(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 July 2013 (04.07.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/101173 Al

(51) International Patent Classification:
G06F 9/30 (2006.0 1) *G06F 9/44* (2006.0 1)

(21) International Application Number:
PCT/US201 1/068067

(22) International Filing Date:
30 December 201 1 (30. 12.201 1)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US):* **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95052 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only):* **KRIG, Scott A.** [US/US]; 4518 Carlyle Ct., #1724, Santa Clara, California 95054 (US).

(74) Agent: **TROP, Timothy N.;** Trop, Pruner & Hu, P.C., 1616 S. Voss Rd., Ste. 750, Houston, Texas 77057-2631 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available):* AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available):* ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *of inventorship (Rule 4.17(iv))*

Published:

— *with international search report (Art. 21(3))*

WO 2013/101173 A1

(54) Title: DECOMPOSING OPERATIONS IN MORE THAN ONE DIMENSION INTO ONE DIMENSIONAL POINT OPERATIONS

(57) Abstract: A processing architecture uses stationary operands and opcodes common on a plurality of processors. Only data moves through the processors. The same opcode and operand is used by each processor assigned to operate, for example, on one row of pixels, one row of numbers, or one row of points in space.

DECOMPOSING OPERATIONS IN MORE THAN ONE DIMENSION INTO ONE DIMENSIONAL POINT OPERATIONS

Background

[0001] This relates generally to processing architectures and particularly to processing architectures adapted for parallel operations on a large amount of data.

[0002] In many processing applications, including those involving graphics and those involving complex mathematical calculations, a large number of simple operations must be done a large number of times. As a result, many of these operations can be done in parallel.

[0003] In a typical Von Neumann architecture, a processing pipeline is executed by a processor. In that pipeline, there are number of stages. Both data to be operated on and code to operate on that data, move through the pipeline in parallel. That is, both the instructions and the data move from stage to stage through the pipeline in the same way.

[0004] In image processing, there are a number of operations that are considered to be point operations. These operations are generally performed only on one pixel and only using that pixel's value. Thus, point operations can also be called one dimensional operations.

[0005] There are also operations that involve not only a pixel of interest but its immediate neighbors as well. These operations use the values of the pixel of interest and its neighbors. Thus, such image processing may be called group processing, area processing, or two-dimensional processing.

Brief Description Of The Drawings

[0006] Some embodiments are described with respect to the following figures:

Figure l is a hardware depiction of one embodiment;

Figure 2 is a sequential depiction of a write operation according to one embodiment;

Figure 3 is a flow chart for the write operation in one embodiment;

Figure 4 is a sequential depiction of a read operation according to one embodiment;

Figure 5 is a flow chart for a read operation in one embodiment;

Figure 6 is an illustration of a spatial convolution;

Figure 7 is a functional depiction of one embodiment;

Figure 8 is a flow chart for one embodiment;

Figure 9 is an example of a transformation of a streaming convolution in one embodiment;

Figure 10 is an example of a transformation of streaming mathematical morphologyinto one embodiment; and

Figure 11 is an example of a transformation of a streaming MIN filter in one embodiment.

Detailed Description

[0007] In some embodiments an instruction stream does not need to be fetched in contrast to the Von Neuman architecture. Instead, instructions and operands are preset into the control and operand registers, and only the data stream needs to be fetched. In some cases this is advantageous for speed of calculations and reduction of memory bandwidth requirements.

[0008] Referring to Figure 1, in accordance with one embodiment, a host controller 12 may be coupled to an orthogonal processor 14 and an orthogonal processor 16a. The difference between the two processors 14 and 16a is that one works on a smaller sized word than the other. Specifically, the orthogonal processor 14 in one embodiment works on 4k words while the orthogonal processor 16a in one embodiment works on 16k words. Other arrangements are also possible. Thus, there may be additional orthogonal processors, each adapted to different word sizes, and there is no limitation on the particular word sizes that any particular processor may be designed to operate on.

