Post-Silicon Failing-Test Generation through Evolutionary Computation

Ernesto Sanchez, Giovanni Squillero. Alberto Tonda
Politecnico di Torino
Torino, ITALY
\{ernesto.sanchez, giovanni.squillero, alberto.tonda\} @polito.it

Abstract—The incessant progress in manufacturing technology is posing new challenges to microprocessor designers. Several activities that were originally supposed to be part of the pre-silicon design phase are migrating after tape-out, when the first silicon prototypes are available. The paper describes a post-silicon methodology for devising functional failing tests. Several silicon prototypes are available. The paper describes how to take into account complex hardware characteristics and architectural details of such complex devices. The experimental evaluation clearly demonstrates the potential of this line of research.

I. INTRODUCTION

Nowadays, manufacturing technology is advancing at a faster pace than designing capability, posing unprecedented challenges in the arena of integrated circuits. Due for example to limitations in current timing analysis tools, the comprehensive analysis of a complex chip can only be performed after tape-out. In industrial practice, once manufacturing is completed and first silicon is produced, the early chips are sent back to their design teams. Like the well-known post-silicon verification, several activities that were originally supposed to be part of the pre-silicon design phase are nowadays migrating to the post-silicon time. The cost of manufacturing prototypical devices is indeed enormous, but this practice is not an option. Designers acknowledged that “very few chips ever designed function or meet their performance goal the first time” [1].

Non-deterministic effects, such as manufacturing variability, are posing great challenges to the designers. It has been long known that several physical defects only appear when the device operates at full speed [2], but nowadays also design criticalities become apparent only at high frequencies. Even worse, they appear only occasionally and possibly only in a percentage of the manufactured chips. “Finding the root cause of at-speed failures remains one of the biggest challenges in any high-performance design”, stated Rob Aitken in his editor’s note for [3].

Microprocessors are a paradigmatic example of the current scenario: devices for the desktop market contain billions of transistors, implement quite complex microarchitectures, and operate into the microwave frequency range. Since last decade, desktop microprocessors include hardware support to efficiently execute multiple independent flows of instructions, called threads. Multithreading allows increasing the overall throughput in a multitasking environment, even when it would be impossible to further speed up the single program with a pipeline or other techniques. Simultaneous multithreading, branded as hyper-threading by Intel, enables multiple threads to be executed concurrently exploiting superscalar designs. More recently, in a multicore architecture, or chip-level multiprocessor, two or more independent processing units work side by side packaged in the same chip and sharing the same memory. Indeed, each individual core also exploits simultaneous multithreading.

A speed path is a path that limits the performance of a chip because a faster clock would cause an incorrect behavior with a relevant probability. Speed paths may be the location where potential design fixes should be applied, and may indicate places where potential holes in the design methodologies or manufacturing technologies exist. At design time, the slowest logic path in a circuit is known as the critical path, and it can be quite easily determined. However, in complex multicore/multithreaded designs it has been recognized that critical paths reported from the pre-silicon timing analysis tools poorly correlate with the actual speed paths. The reason is that obtaining accurate models for nanometer processes is difficult, if not nearly impossible. Analysis algorithms are also approximated and oversimplified because of the complexity involved. Finally, timing behavior on the silicon is a result of several factors mingled together, and during the pre-silicon analysis it is not feasible to consider multiple factors simultaneously [4] [5] [6].

To meet today’s performance requirements, the design flow of a modern microprocessor goes through several iterations of frequency pushes prior to final volume production. Such a process is called speed stepping. The identification and the debug of speed paths is an essential part of speed stepping. A failing test is defined as a sequence of operations that uncovers an incorrect behavior when run at high frequency. Failing tests may be, for example, sequences of inputs to be applied to the microprocessor pins by an automatic test equipment (ATE). In industrial practice, such tests are crafted by engineers starting from the pre-silicon verification test suite; generated by pre-silicon specialized tools, or automatic test pattern generators (ATPGs); or created on silicon, tackling the actual devices [7] [8].

Interestingly, the instruction sets of microprocessors have been successfully exploited to tackle path-delay faults [9] [10].
The underlying idea of these works is that executing a set of carefully designed programs may effectively uncover timing issues. The strengths of the methodology are that the execution of such test programs is per se at-speed and requires no additional hardware, or complex and expensive ATEs. No attempts, however, have been reported to devise failing tests directly at the instruction level. No one has yet proposed a post-silicon methodology able to automatically generate a test program that stresses a speed path causing a detectable failure.

