# MULTIPLE-INPUT FLOATING POINT NUMBER PROCESSING METHOD AND APPARATUS, PROCESSOR AND COMPUTER DEVICE

## RELATED APPLICATION

**[0001]** This application claims priority to Chinese Patent Application No. 2021116348560, filed with the China Patent Office on December 29, 2021 and entitled “MULTIPLE-INPUT FLOATING POINT NUMBER PROCESSING METHOD AND APPARATUS, PROCESSOR AND COMPUTER DEVICE”, which is incorporated herein by reference in its entirety.

## FIELD OF THE TECHNOLOGY

**[0002]** This application relates to the technical field of data processing, in particular to a multiple-input floating point number processing method and apparatus, a processor, a computer device and a storage medium.

## BACKGROUND OF THE DISCLOSURE

**[0003]** With development of a computer technology, an artificial intelligence (AI) technology is also rapidly developing. In the technical field of AI, an AI algorithm is typically implemented through an AI processor. In the AI processor, a matrix operation unit is a core data processing device, and performance and computational power of the matrix operation unit directly determine the performance of the AI processor. In the matrix operation unit, a multiple-input floating-point operation unit is a key to determine the performance.

**[0004]** For a multiple-input floating-point operation mode in a conventional solution, a shifted data bit width will be very wide to realize a purpose that there is no intermediate precision loss. Therefore, a plurality of shifters with high bits are usually required to ensure that there is no intermediate precision loss. An excessive shift range causes significant hardware overhead, resulting in the processor needing to occupy more hardware resources.

## SUMMARY

**[0005]** Various embodiments of this application provide a multiple-input floating point number processing method and apparatus, a processor, a computer device and a storage medium.

**[0006]** According to various embodiments of this application, a multiple-input floating point number processing method is provided. The method includes the following steps:

**[0007]** acquiring a plurality of to-be-processed floating point numbers corresponding to a target task, and extracting an exponential value of an exponent part and a mantissa value of a mantissa part in each floating point number respectively;

**[0008]** sorting, according to a magnitude of the exponential value of each floating point number, the plurality of floating point numbers to obtain a sorting result, and allocating, based on the sorting result, a shifter for each floating point number from a plurality of shifters with different preset bits;

**[0009]** performing, for each floating point number, shift processing on the mantissa value of the corresponding floating point number through the shifter allocated for the floating point number to obtain a shift result; and

**[0010]** determining a floating point number processing result corresponding to the target task based on each shift result.

**[0011]** According to various embodiments of this application, a multiple-input floating point number processing apparatus is provided. The apparatus includes:

**[0012]** an acquiring module, configured to acquire a plurality of to-be-processed floating point numbers corresponding to a target task, and extract an exponential value of an exponent part and a mantissa value of a mantissa part in each floating point number respectively;

**[0013]** an allocating module, configured to sort, according to a magnitude of the exponential value of each floating point number, the plurality of floating point numbers to obtain a sorting result, and allocate, based on the sorting result, a shifter for each floating point number from a plurality of shifters with different preset bits;

**[0014]** a shifting module, configured to perform, for each floating point number, shift processing on the mantissa value of the corresponding floating point number through the shifter allocated for the floating point number to obtain a shift result; and

**[0015]** a determining module, configured to determine a floating point number processing result corresponding to the target task based on each shift result.

**[0016]** According to various embodiments of this application, a processor is provided. The processor includes:

**[0017]** at least one shifter with different preset bits, each shifter being allocated to be used for performing shift processing on a mantissa value of a mantissa part of one of a plurality of to-be-processed floating point numbers, and the shifter allocated to each floating point number being

determined according to a sorting result obtained by sorting exponential values of an exponent part of each floating point number; and

**[0018]** a logic processing unit, configured to perform logic processing on a plurality of shift results obtained by shift processing of each shifter to obtain a floating point number processing result.

**[0019]** According to various embodiments of this application, a computer device is provided. The computer device includes a memory and a processor, the memory storing a computer readable instruction, and the processor, when executing the computer readable instruction, implementing steps of the multiple-input floating point number processing method above.

**[0020]** According to various embodiments of this application, a computer readable storage medium is provided. The computer readable storage medium stores a computer program, the computer program, when executed by a processor, implementing steps of the multiple-input floating point number processing method above.

**[0021]** According to various embodiments of this application, a computer program product is provided. The computer program product includes a computer program, the computer program, when executed by a processor, implementing steps of the multiple-input floating point number processing method above.

**[0022]** Details of one or more embodiments of this application are provided in the accompanying drawings and descriptions below. Other features, objectives, and advantages of this application become apparent from the specification, the accompanying drawings, and the claims.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0023]** FIG. 1 is a schematic flowchart of a multiple-input floating point number processing method in one or more embodiments.

**[0024]** FIG. 2 is a schematic diagram of a format of a floating point number in one or more embodiments.

**[0025]** FIG. 3 is a schematic flowchart of a step of allocating a shifter in one or more embodiments.

**[0026]** FIG. 4 is a schematic flowchart of a step of shift processing in one or more embodiments.

**[0027]** FIG. 5 is a schematic flowchart of a step of shift processing in another one or more embodiments.

**[0028]** FIG. 6 is a schematic flowchart of a step of determining a floating point number processing result corresponding to a target task based on each shift result in one or more embodiments.

**[0029]** FIG. 7 is a schematic flowchart of a step of segmented compression processing in one or more embodiments.

**[0030]** FIG. 8 is a schematic diagram of a principle of segmented compression processing in one or more embodiments.

**[0031]** FIG. 9 is a schematic flowchart of determining a floating point number processing result corresponding to a target task based on a plurality of segmented compression results in one or more embodiments.

**[0032]** FIG. 10 is a schematic flowchart of a step of outputting a selection result by a selector in one or more embodiments.

**[0033]** FIG. 11 is a schematic flowchart of a step of performing normalization processing on a floating point number processing result in one or more embodiments.

**[0034]** FIG. 12 is a schematic flowchart of existing multiple floating point number addition calculation in one or more embodiments.

**[0035]** FIG. 13 is a schematic diagram of composition of a data bit width in one or more embodiments.

**[0036]** FIG. 14 is a schematic flowchart of application of this application to multiple floating point number addition calculation in one or more embodiments.

**[0037]** FIG. 15 is a structural block diagram of a multiple-input floating point number processing apparatus in one or more embodiments.

**[0038]** FIG. 16 is a structural block diagram of a processor in one or more embodiments.

**[0039]** FIG. 17 is a structural block diagram of a logic processing unit in one or more embodiments.

**[0040]** FIG. 18 is a diagram of an internal structure of a computer device in one or more embodiments.

**[0041]** In order to better describe and illustrate the embodiments and/or examples of the inventions disclosed here, reference may be made to one or more accompanying drawings. The additional details or examples used for describing the accompanying drawings are not to be

considered as limiting the scope of any of the disclosed invention, the currently described embodiments and/or examples, and best modes of these inventions understood currently.

## DESCRIPTION OF EMBODIMENTS

**[0042]** To make objectives, technical solutions, and advantages of this application clearer, the following further describes this application in detail with reference to the accompanying drawings and the embodiments. It is to be understood that the specific embodiments described here are only used for explaining this application, and are not used for limiting this application.

**[0043]** This application provides a multiple-input floating point number processing method and apparatus, a processor, a computer device, a storage medium and a computer program product. By optimizing an operation mode of a floating point number and a corresponding processing logic, not only the operation of the plurality of floating point numbers can be efficiently processed, but also an area of an AI processor can be effectively reduced by reducing an area of a shifter, a critical path in timing is reduced, and a master frequency of the AI processor is improved, so that a single chip applying the AI processor may provide higher computational power.

**[0044]** In some embodiments, as shown in FIG. 1, a multiple-input floating point number processing method is provided. This embodiment takes an example of the application of this method to a computer device for illustration. It may be understood that the computer device may be a terminal or a server. The server may be an independent physical server, or a server cluster or distributed system composed of the plurality of physical servers, or a cloud server that provides a cloud computing service. The terminal may be, but is not limited to, one or more of various personal computers, notebook computers, smartphones, tablet computers, IoT devices, portable wearable devices and the like. The IoT device may be one or more of a smart speaker, a smart television, a smart air conditioner, a smart vehicle-mounted device and the like. The portable wearable device may be one or more of a smart watch, a smart wristband, a headset device, and the like. Exemplarily, the multiple-input floating point number processing method provided in this embodiment of this application may be applied to a data processing device in the computer device, such as a processor and a sensor.

