Testing CPU stability using Multi-precision Arithmetic by calculating *n* digits of π

G. Hari Prakash

***Abstract*—Calculating arbitrary precision of π stresses the fixed-point, floating-point, logic, shift, branch prediction and pipelining circuits of a CPU. Testing of a CPU using two versions of algorithmically different programs to generate π and verifying the result by comparison is proposed to ensure the integrity of the CPU. The rapidly convergent compute intensive Borwein-Borwein (BB) algorithm is used as the CPU stressor and Bailey-Borwein-Plouffe (BBP) digit-extraction spigot algorithm as a result-checker is studied in this paper. The faulty CPU is identified, if there is a difference in the result produced by both the algorithms. This paper describes a technique to verify the integrity of the CPU by selecting the first algorithm from the class of iterative convergence based algorithms to generate the first n digits and the second algorithm to extract the *n*th digit to verify the result of the first algorithm. This technique is superior to the existing method of verifying the result with the same class of iterative convergence algorithms where the first n digits need to be re-generated using a different iterative algorithm. The digit-extraction algorithm (BBP) requires less system resources and time compared to iterative algorithms and hence suited as an ideal π result-checker. The proposed method can be included in the CPU diagnostic routines to ensure the arithmetic stability of the computer.**

***Index Terms*— arbitrary precision arithmetic, spigot algorithm, CPU diagnostics.**

I. Introduction

The computation of the numerical value of π has been pursued for centuries for a variety of reasons, both practical and theoretical. A value of π correct to 20 decimal places is sufficient for most practical applications. Even for scientific computations a value of π correct to 100 decimal places is sufficient. The extended precision calculation of π has substantial application as a test of global integrity of computer systems. The extended precision calculation of π uncovers most of the CPU hardware errors [1].

In recent years, the computation of the expansion of π has assumed the role as a standard test of computer integrity. If even one error occurs in a computation then the result will almost certainly be completely in error after an initial correct section. For this reason, programs that compute the decimal expansion of π are frequently used by both manufacturers and purchasers of new computers to certify system reliability [2].

II. Background

A. Brent-Salamin iterative rapid convergence algorithm

An approximate algorithm based on elliptic integrals that yield quadratic convergence to π. In series-based algorithms, the number of correct digits increases only linearly with the number of iterations performed. In Bernt-Salamin (BS) algorithm, single iteration of the algorithm doubles the number of correct digits of π. BS algorithm is also known as Gauss-Legendre (GL) algorithm. Kanada and Tamura employed this algorithm to compute π to over 16 million decimal digits [11] [12]. BS algorithm:

Initial value

$$a_0 = 1, b_0 = \frac{1}{\sqrt{2}}, t_0 = \frac{1}{4}, p_0 = 1$$

Iterate until a_n and b_n is within the desired accuracy

$$a_{n+1} = \frac{a_n + b_n}{2}$$
$$b_{n+1} = \sqrt{a_n b_n}$$
$$t_{n+1} = t_n - p_n (a_n - a_{n+1})^2$$
$$p_{n+1} = 2p_n$$

The approximate value of π:

$$\pi \approx \frac{(a_n + b_n)^2}{4t_n}$$

The calculated π value is verified using the Borwein-Borwein (BB) algorithm given in the next section. The BB algorithm demands higher precision than the desired digits. It is to be noted that the BS algorithm is superior to BB algorithm. The correctness of the digits generated is verified by re-generating digits using different algorithm and comparing the results. Moreover, the re-generation methodology is time and space consuming when applied for verifying the integrity of the CPU. It is just sufficient to verify the specific digits of the result to ensure integrity of the CPU using alternate fast method without re-generating the n digits of π.

B. Borwein-Borwein iterative rapid convergence algorithm

Jonathan and Peter Borwein devised many rapidly quadratically convergent algorithm for calculating π based on the *arithmetic geometric mean* (AGM) [3] [7]. In 1987 they devised a quadratic convergence algorithm to calculate the reciprocal of π. BB algorithm:
Initialize

$$x_0 = \sqrt{2}$$
$$y_0 = \sqrt[4]{2}$$
$$p_0 = 2 + \sqrt{2}$$

Iterate

$$x_k = \frac{1}{2}\left(x_{k-1}^{\frac{1}{2}} + x_{k-1}^{\frac{-1}{2}} \right)$$

$$y_k = \frac{y_{k-1} x_{k-1}^{\frac{1}{2}} + x_{k-1}^{\frac{-1}{2}}}{y_{k-1} + 1}$$