A software-based speed-path failing test is defined as an assembly-language program that produces the correct result only while the microprocessor operating frequency is below a certain threshold. As soon as the frequency is pushed above the threshold, the result yielded by the program becomes incorrect. Let us denote the threshold for a given program as its functional frequency threshold, because the incorrect behavior is functionally observable. That is, it can be theoretically detected by observing the values stored in the main memory and registers. Clearly, the diagnostic capability of a software-based speed-path failing test increases as its functional frequency threshold decreases. A test that produces a failure at a relatively low frequency is preferable to a test that fails only at very high frequencies.

This paper shows for the first time a fully-automatic automatic methodology for devising functional speed-path failing tests. It demonstrates that highly effective software-based tests could be generated directly on-silicon exploiting an evolutionary algorithm. The result advocates for the exploitation of the methodology inside the manufacturer’s facility during the speed stepping phase.

Sections 3 and 4 describe the proposed methodology, detailing the adopted evolutionary algorithm. Sections 5 illustrates the feasibility study and report the obtained results. Section 6 concludes the paper, sketching the future directions of the research.

II. GENERATION AND EVALUATION OF TEST PROGRAMS

The proposed approach for generating software-based speed-path failing tests is feedback-based. Candidate test programs are created without a rigid scheme, and evaluated on the target microprocessor. The data gathered are fed back to the generator and used to generate a new, enhanced set of candidate solutions. The process is then iterated.

To exploit a feedback-based mechanism it is required to evaluate the goodness of each candidate test. As stated before, a software-based speed-path failing test is as good as it fails at low frequencies, and the key parameter in evaluating a test is its functional frequency threshold. However, it should not be forgotten that variability vexes verification engineers. A failing test may not fail always at the same frequency, even if all controllable parameters are exactly reproduced. The variability of speed paths may be caused by non-deterministic factors, such as noise, die temperature or small fluctuation in the external power. Some design criticalities may appear only under particularly unfavorable conditions. All experiments need to be repeated at least several times, when not on different devices.

Consequently, besides the lowest functional frequency threshold detected amongst the repeated experiments, an additional parameter in evaluating a test is the percentage of runs that actually failed at that frequency. It is intuitively plausible that a test failing half of the times at a certain frequency is more useful that a test that fails only once every thousands experiments.

Changing the operating frequency of a microprocessor, however, is not an easy task. To ensure proper synchronization between all the components of the system, only a very limited set of operating clock speeds are available to the end users. While the microprocessor is connected to an ATE after production, such an evaluation is perfectly feasible. However, outside manufacturer laboratories the large steps in frequencies would likely impair the overall usability. Notably, outside manufacturer laboratories, the final aim would hardly be speed stepping. Conversely, end users may be quite interested in performing an incoming inspection on purchased devices. Tackling this latter goal, this paper shows how to adapt the methodology in order to require no test equipment and no additional hardware whatsoever.

The architecture of modern microprocessors includes dynamic performance scaling technologies. Intel branded it as SpeedStep. Similar mechanisms are available as Advanced Micro Devices PowerNow! and Cool'n'Quiet, or VIA Technologies LongHaul. Such technologies are designed to save power and reduce heat, thus they only allow decreasing the operating frequency and the power supply voltage supplied to the microprocessor. The latter is known as undervolting.

Roughly speaking, desktop microprocessors are made using the complementary metal-oxide-semiconductor (CMOS) technology, based on field-effect transistors (FETs). In such devices, reducing the voltage increases the time required to switch between logic values [13]. Increasing frequency and reducing voltage involves significantly different phenomena in the physical world, especially where not all paths have the same Vcc sensitivity or where paths are interconnect dominated. However, the effects of reducing voltage may be related to the effects of increasing the operating frequency. As a matter of fact, whenever a microprocessor is undervolted, its operating frequency is also reduced to guarantee proper functionalities.

Manufactures define sets of safe operating states, sometime called performance states or p-states. While the exact meaning of these p-states is implementation dependent, P0 is always the highest-performance state, with the following P1 to Pn being successively lower-performance and less-consuming states. Thus, to test the methodology, the behavior of a microprocessor could be analyzed intentionally outside the predetermined p-states. Let us define the functional core voltage of a failing test as the lower voltage required not to fail the test at a given operating frequency. Conversely to functional core frequency, a failing test is as good as its functional core voltage is high. That is, all tests would fail with a very low core voltage, but only the interesting ones truly require full power.

