EasyChair Preprint

№ 7323

An Alternative Approach to Rounding Issues in Precision Computing with Accumulators, with Less Memory Consumption: a Proposal for Vectorizing a Branched Instruction Set

Roy Gulla

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

January 10, 2022

An Alternative Approach to Rounding Issues in Precision Computing with Accumulators, with less Memory Consumption: A Proposal for Vectorizing a Branched Instruction Set.

Roy P. Gulla rgullape@gmail.com

***Abstract*— Recent developments in numerical formatting have introduced a new system and a new emphasis on the use of accumulators for numerical computation. There also has been a recent development in this Posits numerical system, designed by John Gustafson, utilizing logarithmic bases. Here an alternative in light of these new developments is presented in a way to incorporate a design feature of Gustafson's format which negates the need for fractional bits in his system. One possible application for implementation of the formatting is also mentioned, which enhances the usage of instruction level parallelism in the arithmetic circuit design.**

1.1 Introduction

Low level hardware designs are rapidly undergoing a major sea level change in response to the new emerging technology surrounding quantum computing. In order to control for some of the inherent volatility in these new products, new formatting for precision computing is taking shape with several performance constraints enacted in order to limit the nearly unlimited bounds for power consumption in these new devices.

Some of the most important constraints are hardly new considerations, but one of them in particular, word size, can now be controlled for more easily, sort of ironically, due to a particularly recent implementation made possible by these new technologies.

In this new development, described in the third section below, a sort of folding of layers of bit streams is pretty clearly detailed as the building block for multiple layers being architecturally networked, or even streamed together. The blocks being connected though are words, and the words are being connected at the bit level.

Very recently, John Gustafson's POSIT formatting has found some implementations in a square root and divide circuit described in [2],and a new standard creation is underway to directly implement the direct dot product as a fifth arithmetic operation. In the first of these developments, power consumption is highlighted as a primary interest of the application designers.

Even more recently a proposed application of the minposis subset of Gustafson's posits has been described where a word the size of a single minposis bit is repeatedly read, or weighted, in a small arithmetic circuit consisting of the additional operations of adding and multiplying.

In this arithmetic circuit design described in [3] these repeated, or branched add and multiply instructions, lead to an expansion of the minposis value in a finite sum approximation of a non-exactly representable value in binary.

In the first portion of the paper the author will describe this proposed arithmetic circuit in light of the new quantum dot cellular automata application implementation with one-bit full adders described in [1], in a way that the instruction set is not in any way bloated with extraneous operands (i.e. store and/or load). In this portion a design tactic for instruction set architectures already implemented in a previous MIPS processor will be re-implemented with very few optimization flags enacted, so as to not increase the level of token matching in the operand matching stage of compilation performed in dataflow or even superscalar architectures.

In the final two sections of this paper, an actual prototyped package, with minimally accessible instruction sets for global registers, will be introduced as a vectorized computer architectural representation of the minposis subset of Gustafson's Posit formatting.

In the interim a brief digression will ensue that will follow the mathematical developments in low level numerical formatting that have lead to the use of these newer hardware designs for the improvement of computing precision.

1.2 The Problem of Rounding with Variable and Large Word Sizes

This new proposed implementation of certain rational numbers avoids the one issue that is common to all of the architectures.mentioned above, storage and retrieval of the correct values. Simply put, the difficulties in the rounding stage of computation can be exacerbated in vectorized machines where the mixing of different levels of precision can lead to denormalization of values. As implementations of these numerical types go through layers of architecture to higher levels, what once was a double type can become mixed with integer and decimal types, and astonishingly, even hexidecimal types when these are perpetuated from the program test registers at lower levels. When designers of higher level programs seek to implement high performance compute nodes and these lower level gates, their design choices teeter dangerously between implementing massive amounts of compute nodes, and their inherent compute power, and incorporating them in re-usable data structures in the spirit of sharing these compute nodes' data with other interested computing clients. And anytime when global access registers are implemented, which necessitates the use of general purpose registers, either data corruption, or performance fall-off becomes an issue. Accordingly, here the design choice is to avoid altogether the use of global general purpose