**[0045]** In this embodiment, the method includes the following steps:

**[0046]** step S102: Acquire a plurality of to-be-processed floating point numbers corresponding to a target task, and extract an exponential value of an exponent part and a mantissa value of a mantissa part in each floating point number respectively.

**[0047]** The target task refers to a computational processing task executed by the computer

device to achieve a certain goal. Computational processing includes, but is not limited to, mathematical operations such as addition, subtraction, multiplication or division. For example, the target task may be a computational processing task in a neural network training process, and includes, but is not limited to, a convolutional summation task or a similarity computational task. For another example, the target task may also be a cloud computing or distributed computing task, used for computing a plurality of pieces of data.

**[0048]** The floating point number is digital representation of a number belonging to a specific subset of rational numbers, and is used for approximate representation of any real number in a computer. Taking a current common floating point number format as an example, the format of the floating point number generally follows an IEEE binary floating point number arithmetic standard (ANSI/IEEE Std 754-1985, usually referred to as IEEE 754) formulated by a microprocessor standards committee (MSC).

**[0049]** The IEEE 754 standard specifies a specific standard for storing a decimal floating point number in a binary form in a computer memory, and formulates four ways to represent a floating point number value: a single-precision floating point number, a double-precision floating point number, an extended single-precision floating point number, and an extended double-precision floating point number.

**[0050]** As shown in FIG. 2, IEEE 754 represents the floating point number as a symbol, an exponent part, and a mantissa part. A symbol S occupies a most significant bit, and is used for representing that the floating point number is a positive number or a negative number; “0” represents that the floating point number is a positive number, and “1” represents that the floating point number is a negative number. An exponential value E (usually also called a level code) in the exponent part is represented by an offset code (also called a biased exponent or a biasing code). Taking a 32-bit single-precision floating point number as an example, a range of the exponential value E is 8bit, representing 0-255 exponential values. The exponential value is used for indicating a location of a decimal point. A mantissa value M of the mantissa part of the floating point number is represented by a source code. Similarly, taking a 32-bit single-precision floating point number as an example, a range of the mantissa value M is 24bit, which determines precision of the real number that can be represented by the floating point number. Due to the fact that a most significant bit of the source code must be a significant bit (i.e. must be 1), the most significant bit of the mantissa part is usually omitted (or hidden) in the computer memory to save a storage space. Therefore, 24bit information may be represented by the mantissa part of 23bit in the figure.

**[0051]** For simplicity and ease of understanding in description, the examples listed in the

following embodiments comply with the IEEE 754 standard, but are not to be understood as limiting the application scope of the embodiments of this application.

**[0052]** In some embodiments, the plurality of to-be-processed floating point numbers are stored in a memory. The memory may be an internal memory set in the computer device, or an external memory that is independent of the computer device and is in communication connection with the computer device.

**[0053]** Specifically, the computer device acquires two or more to-be-processed floating point numbers from the memory, and extracts the exponential value of the exponent part and the mantissa value of the mantissa part of each floating point number according to the format followed by the floating point number. The exponential value is used for subsequent sorting of the floating point numbers to determine the magnitude of the shifter allocated for each floating point number; and the mantissa value is a part used for specific shift processing.

**[0054]** Then taking the 32-bit single-precision floating point number as an example, the computer device extracts a numerical value corresponding to a $31^{st}$ bit (the most significant bit) as a value of a sign bit, extracts numerical values corresponding to a $30^{th}$ bit to a $23^{rd}$ bit as the exponential value of the exponent part, and extracts numerical values corresponding to a $22^{nd}$ bit to a 0 bit (the least significant bit) as the mantissa value of the mantissa part. As mentioned earlier, the computer device uses a representation of a most significant bit 1 of an implicit mantissa part to represent the mantissa value, so the bit extracted by the computer device is from the $22^{nd}$ bit to the 0 bit.

**[0055]** Step S104: Sort, according to a magnitude of the exponential value of each floating point number, the plurality of floating point numbers to obtain a sorting result, and allocate, based on the sorting result, a shifter for each floating point number from a plurality of shifters with different preset bits.

**[0056]** Because the exponential values of all the floating point numbers are different, a dimension of each floating point number may not be consistent. In order to calculate the floating point numbers, it is necessary to first unify the different floating point numbers into the same dimension. Here, shift processing of each floating point number is realized by setting the plurality of shifters, so that all the floating point numbers are located in the same dimension. The quantity of the shifters is determined based on the quantity of the floating point numbers. For example, the quantity of the shifters may be the same as the quantity of the floating point numbers. For another example, since a floating point number corresponding to a maximum value of the exponential value does not need to be subjected to shift processing, the quantity of the shifters may be the quantity of

the floating point numbers minus 1, so as to reduce hardware resources required to be consumed.

**[0057]** All the shifters are preset with different bits, and the preset bit for each shifter includes a maximum range of the mantissa value plus a maximum shiftable range of the shifter during shift processing. For example, a bit of a 50bit shifter is 50 bits, and since the mantissa value is up to 24 bits, the space occupied by the mantissa value is removed, and the maximum shiftable range w of the shifter is 26 bits. In a subsequent shift process, the shifter shifts the mantissa value within its maximum shiftable range.

**[0058]** In order to reduce hardware overhead as much as possible, in some embodiments, the different preset bits possessed by the plurality of shifters are all within a first preset range, and the preset bits of all the shifters are uniformly distributed within the first preset range. Specifically, the maximum value of the preset bits in the plurality of shifters is taken as the first preset range, and the shift ranges of the remaining shifters are all within the first preset range. The preset bits of all the shifters are uniformly distributed, including gradually increasing or gradually decreasing by a certain multiple, presenting as arithmetic progression or proportional progression. Exemplarily, a shifter with a q (e.g. q=24bit+wbit, w=26) bits, a shifter with b 2*w bits, a shifter with c 3*w bits, ..., and a shifter with x n*w bits are set in advance, where a value of n is the number of the floating point numbers minus 1.

**[0059]** For the purpose of differentiation, in this embodiment of this application, the range in which the preset bits of the shifter are located is referred to as the first preset range, and a range in which a numeric value of a compression ratio of a compressor is located is referred to as a second preset range. The terms “first” and “second” above are used in this application for describing different numeric value ranges, but these numeric value ranges are not to be limited by these terms. These terms are merely used for distinguishing one numeric value range from another numeric value range. For example, the first preset range may be referred to as the second preset range, and similarly, the second preset range may be referred to as the first preset range without departing from the scope of various described embodiments, but unless the context explicitly indicates otherwise, they do not refer to the same range. Similar situations include a first domain segment, a second domain segment, and a third domain segment, as well as first symbol identification and second symbol identification, as well as a first shift direction and a second shift direction, and so on.

**[0060]** Specifically, after acquiring the exponential value of each floating point number, the computer device sorts all the floating point numbers according to the magnitude of the exponential values of all the floating point numbers to obtain the sorting result. Exemplarily, the computer device sorts all the floating point numbers in an order of the exponential values from small to large,

so as to obtain the sorting result. For another example, the computer device sorts all the floating point numbers in an order of the exponential values from large to small, so as to obtain the sorting result. Since the floating point number with the maximum exponential value does not need to be subjected to shift processing, except for the floating point number with the maximum exponential value, the computer device allocates one shifter to each of the remaining floating point numbers in turn according to the obtained sorting result of the floating point numbers.

**[0061]** In the above embodiment, by allocating the different shifters for the different floating point numbers, compared with a situation in the related art that a plurality of shifters with the same bit are used for shift processing no matter the magnitude of each floating point number, the hardware overhead is significantly reduced, timing characteristics are good and efficiency is high.

**[0062]** In some other embodiments, the different preset bits possessed by the plurality of shifters may also be non-uniformly distributed within the first preset range. For example, within the first preset range, a preset bit of one shifter is set to be q bit, and the preset bits of the remaining shifters are n*w bits. For another example, within the first preset range, the preset bits of all the shifters gradually increase or gradually decrease exponentially. For yet another example, within the first preset range, the preset bit of each x shifter is the same, and the overall trend is gradually increasing or gradually decreasing.

**[0063]** Step S106: Perform, for each floating point number, shift processing on the mantissa value of the corresponding floating point number through the shifter allocated for the floating point number to obtain a shift result.

**[0064]** Specifically, for one floating point number, the computer device performs shift processing on the mantissa value of the floating point number within the preset bit of the shifter by using the shifter allocated for the floating point number to obtain the shift result of the floating point number. Since the floating point number with the maximum exponential value does not need to be subjected to shift processing, the computer device performs shift processing on the remaining floating point numbers by using the shifter allocated respectively to obtain the plurality of shift results.