[0009] As used herein, an orthogonal processor refers to the fact that the data and instructions do not move through the processor along the same path. Instead, a given word of work is broken into a given number of bits to form a data word. A nanoprocessor is provided to operate on each of thegroups of bits (data words) in parallel. Thus to operate on a 4k word, there would be 4k nanoprocessors in one embodiment. Each nanoprocessor may use a common or shared operating register 28 and a common opcode register 30 because each nanoprocessor is doing the

same operation using the same operand as all the other nanoprocessors in a given orthogonal processor.

[001 0] The output of each nanoprocessor 32 is stored in a row in the cell array 34 which is a two-dimensional memory with rows and columns. A nanoprocessor is any relatively small limited function or dedicated processor.

[001 1] The way that these operations are implemented is equivalent to a direct memory access (DMA). Thus the operations occurat memory write speeds in some embodiments, and faster or slower in other embodiments.

[001 2] Opcode register 30 stores an opcode that is then used by each nanoprocessor to operate on the input data. In some embodiments there may be more than one opcode that is applied to the data. Thus, in some embodiments more than one opcode register may be included. This results in the same data being operated on by more than one opcode. In some embodiments the opcode register 30 may store compound opcodes such as fused multiplyadd opcodes. In such cases, more than one opcode occurs together in the same instruction. Thus, the opcode register may include opcodes fused together to perform both a multiply and an add in the same instruction. Other fused operations include multiply and clip in the same instruction, and add and clip in the same instruction using a plurality of opcode registers. Other compound opcodes may also be used.

[001 3] Referring to Figure 2, in an orthogonal processor, data moves in the vertical direction and operands and opcodes are moving or set into one or more operandand opcode registers in the horizontal direction in each nanoprocessor. The operands and opcodes are stored before the data flow begins.

[0014] Thus the sequence may be, in one embodiment, to provide a word of data having a number of bits equal to the number of nanoprocessors. Each nanoprocessor has access to the particular operands and the particular opcodesto be executed any given number of times. Thus a two-dimensional array of data may include a number of horizontal rows of data. Each row may be processed serially, one after the other. Therefore the nanoprocessors do not need to receive new opcodes or operandsuntil after the entire two-dimensional array has been processed.

[001 5] Once each nanoprocessor has access to the correct operands and the correct opcodes and has the data ready to operate on, the operation is implemented. For example if the operation is a multiply, each nanoprocessor does the multiplication and loads the data into a row of the cell array 34. Thus the operations are done effectively at write speeds corresponding to direct memory accesses. Each cell in the array stores the result of the operation performed on one bitor data word, such as one pixel in a graphics application.

[001 6] The host controller feeds the data to each orthogonal processor 14 or 16a as the case may be. Thus if a given set of operations uses words of one size, the data may be provided to the processor 14, and if the data is of a different size it may be provided to a processor 16a adapted to that particular size.

[001 7] Typically, embodiments of the present invention operate on point operations which are basically one-dimensional. A multiply or an add is an example of a point operation. Area operations involve two or more dimensions and correspond to things like kernel operations, convolutions, and binary morphology, and one skilled in the art will recognize that this invention may be embodied in multiple dimensions.

[001 8] Applications fortwo-dimensional operations include discrete convolution and kernel operations include media post-processing, camera pipeline processing, video analytics, machine vision and general image processing. Key operations may include edge detection and enhancement, color and luminance enhancement, sharpening, blurring, and noise removal.

[001 9] Applications of binary morphologyand gray scale morphologyas two-dimensional area operationsinclude video analytics, object recognition, object tracking and machine vision. Key operations performed in the orthogonal processor may include a erode, dilate, opening and closing.

[0020] Applications for numeric area and point operations include any type of image processing including those described above in connection with discrete convolution, kernel operations, and binary morphology. Key operations include math operators, Boolean operators applied to each point or pixel and numeric precision data type conversions.

