

FAST BRANCH-FREE VECTOR DIVISION COMPUTATION

FIELD

[0001] The present disclosure generally relates to the field of computing. More particularly, an embodiment of the invention generally relates to techniques for fast branch-free vector division computation.

BACKGROUND

[0002] Compared to other simple arithmetic operations, hardware implementations for division operations have been quite slow, for example, due to larger latency. Some speedup could be achieved in vector cases due to various kind of parallelism existence on modern architectures, such as via SIMD (Single-Instruction, Multiple-Data) parallelism, superscalar, and out-of-order execution. For example, reciprocal approximation with further Newton-Raphson refinement iterations method (such as discussed at http://en.wikipedia.org/wiki/Newton%E2%80%93Raphson_method) generally works well for single precision (SP) case providing up to a two-fold speedup over hardware division operation in some implementations. However, this approach loses most of its benefits on double precision (DP) side because of absence of double

precision reciprocal operation in current SSE architectures. Consequently, additional DP to SP and SP to DP conversions may need to be performed, along with exponent field manipulations. Further, the SP and DP above-described approximations generally require special processing of denominator with infinite (INF) or zero values, reducing parallelism and reducing potential performance gains.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] The detailed description is provided with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The use of the same reference numbers in different figures indicates similar or identical items.

[0004] Fig. 1 illustrates a flow diagram of a method according to an embodiment of the invention.

[0005] Figs. 2A-2C illustrate pseudo code segments, which may be used in some embodiments.

[0006] Fig. 3 illustrates a block diagram of a fast vector division, according to an embodiment.

[0007] Figs. 4 and 5 illustrate block diagrams of embodiments of computing systems, which may be utilized to implement some embodiments discussed herein.

DETAILED DESCRIPTION

[0008] In the following description, numerous specific details are set forth in order to provide a thorough understanding of various embodiments. However, various embodiments of the invention may be practiced without the specific details. In other instances, well-known methods, procedures, components, and circuits have not been described in detail so as not to obscure the particular embodiments of the invention. Further, various aspects of embodiments of the invention may be performed using various means, such as integrated semiconductor circuits (“hardware”), computer-readable instructions organized into one or more programs (“software”), or some combination of hardware and software. For the purposes of this disclosure reference to “logic” shall mean either hardware, software (including for example micro-code that controls the operations of a processor), or some combination thereof.

[0009] Reference in the specification to “one embodiment” or “an embodiment” means that a particular feature, structure, or characteristic described in connection with the embodiment may be included in at least an implementation. The appearances of the phrase “in one embodiment” in various places in the specification may or may not be all referring to the same embodiment.

[0010] Also, in the description and claims, the terms “coupled” and “connected,” along with their derivatives, may be used. In some embodiments of the invention, “connected” may be used to indicate that two or more elements are in direct physical or electrical contact with each other. “Coupled” may mean that two or more elements are in direct physical or electrical contact. However, “coupled” may also mean that two or more elements may not be in direct contact with each other, but may still cooperate or interact with each other.

[0011] Some of the embodiments discussed herein may provide improved performance for double precision division/inversion vector computations, e.g., without requiring branches or special actions which were previously necessary. The vector division computations may be performed on SIMD computing platforms. Generally, SIMD is a technique employed to achieve data level parallelism. In particular, multiple data may be processed in multiple corresponding lanes of an SIMD vector processor (such as processors 402 and 502/504 of Figs. 4 and 5, respectively) in accordance with a single instruction.

[0012] In some implementations, only one division operation is performed for several inversions. Taking the following as an example (proposed by I. I. Zavarzin, V. F. Kuryakin, V. V. Lunev, D.M. Obuvalin, V. G. Ryzhih in “Optimizatsiya Vychislenij Vektornyh Funktsyj.” VANT. ser. Matematicheskoe modelirovanie fizicheskikh protsessov. 1997. Vol. 4 (Russian language magazine)):