**[0065]** Shift processing refers to shifting the mantissa value by a certain distance towards a certain shift direction. The shift direction includes left shift and right shift. The shifted bit in the shift processing is a difference between the floating point number and the maximum value of the floating point number. Exemplarily, for the floating point number A, the difference between its exponential value and the maximum exponential value is b, and then the computer device shifts the mantissa value of the floating point number A to the left or right by b bits. The maximum

exponential value is the maximum value among the exponential values of the plurality of floating point numbers.

**[0066]** Step S108: Determine a floating point number processing result corresponding to the target task based on each shift result.

**[0067]** Specifically, after shift is completed, the computer device may perform subsequent processing according to the specific target task. Specifically, according to the respective corresponding shift result of each floating point number, all the shift results are compressed in turn to obtain a compression result; and the computer device then performs post-processing on the compression result to obtain the floating point number processing result corresponding to the target task. Post-processing includes standardization processing, rounding processing and the like. For example, post-processing is used for performing normalization processing on the compression result, so that the compression result conforms to a format specified in IEEE 754, and then the final floating point number processing result is obtained by rounding.

**[0068]** In the above multiple-input floating point number processing method, the plurality of shifters with different preset bits are designed in advance. When processing the plurality of to-be-processed floating point numbers, the plurality of floating point numbers may be sorted according to the magnitude of the exponential value of each floating point number, thus the corresponding shifter is allocated for each floating point number from the plurality of shifters with different preset bits based on the sorting result, and the shift processing is performed on the plurality of floating point numbers by using the allocated shifter, so as to obtain the floating point number processing result corresponding to the target task based on the shift processing result. In this way, an idea of effective shift is introduced to effectively shift the mantissa value with lower sorting, and a shift range of the mantissa value with top sorting is small (even without shifting). Under the premise of ensuring that there is no intermediate precision loss, area overhead of the shifter is greatly reduced, thus saving the hardware overhead of the processor. Under the premise of limited hardware resources, efficiency and accuracy of floating point number processing can be well considered.

**[0069]** As mentioned above, in some embodiments, the quantity of the plurality of shifters is the same as the quantity of the plurality of floating point numbers. As shown in FIG. 3, the allocating, based on the sorting result, the shifter for each floating point number from the plurality of shifters with different preset bits includes:

**[0070]** Step S302: Determine a sorting serial number of each floating point number in the sorting result; and

**[0071]** Step S304: Determine a preset bit respectively corresponding to each sorting serial number, and allocate the plurality of shifters to a floating point number specified by the sorting serial number corresponding to the corresponding preset bit according to the possessed preset bits.

**[0072]** Specifically, the computer device determines the sorting serial number to which each floating point number belongs according to the sorting result of each floating point number. The sorting serial number indicates a bit of the floating point number in the sorting result. Each sorting serial number is preset with the corresponding preset bit, for example, a shifter with a first bit corresponding to an x bit, a shifter with a second bit corresponding to a y bit, etc. For one floating point number, the computer device determines the shifter with the preset bit according to the preset bit corresponding to the sorting serial number to which the floating point number belongs, thus determining an association relationship between the floating point number and the shifter. The computer device may allocate the shifter to process the floating point number.

**[0073]** Exemplarily, the floating point number at the first bit in the sorting result does not need to be shifted; for the floating point number at the second bit, the computer device allocates a q-bit shifter for it; and for the floating point number at the third bit, the computer device allocates a 2q-bit shifter for it, ... , and so on until all the other floating point numbers except the floating point number at the first bit are allocated.

**[0074]** In this embodiment, by allocating the different shifters to the different floating point numbers, the hardware overhead is significantly reduced, the timing characteristics are good and the efficiency is high.

**[0075]** After allocating the corresponding shifter to each floating point number, the computer device uses the shifter to perform shift processing on the floating point number. In the shift processing process, the bit that the shifter shifts the mantissa value of the floating point number may be determined according to the difference between its exponential value and the maximum exponential value. For this purpose, in some embodiments, as shown in FIG. 4, the performing, for each floating point number, shift processing on the mantissa value of the corresponding floating point number through the shifter allocated for the floating point number to obtain the shift result includes:

**[0076]** Step S402: Determine, based on the difference between the exponential value of each floating point number and the maximum exponential value respectively, a shift bit corresponding to the respective mantissa value of each floating point number, and the maximum exponential value being the maximum value of the exponential values of the plurality of floating point numbers; and

**[0077]** Step S404: Perform, for each floating point number, shift processing on the mantissa value of the corresponding floating point number through the shifter allocated for the floating point number based on the shift bit corresponding to the mantissa value of the corresponding floating point number to obtain the shift result.

**[0078]** Specifically, for one floating point number, the computer device determines, based on the difference between the exponential value of the floating point number and the maximum exponential value, the shift bit corresponding to the respective mantissa value of each floating point number, and then performs shift processing by using the shifter allocated to the floating point number in combination with the determined shift bit. The difference may be a difference value between the exponential value and the maximum exponential value, or a ratio of the exponential value to the maximum exponential value, or a multiple of the difference value between the exponential value and the maximum exponential value.

**[0079]** Exemplarily, if the difference value between the exponential value $E_1$ of the floating point number A and the maximum exponential value $E_{max}$ is x, the computer device determines that the shift bit corresponding to the mantissa value $M_1$ of the floating point number A is x bit. After determining the shift bit, the computer device shifts the mantissa value of the floating point number by x bit by using the shifter allocated for the floating point number to obtain the shift result of the floating point number. Except for the floating point number (the floating point number does not need to be shifted) corresponding to the maximum exponential $value_{max}$, the computer device performs shift processing on the remaining floating point numbers to obtain the plurality of shift results.

**[0080]** In this embodiment, by allocating the different shifters for the different floating point numbers, the hardware overhead is significantly reduced, and shifter resources required by shift processing are saved.

**[0081]** In the shift process, it is possible to encounter a situation where the determined shift bit is larger than the preset bit of the shifter. Here, in some embodiments, as shown in FIG. 5, the performing, for each floating point number, shift processing on the mantissa value of the corresponding floating point number through the shifter allocated for the floating point number based on the shift bit corresponding to the mantissa value of the corresponding floating point number to obtain the shift result includes:

**[0082]** Step S502: Determine, for each floating point number, whether the shift bit corresponding to the mantissa value of the floating point number is within a shift range, the shift range matching the preset bit of the shifter allocated for the floating point number;

**[0083]** Step S504: Shift, in a case that the shift bit is located within the shift range, each mantissa member constituting the mantissa value in the floating point number by the shift bit towards a same shift direction within the corresponding shift range through the shifter allocated for the floating point number, the shift direction including left shift or right shift;

**[0084]** Step S506: Shift, in a case that the shift bit is located outside the shift range, each mantissa member constituting the mantissa value in the floating point number by the preset bit towards the same shift direction through the shifter allocated for the floating point number; and

**[0085]** Step S508: Take the mantissa value of each floating point number obtained after shift processing as the respective shift result of each floating point number.

**[0086]** Specifically, for one floating point number, the computer device acquires the shift bit corresponding to the mantissa value of the floating point number by executing the above step S502. Meanwhile, the computer device determines the shifter allocated for the floating point number and acquires the preset bit of the shifter. The computer device compares the shift bit with the preset bit to determine whether the shift bit is within the shift range corresponding to the preset bit. In the case that the shift bit is located within the shift range, the computer device shifts the mantissa value by the shift bit towards the shift direction by using the shifter; and in the case that the shift bit is located outside the shift range, the computer device shifts the mantissa value by the preset bit towards the shift direction by using the shifter. After the computer devices performs shift processing on the mantissa value of the floating point number, the obtained mantissa value is taken as the shift result of the floating point number. Except for the floating point number (without needing to be shifted) corresponding to the maximum exponential value, the computer device performs shift processing mentioned above on the remaining floating point numbers to obtain the plurality of shift results.

**[0087]** Exemplarily, for the floating point number A, the computer device determines that the shift bit corresponding to its mantissa value $M_1$ is 22 bits, and the preset bit of the shifter $S_A$ allocated for the floating point number A is 50 bits, that is, the shift range is 26 bits. Then the computer device judges that the shift bit of the floating point number A is within the shift range of the shifter $S_A$, so the computer device shifts each mantissa member of the mantissa value of the floating point number A by 22 bits, for example, 22 bits to the right, towards the same shift direction by using the shifter $S_A$.

**[0088]** For another example, for a floating point number B, the computer device determines that the shift bit corresponding to its mantissa value $M_2$ is 56 bits, and the preset bit of the shifter $S_B$ allocated for the floating point number B is 76 bits, that is, the shift range is 52 bits. Then the

computer device judges that the shift bit of floating point number B is outside the shift range of shifter $S_B$. In other words, the shift bit of the floating point number B has exceeded the maximum shift range that shifter $S_B$ can shift. Therefore, the computer device shifts each mantissa member of the mantissa value of the floating point number B by 52 bits towards the same shift direction by using the shifter $S_B$.

