A New Speculative Addition Architecture Suitable for Two's Complement Operations

Alessandro Cilardo
Università degli Studi di Napoli Federico II
Dipartimento di Informatica e Sistemistica
Via Claudio 21, 80125 Napoli, Italy
Email: acilardo@unina.it

***Abstract*—Existing architectures for speculative addition are all based on the assumption that operands have uniformly distributed bits, which is rarely verified in real applications. As a consequence, they may be disadvantageous for real-world workloads, although in principle faster than standard adders. To address this limitation, we introduce a new architecture based on an innovative technique for speculative global carry evaluation. The proposed architecture solves the main drawback of existing schemes and, evaluated on real-world benchmarks, it exhibits an interesting performance improvement with respect to both standard adders and alternative architectures for speculative addition.**

I. Introduction

Microprocessor performance is deeply affected by arithmetic operations, especially integer addition. Finding fast and efficient implementations for this operation, therefore, has always been one of the major challenges in digital design. Most of the current high performance processors employ one of the known parallel adders [3], [5], such as Carry Lookahead, Brent-Kung, Kogge-Stone, and Carry Select adders. In synchronous designs, the worst case delay of such adders clearly limits the maximum frequency. Speculative addition, on the other hand, is based on the idea of performing an incomplete, but much faster addition, which turns out to be correct most of the time. Most existing speculative architectures are efficient only under the assumption that input operands have uniformly distributed random bits, producing short carry propagation chains on average. As a preliminary contribution in this work, we present a framework for dynamic profiling of software programs, which provides an experimental characterization of carry chain lengths in real-world benchmarks and confirms the limitations of existing schemes. Based on the above characterization, we develop an innovative architecture for speculative addition. The new scheme has the same critical path than previously proposed speculative adders, but can effectively handle the long carry chains occurring in real benchmarks, mostly depending on two's complement operations. Experimental results and comparisons with traditional adders confirm the effectiveness of our approach and exhibit an interesting performance improvement measured on real-world programs.

II. Speculative Addition

The most significant bit of an addition result depends, in the worst case, on all input bits. This happens, in particular, when each input bit pair a_i/b_i determines a carry-propagate condition, i.e. $a_i \oplus b_i = 1$. As a consequence of this dependence, all circuits for n-bit integer addition have a time complexity of at least $O(\log n)$ [5]. Figure 1 gives an example of different "carry chains", i.e. sequences of consecutive pairs $a_i \oplus b_i = 1$ causing a propagation of a possible carry and, thus, the dependence of output bits on a larger subset of inputs. A fundamental observation, at the heart of the so-

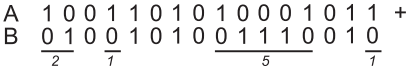

Fig. 1. Various carry chain lengths.

called speculative addition, is that such carry chains are, on average, much shorter than the full operand size n, at least assuming uniformly distributed random inputs. Based on this observation, speculative adders are structured in such a way that each output bit depends only on the previous k bits, with k much smaller than n. The resulting circuit is clearly faster, but unfortunately it produces a wrong result when there are carry chains of k bits or more. In such occurrences, a complete carry propagation is required before producing the correct result. Besides the adder itself, thus, we also need a suitable circuit able to detect and notify this error condition. In an asynchronous design, this corresponds to delaying the completion of the operation. In a synchronous design, on the other hand, the error condition will be handled by some external control circuitry causing one or more additional clock cycles for further processing.

An interesting question relates to how to decide the value k of the maximum carry chain to be detected. Intuitively, since $a_i \oplus b_i = 1$ occurs with a probability of $1/2$, the probability of a single k-bit carry chain is $(1/2)^k$, assuming uniformly distributed random inputs. It is less trivial to study the distribution of the *maximum* carry chain length generated by an n-bit addition. This is equivalent to solving the "longest run of heads" problem in a fair coin-tossing experiment. Although an exact treatment is not straightforward, all previous works on speculative addition [1], [7], [4], with different arguments, converge on the fact that such longest chain is in the order of $\log n$, i.e. much smaller than n, making speculative adders much faster than complete adders.