Thus, for the sake of a feasibility study, the evaluation of a candidate test could be based on its functional core voltage, and
on the percentage of runs that actually failed. It must be noted that there is no conceptual difference between overclocking and undervolting from the point of view of the proposed algorithm.

III. EVOLUTIONARY CORE

The optimizer used in the feedback approach is an evolutionary algorithm, that is, some of its internal mechanism loosely mimic principles of the Neo-Darwinian paradigm, namely variation, inheritance, and selection. The toolkit exploited in this work is called µGP (MicroGP) [14], available under the GNU Public License from Sourceforge.

Since the toolkit has already been used in several works, its description is out of the scope of this text. However, it could be useful to remind that programs are encoded as directed multigraphs. During the optimization process, a test program undergo several types of modifications that ape both sexual and asexual reproduction. For example one or more instructions can be added; one or more instructions may be removed; the operands of certain instructions can be modified. µGP also mimics sexual reproduction, thus new programs may be obtained by mixing, in different ways, existing ones. The multigraph representation helps ensuring that the result of the crossover is still a sensible program, resembling to both parents and, thus, inheriting potentially good characteristics from both of them.

For the core can be simply used out of the box, the efficacy of an evolutionary algorithms depend on other factors. The two most important one are: what feedback is used to evaluate candidate solutions, termed fitness by the evolutionary algorithm scholars; what is encoded inside individuals. While exploiting an evolutionary approach is per-se of little interest, selecting and tuning such elements can effectively enable to find a solution.

A. Fitness Function

The fitness function must not only be able to evaluate candidate solutions, but also be able to rank them, identifying the more promising ones. In these kind of algorithms, candidate solutions are optimized through the accumulation of slight but useful variations [15]. Thus, if all individuals are indistinguishable because they have the same fitness, an evolutionary algorithm is completely useless.

The first and most important component of the fitness is simply the functional voltage threshold. The second is the number of failures detected over the \( R \) repetitions at the maximum voltage.

Similarly to software-based self test [16], candidate test programs include a mechanism that helps checking their own correctness: all the results of the calculations performed by the test program are compacted in a single signature using a hash function. The evaluator first runs the test program in safe conditions, i.e., at full power, and store the signature. Then it runs the program again at decreasing CPU core voltages, checking that the signature is not modified. As soon as a difference is detected, the functional voltage threshold is recorded. The whole process is repeated \( R \) times to tackle variability.

It must be stressed out that the actual result of the calculations is of no interest, the only relevant detail being that it changes when the test is executed undervolting the CPU below the functional core voltage.

Operatively, µGP creates assembly functions that are assembled and linked with a manager module. These functions contain a loop that execute \( L \) times a set of instructions. The instructions themselves are devised by the evolutionary core, while the framework is fixed. At the end of the loop, before the next iteration, the values in the registers are used to update the signature.

During the experiments the system frequency is first increased using the so-called overclocking features of modern main boards. An excessive increase of the frequency may cause overheating or otherwise irreparably damage the microprocessor, but increasing it slightly is usually perfectly safe. Then the evaluation is performed by reducing the core voltage, only.

B. Internal Representation, Multithreading and Multicore

The internal representation is another key aspect. The evolutionary algorithm must be given the opportunity to generate useful solutions.

Modern processors may implement a multithreaded design; or they can exploit a multicore architecture; or even both. From the perspective of the test-program generator details are not relevant, but it is vital to create multiple independent instruction flows. A single individual is composed of different independent functions. The manager activates them as different threads on different cores using appropriate operating system calls, or directly when no operating system is used. Such blocks, in the individual, are represented as disjoint subgraphs. Notably, different blocks may be forced to have different structural characteristics, or use different subsets of instructions.

For the generation of failing test is performed during the speed stepping or incoming inspection, it is essential to test all possible instructions, and especially the newest. The assembly instructions made available to µGP can be divided in three main classes.