**[0089]** In this embodiment, by allocating the shifters with the different preset bits to perform shift processing on the floating point number respectively, it is unnecessary to use a shifter with a larger bit width to ensure that there is no intermediate precision loss, thereby saving the hardware resources of the shifter, and reducing the area overhead of the shifter.

**[0090]** As mentioned above, after determining the shift results of the plurality of floating point numbers, the computer device further performs compression processing on the floating point number. In order to reduce an area of the compressor used and reduce a timing path, in some embodiments, as shown in FIG. 6, the determining the floating point number processing result corresponding to the target task based on each shift result includes:

**[0091]** Step S602: Divide, based on a first preset range where the shifters with the different preset bits are located, the first preset range to obtain a plurality of domain segments, and determine compressors respectively corresponding to the plurality of domain segments, the different compressors having different preset compression ratios.

**[0092]** The compression ratio refers to a ratio of the quantity of inputs to the quantity of outputs of the compressor. For example, a compressor with a preset compression ratio of (n:2) has n inputs and 2 outputs. For another example, a compressor with a preset compression ratio of (n:3) has n inputs and 3 outputs.

**[0093]** In some embodiments, numeric values of different preset compression ratios possessed by the plurality of compressors are all within a second preset range, and the numeric values of the preset compression ratios of all the compressors are uniformly distributed within the second preset range. The second preset range is obtained based on the quantity of the floating point numbers. Specifically, the preset compression ratio of the compressor is within the second preset range determined by the range maximum value of (n:2), and the second preset range further includes, for example, (n-1):2, (n-2):2, etc. Where n is the quantity of the floating point numbers.

**[0094]** Specifically, the computer device divides the first preset range where the shifters with the different preset bits are located, to obtain the plurality of domain segments. For example, according to the maximum preset bit between all the shifters, a range from the most significant bit

to the maximum preset bit is determined, and the range is equally divided into the plurality of domain segments. The ranges of all the domain segments are equal.

**[0095]** Exemplarily, for a compression process with four floating point numbers, for the domain segment at the most significant bit, the computer device determines that a compression ratio of a compressor corresponding to the domain segment is 4:2; for a domain segment at the second most significant bit, the computer device determines that a compression ratio of a compressor corresponding to the domain segment is 3:2; and for domain segments at the least significant bit and a second least significant bit, there is no need to allocate the compressor as there is no need for compression. Since the mantissa values in the domain segments at the least significant bit and the second least significant bit does not need to be compressed, the quantity of the compressors is the quantity of the floating point numbers minus 2.

**[0096]** Step S604: Determine, for each domain segment, a plurality of intra-domain shift results within the corresponding domain segment respectively, the single intra-domain shift result being an intra-domain part in the shift result corresponding to the single floating point number.

**[0097]** Specifically, for one domain segment, the computer device determines a part within the domain segment whose shift results of all the floating point numbers are within the domain segment. The part is called the intra-domain part. For example, if the domain segment at the most significant bit is the highest w bit, then within the domain segment, the computer device acquires a value of the highest w bit of the shift result of each floating point number, and the intra-domain shift results of all the acquired floating point numbers are the plurality of intra-domain shift results within the domain segment.

**[0098]** Step S606: Perform, through each compressor, segmented compression processing on the plurality of intra-domain shift results within the domain segment corresponding to the corresponding compressor to obtain a plurality of segmented compression results.

**[0099]** Specifically, the computer device performs segmented compression processing on the plurality of intra-domain shift results within the divided domain segment by using the compressor allocated for each domain segment to obtain the plurality of segmented compression results.

**[00100]** Exemplarily, for a compression process with n floating point numbers, for the domain segment at the most significant bit, the computer device performs segmented compression processing by using a compressor with a compression ratio of (n:2); for the domain segment at the second most significant bit, the computer device performs segmented compression processing by

using a compressor with a compression ratio of (n-1):2,..., and so on; and for the domain segments at the least significant bit and the second least significant bit, there is no need to perform compression processing.

**[00101]** Step S608: Determine the floating point number processing result corresponding to the target task based on the plurality of segmented compression results.

**[00102]** Specifically, the computer device further processes all the segmented compression results based on the segmented compression result corresponding to each domain segment, so as to determine the floating point number processing result corresponding to the target task. For example, for the plurality of segmented compression results after segmented compression processing, the computer device stitches all the segmented compression results according to the high and low order of the bits to form a complete compression result, then inputs the compression result to an adder for processing, and finally obtains the floating point number processing result corresponding to the target task.

**[00103]** In this embodiment, by dividing the plurality of domain segments and performing segmented compression, the compressor at the low-bit domain segment has less input (or even there is no need for compression), greatly reducing the area of the compressor and reducing the timing path.

**[00104]** As shown in FIG. 7, in some embodiments, the performing, through each compressor, segmented compression processing on the plurality of intra-domain shift results within the domain segment corresponding to the corresponding compressor to obtain the plurality of segmented compression results includes:

**[00105]** Step S702: Take, for each compressor, the plurality of intra-domain shift results within the respective corresponding domain segments as inputs of the corresponding compressor; and

**[00106]** Step S704: Perform, by each compressor, segmented compression processing on respective input according to respective corresponding preset compression ratios respectively to obtain a standard result and a carry result, the standard result and the carry result constituting a segmented compression result corresponding to a corresponding sectionalizer.

**[00107]** Specifically, for one compressor, the computer device takes the plurality of intra-domain shift results within the domain segment corresponding to the compressor as inputs of the compressor, and then performs segmented compression processing on the plurality of intra-domain shift results within the domain segment according to the preset compression ratio of the compressor

to obtain the standard result and the carry result. The standard result is a value of a sum obtained after compressing all the intra-domain shift results, and the carry result is a carry value of the value of the sum.

**[00108]** For example, as shown in FIG. 8, taking a compression process of the four floating point numbers (the floating point number A, the floating point number B, a floating point number C, and a floating point number D) as an example, for the domain segment at the highest w bit, the computer device performs segmented compression processing on the four intra-domain shift results within the domain segment by using the 4:2 compressor to obtain the standard result and the carry results. When there is a carry in the standard result, the corresponding carry result is 1, otherwise it is 0. For the domain segment at the second highest w bit, the computer device performs segmented compression processing on the three intra-domain shift results within the domain segment by using the 3:2 compressor to obtain the standard result and the carry result. For the domain segments at the least significant bit and the second least significant bit, there is no need to perform compression processing.

**[00109]** In this embodiment, by respectively setting the different compressors for compression according to domain segment division, the area of the compressor is greatly reduced, and the overhead of the hardware resources is reduced.

**[00110]** As mentioned above, the computer device further processes all the segmented compression results after obtaining the segmented compression result corresponding to each domain segment, so as to determine the final floating point number processing result. In the related art, after compressing to obtain carry (corresponding to the standard result in this embodiment of this application) and sum (corresponding to the carry result in this embodiment of this application), the computer device uses a carry propagate adder (CPA) to perform addition of carry propagation so as to obtain the floating point number processing result.

**[00111]** However, the mode adopted in the related art requires to perform addition of carry propagation on two inputs (carry and sum) with full bit width, which requires at least one 128bit adder, thus occupying a lot of hardware resources and having poor timing. For this purpose, the computer device allocates the different adders for the different domain segments for addition processing respectively; and due to the segmented addition process, it is also necessary to consider the carry situation of each domain segment. Therefore, in order to reduce the occupied hardware resources, in some embodiments, as shown in FIG. 9, the determining the floating point number processing result corresponding to the target task based on the plurality of segmented compression results includes:

**[00112]** Step S902: Take, for a first domain segment that has not undergone compression processing and only corresponds to the single intra-domain shift result among the plurality of divided domain segments, the single intra-domain shift result as a selection result of the first domain segment;

**[00113]** Step S904: Generate, for a second domain segment that has not undergone compression processing and corresponds to more than one intra-domain shift result among the plurality of divided domain segments, a truth value result and a pseudo value result of the second domain segment based on the more than one intra-domain shift result within the second domain segment;

**[00114]** Step S906: Generate, for a third domain segment subjected to compression processing among the plurality of divided domain segments, a truth value result and a pseudo value result of the corresponding third domain segment based on the segmented compression result corresponding to each third domain segment;

**[00115]** step S908: Determine, according to a bit field height of the domain segment and starting from a domain segment at a least significant bit, a selection result corresponding to each domain segment sequentially until a selection result of a domain segment at a most significant bit is obtained, selection results of other domain segments among the domain segments except for the first domain segment being one of a truth value result and a pseudo value result of the corresponding domain segment; and

**[00116]** Step S910: Determine the floating point number processing result corresponding to the target task based on the selection result of each domain segment.