[0021] In some embodiments area operations are converted into point operations, where area operations may be two or three cubic, or higher dimensions,and the reduction of said area operations into one-dimensional point operations is advantageous in some embodiments reducing the computational and memory bandwidth overhead for all point operations. For example, a convolution is an area operation that can be converted into a series of successively shifted multiplications with accumulation, which are simple one-dimensional point operations that are accelerated.Then in the first passthrough an orthogonal processor, a shift in the dataset origin is implemented and in the second pass, a multiplication may be implemented with accumulation on a shifted version of the source dataset.

[0022] By decomposing the computation into a set of point operations implemented as a set of DMA or other memory writes, the data set may be kept in a local cache portion of the memory hierarchy and thus be much more efficient in terms of memory bandwidth, system performance, and lower power in some embodiments.

[0023] In a more specific example, the operationmay be accumulation or summing. Each orthogonal processor cell is an accumulator that sums the results of each memory write into itself by combining the write value or operand according to an opcode. Only a write into memory is needed for the memory cell to perform the computation. At page writes and corresponding vectorization of computations such as 4,096 page writes and 4,096 vectorized operations may occur a direct memory access speeds. In this example, the memory cell is the accumulator for a set of sequential operands, and the cumulative result of a set of operations is accumulated in the memory cell, for example, a set of nine (9) MULTIPLY-ADD instructions used to implement a convolution kernel where the result is accumulated into the memory cell.

[0024] The memory cell may also used as an operand for some operations or opcodes. An opcode may take as an input a memory cell and an operand from a register, where the result is stored into the memory cell, for example, as may be the core witha MULTIPLY-ADD instruction.

[0025] Each nanoprocessor may operate as follows in one embodiment. For each opcode, the operation bit precision and numeric conversion is defined.Assuming a 32-bit opcode embodiment, there are zero to fifteenbits to define the opcode and

sixteen to twenty-one bits to define the precision and conversion of the operation. The decoding of the instructions may occur in an orthogonal path to the data path.

[0026] Accumulation may effectively be done in the cell array 34. Opcodes may be implemented in the nanoprocessors 32 and numeric conversions may occur on read or write to each memory cell. Each memory cell applies a data format conversion operation as follows. For read operations, the cell numeric format is converted on memory read using a convert operator. Numeric conversions can be specified using opcodesto set up the nanoprocessors prior to the memory reads or writes to enforce the desired data conversion and numeric precision. The numeric conversions are implicit and stay in effect until a new opcode is sent to the nano processors. For write operations, a final value is converted to a desired numeric format according to the convert operator. This allows any sort of common operation to be implemented such as area convolution, other types of mathematical area operations, point operations, binary morphology andgray scale morphology,with options available to be set into control or opcode registers to specify the numeric conversions between float, double, and integer. In some embodiments precision may be fixed or limited to save silicon real estate and to reduce power consumption.

[0027] The cell array is an array of memory cells or registers with attached compute capabilities in the form of the nanoprocessors shown in Figure 1. Each memory cell is also an accumulator storing results with varying precision calculated by the nanoprocessors. Cell array processing occurs at the speed of memory writeseliminating memory reads for kernels and source pixels and providing vectorized processing at the speed of direct memory access writes into the cell array in some embodiments.

[0028] The array can be used simply for data conversions instead of calculations, since data conversions are very common, and the array can accelerate them.

[0029] An array can also be used for memory read operations simply for numeric conversions via DMA reads, since the numeric conversions are fast and occur at DMA rates with no need for processing the data. The numeric conversions may be between integer and floating point, various integer lengths, and various floating point lengths using sign extension, rounding, truncation, and other methods as may be desired and set using opcodes.

[0030] The cell array operation is similar to a hardware raster operation in a display system. In a display system, the raster operations are applied for each pixel written into a display memory cell or pixel.

[0031] Standard area operations like convolution may be broken down into point operations, allowing the computations to be performed as a series of DMA writes at very high performanceinsome embodiments.