Integer instructions include all usual instructions, such as logical and arithmetical ones. They operate on internal registers or memory. In the adopted scheme, only two registers are employable, while the others are used by the manager. However, this restriction should not impair the global result. Comparisons, tests and branches are also included in this class. To avoid endless loops, µGP was forced to create only forward branches in the generated code.

x87 instructions are the subset of the Intel 32-bit architecture (IA32) related to the floating point unit (FPU). The name stems from the old separate floating point coprocessors, like 80287 and 80387. They provide single precision, double precision and 80-bit double-extended precision binary floating-point arithmetic according to the IEEE 754-1985 standard. x87 instructions operates on a stack of eight 80-bit wide registers,
but some instruction modifiers allow the use of the stack as a set of registers. In the actual version, µGP uses x87 instructions in only one thread.

The third class of instructions requires a slightly longer introduction. In 1996, Intel introduced single-instruction/multiple-data (SIMD) instructions in the Pentium microprocessor, its first superscalar implementation of the x86 instruction set architecture. In a SIMD instruction, multiple processing elements perform the same operation simultaneously on different data. Matter-of-factly, the technique is called data-level parallelism. Pentium SIMD instructions were originally branded as MMX extension, and operate on eight 64-bit wide registers. Advanced Micro Devices offered its own enhanced version of the SIMD instructions two years later, marketing them as 3DNow!. In 1999, Intel outbid with the so-called Streaming SIMD Extensions (SSE). Followed in 2001 by SSE2; in 2004 by SSE3; in 2006 by the Supplemental Streaming SIMD Extensions 3 (SSSE3, with three “S”), and eventually by SSE4. Advanced Micro Devices is planning to include SSE5 in its Bulldozer processor core in 2011.

Not surprisingly, SIMD instructions are particularly critical during speed stepping. The complex calculations involved by these instructions cause data to go through several functional units, and the resulting datapaths are prone to be source of problems when the operating frequency is increased.

Cache memories must be taken into account as well, since there may be a significant difference in performance and power consumption between a L1 cache hit and a L1 cache miss. In order to give µGP the possibility to generate cache hits and cache misses, a special set of C variables was defined. The variables are carefully spaced so that all their memory locations will be cached in the very same cache location. If the microprocessor uses a k-way set associative L1 cache and \( C > k \), a shrewd sequence of read and write operations on such variables may generate the desired cache activity.

It must be noted that the goal of adding such variables is to let the evolutionary core to control the cache activity, but no suggestions are given on how to exploit them. µGP would find autonomously which sequence of operations is more useful to generate a failing test.

IV. EXPERIMENTAL EVALUATION

While no working methodology for functional failing-test generation has been reported in the specialized literature yet, a related problem is faced by a community of computer enthusiasts. Overclockers try to push the performance by increasing the operating frequencies of their microprocessors and the CPU core voltages [17]. However, after pushing their computers to astonishing frequencies, they need to assess the stability of their systems. The test suites that are used to stress the systems and highlight criticalities may be regarded as generic fail tests not focused on a specific microprocessor. Thus, they can be used as a baseline to evaluate the performances of the proposed methodology.

While all the stability tests are quite different, a common point is that modern ones do extensive SIMD calculation. Another common point is their ability to increase the temperature of the microprocessor. It is well known that high temperature may cause both reversible and irreversible effects on electronic devices. Heating may increase the skew of the clock net and alter hold/setup constraints, causing design criticalities to become manifest and the circuit to operate incorrectly [18].

However, while such an effect is sensible when assessing the stability of a system, it may not be desirable when the goal is to find a failing test during speed stepping. The main reason is that the failing test should be as repeatable as possible, while increasing the temperature also increase non-deterministic phenomena. Nevertheless, since no other comparison is possible, the proposed approach was tested against the state-of-the-art stress tests used by the overclocking community.

A. Overclockers’ Stress Tests

Most of the information about stability stress tests is available through forums and web sites on the internet, with few or none official sources. However, there is quite a generalized agreement in the overclockers community on these tools.

SuperPI is a version of the program used by Yasumasa Kanada in 1995 to compute \( \pi \) to 232 digits. It is based on the Gauss–Legendre algorithm. SuperPI implementation makes use of x87 instructions only, it exploits no SIMD instructions, and it is strictly single threaded. CPU BurnIn is a stress test developed by Michal Mienik in the beginning of 2000s. Like SuperPI it uses no SIMD instructions and is single threaded. These two programs are rather old, but have been included for the sake of comparison.

Prime95 is the name of an application written by George Woltman and used by a project for finding Mersenne prime numbers\(^2\). It makes extensive use of the fast Fourier transform, or FFT, with a highly efficient implementation that exploits SIMD instructions. Over the years, it has become extremely popular among overclockers as a stability test. It includes a “Torture Test” mode designed specifically to test systems and highlight problems. In the overclocking community, the rule of thumb is to run it for some tens of hours.