978-3-9810801-5-5/DATE09 © 2009 EDAA

Speculative addition has been firstly proposed in the context of asynchronous design. In [6], a fast asynchronous k-bit speculative adder is extended with different abort detection networks, each associated with a different delay condition occurring in the addition circuit. Abort detection is computed in parallel with the main datapath computation. Authors in [6] also consider non-random input distributions obtained from actual software programs. Based on this, they introduce some variants to the base architecture, which can handle long carry propagation due to two's complement numbers, although limited to some specific portions of the input bits. A synchronous solution for speculative execution is used in [2]. It involves a two-cycle operation. The addition is started in the first cycle, and the result is assumed to be correct. A parallel carry propagation network checks whether or not the operation has long carry paths. Should this be the case, the system is stalled for the duration of an additional clock cycle where the original addition operation has sufficient time to complete. Some bypass logic is required, to allow the sum to be generated using these early carries. The detection of long carry chains takes place only in the second cycle, so the previous result may need to be overwritten. A similar construction is presented in [1], where some solutions for early termination detection are described, and a statistical approach for their area-efficient implementation is discussed. In [4], the concept of "approximate" addition is proposed, although authors do not go into detail of its implementation.

A recent contribution has been presented in [7]. We will describe it in more detail. In its first stage, it has a parallel prefix structure, as most other solutions. Propagate/Generate (P/G) signals for single digits –or for groups of digits– represent respectively carry propagation (the output carry for those digits is equal to the input carry) and carry generation (the sum of those digits always produces an output carry independent of previous carries). For a one-bit digit, P/G are computed as

$$P_i = a_i \oplus b_i \qquad G_i = a_i \cdot b_i$$

When applied to a group of digits in position $_\alpha \ldots _\delta$, generation/propagation signals will be denoted as $P_{\delta-\alpha}$ and $G_{\delta-\alpha}$. The combination of P/G for consecutive, possibly overlapping, digit groups is obtained as follows:

$$P_{\delta-\alpha} = P_{\delta-\gamma} \cdot P_{\beta-\alpha} \qquad G_{\delta-\alpha} = G_{\delta-\gamma} + P_{\delta-\gamma} \cdot G_{\beta-\alpha}$$

where $\delta \geq \beta \geq \gamma \geq \alpha$, and $\oplus$, $\cdot$, and $+$ denote the XOR, AND, and OR function, respectively. In a k-bit speculative adder, each output bit depends only on the previous k bits, so for each position i, we need the signals $P_{(i-1)-(i-k)}$ and $G_{(i-1)-(i-k)}$. They can be built with the scheme depicted in Figure 2 (where $k = 4$), also used in [6]. Sum output bits s_i are then easily obtained by P/G signals:

$$s_i = P_i \oplus G_{(i-1)-(i-k)}$$

while the error condition is raised if at least one k-bit long carry chain exists:

$$E = \sum_{i=k}^{n} P_{(i-1)-(i-k)} \tag{1}$$

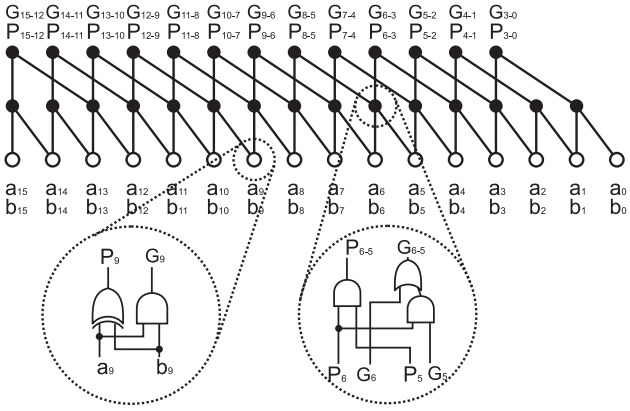

Fig. 2. Generation of partial P/G signals.

Notice that the critical path of the architecture depends on the error condition, which is a *global* signal, and still requires an $O(\log n)$ time complexity. The advantage of the scheme is that the large OR required by the error condition can be computed more efficiently than the global carries required by Carry Lookahead and Parallel Prefix adders. Like other schemes, there is also an "error recovery circuit" which completes the addition (based on a Carry Lookahead circuit) in less than two clock periods and is used in the case $E = 1$, but only after the second cycle. The structure of the speculative adder proposed in [7] is depicted in Figure 3.