[0032] For example in connection with a convolution, a series of pixel offset writes can occur into the orthogonal processor memory cells where the desired operation for each pixel may occur within the nanoprocessors that act on the individual cells. Each kernel value is preset into the cell array operand register prior to the pixel blit. The cell array operates by simply writing the entire image which causes the nanoprocessors to perform convolution operations for each pixel. This arrangement transfers pixel by pixel area convolution into a vectorized write operation, eliminating kernel and pixel reads and performing a fused multiply-add accumulation in each cell. In an embodiment, if the memory system employs cache memory levels and the data set resides in the cache at all times or even most of the time, this method will exploit cache memory and thus provide additional power & performance advantages.

[0033] Referring to Figure 6, spatial convolution is performed as a summation of elements multiplied by constants. In the case of an image processing application, the elements are pixel brightnesses of a kernel. A kernel is the input pixel plus its immediate neighbors. The number of immediate neighbors is variable. For example 3x3, 5x3, or any odd numbered array of pixels around the pixel of interest may be chosen. The constants are weights or convolution coefficients. A convolution mask is an array of convolution coefficients.

[0034] Thus, in Figure 6, a 3x3 array of pixels is illustrated. In this case a 3x3 convolution will be done of the central pixel x, y as an example. Convolution coefficients a through i make up a convolution mask. Thus each of the convolution coefficients is used in a multiply and accumulate operation with its corresponding pixel location. That is, as an example, the convolution coefficient a corresponds to the pixel position x-1 , y-1 in this example. Then the convolution is the sum of the multiplications of each convolution coefficients times its corresponding input image pixel. The result is a new output pixel value for the pixel x, y. The specific

multiplication and accumulation operations for this example are also shown in Figure 6.

[0035] Then in order to perform the convolution of the next pixel, the convolution mask can simply be shifted by one to the left or the right or up or down successively until all the pixels in the input image have their convolution determined. The shift operation can be effectively accomplished with the memory array and associated nanoprocessors. Particularly, the multiply and add and shift can all be done in this example using nine pixel writes to the memory array. This is a considerable simplification relative to existing techniques.

[0036] The shifting operation is also shown in Figure 7. The convolution mask is B and the kernel is A in Figure 7. A larger portion of the array is shown because more neighboring pixels, than what is shown in Figure 6, are involved. As can be seen, the entire group of pixels shown in Figure 6, nine in number, can be subjected to convolution by nine shifts each with nine multiply and addition operations.

[0037] Finally referring to Figure 8, a convolution sequence 70 may be implemented in software, firmware and/or hardware. In software and firmware embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium such as a semiconductor, magnetic, or optical storage.

[0038] Initially, as stated in block 72, the array is initialized by writing the image pixels into the array. Then the opcode register is initialized as multiply as indicated in block 74. The operand register is initialized with the current kernel value as indicated in block 76. Next the entire image is written into the array at the offset for each kernel as indicated in block 78. A check at diamond 80 determines whether all the necessary iterations have been done. The number of iterations is determined by the size of the array. Thus in the example given in Figure 6, nine iterations would be needed.

[0039] The orthogonal processor may perform 3x3 convolution with nine pixel writes of the image frame onto itself and offsets according to the kernel size, eliminating explicit read operations. In contrast a normal 3x3 convolution involves nine kernel reads, nine pixel reads and nine fusedmultiply-add instructions for each pixel in addition to a final pixel write. Thus the orthogonal processor may provide a

significant speed-up in some embodiments. The pseudo code for 3x3 convolution using nine image frame writes plus kernel set-up is as follows:

```
sobel[3][3] =
        {
        {-1, -2, -1,}
        { 0, 0, 0,}
        { 1, 2, 1}
        };

        // Initialize cells by writing entire image into XCELLARRAY
        writeImage(source_image, &xcellarrray.memory, /*X OFFSET*/ 0, /*Y OFFSET */ 0);

        // Initialize opcode register with MULTIPLY
        Xcellarray.opcode = OP_MULTIPLY;

        // Iterate 9 times to write the entire image, one line at a time, into the memory array
        // and for each write, use a different kernel value
        XSIZE = 3;
        YSIZE = 3;

        XOFFSET = (XSIZE / 2);
        YOFFSET = (YSIZE / 2);

        for (x=0; x < XSIZE; x++)
        {
           for (y=0, y < YSIZE; y++)
           {
           // Initialize operand register with the current kernel value [x,y]
           Xcellarray.operand[0] = sobel[x,y];

           // Write source image into cell array at the offset for each kernel element
           // This Write performs a MADD instruction -> CELL += (CELL * operand)
           writeImage(source_image, &xcellarrray.memory, x - XOFFSET, y - YOFFSET);
           }
        }
```

A pictorial illustration of the transformation of a streaming convolution is shown in Figure 9.