**[00117]** For each domain segment, the computer device first performs addition processing on the segmented compression results in that domain segment respectively to obtain the truth value result and the pseudo value result. The truth value result is an actual summation result obtained by performing addition processing on the intra-domain shift result, and the pseudo value result is a simulated summation result obtained by calculation when performing addition processing on the intra-domain shift result and assuming there is the carry. Then, the computer device inputs the truth value result and the pseudo value result within each domain segment into a selector for selection, and the selector ultimately determines which result to output.

**[00118]** Specifically, if in the plurality of domain segments divided by the computer device, the domain segment at the least significant bit (referred to as the first domain segment) does not need to undergo compression processing, and there is only one intra-domain shift result in the

domain segment, the computer device does not need to perform addition processing on the domain segment and does not need to select. The intra-domain shift result may be directly used as a selection result of the first domain segment without allocating the adder and the selector.

**[00119]** The domain segment at the second least significant bit (referred to as the second domain segment) does not need to undergo compression processing as well. There are more than one intra-domain shift result corresponding to the second domain segment, which requires addition processing. Then the computer device performs addition processing on all the intra-domain shift results in the second domain segment, and obtains the truth value result and the pseudo value result of the second domain segment by calculation.

**[00120]** Except for the domain segment at the least significant bit and the domain segment at the second least significant bit, the intra-domain shift results in all remaining domain segments (referred to as the third domain segment) are subjected to compression processing, and there are the plurality of intra-domain shift results corresponding to the third domain segment. Therefore, for each third domain segment, the computer device performs addition processing on the segmented compression results in the third domain segment to generate the truth value result and the pseudo value result of the third domain segment.

**[00121]** After determining the truth value result and the pseudo value result of each domain segment except for the first domain segment, the computer device inputs the truth value result and the pseudo value result of each domain segment into the selector respectively, and the selector determines whether the output result is the truth value result or the pseudo value result. In order to improve efficiency, the computer device sequentially determines the selection results of the domain segment at the least significant bit, the domain segment at the second least significant bit, ..., the domain segment at the most significant bit starting from the domain segment at the least significant bit according to the bit field height of the domain segment. The selection result corresponding to the domain segment at the least significant bit is the original intra-domain shift result in the domain segment. Due to the fact that the domain segment at the least significant bit does not undergone compression and addition processing, there is no carry, so the selection result of the domain segment at the second least significant bit is the truth value result. For each third domain segment, the selection result of the domain segment at the higher bit needs to consider whether there is the carry in the domain segment at the lower bit in adjacent domain segments. When there is the carry in the domain segment at the lower bit, the selector outputs the selection result of the domain segment at the higher bit as the pseudo value result. On the contrary, the selector outputs the selection result of the domain segment at the higher bit as the truth value result.

**[00122]** Exemplarily, as shown in FIG. 10, for the domain segment at the most significant bit, the computer device inputs the carry case through the selector of the domain segment at the second most significant bit, and the selector of the domain segment at the most significant bit outputs the truth value result or the pseudo value result. Similarly, for the domain segment at the second most significant bit, the computer device also performs the above processing. For the domain segment at the second least significant bit, since there is no carry in the domain segment at the least significant bit, the selector of the domain segment at the second least significant bit outputs the truth value result. For the domain segment at the least significant bit, it is directly the original intra-domain shift result as there is no need for compression and addition processing.

**[00123]** As a result, the computer device outputs the selection results corresponding to all the domain segments through the selector.

**[00124]** In this embodiment, by adopting a segmented addition strategy, different adders are allocated to different domain segments for segmented addition processing, and the plurality of selectors are used to perform carry propagate according to the carry situation of each domain segment. Compared to an addition using a full bit width for carry propagate in the related art, hardware resources needing to be occupied are reduced, a length of a critical path is effectively reduced, and timing characteristics are good.

**[00125]** Then, the computer device stitches the selection results under each domain segment to obtain the complete floating point number processing result under the first preset range. For this purpose, in some embodiments, the determining the floating point number processing result corresponding to the target task based on the selection result of each domain segment includes: stitching the selection result of each domain segment sequentially according to the bit field height of each domain segment to obtain the floating point number processing result corresponding to the target task. Specifically, the computer device sequentially stitches the selection results under the adjacent two domain segments one by one from high to low (or from low to high) according to the bit field height of each domain segment, so as to obtain the floating point number processing result under the whole first preset range.

**[00126]** In this embodiment, since the strategies of segmented compression and segmented addition are adopted, the obtained results of all the domain segments are stitched again to obtain the complete floating point number processing result. This mode does not need the use of the full-bit-width compressor and adder, which reduces the length of the critical path and has good timing characteristics.

**[00127]** The floating point number is high in effective precision, and is more suitable for

scientific computing and engineering computing. In a scientific notation, if the expression of the floating point number is not clearly specified, coded representation of one floating point number in the computer will not be unique, which is not conducive to computer recognition and processing. For example, decimal numbers may be represented as $1.11\times10^{0}$, $0.111\times10^{1}$, $0.0111\times10^{2}$ and other various representations. Since the normalized floating point number has the unique representation, it is necessary to normalize the floating point number in floating point operations.

**[00128]** In order to ensure that the obtained floating point number processing result conforms to the normalized floating point number standard, after the floating point number processing result is obtained, in some embodiments, the computer device further performs normalization processing on the floating point number processing result, so that the floating point number processing result conforms to a preset floating point number standard. By performing normalization processing on the floating point number processing result to ensure that the obtained floating point number processing result conforms to the normalized floating point number standard, the computer does not need to recognize and convert all the floating point number processing results during processing, the processing efficiency is higher, and a problem of inaccurate computing caused by the non-unique coded representation of the floating point number is avoided.

**[00129]** Normalization processing, also known as formatted output, refers to converting one floating point number according to a specified format. An absolute value of mantissa M of the floating point number processing result after normalization processing is to meet $1/r \leq |M| < 1$, where r is a cardinal number, and r is usually 2 or 8 or 16.

**[00130]** Normalization processing refers to that a non-zero floating point number is guaranteed to be a valid value at the most significant bit of the mantissa value by adjusting the magnitude of the mantissa value and exponential value of a non-normalized floating point number. In some embodiments, as shown in FIG. 11, the performing normalization processing on the floating point number processing result includes:

**[00131]** Step S1102: Determine first symbol identification and second symbol identification in the floating point number processing result;

**[00132]** Step S1104: Perform, in a case that the first symbol identification and the second symbol identification are the same, shift processing on the mantissa value in the floating point number processing result according to a first shift direction; and

**[00133]** Step S1106: Perform, in a case that the first symbol identification and the second symbol identification are different, shift processing on the mantissa value in the floating point

number processing result according to a second shift direction, the second shift direction being opposite to the first shift direction.

**[00134]** Normalization processing of the floating point number includes to modes: left normalization and right normalization. The left normalization refers to performing normalization processing when the result of floating point number operation is denormalized, shifting the mantissa value to the left by one bit, and subtracting the level code by 1 (when the cardinal number r=2); and the left normalization may be performed for multiple times. Right normalization refers to shifting the mantissa value to the right by one bit, and adding the level code by 1 (when the cardinal number r=2) when the mantissa value overflows in the result of the floating point number operation. The right normalization only needs to be performed once.

**[00135]** Specifically, the computer device acquires the first symbol identification and the second symbol identification in the floating point number processing result. In the case where the first symbol identification and the second symbol identification are the same (i.e. the first symbol identification and the second symbol identification constitute 00 or 11), shift processing is performed on the mantissa value in the floating point number processing result according to the first shift direction. In the case where the first symbol identification and the second symbol identification are different (i.e. the first symbol identification and the second symbol identification constitute 01 or 10), shift processing is performed on the mantissa value in the floating point number processing result according to the second shift direction. The second shift direction is opposite to the first shift direction. Taking the normalization mode of the floating point number under the IEEE 754 standard as an example, the first shift direction is left shift, and the second shift direction is right shift.

**[00136]** Exemplarily, when the computer device judges that the two pieces of symbol identifications are the same, it indicates that there is no overflow. However, a highest numeric value bit of the floating point number processing result is the same as the symbol identification, so the left normalization processing is required at this time, that is, the mantissa value is shifted to the left until the highest numeric value bit is different from a numeric value of the symbol identification. Exemplarily, for the floating point number processing result of the following two cases: 111××× and 000×××, a result of shifting 111××× to the left by one bit is 11×××0; and a result of shifting 000××× to the left by one bit is 00×××0, and finally, the number of shifting is subtracted from the exponential value.