$$\frac{1}{x_1} = x_2 \cdot x_3 \cdot x_4 \cdot R,$$

$$\frac{1}{x_2} = x_1 \cdot x_3 \cdot x_4 \cdot R,$$

$$\frac{1}{x_3} = x_1 \cdot x_2 \cdot x_4 \cdot R,$$

$$\frac{1}{x_4} = x_1 \cdot x_2 \cdot x_3 \cdot R,$$

$$\text{where } R = \frac{1}{x_1 \cdot x_2 \cdot x_3 \cdot x_4}$$

[0013] The inversion of every argument may be computed using three additional multiplications by R and three other arguments which is generally faster than four hardware divisions that involve large latency and large throughput value. More specifically, for the general case of N values, the next maximum possible performance gain estimation for this technique (D , M –throughput values for Division and Multiplier, respectively) may be:

$$\text{gain} = \frac{D}{\frac{D}{N} + \frac{(N-1) \cdot M}{N} + (N-1) \cdot M}$$

[0014] A weakness of this approach lies in high possibility to encounter overflow or underflow for the product $x_1 \cdot x_2 \cdot x_3 \cdot x_4$ because of argument exponents variety which may result in incorrect outputs for the whole four-arguments bundle.

For example, if $x_1 = 0.0$, but $x_2 \neq 0.0$, $x_3 \neq 0.0$, $x_4 \neq 0.0$, then $R = INF$ and three of

four results are incorrect: $\frac{1}{x_2} = \frac{1}{x_3} = \frac{1}{x_4} = INF$

[0015] To address this problem, there needs to be a guarantee that the sum of all input exponents is not going to cause underflow or overflow. This makes the work interval quite narrow and requires argument comparisons with possible branches for special cases that cannot be processed properly in the main path. In an embodiment, the above-mentioned problems may be addressed by argument scaling and reconstruction.

[0016] More specifically, **Fig. 1** illustrates a flow diagram of a method 100 in accordance with one embodiment. At an operation 102, one or more arguments (e.g., x_1, x_2 , etc.) are received. The arguments may be represented as $x = (-1)^s b^n \cdot f$, where the terms are defined as follows:

$s = \{0,1\}$ – sign for “s” in $(-1)^s$,

b – base (e.g., for binary case $b = 2$),

n – exponent ($E_{\max} \leq n \leq E_{\min}$), where $E_{\max}$ and $E_{\min}$ respectively refer

to the minimum and maximum exponents for corresponding data type according to ANSI/IEEE Std 754-2008.

$f = 2^0 + f_1 \cdot 2^{-1} + f_2 \cdot 2^{-2} + \dots + f_p \cdot 2^{-p} = 1.f_1 f_2 \dots f_p B$ – mantissa

$(1 \leq f < 2)$.

[0017] The inversion can be determined: $\frac{1}{x} = \frac{1}{(-1)^s 2^n \cdot f} = (-1)^s \cdot 2^{-n} \cdot \frac{1}{f}$.

[0018] Assume that (x_1, x_2, x_3, x_4) are input arguments where $x_i = (-1)^{s_i} 2^{n_i} \cdot f_i$ for $i = \overline{1,4}$. These are scaled at operation 104 by 2^{0_i} as $z_i = 1.0 \cdot f_i = 2^0 \cdot f_i$. The scaled arguments are multiplied at operation 106 (e.g., $M = z_1 z_2 z_3 z_4$) and the multiplication is inversed at operation 108 (e.g., $R = 1/M$). For example, every $\frac{1}{z_i}$ for $i = \overline{1,4}$ is calculated by:

$$\frac{1}{z_i} = \frac{\prod_{j \neq i} z_j}{z_1 z_2 z_3 z_4}$$

[0019] At an operation 110, the result is reconstructed as $\frac{1}{x_i} = (-1)^{s_i} \cdot 2^{m_i} \cdot \frac{1}{z_i}$

where $m_i = \sum_{j \neq i} n_j - \sum_{j=1,4} n_j = -n_i$, in an embodiment, by an “insertion” of negated input exponent 2^{-n_i} and source sign into $\frac{1}{z_i}$.

[0020] This approach provides sufficient accuracy and processes Institute of Electrical and Electronics Engineers, Inc. (IEEE) special values within the main path. As much as $z_i \in [1,2)$, no results with over/under-flow may occur during

computation, e.g., a product of any two values $z_i \cdot z_j \in [1,4)$, of any three values $z_i \cdot z_j \cdot z_k \in [1,8)$, of all four arguments $z_1 \cdot z_2 \cdot z_3 \cdot z_4 \in [1,16)$ and $\frac{1}{z_i} \in (0.5,1]$, etc.