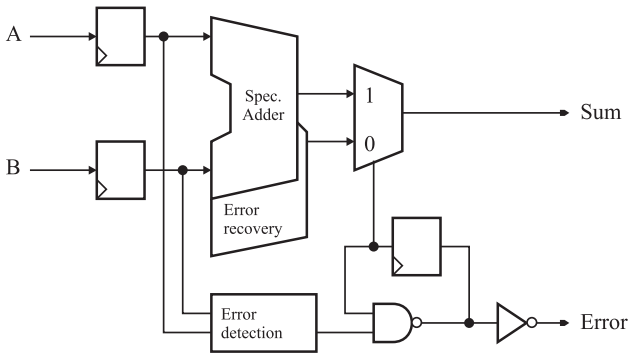

Fig. 3. Speculative Adder in [7]

III. Characterization of Real Workloads

As mentioned in the introductory section, most previous works are based on the assumption that input operands have uniformly distributed random bits, and hence produce short average carry chains. This condition largely simplifies the design of speculative adders, since one can just make each output bit have a purely "local" dependence on addition operands, i.e. depend only on the previous k bits. Unfortunately, this

assumption is unlikely to be verified in practice. A very simple example will help clarify the problem. Consider, for instance, a small program computing the Greatest Common Divisor (GCD) based on Euclide's algorithm:

```
while (b!=0){
    if (a > b)    a = a - b;
    else          b = b - a;    }
```

The above code corresponds to two comparisons (at least one is likely to be implemented as a subtraction) and one subtraction performed in each iteration. All of them require a negation of the bits of the subtrahend, followed by an addition with carry-in equal to 1. Following is a trace of the algorithm execution, with initial values a=15 and b=25. The trace includes all additions performed on two's complement numbers, including those for comparisons. x and y denote the actual values processed by the adder. The comparison b!=0 is assumed to be performed by a dedicated zero-detection circuit and is not included in the trace.

	a and b in 2'complement representation
x = a = 15	00000000000000000000000000001111
y = –b = –25	11111111111111111111111111100111
x = b = 25	00000000000000000000000000011001
y = –a = –15	11111111111111111111111111110001
x = a = 15	00000000000000000000000000001111
y = –b = –10	11111111111111111111111111110110
x = a = 5	00000000000000000000000000000101
y = –b = –10	11111111111111111111111111110110
x = b = 10	00000000000000000000000000001010
y = –a = –5	11111111111111111111111111111011
x = a = 5	00000000000000000000000000000101
y = –b = –5	11111111111111111111111111111011
x = b = 5	00000000000000000000000000000101
y = –a = –5	11111111111111111111111111111011

As emphasized above, very long carry chains, and hence *non-local* dependences of output bits on input bits, are very likely to happen, especially when manipulating number in two's complement representation. Because of carry chains longer than k, existing speculative adders would incur a significant penalty in the execution time due to misprediction (e.g., two clock cycles for each addition with the adder in [7]), which may make them totally ineffective.

In order to emphasize the limitations of existing speculative architectures with real-world workloads, we developed an ad-hoc framework for dynamic profiling of software programs, i.e. for tracing the actual operand values driven to the processor adder as they are produced during execution. The traces collected, with special emphasis on data-intensive algorithms, are also representative of hardware implementations whose architecture is based on a single parallel adder.

Figure 4 depicts the structure of the framework. Unlike the approach taken, for example, in [6], we did not modify the implementation of a general-purpose processor simulator in order to collect the traces. Rather, we decided to directly instrument native code running on a physical machine (e.g. an IA-32 machine) so as to run benchmarks at real-machine speed.