**[00137]** When the computer devices judge that the two pieces of symbol identifications are different, it indicates that the operation result overflows. At this time, it needs to perform right normalization processing, that is, the mantissa of the floating point number processing result is

shifted to the right, and shifting is stopped until there is no overflow; and then the exponential value is added by the number of shifting. Exemplarily, for the floating point number processing result of the following two cases: 01×××× and 10××××, a result of shifting 01×××× to the right by one bit is 001×××; and a result of shifting 10×××× to the right by one bit is 110×××, and finally, the exponential value is added by 1.

**[00138]** In this embodiment, by performing normalization processing on the floating point number processing result, the effective bits of the mantissa value are fully utilized, and the precision of the floating point number operation is improved.

**[00139]** In some embodiments, after performing normalization processing on the floating point number, some values may be increased to the lower bit of the mantissa part. These added values need to be subjected to rounding processing. For example, 1.2349999 is rounded down to 1.23, or 1.2350001 is rounded up to 1.24. The IEEE 754 standard specifies the following several rounding modes: rounding to the nearest even number, rounding up, rounding down, and rounding towards 0. Certainly, it is not limited to this. Different floating point number standards may be formulated with different rounding modes. In practical applications, appropriate rounding modes may be selected according to needs.

**[00140]** This application further provides an application scenario, applying the above multiple-input floating point number processing method. In some embodiments, the application of the multiple-input floating point number processing method in the application scenario is, for example, processing a target task, where the target task is one of subtasks in a neural network processing task, and the neural network processing task at least includes one of a convolutional processing task or a similarity processing task. Under the application scenario, the above multiple-input floating point number processing method further includes: executing each subsequent subtask in the neural network processing task based on the floating point number processing result to obtain a neural network processing result.

**[00141]** Specifically, in a process of training and applying a neural network, for example, it is necessary to perform computational processing on image data, audio data, and text data. The image data, audio data, and text data processed by the neural network are usually represented by the floating point numbers in the computer. For example, each pixel in an image is represented by a 32-bit single-precision floating point number between 0 and 255, where 255 represents white and 0 represents black. For another example, the read audio data are the floating point number within a sampling range.

**[00142]** In the neural network processing task, there are the plurality of subtasks involving

the processing of the floating point number. For example, convolution or deconvolution operation is performed on the inputted image data. Taking the convolution operation as an example, convolution is a result of summation after two data multiply within a certain range. In the convolution process, the computer device may use the processor to execute the above multiple-input floating point number processing method, so as to realize the summation of the plurality of floating point numbers, and obtain the floating point number processing result. Then, the subsequent convolution operation is completed based on the floating point number processing result. Therefore, in the neural network processing task, the computer device may execute each subsequent subtask in the neural network processing task based on the floating point number processing result to obtain the neural network processing result, such as outputting the processed image data.

**[00143]** For another example, the above multiple-input floating point number processing method may further be used for the similarity processing task in the neural network processing task. Taking the image data as an example, in the similarity processing task, it is necessary to compare the processed image data with the preset standard image data to calculate the similarity between the two. In the process of calculating the similarity, the computer device may use the above multiple-input floating point number processing method to calculate the difference of the floating point number corresponding to the image data, so as to obtain the floating point number processing result. According to the obtained floating point number processing result, the computer device may continue to execute the subsequent sub tasks in the neural network processing task accordingly, for example, the image data are classified according to the difference value indicated by the floating point number processing result, and finally the neural network processing result (such as an image classification result) is outputted.

**[00144]** Certainly, it is not limited to this. It is clear to a person in the art that, without departing from the inventive concept and idea disclosed in this application, any computational processing task applicable to the plurality of floating point numbers may be taken as the above target task, such as a data computing task in a cloud computing scenario, or a computing task in a data processing process executed by an intelligent sensor (such as an edge sensor).

**[00145]** In this embodiment, by applying the above multiple-input floating point number processing method to the tasks such as neural network processing, high-precision floating point number processing of the neural network can be realized, and the computing performance of the neural network is improved.

**[00146]** In order to explain the invention idea of this application as clearly as possible, an example of specifically executing the addition calculation of the plurality of floating point numbers

is used here for illustration, and the differences and advantages from a conventional mode are illustrated in detail.

**[00147]** In a specific example, the flow of the conventional floating point number processing mode is shown in FIG. 12. The computer device first acquires the exponents of n floating point numbers, and obtains the maximum exponential value by comparison with a comparator; and meanwhile, the computer device acquires the mantissa values (mantissa) of n floating point numbers. Then, for one floating point number, the computer devices inputs the difference value between the exponential value and the maximum exponential value of the floating point number, as well as the mantissa value of the floating point number, into a fixed-size shifter to perform shift processing on the floating point number with the shifter. For other floating point numbers, the computer device performs the same processing. Since the mantissa value of each floating point number varies from large to small, in order to ensure that there is no intermediate precision loss, the range of the shifter used needs to be the maximum value m*w bit, and m is the quantity of the floating point numbers minus 1.

**[00148]** In the shift process, in order to achieve the objective of no intermediate precision loss, the data bit width after shifting will be very large. Taking five 32-bit single-precision floating point numbers as an example, the data bit width after shifting needs to reach 128bit to ensure that there is no intermediate precision loss (the specific composition of 128bit is shown in FIG. 13, where the mantissa of each floating point number after expansion is 24bit, and 2bit is reserved as a carry propagate bit). The oversize shift range causes significant hardware overhead and poor timing, making it difficult for the processor in the computer to achieve a high master frequency.

**[00149]** Similarly, after shifting is completed, the computer device inputs the obtained shift result into the n:2 compressor for compression (where n is the number of inputs of a multiple-input floating-point adder), to obtain two outputs: carry and sum, and then performs carry propagate addition on carry and sum. The final addition result is sent to a normalize unit to complete a standardization operation of the floating point numbers. Finally, the standardized data are subjected to rounding operation to finally obtain the addition result of the floating point numbers. The compressor uses n:2 compression for the full bit width. Taking five 32-bit single-precision floating point numbers as an example, the compressor uses a 128bit 5:2 compressor, which has a huge area overhead. In a subsequent addition stage, the carry propagate addition is also performed on 2 inputs (carry and sum) of the full bit width. Similarly, taking five 32-bit single-precision floating point numbers as an example, one 128bit adder is required, and the timing is poor.

**[00150]** Compared with a conventional mode, as shown in FIG. 14, a whole processing flow

of the multiple-input floating point number processing method provided in this embodiment of this application mainly includes: first acquiring, by a computer device, exponents of n floating point numbers, and obtaining a maximum exponential value by comparison with a comparator; and meanwhile, acquiring, by the computer device, mantissa values (mantissa) of n floating point numbers. Before being inputted to a shifter for shift processing, the computer device performs sorting processing on all the floating point numbers according to magnitude of the exponential values of all the floating point numbers. That is, for a floating point number with a maximum exponent, no shift operation is required; for a floating point number sorting the second bit, only w bit needs to be shifted; and for a floating point number sorting the third bit, only 2*w bit needs to be shifted, ... , and so on. For a floating point number with a minimum exponent (sorting the last bit), the shift range is m*w. Taking five 32-bit single-precision floating point numbers as an example, m=4, and w=26. According to a sorting result, the computer device dynamically allocates the shifter of each floating point number, instead of using the same fixed-size shifter for each floating point number. Compared with the related art, the area overhead of the shifter is greatly reduced.

**[00151]** Then, the computer device inputs the mantissa value of each floating point number into the corresponding shifter for shift processing. The obtained shift results are then inputted into the corresponding compressors respectively. Specifically, the computer device sets compressors with different compression ratios for each domain segment. That is, for the highest w bit, an n:2 compressor is used; and for a second highest w bit, a (n-1):2 compressor is used, ... , and so on. For the lowest w bit and the second lowest w bit, there is no need to compress through the compressor. Therefore, there is no need to allocate the compressor as well. Compared with the related art, the area overhead of the compressor is greatly reduced.