[0021] Furthermore, every multiplication may be done by rounding to working precision with error as much as 0.5 ulp (Unit in the Last Place or Unit of Least Precision; see, e.g., <http://en.wikipedia.org/wiki/Ulp>; see, also, e.g., “On the definition of ulp(x)” by Jean-Michel Muller, INRIA Technical Report 5504), so the error of calculating $z_1 \cdot z_2 \cdot z_3 \cdot z_4$ will not be higher than $3 \cdot 0.5 = 1.5$ ulp.

[0022] In an embodiment, to find $\frac{1}{z_i} = \frac{\prod_{j \neq i} z_j}{z_1 z_2 z_3 z_4}$ another three multiplications

may be used with additional $3 \cdot 0.5 = 1.5$ ulp and one inversion which is error-free this case. Thus, error of the result $\frac{1}{z_i}$ will be less or equal to $(3+3) \cdot 0.5 = 3.0$ ulp.

Final reconstruction $\frac{1}{x_i} = (-1)^{s_i} \cdot 2^{m_i} \cdot \frac{1}{z_i}$ does not add any additional errors due to

IEEE floating point numbers representation. Resulting 3.0 ulp is within requirements for ICL (Intel Compiler), SVML (Short Vector Math Library), MKL (Math Kernel Library) and IPP (Intel Performance Primitives) vector math libraries LA (Low Accuracy) default flavor – up to four ulp that corresponds to two incorrect mantissa bits which is enough for large majority of applications.

[0023] Furthermore, even though a case with a four-value bundle is discussed herein, these techniques could be applied to any bundle size.

[0024] Figs. 2A to 2C illustrate pseudo codes to perform fast branch-free vector division computation, according to some embodiments. More specifically, Fig. 2A illustrates arguments, results, casts, and other definitions. Fig. 2B illustrates pseudo code for scaling arguments and saving original exponent values, along with multiplication of scaled arguments, e.g., as discussed with reference to Fig. 1. Fig. 2C illustrates pseudo code for result reconstruction. In some embodiments, for full division $r[i]$ may be multiplied by numerators.

[0025] Fig. 3 illustrates a block diagram of a fast vector division, according to an embodiment. As shown, exponent and fraction portions of a floating point number 302 are separately handled 304. While in the shown example, the number (labeled as “arg”) is indicated within a shown range, other ranges may be used depending on the implementation, processor, datapath width, etc. Moreover, to obtain the inversion of an argument, inversion of exponent 306 and inversion of mantissa 308 may be determined separately, and then the final result is determined based on their product. In one embodiment, the inversion of exponent may be based on a sign replacement (310 and 312), such as discussed with reference to Figs. 1-2C. This may be done via integer operations and without any accuracy loss in some embodiments. To inverse the fraction (314 and 316), it may be represented as a

floating point number with zero exponent. Lastly, the final result is determined 318 based on the multiplication of the inverted fraction and exponent (which is considered to be a precise result since at least one of multipliers has a zero fraction).

[0026] **Fig. 4** illustrates a block diagram of an embodiment of a computing system 400. In various embodiments, one or more of the components of the system 400 may be provided in various electronic devices capable of performing one or more of the operations discussed herein with reference to some embodiments of the invention. For example, one or more of the components of the system 400 may be used to perform the operations discussed with reference to Figs. 1-3, e.g., by generating values corresponding to division/inversion by enhanced performance through use of SIMD, etc. in accordance with the operations discussed herein. Also, various storage devices discussed herein (e.g., with reference to Figs. 4 and/or 5) may be used to store data, operation results, etc. In one embodiment, data associated with operations of method 300 of Fig. 3 may be stored in memory device(s) (such as memory 412 or one or more caches (e.g., L1 caches in an embodiment) present in processors 402 of Fig. 4 or 502/604 of Fig. 5). These processors may then apply the operations discussed herein for fast division/inversion (such as one or more of the operations of Figs. 1-3). Accordingly, in some embodiments, processors 402 of Fig. 4 or 502/604 of Fig. 5 may be vector processors that are capable of supporting SIMD operations.