$$p_k = p_{k-1} \frac{x_{k+1}}{y_{k+1}}$$

$$p_k converges \xrightarrow{to} \pi$$

$$p_k - \pi = 10^{-2^{k-1}}$$

C. Bailey-Borwein-Plouffe digit-extraction algorithm

The BBP algorithm is based on polylogarithmic ladder in integer base [4] [7]. The BBP π summation formula was founded in 1995 by Simon Plouffe using PSLQ integer relation algorithm [5]. The BBP formula provides a spigot algorithm [6] for the computation of the n^{th} binary digit of π without calculating the previous digits of π. BBP algorithm:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

$$\pi = 4\sum_{k=0}^{\infty} \frac{1}{(16^k)(8k+1)} - 2\sum_{k=0}^{\infty} \frac{1}{(16^k)(8k+4)} - \sum_{k=0}^{\infty} \frac{1}{(16^k)(8k+5)} - \sum_{k=0}^{\infty} \frac{1}{(16^k)(8k+6)}$$

Using spigot algorithm to find n^{th} hexadecimal digit of π, take the first sum and split the sum to infinity across the n^{th} term.

$$\sum_{k=0}^{\infty} \frac{1}{(16^k)(8k+1)} = \sum_{k=0}^{n} \frac{1}{(16^k)(8k+1)} + \sum_{k=n+1}^{\infty} \frac{1}{(16^k)(8k+1)}$$

Multiply by 16^n so that the hexadecimal point (the divide between fractional and integer parts of the number) is in the *n*th place.

$$\sum_{k=0}^{\infty} \frac{16^{n-k}}{8k+1} = \sum_{k=0}^{n} \frac{16^{n-k}}{8k+1} + \sum_{k=n+1}^{\infty} \frac{16^{n-k}}{8k+1}$$

For digit-extraction only the fractional part of the sum is required. The first sum is able to produce whole numbers whereas the second sum cannot produce whole numbers since the numerator can never be larger than the denominator for $k > n$. To remove the whole numbers for the first sum perform modulo operation of the numerator by $8k + 1$. The sum for the first fractional part then becomes:

$$\sum_{k=0}^{n} \frac{16^{n-k} \bmod 8k+1}{8k+1} + \sum_{k=n+1}^{\infty} \frac{16^{n-k}}{8k+1}$$

The modulo operator always guarantees that only the fractional sum is retained. The modulo operation is performed when the running product becomes greater than one. In order to complete the calculation apply this to each of the four sums in turn. Finally the four summations and put back into the sum to π.

$$\pi = 4\sum\nolimits_1 - 2\sum\nolimits_2 - \sum\nolimits_3 - \sum\nolimits_4$$

Leave the integer part since only the fractional part is accurate and multiply the final sum by 16 to extract the hexadecimal digit.

This process is similar to performing long multiplication, but only having to perform the summation of some middle columns. While there are some carries that are not counted, computers usually perform arithmetic for many bits (32 or 64) and they round and we are only interested in the most significant digit. There is a small possibility that a particular computation will be akin to failing to add a small number (e.g. 1) to the number 999999999999999 and that the error will propagate to the most significant digit, but being near this situation is obvious in the final value produced.

The BBP algorithm computes π without requiring custom data types having thousands or even millions of digits. The method calculates the *n*th digit *without* calculating the first $n - 1$ digits, and can use small, efficient data types.

The algorithm is the fastest way to compute the n^{th} digit (or a few digits in a neighborhood of the n^{th}), but π-computing algorithms using large data types remain faster when the goal is to compute all the digits from 1 to n. This algorithm has proven popular and many software implementations exist.

III. Implementation

In the previous section we have mentioned that the BS algorithm is superior to BB algorithm, but we still prefer using the BB algorithm for the following reason. Higher precision of BB algorithm implies more stress to the circuits. Compared to the BS algorithm, BB algorithm is a good CPU stressor. The BB and BBP algorithms are implemented in C++ language. The digits produced in BB algorithm are implemented in 256 base, they are converted to hexadecimal digits at the end of the calculation [8]. Single byte consisting of two hexadecimal digits are extracted from the BBP algorithm and compared with the BB result for checking the stability of the CPU during the execution of BB algorithm. Incase of digit mismatch, a CPU fault signal is triggered.

Similar to N-version software fault-tolerance model, the BB and BBP algorithms are diverse and least overlap on the utilization of CPU resources to avoid common mode failure.