[0040] The example below shows pseudo-code for a 3x3 morphological DILATE operation illustrating the cell array optimization method according to one embodiment.

```
        dilate[3][3] =
        {
        {0, 1, 0,}
        {1, 0, 1,}
        {0, 1, 0}
        };

        // Initialize cells by writing entire image into
        writeImage(source_image, Sxcellarrray. memory, /*X OFFSET*/ 0, /*Y OFFSET */ 0);

        // Initialize opcode register with MULTIPLY
```

```
Xcellarray.opcode  = 0 P_0R; // Boolean OR

// Iterate 9 times to write the entire image into the memory array
// and for each write, use a different kernel value
XSEZE = 3;
YSIZE = 3;

XOFFSET = (XSIZE / 2);
YOFFSET = (YSIZE / 2);

for (x=0; x < XSIZE; x++)
{
   for (y=0, y < YSIZE; y++)
   {
      // OPTMIIZATION: for DILATE, we only use truth values of 1 (ignore 0)
      if (dilate[x,y]  != 0)
      {
         // Initialize operand register with the current kernel value [x,y]
         Xcellarray.operand[0]  = dilate[x,y];

         II Write source image into memory array at the offset for each kernel element
         II This Write performs a MADD instruction  -> CELL += (CELL * operand)
         writelmage(source _image. Sxcellarrray. memory, x - XOFFSET, y - YOFFSET);
      }
   }
}
```

[0041] An example of a transformation of streaming mathematic morphology is shown in Figure 10 and an example of the transformation of a streaming MIN filter is shown in Figure 11.

[0042] Each cell in the memory 34 contains the following three features: 1) accumulation or summing into the cell, 2) operations or opcodes that act on the cell and a set of operands in programmable registers, and 3) numeric and data format conversions between various integer and floating point data types and bit resolutions.

[0043] In an embodiment, a specific set of opcodes may be implemented as needed to suit a specific task, including mathematical operations, Boolean logic operations, logical comparison operations, data conversion operations, transcendental function operations, or other operations that may be devised by one skilled in the art.

[0044] The nanoprocessors provide a set of mathematical and logical operations and numeric format conversions using an input operand and the current cell value

accumulated in the cell as shown below in equation 1, where one or more operands may be used in an embodiment:

Equation 1: Cell = Precision (Opcode(Cell * Operandi ... Operand *n*))

where:

Cell = existing value of the memory cell

Operand *1..n.* values to combine with the cell value via the opcode

Opcode: *math (+,-,*,/, ||,...) or Boolean (AND, OR, NOT, XOR) result accumulated in cell

Precision: numeric format conversions int(8,10,12,14,16,24,32,64), float(24,32,64), etc.

[0045] Each memory cell is an accumulator, and sums the results of each memory write into itself by combining the write value (operand) according to an opcode. Only a write into memory is needed for the memory cell to perform the computations, which allows DMA rate page writes and corresponding vectorization of computations, such as 4096 page writes and 4096 vectorized operations.