[0027] Moreover, the computing system 400 may include one or more central processing unit(s) (CPUs) 402 or processors that communicate via an interconnection network (or bus) 404. The processors 402 may include a general purpose processor, a network processor (that processes data communicated over a computer network 403), or other types of a processor (including a reduced instruction set computer (RISC) processor or a complex instruction set computer (CISC)). Moreover, the processors 402 may have a single or multiple core design. The processors 402 with a multiple core design may integrate different types of processor cores on the same integrated circuit (IC) die. Also, the processors 402 with a multiple core design may be implemented as symmetrical or asymmetrical multiprocessors. Additionally, the processors 402 may utilize an SIMD architecture. Moreover, the operations discussed with reference to Figs. 1-3 may be performed by one or more components of the system 400.

[0028] A chipset 406 may also communicate with the interconnection network 404. The chipset 406 may include a memory control hub (MCH) 408. The MCH 408 may include a memory controller 410 that communicates with a memory 412. The memory 412 may store data, including sequences of instructions that are executed by the CPU 402, or any other device included in the computing system 400. In one embodiment of the invention, the memory 412 may include one or more volatile storage (or memory) devices such as random access memory (RAM), dynamic RAM

(DRAM), synchronous DRAM (SDRAM), static RAM (SRAM), or other types of storage devices. Nonvolatile memory may also be utilized such as a hard disk. Additional devices may communicate via the interconnection network 404, such as multiple CPUs and/or multiple system memories.

[0029] The MCH 408 may also include a graphics interface 414 that communicates with a display 416. The display 416 may be used to show a user results of operations associated with the fast division/inversion discussed herein. In one embodiment of the invention, the graphics interface 414 may communicate with the display 416 via an accelerated graphics port (AGP). In an embodiment of the invention, the display 416 may be a flat panel display that communicates with the graphics interface 414 through, for example, a signal converter that translates a digital representation of an image stored in a storage device such as video memory or system memory into display signals that are interpreted and displayed by the display 416. The display signals produced by the interface 414 may pass through various control devices before being interpreted by and subsequently displayed on the display 416.

[0030] A hub interface 418 may allow the MCH 408 and an input/output control hub (ICH) 420 to communicate. The ICH 420 may provide an interface to I/O devices that communicate with the computing system 400. The ICH 420 may communicate with a bus 422 through a peripheral bridge (or controller) 424, such as

a peripheral component interconnect (PCI) bridge, a universal serial bus (USB) controller, or other types of peripheral bridges or controllers. The bridge 424 may provide a data path between the CPU 402 and peripheral devices. Other types of topologies may be utilized. Also, multiple buses may communicate with the ICH 420, e.g., through multiple bridges or controllers. Moreover, other peripherals in communication with the ICH 420 may include, in various embodiments of the invention, integrated drive electronics (IDE) or small computer system interface (SCSI) hard drive(s), USB port(s), a keyboard, a mouse, parallel port(s), serial port(s), floppy disk drive(s), digital output support (e.g., digital video interface (DVI)), or other devices.

[0031] The bus 422 may communicate with an audio device 426, one or more disk drive(s) 428, and a network interface device 430, which may be in communication with the computer network 403. In an embodiment, the device 430 may be a NIC capable of wireless communication. Other devices may communicate via the bus 422. Also, various components (such as the network interface device 430) may communicate with the MCH 408 in some embodiments of the invention. In addition, the processor 402 and the MCH 408 may be combined to form a single chip. Furthermore, the graphics interface 414 may be included within the MCH 408 in other embodiments of the invention.

[0032] Furthermore, the computing system 400 may include volatile and/or nonvolatile memory (or storage). For example, nonvolatile memory may include one or more of the following: read-only memory (ROM), programmable ROM (PROM), erasable PROM (EPROM), electrically EPROM (EEPROM), a disk drive (e.g., 428), a floppy disk, a compact disk ROM (CD-ROM), a digital versatile disk (DVD), flash memory, a magneto-optical disk, or other types of nonvolatile machine-readable media that are capable of storing electronic data (e.g., including instructions). In an embodiment, components of the system 400 may be arranged in a point-to-point (PtP) configuration such as discussed with reference to Fig. 5. For example, processors, memory, and/or input/output devices may be interconnected by a number of point-to-point interfaces.