Benchmarks can be made of one or more .c/.s modules. We may be not interested in profiling the entire benchmark, e.g. there may be a main module preparing test cases, which

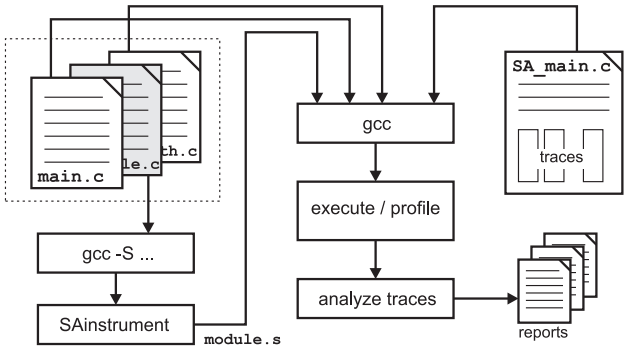

Fig. 4. The profiling framework.

should not be profiled in itself. The framework, therefore, allows the selection of some submodules in the benchmark. These are separately pre-processed by compiling them to an .s assembler file (using the GCC GNU compiler), which are subsequently instrumented by an ad-hoc tool. Instrumentation, basically, consists in replacing each profiled instruction (e.g., ADD, SUB, or CMP), with special code fragments that save the instruction operands to some trace memory areas. The entire benchmark is then compiled and linked with a higher-level main (SA_main), which contains the instantiation of memory areas used for traces, some definitions, the invocation of the benchmark main, and the functions processing the traces after execution and generating textual reports.

For the evaluation presented in this work, we considered a cryptographic workload including: RSA encryption/decryption (RSA), Elliptic curve ElGamal encryption/decryption over prime fields (ECELGP), Diffie-Hellman key exchange (DH), and Elliptic curve digital signature algorithm over prime fields (ECDSP). The results of profiling are depicted in Figure 5, as the relative distribution of maximum carry chain lengths occurring during addition operations. For each benchmark, the charts distinguish the chains due to sign extension (white bars) from the remaining generic carry chains (gray bars). On the left side, the charts include carry chains up to a certain length, for which a significant number of samples was found. On the right side, they include carry chains with lengths close to n, the full operand width, which are mainly due to two's complement operations. This separation makes the characterization mostly independent of the actual adder width n, due to sign extension that takes place for two's complement numbers. It can be easily seen from such results that long carry chains are far from rare. As a consequence, the performance of existing solutions for speculative additions dramatically drops compared to the assumption of randomly distributed input bits. A quantitative evaluation will be provided for comparison in the following sections.

IV. Speculative Adder Architecture

Although speculative adders in themselves are based only on local dependences, the associated error detection circuit, implementing a function like that in Equation 1, depends

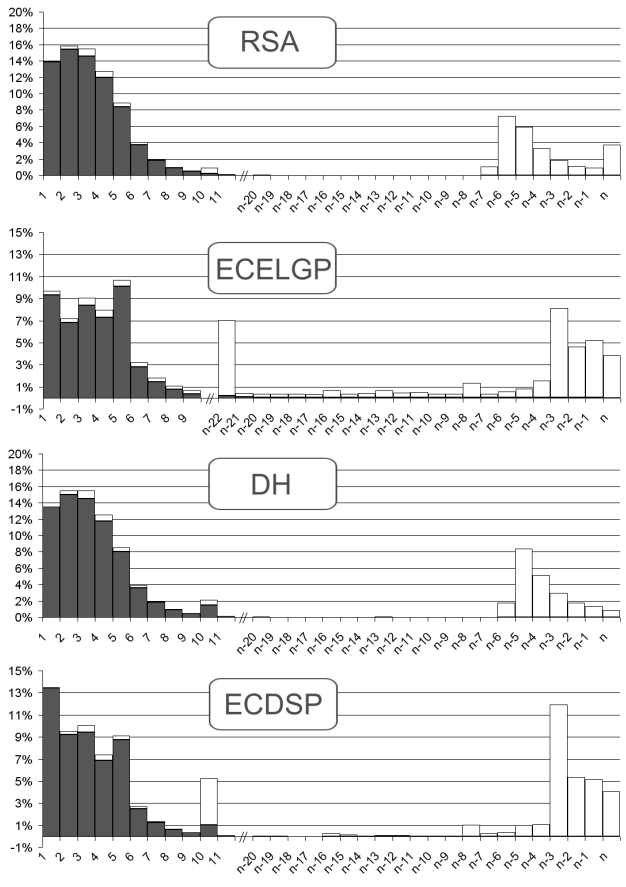

Fig. 5. Characterization of carry lengths.