[0046] An opcode may use one or more operands. For example, a Write opcode operation using a single operandmay includethe following instruction format:

MADD cell = (cell * in + cell)
ADD cell = (cell + in)
SUBTRACT cell = (cell - in)
MULTIPLY cell = (cell * in)
DIVIDE cell = (cell / in)
XOR cell = (cell ^ in)
OR cell = (cell | in)
AND cell = (cell * in)
NOR cell = (!(cell | in))
NAND cell = (!(cell * in))
CONVERT (INT <-> FLOAT, resolution, truncation, etc. - this is a part of opcode)
OPERAND (the incoming value being written into the cell)

An example of an opcode using multiple operands in an embodiment could be an ADDCLIP instruction as follows:

ADDCLIP OPERANDI OPERAND2 CELL
Where:

OPERANDI = value to add to the cell
OPERAND2 = value to clip the addition result,

so that the result cannot be larger than OPERAND2
CELL = the memory cell where the addition result is stored

And the equation or pseudo code showing this operation is:

```
RESULT = CELL + OPERANDI
IF (RESULT > OPERAND2) RESULT = OPERAND2 // clipped result
CELL = RESULT
```

[0047] Each memory cell applies a data format conversion operation using the convert operation as follows. For read operations convert cell numeric format on memory read using convert operation. For write operations convert final value to desired numeric format according to convert operator.This allows any sort of common operation to be implemented such as area convolution, point operations, binary morphology, numeric conversions between float, double, int, etc.

[0048] In some embodiments, multiformat read and multiformat writes may be supported. This allows various numeric precisions to be used and converted on the fly. Numeric formats may include integer and float of various bit sizes. In one embodiment, only a subset of the numeric formats may be implemented to save silicon real estate and reduce power consumption. For example, one embodiment may support only integer (8, 12, 16, 32 bits) and float (24, 32 bits) numeric formats and conversions.

[0049] Each cell may store numeric data in an appropriate canonical numeric format to support the numeric conversions. The canonical format may vary in some embodiments.

[0050] Each memory cell in the array may have a dedicated nanoprocessor. However in other embodiments, a single vector of nanoprocessors corresponding to the memory page width may be shared among all the cells to support direct memory access page writes of 4,096 words together with the necessary processing. Thus some embodiments allow a single vector processing unit of a given size to be shared among vectors of memory cells rather than actually providing a dedicated nanoprocessor at each cell.

[0051] Figure 2 shows a streaming calculation by a direct memory access write operation. In this example, the data stream may be a 1920X1 080 image.A portion of

the width of the image, in one embodiment a 4K portion, is written to the receive buffer 20 as indicated by the write arrow in Figure 2.That 4K chunk is then moved to the working buffer 24 and another 4K chunk may be read across the width of the data stream to get it ready for subsequent operations in the controller. In the controller 26, there may be in one embodiment be 4K nanoprocessors each with an opcode 30 and an operand 28. Thus, a controller may include a nanocontroller for each bit of the chunk in one embodiment. It may also transfer each bit to the precision converter which changes either the precision or the type of data from integer to float or from float to integer. Then the data is stored into a row of memory cells in the memory array 34.

[0052] Thus referring to Figure 3, a sequence may be implemented in hardware, software and/or firmware. In software and firmware embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium such as an optical, magnetic or semiconductor memory. For example, the sequence of instructions may be stored in the controller 26 in Figure 2 in one embodiment.

[0053] The sequence begins when the host controller 12 (Figure 1) writes the opcode and operand to the controller 26 registers as indicated in block 46. The block code contains a bit precision information. In some embodiments, there may be multiple operands.

[0054] Then the host does a DMA write into a cell memory address as indicated in block 48. More particularly data may be copied into a receive buffer for calculations prior to going into the cell memory.

[0055] Next the controller 26 copies the DMA data into the working buffer 24 in Figure 2 as indicated in block 50. Next the controller reads the effected memory cells 34 to implement the calculation (block 52). Precision conversion may occur as set forth in the particular opcode.

[0056] Next the controller performs the operations specified by the opcode as indicated in block 54. He uses the operands and memory cellsas specified in the opcode in some embodiments. Finally the controller 26 writes the result into the effected memory cells as indicated in block 56.

[0057] The same thing can be done in the reverse order by using a DMA read operation for data format conversion. Thus looking at Figure 4, data may be read from the memory cells to the precision converter and passed by the controller to the working buffer 24 to receive buffer 20 and then read out to form a data stream.