registers, and in particular to avoid the implementation of the proposed instruction sets in any higher level language module, and instead pursue the course of instruction level parallelism. This both enables the avoidance of global general purpose registers, and implements more local and effective usage of memory, making global memory registers unnecessary. And as architectures trend more and more towards multithreaded ones, the larger blocks of instructions which are mapped into ISBs should be grouped together at the earliest point in the control flow graph anyhow (i.e. in the closest sequential proximity to each other in the code block). In the case of dataflow architectures where token matching is a crucial part of the compilation process, this can prove vital to processor performance.

1.3 The Number Rings of these Non-Exactly Representable Numbers

Certain subunitary numbers exhibit certain properties which make them primary candidates for the alternative arithmetic circuit design proposed here. Particularly, those numbers which can be represented in the following geometric series form fitting the proposed formatting:

$$j \in \mathrm{N} \sum \; n \; /1 - n$$

where n is the jth binary number. [3] gives a detailed description of the expansions of these non exactly representable numbers. And although these expansions are ideally suited for those of the form given above, the use of weights in these expansions also makes a much larger class of subunitary numbers expressible in these binary expansions.[3] A simple example from [3] is shown below:

Table I. Expansion in Geometric Series of $1/2^n$

Decimal Value.	**.333333...(1/3)**
Alternate Expanded form, in as much binary as possible->	**1/4 + 1/12 = 1/4(1+1/3) =1/4(1+1/4(1+1/3)) =1/4(1+1/4(1+1/4(1+1/3))) etc.**
Floating Term	**1+1/3 = 4/3**

The use of repeated terms(i.e. bits) in these circuits hints at the idea of the repeated use of bit strings in representing a very small subset of the above subunitaries with certain cyclic properties. The numbers exhibiting these cyclic properties can be represented as shown below:

$$1/x^2 - y^3 \qquad x, y \in 0, 1 \; mod \; 9$$

These numbers, and these alone, are recommended for representation in either hexidecimal or other constant character formatting to limit the chances of corruption when the bit strings are repeatedly written into memory offsets.

2.1 An Architecture Which May Exploit Instruction Level Parallelism

Implementing one bit flags or tokens in registers is nothing new. Here a recent architecture has been shown as an effective implementation of a quantum dot cellular automata showing promise for the use of one bit full adders put to use in a future CMOS circuit design, where full bit adders have already been in use.

In this architecture, one bit full adders are implemented in a multi gate input- single gate output design. Below is a depiction of the idea behind a sum-carry adder circuit, implemented with a one bit full adder, which is a primary candidate for implementing the compound arithmetic circuit proposed in this paper. Although the diagram below depicts a full word with two bits summed in the adder, and the carry propagating forward, the proposal in [1] is more for a six -layer cellular design, where the polarization of the input gates needs to be carefully read at each gate (three in total) of each level. Here for simplicity, and for the purposes of this paper, we are merely interested in the potential use of these newer proposed hardware models as simple well known CMOS circuit designs, once technology has made such advances possible.

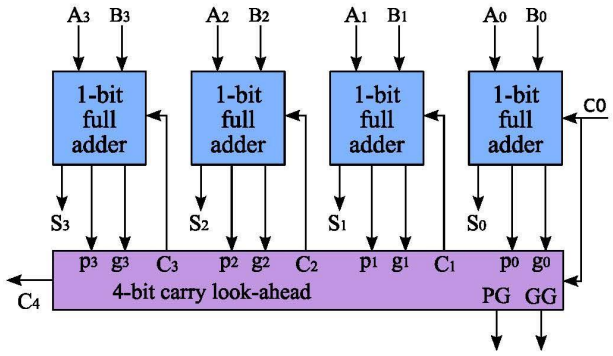

Figure 1 The eventual simplified design of the above described arithmetic circuit using the QCA architecture, as referenced in [1].