on all input bits (it includes a large OR gate with a fan-in close to n). This circuit should operate in parallel with the speculative adder and is thus likely to determine the delay of the overall structure. Nevertheless, as observed in [7], the error detection function has a simpler structure and enables a faster implementation than an complete adder. On the other hand, the presence of long carry chains, inherent in signed operations, has the main effect of creating a "global" dependence of output bits on input bits, which makes existing speculative adders ineffective. The essential idea behind our architecture is to introduce a new speculation circuit which, based only on local information, is able to anticipate the generation of a carry due to signed operations. As shown later, the complexity of such a *global carry* speculation circuit is very similar to the error detection circuitry, so its presence does not necessarily increase the delay of the overall speculative adder.

Figure 6 presents the structure of a cell in our speculative addition architecture. Consider the cell i and call P_R and G_R the P/G signals coming from the k cells on the right, and P_L the propagate signal coming from the k cells on the left. Notice that these signals are always available when a structure like that in Figure 2 is adopted. As can be seen in Figure 6, each cell implements the usual sum equation $s_i = P_i \oplus (G_R + P_R \cdot C)$ by using the P_R/G_R signals and

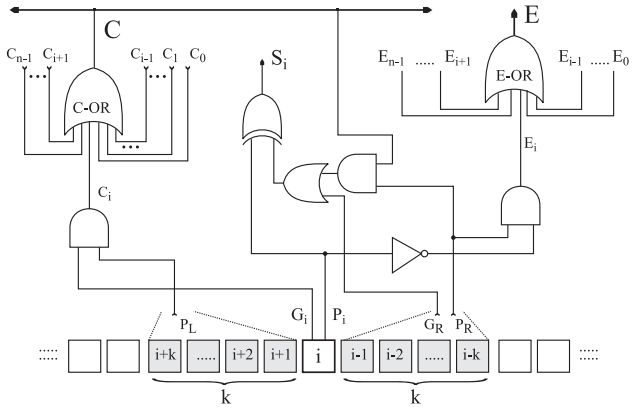

Fig. 6. A Speculative Adder cell.

the global carry C as the incoming carry. The global carry, in turn, can be high only if there is (at least) one cell which generates ($G_i = 1$) followed by k propagates on the left ($P_L = 1$). The error condition E_i is raised if there is at least one cell having k consecutive propagates on the right ($P_R = 1$), *unless* these are followed by a further propagate on the left ($P_i = 1$). This structure is sufficient to produce a correct result in presence of long carry chains due to signed operations, independent of their length. This can be justified as follows. If there is a long propagate chain reaching the left end of the adder (due to sign extension), the first cell from the left which does not propagate, say the ith cell, is responsible for carry generation. If $G_i = 1$, then the global carry C will be 1 and will be used as the carry-in by all subsequent cells up to the most significant bit. Indeed, in the case there are k consecutive propagates determining $P_R = 1$, *not* included in the sign extension chain, the global carry will be erroneously used as the carry-in also by that chain, regardless of where it is located in the adder. If present, such an "inner" chain will however be recognized by the error detection circuitry, leading to an invalid condition. A long propagation chain which *does not* reach the left end of the adder, in fact, is always detected, because there will be at some point a non-propagating cell ($P_i = 0$, $P_R = 1 \Rightarrow E_i = 1$) raising the error condition. In other words, the error detection circuit used in the base speculative adder scheme is mostly reused for detecting a mispredicted global carry. Notice that, according to our experimental results, long chains not due to sign extension, i.e. not reaching the most significant bit, are *actually* rare in real workloads, so the occurrence of such error conditions can be tolerated in practice. Some examples will help clarify the scheme. Assume $n = 16$ and $k = 4$:

(1)	P_i	1111111100001110	(2)	P_i	1111111111000111
	G_i	0000000000000000		G_i	0000000000100000
	C_i	0000000000000000		C_i	0000000000100000
	S_i	1111111100001110		S_i	1000000000000111
	E_i	0000000000000000		E_i	0000000000000000

(3)			(4)		
	P_i	1111110000111110		P_i	1111110000111110
	G_i	0000001000000000		G_i	0000000000000001
	C_i	0000001000000000		C_i	0000000000000001
	S_i	1000000001011110		S_i	1001111000100000
	E_i	0000000001000000		E_i	0000000001000000

The result S is always correct when all E_i are zero. In example (3), the global carry $C = \sum C_i$ is high due to a generate followed by a long propagate chain on the left. Unfortunately, C is also spuriously used as the carry-input by another 4-bit propagate inner chain. However, the same 4-bit propagate chain is responsible for an error generation, so the result is not valid in this case. As emphasized by example (4), any propagate chain which does not reach the most significant bit may determine a spurious global carry C, but it is always detected since the cell where the propagate chain ends will raise an error.