LINPACK is a software library for performing numerical linear algebra on digital computers. It was originally written in Fortran in the 1970s and early 1980s. Newer implementations of LINPACK exploit SIMD instructions and are highly optimized. Significantly, Intel includes a benchmark based on an optimized version of LINPACK in its Math Kernel Library\(^3\). Different applications exploited such benchmark to assess the stability. The most common are LinX\(^4\), IntelBurnTest\(^5\), and OCCT\(^6\). The last one also includes a proprietary stress test.

---

2 A Mersenne number is a positive integer that is one less than a power of two: \( M = 2^p - 1 \). The name came from the French theologian, philosopher, mathematician and music theorist Marin Mersenne, sometimes referred to as the “father of acoustics”. As of December 2010, only 47 Mersenne prime numbers are known. Remarkably, the largest known prime number is also a Mersenne number: \( N = 2^{43,112,609} - 1 \). For more information see “The Great Internet Mersenne Prime Search” at http://www.mersenne.org/.


4 Originally posted on http://forums.overclockers.ru/
B. Target Systems

Experiments were run on two different systems. In both cases, the only non-standard devices were in-house manufactured water cooling systems.

The first benchmark is an Intel Pentium Core 2 Duo E2180 on a MSI motherboard NEO2-FR with the Intel chipset P35. The system was equipped with 3 Gb RAM memory DDR2-800, and a Sparkle Nvidia 8800GT graphic card. While the default clock was 2GHz, for the purpose of the experiments the system was overclocked to 2.93GHz.

<table>
<thead>
<tr>
<th>TABLE I. µGP PARAMETERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>( \mu )</td>
</tr>
<tr>
<td>( v )</td>
</tr>
<tr>
<td>( \lambda )</td>
</tr>
<tr>
<td>( R )</td>
</tr>
<tr>
<td>( L )</td>
</tr>
<tr>
<td>( C )</td>
</tr>
</tbody>
</table>

The Intel Pentium Core 2 Duo E2180 is based on the Core architecture. Its design can be traced back to the P6, introduced in 1995 with the Pentium PRO and revived in 2000 with the Pentium M line. It supports SIMD instructions up to SSE3 and SSSE3, and the Enhanced Intel SpeedStep (EIST) technology. Unlike its predecessor NetBurst and its successor Nehalem, the Core 2 Duo architecture does not exploit simultaneous multithreading. The E2180 is a dual-core microprocessor. Each core has two separate 32 KiB L1 caches for data and instructions, both implementing an 8-way set associative architecture. Each core has also an L2 cache of 1 MiB, 8-way set associative that is used for both data and instructions.

The second benchmark is an Intel Pentium Core i7-950 on an ASUS motherboard Rampage III Extreme with the Intel chipset X58. The system was equipped with 6 Gb RAM memory DDR3 1600 MHz, and a Radeon HD 5870 graphic card. The default clock ranges between 3.06GHz and 3.48GHz, for the purpose of the experiments the system clock was fixed to 3.82 GHz (166MHz x 23).

The i7-950 is based on Nehalem architecture, the successor of the Core architecture. It supports the SSE 4.2 instructions, adding 7 new instructions to the SSE 4.1 set available in the Core 2 series. The i7-950 is a quad-core microprocessor, able to run up to 8 threads with simultaneous multithreading. Each core has two separate 32 KiB L1 caches for data and instructions, both implementing an 8-way set associative architecture. Each core has also an L2 cache of 1 MiB, 8-way set associative that is used for both data and instructions. There is an additional 8 MiB L3 cache, 16-way set associative that is shared by the 4 cores using a design branded as Intel smart cache.

C. Experimental Results

The failing test devised by the proposed approach on the target system was compared with the state-of-the-art stress tools used by the overclocking community. Results are reported in Table II and Table III. Columns are labeled with the name of the program used to test the system. The last column reports data of the test generated by µGP. Rows indicate the CPU core voltage at which the experiments were run. Cells show the time required for the given stress test to report a failure. To reduce overheating effects, all tests were stopped after 10 minutes. Thus “more than 10 minutes” means that no failure has been detected. Almost instantaneous failures are reported as “less than 1 second”. All experiments have been repeated 10 times. µGP parameters are shown in Table I.

