3D testbench [1].

6 Conclusion

In this paper, we have presented a modified version of the DIVA architecture, which avoids the assumption that the checker unit is completely reliable. The architectural aspects for the reliability and performance improvements of this modified DIVA architecture have been examined. Finally, the adaptation of DIVA to a RISC processor pipeline has been presented, and results from a VHDL implementation with automatic fault injections were reported.

7 References

[1] Austin T.: DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design, in MICRO1999

[2] Borkar B.: Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. IEEE Micro 25(6): 10-16 (2005)

[3] Bouajila et al.: Error Detection Techniques Applicable in an Architecture Framework and Design Methodology for Autonomic SoCs, in IFIP BICC 2006, Chile, August 2006

[4] Leon3 processor and datasheet available at Gaisler Research website: www.gaisler.com

[5] Weaver C. et al., A Fault Tolerant Approach to Microprocessor Design: IEEE DSN-2001, June 2001.

[6] Mibench testbenches available at http://www.eecs.umich.edu/mibench/

[7] Chatterjee S. et al.: Efficient Checker Processor Design: ACM/IEEE 33rd International Symposium on Microarchitecture (MICRO-33), December 2000

[8] David A. Patterson, John L. Hennessy: Computer Organization and Design: The Hardware/Software Interface. Third Edition, Elsevier, 2005

[9] Dan Ernest, et al "Razor: A Low-Power pipeline Based on Circuit- Level Timing Speculation", the 36th Annual International Symposium on Microarchitecture (Micro-36), December 2003razorpipeline03

[10] Chardonnereau D, Nicolaidis M: Fault Tolerant 32-bit RISC Processor: Implementation and Radiation Test Results, downloaded on july 2006 from http://www.iroctech.com/pdf/RISC_rad_results.pdf

[11] Michael Nicolaidis: Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer technologies, 7th IEEE VLSI Test Symposium 1999

[12] Abdelmajid Bouajila, Johannes Zeppenfeld et al.: Organic Computing at the System on Chip Level. IFIP VLSI-SoC 2006, IFIP WG 10.5 International Conference on Very Large Scale Integration of System-on-Chip, Nice, France, 16-18 October 2006. IEEE 2006

[13] Tamir et al.: High-Performance Fault-Tolerant VLSI Systems Using Micro-rollback, IEEE Transactions on Computers, Vol. 39, NO. 4, April 1990