As for similar solutions [7], the speculative adder is completed with some "error recovery" circuitry, which takes an additional clock cycle to complete the operation upon error. The error recovery circuit is based on a carry-lookahead scheme and is not critical for the timing of the overall architecture, since its delay spans two clock cycles.

As emphasized in the upper portion of Figure 7, the global carry C is shared among all cells, so a given output bit s_i may be influenced by cells in higher positions. In fact, this can only happen if s_i is preceded by a long "inner" chain ($P_R = 1$), and is thus affected by the global carry C even if this is generated in higher positions. Such an inner chain, however, corresponds to a "don't care" situation for the speculative adder output, since it is always associated to an error condition. In other words, the dependence of output bits s_i can be limited to local carries coming from the right, and this can create some opportunities for further optimization. In particular, we may want to reduce the fan-out of the large OR gate generating C. This can be done as in the lower portion of Figure 7, where incoming carries are partitioned into different groups. Each

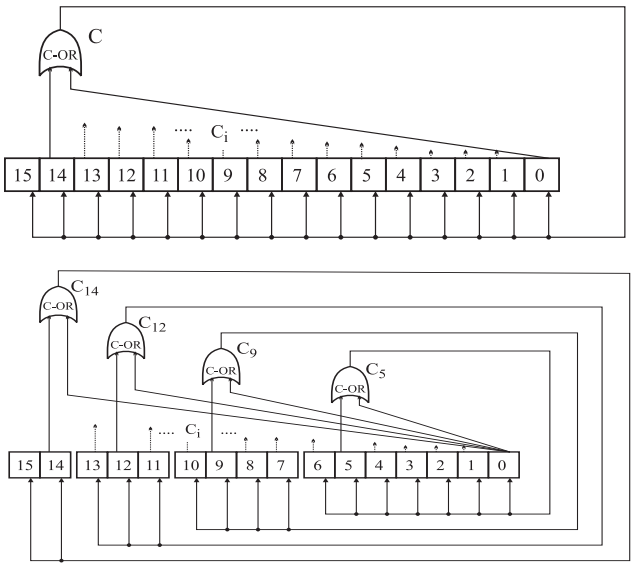

Fig. 7. Two different solutions for generation of global carries.

cell receives a carry input which is an OR of (at least) all local carry outputs on its right side. OR gates with a lower fan-in (those used to drive cells in lower positions) are less critical and may have a larger fan-out. Clearly, a great deal of sub-expression sharing can be exploited among the different OR gates, and the partitioning can be chosen (possibly by a synthesis tool) so as to privilege more critical OR gates. As our experimental results confirm, the circuits for generating global carries and the error signal E enable a faster implementation than ordinary adders, even when compared to highly optimized library cores already available in synthesis tools. As a last observation, notice that –although affected by a large fan-out– the global carry is a purely internal signal. In a complete design, it is thus less critical than the error signal E which, on the other hand, will exit the adder and will be processed by an external controller, possibly during the same clock cycle in order to avoid stalls.

V. Experimental Results and Comparisons

As explained in the previous section, our speculative adder scheme can correctly process in a single clock cycle any propagate chain caused by signed operations, i.e. *all* white bars in the charts of Figure 5, in addition to generic chains with length less than k. In order to provide an experimental evaluation, the speculative adder architecture was synthesized for a CM0S $0.18\mu m$ standard cell library technology by using Cadence Build Gates synthesis tool. The results were compared with a similar scheme recently proposed [7] and with optimized library adders already available to the synthesis tool. For uniform comparisons, the three designs were synthesized with the same environment, so our results for clock periods are slightly different than those in [7]. Concerning area complexity, our speculative adder requires slightly more gates than the previous solution in [7]. In all cases, however, this increment was not above 25% with respect to the previous speculative adder solution. For time performance, we compared the minimum time spent for addition operations alone, evaluated as $T = T_{CLK} \cdot (N_1 + 2 \cdot N_2)$, where T_{CLK} is the minimum clock period, N_1 is the number of correct speculative additions, which only require one clock cycle, and N_2 is the number of additions that require a further clock cycle due to a speculation miss. For a standard adder, clearly, $N_2 = 0$. We consider 128, 256, 512, and 1024-bit adders, with $k = 12, 13, 14, 15$, respectively, as done in [7]. Total addition times are normalized to library adder times.