**[00152]** Then, for each domain segment, the computer device uses a CPA for an addition operation. For the lowest w bit, there is no need to perform the addition operation, and the shift result of the w bit is directly acquired. For the second lowest w bit, since the shift result of two floating point numbers (corresponding to the intra-domain shift result in the aforementioned embodiment) exists in the domain segment, the computer device uses one CPA to perform the addition operation. Because there is no carry in the last w bit (the least significant bit) of the w bit (the second least significant bit), the w bit outputs the actual summation result. For other domain segments, according to whether the next domain segment (an adjacent low-bit domain segment) in the domain segment has a carry, MUX (a selector) selects whether to output the actual summation result (corresponding to the truth value result in the aforementioned embodiment) or a summation result containing the carry (i.e. the pseudo value result in the aforementioned embodiment), thus obtaining a plurality of selection results.

**[00153]** Finally, the computer device stitches all the selection results to obtain the final addition result. Then post-processing is performed, normalization processing is performed on the addition result to make it conform to a format specified in IEEE 754, and then rounding operation is performed to obtain the final result.

**[00154]** Compared to a conventional mode, on the one hand, the idea of effective shift is introduced to effectively shift the sorted mantissa. The shift range of the mantissa sorting top is small (or even there is no need to shift), greatly reducing the area overhead of the shifter. On the other hand, based on the idea of effective shift introduced, domain segments are divided for the full bit width and segmented compression is performed, the compressor at the low-bit domain segment has less inputs (or even there is no need for compression), greatly reducing the area of the compressor and reducing the timing path. In the final addition stage, the strategy of segmented addition is adopted, effectively reducing a length of a critical path. Thus, lossless multiple-input floating point number processing can be realized with the minimum hardware area, which can significantly reduce the area of the entire processor and effectively improve a core frequency of the processor.

**[00155]** Taking five 32-bit single-precision floating point numbers as an example, and taking a Global Foundries (GF) 12nm process technology as an area estimation standard, the comparison between the shifter resources required by the present disclosure and the related art is shown in Table 1 below:

| Shifter resources used in the related art | Shifter area used in the related art ($um^2$) | Shifter resources used in the present disclosure | Shifter area used in the present disclosure ($um^2$) |
|---|---|---|---|
| 128bit shifter<br>Five | 800 | 50bit shifter<br>One | 62.5 |
| / | / | 76bit shifter<br>One | 95 |
| / | / | 102bit shifter<br>One | 127.5 |
| / | / | 128bit shifter<br>One | 160 |
| Total | 800 | / | 445 |

Table 1

**[00156]** It can be seen that the shifter resources used in the present disclosure are only 55.6% of the related art, greatly saving the area overhead of the shifter.

**[00157]** In a compression stage, the comparison between compressor resources used in the present disclosure and compressor resources used in the related art is shown in Table 2 below:

| Compressor resources used in the related art | Compressor area used in the related art ($um^2$) | Compressor resources used in the present disclosure | Compressor area used in the present disclosure ($um^2$) |
|---|---|---|---|
| 128bit 5:2 compressor One | 192 | 24bit 5:2 compressor One | 36 |
| / | // | 26bit 4:2 compressor One | 26 |
| | / | 26bit 3:2 compressor One | 13 |
| Total | 192 | / | 75 |

Table 2

**[00158]** It can be seen that the compressor resources used in the present disclosure are 39% of the related art, greatly saving the area overhead of the compressor.

**[00159]** In addition, in terms of timing, the critical path from shift processing to adder output in the related art is: 128bit shift (shifter) ->5:2 compressor ->128bit adder. The critical path from shift processing to adder output in the present disclosure is: 128bit shift (shifter) ->5:2 compressor ->24bit adder. The length of the critical path is effectively reduced, and a higher processor master frequency may be realized.

**[00160]** It is to be understood that, although the steps are displayed sequentially according to the instructions of the arrows in the flowcharts involved by all the embodiments above, these steps are not necessarily performed sequentially according to the sequence instructed by the arrows. Unless otherwise explicitly specified in this specification, execution of the steps is not strictly limited, and the steps may be performed in other sequences. Moreover, at least some of the steps in the flowcharts involved by all the embodiments above may include a plurality of steps or a plurality of stages. The steps or stages are not necessarily performed at the same moment but may be performed at different moments. Execution of the steps or stages is not necessarily sequentially performed, but may be performed alternately with other steps or at least some of sub-steps or stages

of other steps.

**[00161]** Based on the same invention concept, an embodiment of this application further provides a multiple-input floating point number processing apparatus used for implementing the multiple-input floating point number processing method involved above. The implementation solution provided by the apparatus to solve the problem is similar to the implementation solution recorded in the above method. Therefore, the specific limitations in one or more embodiments of the multiple-input floating point number processing apparatus provided below may be referred to the limitations on the multiple-input floating point number processing method above.

**[00162]** In some embodiments, as shown in FIG. 15, a multiple-input floating point number processing apparatus 1500 is provided. The apparatus may adopt a software module or a hardware module, or a combination of the two to become a part of a computer device. The apparatus specifically includes: an acquiring module 1501, an allocating module 1502, a shifting module 1503, and a determining module 1504.

**[00163]** The acquiring module 1501 is configured to acquire a plurality of to-be-processed floating point numbers corresponding to a target task, and extract an exponential value of an exponent part and a mantissa value of a mantissa part in each floating point number respectively.

**[00164]** The allocating module 1502 is configured to sort, according to a magnitude of the exponential value of each floating point number, the plurality of floating point numbers to obtain a sorting result, and allocate, based on the sorting result, a shifter for each floating point number from a plurality of shifters with different preset bits.

**[00165]** The shifting module 1503 is configured to perform, for each floating point number, shift processing on the mantissa value of the corresponding floating point number through the shifter allocated for the floating point number to obtain a shift result.

**[00166]** The determining module 1504 is configured to determine a floating point number processing result corresponding to the target task based on each shift result.

**[00167]** In some embodiments, the different preset bits possessed by the plurality of shifters are all within a first preset range, and the preset bits of all the shifters are uniformly distributed within the first preset range.

**[00168]** In some embodiments, the quantity of the plurality of shifters is the same as the quantity of the plurality of floating point numbers. The allocating module is further configured to determine a sorting serial number of each floating point number in the sorting result; and determine a preset bit respectively corresponding to each sorting serial number, and allocate the plurality of

shifters to a floating point number specified by the sorting serial number corresponding to the corresponding preset bit according to the possessed preset bits.

**[00169]** In some embodiments, the shifting module is further configured to determine, based on a difference between the exponential value of each floating point number and a maximum exponential value respectively, a shift bit corresponding to the respective mantissa value of each floating point number respectively, the maximum exponential value being a maximum value of the exponential values of the plurality of floating point numbers; and perform, for each floating point number, shift processing on the corresponding mantissa value through the shifter allocated for the floating point number based on the shift bit corresponding to the mantissa value of the corresponding floating point number to obtain the shift result.

**[00170]** In some embodiments, the shifting module is further configured to determine, for each floating point number, whether the shift bit corresponding to the mantissa value of the floating point number is within a shift range, the shift range matching the preset bit of the shifter allocated for the floating point number; shift, in a case that the shift bit is located within the shift range, each mantissa member constituting the mantissa value in the floating point number by the shift bit towards a same shift direction within the corresponding shift range through the shifter allocated for the floating point number, the shift direction including left shift or right shift; shift, in a case that the shift bit is located outside the shift range, each mantissa member constituting the mantissa value in the floating point number by the preset bit towards the same shift direction through the shifter allocated for the floating point number; and take the mantissa value of each floating point number obtained after shift processing as the respective shift result of each floating point number.

**[00171]** In some embodiments, the determining module further includes a compression unit. The compression unit is configured to divide, based on a first preset range where the shifters with the different preset bits are located, the first preset range to obtain a plurality of domain segments, and determine compressors respectively corresponding to the plurality of domain segments, the different compressors having different preset compression ratios; determine, for each domain segment, a plurality of intra-domain shift results within the corresponding domain segment respectively, the single intra-domain shift result being an intra-domain part in the shift result corresponding to the single floating point number; perform, through each compressor, segmented compression processing on the plurality of intra-domain shift results within the domain segment corresponding to the corresponding compressor to obtain a plurality of segmented compression results; and determine the floating point number processing result corresponding to the target task based on the plurality of segmented compression results.

**[00172]** In some embodiments, numeric values of different preset compression ratios possessed by the plurality of compressors are all within a second preset range, and the numeric values of the preset compression ratios of all the compressors are uniformly distributed within the second preset range.

**[00173]** In some embodiments, the compression unit is further configured to take, for each compressor, the plurality of intra-domain shift results within the respective corresponding domain segments as inputs of the corresponding compressor; and perform, by each compressor, segmented compression processing on respective input according to respective corresponding preset compression ratios respectively to obtain a standard result and a carry result, the standard result and the carry result constituting a segmented compression result corresponding to a corresponding sectionalizer.