<table>
<thead>
<tr>
<th>TABLE II. Failing-test required time for E2180</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE V</td>
</tr>
<tr>
<td>1.2625</td>
</tr>
<tr>
<td>1.2750</td>
</tr>
<tr>
<td>1.2875</td>
</tr>
<tr>
<td>1.3000</td>
</tr>
<tr>
<td>1.3125</td>
</tr>
<tr>
<td>1.3250</td>
</tr>
</tbody>
</table>

Table II and table III on the other side, report the comparison against newer stress tests. For the Intel Pentium Core 2 Duo E2180, all programs use two threads, that is, one for each core. For the Intel Pentium Core i7-950, all programs use eight threads, that is, two for each core.

<table>
<thead>
<tr>
<th>TABLE III. Failing-test required time for i7-950</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE V</td>
</tr>
<tr>
<td>1.21250</td>
</tr>
<tr>
<td>1.21875</td>
</tr>
<tr>
<td>1.22500</td>
</tr>
<tr>
<td>1.23125</td>
</tr>
<tr>
<td>1.23750</td>
</tr>
<tr>
<td>1.24375</td>
</tr>
<tr>
<td>1.25000</td>
</tr>
<tr>
<td>1.25625</td>
</tr>
<tr>
<td>1.26250</td>
</tr>
<tr>
<td>1.26875</td>
</tr>
<tr>
<td>1.27500</td>
</tr>
<tr>
<td>1.28125</td>
</tr>
<tr>
<td>1.28750</td>
</tr>
<tr>
<td>1.29375</td>
</tr>
<tr>
<td>1.30000</td>
</tr>
<tr>
<td>1.30625</td>
</tr>
<tr>
<td>1.31875</td>
</tr>
<tr>
<td>1.32500</td>
</tr>
</tbody>
</table>

Failing tests devised with the proposed methodology clearly outperform all the other approaches. Remarkably, µGP was asked to find a very fast failing test for a specific microprocessor, and therefore there is no guarantee that the devised program would fail on a different model. Moreover, the test was required to be very short, to avoid heating effects. On the contrary, stress tests intentionally exploit overheating and are designed to work with different architectures.

The failing test for the E2180 is 614 line long. The two functions executed by the two cores are respectively 280 and...
The failing test for the i7-950 is 997 line long. The four functions executed by the four cores are respectively 236, 206, 153, 174 line long. The remaining lines are mainly used to define and initialize variables or other program parts.

It should also be noted that µGP required about 5 hours to generate the best failing test for the E2180, and 40 hours for the i7-950. The difference in time can be explained taking into account the greater number available steps, and the length of the test itself.

It must be also noted that the temperature of the microprocessor during the experiments never exceeded 40°C for the E2180 and 50°C for the i7-950, while running LINPACK-based stress tests are significantly higher, even with the liquid cooling.

D. Feedback from the Overclockers Community

The generated tests were made available to the overclockers community as ultra-fast stability test 7. Although not systematic, the feedback fully confirmed our claims: results on i7-950 microprocessors show the superiority of the µGP test. Similar results are achieved on i7-920 units. Interestingly, the failing test is not effective on the i7-860 family. Thus, it is reasonable to presume that the test stress some microarchitectural features present only in certain family.

V. CONCLUSIONS AND FUTURE WORKS

The paper proposed an efficient post-silicon methodology for devising functional failing tests. Experimental results clearly demonstrate that tests are able to highlight criticalities specific of the target microarchitecture: they work equally well with all i7-950 units, but not with devices of different families. More interestingly, it is able to do it without any information about the design. Such result is not completely surprising: µGP already managed to stress specific microarchitectural features tackling a Pentium 4 as a mere black box [17].

The proposed methodology could be exploited by microprocessor manufacturers during verification or speed stepping. The knowledge of the internal design and the physical access to the device under test would be necessary to continue on this line of research.

On the other hand, the methodology could also be used to generate a fast test able to check the reliability of a system. This usage, can be important for the incoming inspection of a set of purchased devices.

Future works include enhancing the evolutionary algorithm, letting it set the number of repetitions in each test. The interaction between x87 and SIMD instructions also deserves a closer examination. A customized version of the µGP requiring no operating systems can be devised in order to more easily run experiments on the microprocessor. Also, the signature could be improved by including more information on the state of the execution, such as the internal performance monitor.

References


http://www.cad.polito.it/research/Evolutionary_Computation/Overclocking.html