	$n = 128$			$n = 256$		
	libr.	[7]	here	libr.	[7]	here
T_{CLK} (ns)	**0.84**	**0.69**	**0.78**	**0.95**	**0.76**	**0.86**
RSA	1.00	1.01	0.93	1.00	0.97	0.90
ECELGP	1.00	1.08	0.93	1.00	1.05	0.91
DH	1.00	1.01	0.93	1.00	0.98	0.91
ECDSP	1.00	1.08	0.93	1.00	1.05	0.91

	$n = 512$			$n = 1024$		
T_{CLK} (ns)	**1.07**	**0.84**	**0.95**	**1.19**	**0.91**	**1.04**
RSA	1.00	0.95	0.89	1.00	0.93	0.87
ECELGP	1.00	1.03	0.89	1.00	1.00	0.88
DH	1.00	0.96	0.89	1.00	0.93	0.87
ECDSP	1.00	1.03	0.89	1.00	1.00	0.87

We found that our adder guesses the correct result in *at least* 99.46% of all additions (the worst case is the ECELGP benchmark, $n = 128$). In other words, our scheme reduces N_2 almost to 0, making the average time for an addition very close to a single period T_{CLK}. Due to the reduced delays enabled by speculation, this allows a significant improvement in the overall performance. We are also confident that a technology-aware, optimized implementation may further decrease this delay.

VI. Conclusions

Speculative adders, as they have been proposed so far, suffer from some limitations that may make them disadvantageous for real-world benchmarks. The new architecture proposed in this paper removes the assumption of uniformly distributed input bits and, based on an innovative technique for speculative global carry evaluation, it solves the main drawback of existing solutions, related to two's complement arithmetic. Unlike most previous works on speculative addition, our solution exhibits very high speculation hit-rates evaluated on real workloads.

Acknowledgment

This work was partially supported by Regione Campania and Ditron S.R.L. in the framework of "Progetto Metadistretto del settore ICT - Misura 3.17: Sistema di comunicazione per l'integrazione delle informazioni nella distribuzione commerciale dei punti vendita".

References

[1] A. Burg, F. K. Girkaynak, H. Kaeslin, W. Fichtner, "Variable delay ripple carry adder with carry chain interrupt detection," in *Proc. of the 2003 Int. Symposium on Circuits and Systems, 2003*, vol. 5, no. 25-28, pp. 113-116, 2003.

[2] Y. Kondo, N. Ikumi, K. Ueno, J. Mori, and M. Hirano, "An Early-Completion-Detecting ALU for a 1GHz 64b Datapath," in *Proc. of the IEEE Int. Solid-State Circuits Conference (ISSCC)*, pp 418-419, 1997.

[3] I. Koren, *Computer Arithmetic Algorithms*, Prentice-Hall Inc., New Jersey, 1993.

[4] S.-L. Lu, "Speeding Up Processing with Approximation Circuits," *Computer*, vol. 37, no. 3, pp. 67-73, 2004.

[5] C. Nagendra, M.J. Irwin, and R.M. Owens, "Area-time-power tradeoffs in parallel adders," *IEEE Trans. on Circuits and Systems II*, vol. 43, no. 10, pp. 689-702, 1996.

[6] S. M. Nowick et al., "Speculative Completion for the Design of High-Performance Asynchronous Dynamic Adders," in *Proc. 3rd Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC '97)*, pp. 210-223, 1997.

[7] A. K. Verma, P. Brisk, and P. Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design," in *Proc. of Design, Automation and Test in Europe (DATE) 2008*, pp. 1250-1255, 2008.