[0058] Referring to Figure 5, the sequence for a streaming data format conversion using a DMA read operation may be implemented in software, firmware and/or hardware. In software and firmware embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium such as semiconductor, optical or magnetic storage. In some embodiments the sequence may be part of the controller 26.

[0059] The sequence begins when the host writes opcodes and operands to the controller registers as indicated in block 58. Then there is a host DMA read of the cell memory addresses as indicated in block 60.

[0060] Thereafter the controller copies (block 62) memory cell data through the precision converter 40 into the working buffer 24. Next the controller copies a working buffer into the receive buffer as indicated in block 64. Finally the host receives the receive buffer 20 DMA page as indicated in block 66.

[0061] While the 4K chunk is used in one embodiment, other chunk sizes may of course be used. The controller then performs the operation on each bit of the chunk in one embodiment.

[0062] Thegraphics processing techniques described herein may be implemented in various hardware architectures. For example, graphics functionality may be integrated within a chipset. Alternatively, a discrete graphics processor may be used. As still another embodiment, the graphicsfunctions may be implemented by a general purpose processor, including a multicore processor.

[0063] References throughout this specification to "one embodiment" or "an embodiment" mean that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one implementation encompassed within the present invention. Thus, appearances of the phrase "one embodiment" or "in an embodiment" are not necessarily referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may

be instituted in other suitable forms other than the particular embodiment illustrated and all such forms may be encompassed within the claims of the present application.

[0064] While the present invention has been described with respect to a limited number of embodiments, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this present invention.

What is claimed is:

1. A method comprising:

in a computer processor, converting operations in more than one dimension into a series of one dimensional operations.

2. The method of claim 1 including converting an area operation into a series of one dimensional operations.

3. The method of claim2 including converting an area operation into point operations implemented by memory writes into memory cells.

4. The method of claim 3 including combininga memory cell value with an operand according to a processing opcode and accumulating results back into the memory cell.

5. The method of claim 1 including:

programming a plurality of parallel processorswith the same operand and the same opcode; and

performing a plurality of parallel operations and storing the results in one line in a memory.

6. The method of claim 5 including performing a precision and numeric conversion in said processors.

7. The method of claim 5 wherein moving only data, and not instructions, along a processing pipeline.

8. The method of claim 5 including providing a parallel processor for each row of pixels in a frame.

9. The method of claim 8 including providing a storage and accumulationcell in said memory for each pixel.

10. The method of claim 9 including enabling each processor to perform both a point operation and an accumulationinto the storage cell.

11. A non-transitory computer readable medium storing instructions to implement a method comprising:

converting operations in more than one dimension into a series of one dimensional operations.

12. The medium of claim 11 including converting an area operation into a series of one dimensional operations.

13. The medium of claim 12 including converting an area operation into point operations implemented by memory writes into memory cells.

14. The medium of claim 13 including combininga memory cell value with an operand according to a processing opcode and accumulating results back into the memory cell.

15. The medium of claim 11 including:

programming a plurality of parallel processors with the same operand and the same opcode; and

performing a plurality of parallel operations and storing the results in one line in a memory.

16. An apparatus comprising:

a processor to convert operations in more than one dimension into a series of one dimensional operations; and

a storage coupled to said processor.

17. The apparatus of claim 16, said processor to convert area operations into a series of one dimensional operations.

18. The apparatus of claim 17, said processor to convert an area operation into point operations implemented by memory writes into memory cells.

19. The apparatus of claim 18, said processor to combine a memory cell value with an operand according to a processing opcode and accumulating results back into the memory cell.

20. The apparatus of claim 16 including a plurality of parallel processors programmed with the same operand and the same opcode, said processors to perform a plurality of parallel operations and store the results in one line in a memory.