The number of digits to calculate depends on the CPU diagnostic run level. Higher the run level, longer the run-time and higher the precision of π is calculated, providing more stress to the arithmetic and instruction pipelines.

IV. Performance

The Borwein-Borwein (BB) algorithm is asymptotically faster algorithm to calculate the first n digits of π uses Fast Fourier Transform (FFT) based multiplication but it is slower than Bailey-Borwein-Plouffe (BBP) algorithm to calculate the n^{th} digit of π. The BB algorithm is used to stress the CPU using iterative algorithm and requires multi-precision arithmetic. Whereas, the BBP algorithm is used to cross-check the results produced by the BB algorithm. The BBP algorithm requires no multi-precision support and virtually no memory. The algorithm runs at near linearly with the order of the desired digit. Ordinary IEEE 64-bit arithmetic is sufficient to obtain a numerically significant result for even a large computation, and quad precision (i.e. 128-bit) arithmetic, can insure that the final result is accurate to several digits beyond the one desired. One can check the significance of a computed result beginning at position n by also performing a computation at position $n + 1$ or $n - 1$ and comparing the trailing digits produced.

V. Coverage

The coverage of BB algorithm can be evaluated using the static and dynamic analysis of instructions of the executable program. The operation count depends on Instruction Set Architecture (ISA). The frequency of the pipeline also influences the timing coverage of the CPU. Running the BB algorithm in a cycle accurate CPU simulator like SimpleScalar [9] will provide more details of the coverage property of calculating arbitrary precision digits of π. A multi-threaded version of BB and BBP algorithm can be coded for testing multi-core processors.

VI. CONCLUSION

Calculating arbitrary precision value of π will improve the arithmetic and instruction pipeline coverage of CPU. Generating few thousand digits of π using BB algorithm and checking using BBP algorithm ensure reasonable stability of the highly used arithmetic and instruction pipelines of a CPU. The BB and BBP algorithms are diverse and least overlap on the utilization of CPU resources to avoid common mode failure.

References

[1] J. M. Borwein and P. B. Borwein, D.H.Bailey, “Ramanujan, Modular Equations, and Approximations to Pi or How to Compute One Billion Digits of Pi”. American Mathematical Monthly, Vol.96. No. 3. March 1989.

[2] David H. Bailey, “The Computation of π to 29,360,000 Decimal Digits Using Borwein’s Quartically Convergent Algorithm”, Mathematics of Computation Vol. 50, number 181. Jan. 1988, pages 283-296.

[3] Jonathan Borwein, Peter Borwein, “Pi and the AGM: A Study in Analytic Number Theory and Computational Complexity”, John Wiley, 1987.

[4] David Bailey, Peter Borwein and Simon Plouffe, “On the Rapid Computation of Various Polylogarithmic Constants”, *Mathematics of Computation*, vol. 66, no. 218 (April 1997), pages. 903–913.

[5] Weisstein, Eric W. “PSLQ Algorithm.” From *Math World – A Wolfram Web Resource*. http://mathworld.wolfram.com/PSLQAlgorithm.html

[6] Spigot algorithm, *Wikipedia* [Online] http://en.wikipedia.org/wiki/Spigot_algorithm.

[7] Lennart Berggren, Jonathan Borwein, Peter Borwein, “Pi: A Source Book”, Springer Verlag, 3rd edition. 1997.

[8] William.H. Presss, Saul A. Teukolsky, William T. Vellerling, Brain P.Flannery, “Numerical Recipes in C”, 2nd Edition, Cambridge University press. 1993.

[9] Todd Austim. “SimpleScalar LLC – Processor simulator”, [Online] http://www.simplescalar.com

[10] LAPACK, “Linear Algebra Package“. [Online]. http://www.netlib.org/linpack

[11] Yasumasa Kanada,”Super_Pi program to calsulate Pi upto 32 million digits”, [Online]. http://en.wikipedia.org/wiki/Super_PI

[12] Yasumasa Kanada, “ Vectorization of Multiple-Precision Arithmetic Program and 201,326,000 Decimal Digits of π Calculation”, *Scientific American*. 1988.

G. Hariprakash, Research Advisor, Computer Science and Engineering department of Srinivasa Ramanujan Centre, SASTRA University, Kumbakonam, TN, India. His area of interest includes CPU diagnostics, Computer architecture, Processor micro-architecture and Operating systems. He can be reached through email *logichari@gmail.com*