[0033] More specifically, **Fig. 5** illustrates a computing system 500 that is arranged in a point-to-point (PtP) configuration, according to an embodiment of the invention. In particular, Fig. 5 shows a system where processors, memory, and input/output devices are interconnected by a number of point-to-point interfaces. The operations discussed with reference to Figs. 1-4 may be performed by one or more components of the system 500.

[0034] As illustrated in Fig. 5, the system 500 may include several processors, of which only two, processors 502 and 504 are shown for clarity. The processors 502 and 504 may each include a local memory controller hub (MCH) 506 and 508 to

couple with memories 510 and 512. The memories 510 and/or 512 may store various data such as those discussed with reference to the memory 412 of Fig. 4.

[0035] The processors 502 and 504 may be any suitable processor such as those discussed with reference to the processors 402 of Fig. 4. The processors 502 and 504 may exchange data via a point-to-point (PtP) interface 514 using PtP interface circuits 516 and 518, respectively. The processors 502 and 504 may each exchange data with a chipset 520 via individual PtP interfaces 522 and 524 using point to point interface circuits 526, 528, 530, and 532. The chipset 520 may also exchange data with a high-performance graphics circuit 534 via a high-performance graphics interface 536, using a PtP interface circuit 537.

[0036] At least one embodiment of the invention may be provided by utilizing the processors 502 and 504. For example, the processors 502 and/or 504 may perform one or more of the operations of Figs. 1-4. Other embodiments of the invention, however, may exist in other circuits, logic units, or devices within the system 500 of Fig. 5. Furthermore, other embodiments of the invention may be distributed throughout several circuits, logic units, or devices illustrated in Fig. 5.

[0037] The chipset 520 may be coupled to a bus 540 using a PtP interface circuit 541. The bus 540 may have one or more devices coupled to it, such as a bus bridge 542 and I/O devices 543. Via a bus 544, the bus bridge 543 may be coupled to other devices such as a keyboard/mouse 545, the network interface device 530

discussed with reference to Fig. 5 (such as modems, network interface cards (NICs), or the like that may be coupled to the computer network 403), audio I/O device, and/or a data storage device 548. The data storage device 548 may store code 549 that may be executed by the processors 502 and/or 504.

[0038] In various embodiments of the invention, the operations discussed herein, e.g., with reference to Figs. 1-5, may be implemented as hardware (e.g., logic circuitry), software (including, for example, micro-code that controls the operations of a processor such as the processors discussed with reference to Figs. 4-5), firmware, or combinations thereof, which may be provided as a computer program product, e.g., including a tangible machine-readable or computer-readable medium having stored thereon instructions (or software procedures) used to program a computer (e.g., a processor or other logic of a computing device) to perform an operation discussed herein. The machine-readable medium may include a storage device such as those discussed herein.

[0039] Additionally, such tangible computer-readable media may be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in tangible propagation medium via a communication link (e.g., a bus, a modem, or a network connection).

[0040] Thus, although embodiments of the invention have been described in language specific to structural features and/or methodological acts, it is to be understood that claimed subject matter may not be limited to the specific features or acts described. Rather, the specific features and acts are disclosed as sample forms of implementing the claimed subject matter.

CLAIMS

1. A method performed in an apparatus which comprises a memory to store a plurality of data values corresponding to an SIMD (Single Instruction, Multiple Data) instruction, and a processor having a plurality of SIMD lanes, wherein each of the plurality of the SIMD lanes is to process one of the plurality of data stored in the memory in accordance with the SIMD instruction, the method comprising:
 - scaling a plurality of arguments representing floating point numbers to generate a plurality of corresponding scaled arguments;
 - multiplying the plurality of scaled arguments to generate a first value;
 - inverting the first value to generate a second value; and
 - reconstructing a plurality of results, wherein reconstructing each of the plurality of results is based on a multiplication of the second value with all of the plurality of scaled arguments except a corresponding one of the plurality of scaled arguments for which the each of the plurality of results is being reconstructed,
 - wherein the plurality of results correspond to inverted versions of the plurality of arguments.
2. The method of claim 1, wherein inverting the first value is performed by changing a sign of an exponent portion of the first value.
3. The method of claim 1, further comprising converting the floating point numbers of the plurality of arguments to respective integer values.
4. The method of claim 1, wherein scaling the plurality of arguments comprises scaling the plurality of arguments by 1.0.