2.2 A Description of the Program Template

As is demonstrated in the architectural hardware design briefly referenced above, branched instructions, depending on the size of the code blocks being implemented, will not necessarily lead to greater sized code generation, and here the intent is not to branch anywhere outside our current block of instructions. The proposition is to generate branched, or "compound" machine level instructions. Generally put, the design's purpose is to make use of an SIMD architecture at the lowest possible level by incorporating a very parallelized instruction set, at the finest granular level.

Although consideration has been given for a proposal to generate a new pragma directive, this would be far too specialized and intensive of an effort, and in order to keep the linkage stage as simple as possible, the decision was made to simply incorporate a code design already in place in some legacy and most newer arm neon processors. There were several criteria used in making this choice, but the one most heavily considered was that of a reusable sublibrary header which would not fall under a large dependency chain, creating the need for losing unnecessary cycles of processor time during the compilation and linking stages.

The general idea of a program template is displayed below in the following section.

AN EXISTING MIPS BRANCHED INSTRUCTION SET FOR A COMPOUND ARITHMETIC CIRCUIT

```
#define D0(X) X
#define D1(X) X "\n\t" X
#define D2(X) D1 (D1 (X))
#define D3(X) D2 (D1 (X))
#define D4(X) D2 (D2 (X))
#define D5(X) D4 (D1 (X))
#define D6(X) D4 (D2 (X))
#define D7(X) D4 (D2 (D1 (X)))
#define D8(X) D4 (D4 (X))
#define D9(X) D8 (D1 (X))
#define D10(X) D8 (D2 (X))
#define D11(X) D8 (D2 (D1 (X)))
#define D12(X) D8 (D4 (X))
#define D13(X) D8 (D4 (D1 (X)))
#define D14(X) D8 (D4 (D2 (X)))
```

Figure 2. DN(X) generates 2**N copies of asm instruction X. Here for our purposes the instruction will be exclusively add and/or mlt. So, for example, we implement D6(X) D4(D2(X)) when X is add, and D4 is add instruction twice, or multiply("mlt") 2*operand (and, similarly, D2 is the add instruction twice).

For the multiply instruction, utilizing the traditional bitshift approach as in Intel processors, but this time to shift right for what is essentially a divide (i.e. multiply by 1/16, as these are minposis bit values) simply define our header file macro as follows:

```
#define D() asm volatile (" %d0, %1.d[0]"    \
            : "=w"(shrl))    //
```

Figure 3a. Because of the use of general purpose register which is what will be the case during testing, d[0] must be offset to the correct bit inside of D1(X), D2(X), D3(X), etc. etc.

```
#define D2(X) D1 (D1 (X) D(D(D())) )
```

Figure 3b. So for 3 increments from base address, the macros are shown here.

Then all subsequent macros, D3, D4, etc. will propagate the multiplication by minposis (or division) factor up the iterative scheme.

The section to follow is simply included to show an initialization test of the above macro and precompilation directives portion of the header file.

Since we are focusing on localized registers for our threads, any sort of buffer mapping will not be necessary at this stage of development. So we would make the width as wide as might be necessary on an architecture which might not perform any peeling or other such optimizations of vectorized instruction sets. Here, simply ensuring alignment with register widths is imperative, i.e. ensuring no split lines happen(as can occur, historically in x86 processors). But this is outside of the scope of the current paper. Here, the focus is simply on not implementing shared registers, at least at this layer. We are emphasizing ILP compilation, and by increasing the instruction size inside of blocks, parallelism can be exploited here, even when branching occurs. In fact, parallelism is being enhanced here because of this vectorized model of instructions.

In the following section, an example is given of how this vectorization, or perhaps better termed, instruction binding, can occur at a higher level in the compiler linker chain.