**[00174]** In some embodiments, the determining module further includes an addition unit. The addition unit is configured to take, for a first domain segment that has not undergone compression processing and only corresponds to the single intra-domain shift result among the plurality of divided domain segments, the single intra-domain shift result as a selection result of the first domain segment; generate, for a second domain segment that has not undergone compression processing and corresponds to more than one intra-domain shift result among the plurality of divided domain segments, a truth value result and a pseudo value result of the second domain segment based on the more than one intra-domain shift result within the second domain segment; generate, for a third domain segment subjected to compression processing among the plurality of divided domain segments, a truth value result and a pseudo value result of the corresponding third domain segment based on the segmented compression result corresponding to each third domain segment; determine, according to a bit field height of the domain segment and starting from a domain segment at a least significant bit, a selection result corresponding to each domain segment sequentially until a selection result of a domain segment at a most significant bit is obtained, selection results of other domain segments among the domain segments except for the first domain segment being one of a truth value result and a pseudo value result of the corresponding domain segment; and determine the floating point number processing result corresponding to the target task based on the selection result of each domain segment.

**[00175]** In some embodiments, the determining module further includes a stitching unit. The stitching unit is configured to stitch the selection result of each domain segment sequentially according to the bit field height of each domain segment to obtain the floating point number processing result corresponding to the target task.

**[00176]** In some embodiments, the above apparatus further includes a post-processing module. The post-processing module is configured to perform normalization processing on the floating point number processing result to make the floating point number processing result conform to a preset floating point number standard.

**[00177]** In some embodiments, the post-processing module is further configured to determine first symbol identification and second symbol identification in the floating point number; perform, in a case that the first symbol identification and the second symbol identification are the same, shift processing on the mantissa value in the floating point number processing result according to a first shift direction; and perform, in a case that the first symbol identification and the second symbol identification are different, shift processing on the mantissa value in the floating point number processing result according to a second shift direction, the second shift direction being opposite to the first shift direction.

**[00178]** In some embodiments, the target task is one of subtasks in a neural network processing task, the neural network processing task at least includes one of a convolutional processing task or a similarity processing task. The above apparatus further includes a task module. The task module is configured to execute each subsequent subtask in the neural network processing task based on the floating point number processing result to obtain a neural network processing result.

**[00179]** The specific limitations about the multiple-input floating point number processing apparatus may refer to the limitations on the multiple-input floating point number processing method above. Each module in the above multiple-input floating point number processing apparatus may be implemented entirely or partially through software, hardware, or a combination thereof. All the above modules may be embedded in or independent of a processor in a computer device in a hardware form, or stored in a memory in the computer device in a software form for the processor to call and execute the operations corresponding to all the above modules.

**[00180]** Based on the same invention concept and idea, an embodiment of this application further provides a processor, used for implementing the multiple-input floating point number processing method involved in the embodiments above. In some embodiments, as shown in FIG. 16, the processor 1600 includes at least one shifter 1601 with different preset bits and a logic processing unit 1602. In different implementation scenarios, the processor may present various encapsulation structures according to the requirements applied to different computer devices.

**[00181]** For the at least one shifter with different preset bits 1601, each shifter is allocated to be used for performing shift processing on a mantissa value of a mantissa part of one of a plurality

of to-be-processed floating point numbers. The shifter correspondingly allocated to each floating point number is determined according to a sorting result obtained by sorting exponential values of an exponent part of each floating point number.

**[00182]** The logic processing unit 1602 is configured to perform logic processing on a plurality of shift results obtained by shift processing of each shifter to obtain a floating point number processing result.

**[00183]** In some embodiments, the different preset bits possessed by at least one shifter are all within a first preset range, and the preset bits of all the shifters are uniformly distributed within the first preset range.

**[00184]** In some embodiments, as shown in FIG. 17, the logic processing unit 1602 includes:

**[00185]** at least one compressor 16021 with different preset compression ratios, connected to the shifter respectively, each compressor being allocated to be used for performing segmented compression processing on a plurality of shift results obtained through shift processing to obtain a segmented compression result;

**[00186]** at least one adder 16022, connected to the compressor 16021 respectively, and configured to generate truth value results and pseudo value results of a plurality of domain segments based on the segmented compression result; and

**[00187]** at least one selector 16023, connected to the adder 16022 respectively, and configured to determine a selection result of each domain segment based on the truth value results and the pseudo value results of the plurality of domain segments. The selection results are used for being stitched to generate the floating point number processing result.

**[00188]** In some embodiments, numeric values of different preset compression ratios possessed by at least one compressor are all within a second preset range, and the numeric values of the preset compression ratios of all the compressors are uniformly distributed within the second preset range.

**[00189]** The specific limitations about the processor may refer to the limitations on the multiple-input floating point number processing method above. All components in the above processor may be fully or partially implemented through combinations of other circuit components such as a gate circuit, and a switch circuit. All the above components may be integrated in the processor of the computer device, for the processor to call and execute operations corresponding to all the above components.

**[00190]** In some embodiments, a computer device is provided. The computer device may be a

server or a terminal containing the above processor. The server may be an independent physical server, or a server cluster or distributed system composed of the plurality of physical servers, or a cloud server that provides a cloud computing service. The terminal may be a smartphone, a tablet computer, a notebook computer, a desktop computer, a smart speaker, a smartwatch, a vehicle-mounted terminal, a smart television, and the like. An internal structure diagram of the computer device may be shown in FIG. 18. The computer device includes a processor, a memory, and a network interface that are connected through a system bus. The processor of the computer device is configured to provide computing and control capabilities. The memory of the computer device includes a non-volatile storage medium and an internal memory. The non-volatile storage medium stores an operating system, a computer readable instruction and a database. The internal memory provides an environment for running of the operating system and the computer readable instruction in the non-volatile storage medium. The database of the computer device is used for storing floating point numbers. The network interface of the computer device is used for communicating with an external terminal through a network connection. The computer readable instruction, when executed by the processor, implements a multiple-input floating point number processing method.

**[00191]** A person skilled in the art may understand that, the structure shown in FIG. 18 is merely a block diagram of a partial structure related to a solution in this application, and does not constitute a limitation to the computer device to which the solution in this application is applied. Specifically, the computer device may include more components or fewer components than those shown in the figure, or may combine some components, or may have a different component deployment.

**[00192]** In some embodiments, a computer device is further provided, including a memory and a processor. The memory stores a computer readable instruction, and the processor, when executing the computer readable instruction, implements the steps in the method embodiments above.

**[00193]** In some embodiments, a computer readable storage medium is provided, storing a computer program. The computer program, when executed by a processor, implements the steps in the method embodiments above.

**[00194]** In an embodiment, a computer program product is provided, including a computer program. The computer program, when executed by a processor, implements the steps in the method embodiments above.

**[00195]** A person of ordinary skill in the art may understand that all or some of flows in the methods of the above embodiments may be implemented by instructing relevant hardware through

the computer program. The computer program may be stored in a non-volatile computer readable storage medium. The computer program, when executed, may include the flows of the embodiments of the methods above. Any reference to the memory, the database, or other mediums used in the embodiments provided in this application may all include at least one of a non-volatile memory or a volatile memory. The non-volatile memory may include a read-only memory (ROM), a magnetic tape, a floppy disk, a flash memory, an optical memory, a high-density embedded non-volatile memory, a resistive random access memory (ReRAM), a magnetoresistive random access memory (MRAM), a ferroelectric random access memory (FRAM), a phase change memory (PCM), a graphene memory, etc. The volatile memory may include a random access memory (RAM) or an external cache memory, etc. As illustration rather than limitation, the RAM may take various forms, such as a static random access memory (SRAM) or a dynamic random access memory (DRAM). The database involved in the embodiments provided by this application may include at least one of a relational database and a non relational database. The non relational database may include, but is not limited to, a blockchain-based distributed database, etc. The processor involved in the embodiments provided by this application may be, but is not limited to, a general purpose processor, a central processing unit, a graphics processing unit, a digital signal processor, a programmable logic unit, a data processing logic unit based on quantum computing, etc.

**[00196]** The technical features of the above embodiments may be randomly combined. For concise description, not all possible combinations of the technical features in the above embodiments are described. However, provided that the combinations of the technical features do not conflict with each other, the combinations of the technical features are considered as falling within the scope described in this specification.

**[00197]** The above embodiments merely express several implementations of this application. The descriptions thereof are relatively specific and detailed, but are not to be understood as limitations to the scope of the present disclosure. For a person of ordinary skill in the art, several transformations and improvements can be made without departing from the idea of this application. These transformations and improvements belong to the protection scope of this application. Therefore, the protection scope of the patent of this application shall be subject to the appended claims.