5. The method of claim 1, further comprising storing generated values in a memory.
6. An apparatus comprising:
 - a memory to store a plurality of data values corresponding to an SIMD (Single Instruction, Multiple Data) instruction; and
 - a processor having a plurality of SIMD lanes, wherein each of the plurality of the SIMD lanes is to process one of the plurality of data stored in the memory in accordance with the SIMD instruction, wherein the processor is to:
 - scale a fraction portion of a first value of the plurality of data values represented by floating point numbers to generate a second value;
 - negate an exponent portion of the first value to generate a third value;
 - invert the second value to generate a fourth value; and
 - multiply the fourth value and the third value to generate an inverse version of the first value, wherein the exponent portion is to be negated by changing a sign of the exponent portion of the first value.
7. The apparatus of claim 6, wherein the processor is to determine the exponent portion and fraction portion of the first value.
8. The apparatus of claim 6, wherein the processor is to scale the fraction portions of the first value by 1.0 to generate the second value.
9. The apparatus of claim 6, wherein the processor is to convert the floating point numbers of the plurality of data values to respective integer values.
10. The apparatus of claim 6, wherein the memory comprises a cache.

11. The apparatus of claim 6, wherein the processor comprises one or more processor cores.
12. The apparatus of claim 6, wherein the processor is to cause storage of generated values in the memory.
13. The apparatus of claim 6, further comprising a display device to display the inverse version of the first value.
14. A computer-readable medium comprising one or more instructions that when executed on a processor configure the processor to perform one or more operations to:
 - scale a plurality of arguments representing floating point numbers to generate a plurality of corresponding scaled arguments;
 - multiply the plurality of scaled arguments to generate a first value;
 - invert the first value to generate a second value; and
 - reconstruct a plurality of results, wherein reconstructing each of the plurality of results is based on a multiplication of the second value with all of the plurality of scaled arguments except a corresponding one of the plurality of scaled arguments for which the each of the plurality of results is being reconstructed, wherein the plurality of results correspond to inverted versions of the plurality of arguments.
15. The computer-readable medium of claim 14, further comprising one or more instructions that when executed on a processor configure the processor to invert the first value by changing a sign of an exponent portion of the first value.
16. The computer-readable medium of claim 14, further comprising one or more instructions that when executed on a processor configure the processor to convert the floating point numbers of the plurality of arguments to respective integer values.

17. The computer-readable medium of claim 14, further comprising one or more instructions that when executed on a processor configure the processor to scale the plurality of arguments by 1.0.

18. The computer-readable medium of claim 14, further comprising one or more instructions that when executed on a processor configure the processor to store generated values in a memory.

19. The computer-readable medium of claim 14, further comprising one or more instructions that when executed on a processor configure the processor to multiply an inverted exponent portion and an inverted fraction portion of the plurality of arguments.

20. A method performed in an apparatus which comprises a memory to store a plurality of data values corresponding to an SIMD (Single Instruction, Multiple Data) instruction, and a processor having a plurality of SIMD lanes, wherein each of the plurality of the SIMD lanes is to process one of the plurality of data stored in the memory in accordance with the SIMD instruction, the method comprising:

scaling a fraction portion of a first value of the plurality of data values represented by floating point numbers to generate a second value;

negating an exponent portion of the first value to generate a third value;

inverting the second value to generate a fourth value; and

multiplying the fourth value and the third value to generate an inverse version of the first value, wherein the exponent portion is to be negated by changing a sign of the exponent portion of the first value.

21. A computer-readable medium comprising one or more instructions that when executed on a processor configure the processor to perform one or more operations to:

scale a fraction portion of a first value of the plurality of data values represented by floating point numbers to generate a second value;

negate an exponent portion of the first value to generate a third value;

invert the second value to generate a fourth value; and

multiply the fourth value and the third value to generate an inverse version of the first value, wherein the exponent portion is to be negated by changing a sign of the exponent portion of the first value.