2.3 A Proposed Implementation which Avoids Higher Level Compilation and Global Namespaces

One of the biggest hurdles to instruction level parallelism is register alignment and/or spilling. So again, however fine grained we design our proposed parallelism, register overwrites and/or data corruption will kill performance of a precision computing superarchitecture.

In an effort to show how even vectorized models may take advantage of smaller word sizes, we can assign a word with one set bit (in the context of full adder circuits)in the initialization stages- much as making only one component of a vector exposed during compilation and link time improves performance and limits chances for data corruption by exposing more components to register reads and writes.

We need only access minimal amounts of components during vectorization by an ILP compiler, as described in [6] below, so the vectorized versions of the following numerics is already perfectly designed for streamlining this process, as they themselves are multi-dimensional vectors of binary bits. The goal in this intermediate level program is to create threads where registers are "local" to each one. In order to avoid the data corruption inherent when multi-threaded processes share registers, we employ local structures and only implement them in strongly typed applications, and even then not in C++ global namespaces.

A SUBHEADER W/OUT EXTERN DIRECTIVES

```
typedef struct minposis4x4x2_t
{
  int8x8_t val[2];
} int8x8x2_t;

typedef struct minposis4x2x2_t
{
  int8x16_t val[2];
} int8x16x2_t;
```

```
typedef struct minposis4x1x2_t
{
  int16x4_t val[2];
} int16x4x2_t;

typedef struct minposis4x8x2_t
{
  int16x8_t val[2];
} int16x8x2_t;

typedef struct minposis4x16x2_t
{
  int32x2_t val[2];
} int32x2x2_t;

typedef struct minposis4x32x2_t
{
  int32x4_t val[2];
} int32x4x2_t;

typedef struct minposis4x64x2_t
{
  int64x1_t val[2];
} int64x1x2_t;
```

Figure 4.. The above code segment is based on the arm_neon.h header file auto generated for the arm-neon architecture. Minposis here refers to the subunitaries of John Gustafson's Unum Posits formatting [4]. The appropriate minposis types can be set as global typedefs in pre-compiler directives. Since doing this here would preclude assuming authorship for the above code, this is outside of the scope of this paper.

3. Suggestions for Future Development of the Program Template

As the proposed sublibrary header file depicted in the second section of this paper contains non-initialized global variables, the obvious first suggestion here is to incorporate them in typedef structure definitions, much as int and double have their own unique identifiers in any strongly typed language, as the language shown here is (C/C++).

And although no specific recommendation is made for the creation of any new pragma directives, which might create the case for a more direct hardware implementation with specific alignment and/or data width requirements, that possibility has been alluded to in the first part of this paper.

References

1. Angizi, Shaahin, , Fartash, Mehdi, Sarmadi, Soheil, Sayedsalehi, Samira, (2016) "A Structured Ultra Dense QCA One-Bit Full Adder Cell" Quantum Matter 5(1):118-123 .
2. Cheng, S. Liang,F. Liang, J., Xiao, F., Wu, B Zhang, G. "Posit Arithmetic Hardware Implementations with the Minimum Cost Divider and Square Root." Electronics 2 October 2020
3. Gulla, Roy P. "Two Alternative Approaches to Rounding Issues in Precision Computing with Accumulators, with Less Memory Consumption." Lecture Notes in Engineering and Computer Science:
   Proceedings of The International MultiConference of Engineers and Computer Scientists 2021,
   20-22 October, 2021, Hong Kong, pp 240-243
4. Gustafson, John. (2015). The End of Error: Unum Computing. 10.1201/9781315161532.
5. Koenig, Jack & Biancolin, David & Bachrach, Jonathan & Asanovic, Krste. (2017). A Hardware Accelerator for Computing an Exact Dot Product. 114-121. 10.1109/ARITH.2017.38.
6. Kumar, Rajendra, PK Singh (2011). "A New Compiler for Space-Time Scheduling of ILP Processors", in Proceedings of International Journal of Computer and Electrical Engineering, Vol. 3, No. 4, August 2011.