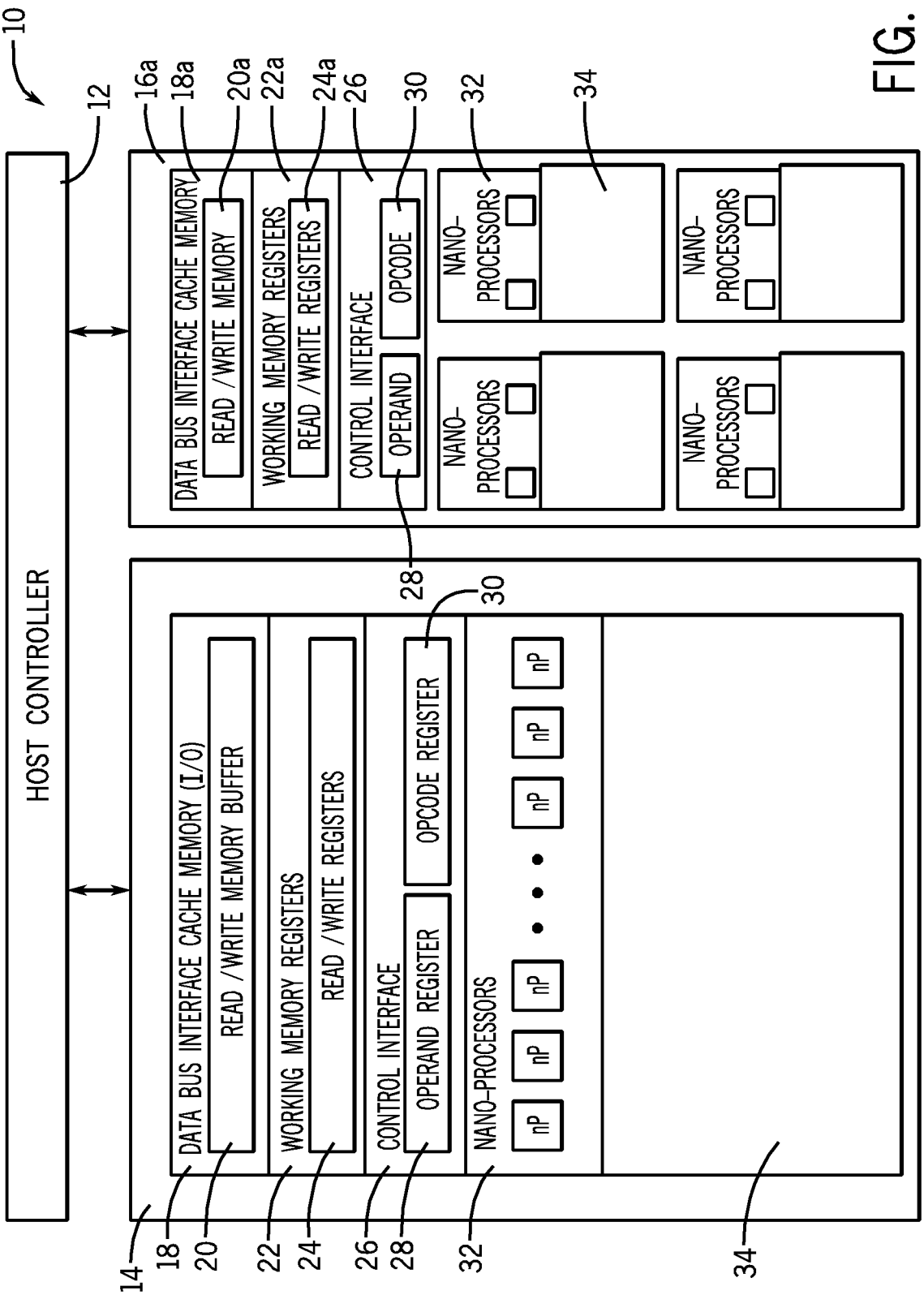
10
12
HOST CONTROLLER
16a
18a
20a
22a
24a
26
30
32
34
DATA BUS INTERFACE CACHE MEMORY
READ /WRITE MEMORY
WORKING MEMORY REGISTERS
READ /WRITE REGISTERS
CONTROL INTERFACE
OPCODE
OPERAND
NANO-PROCESSORS
28
DATA BUS INTERFACE CACHE MEMORY (I/O)
READ /WRITE MEMORY BUFFER
WORKING MEMORY REGISTERS
READ /WRITE REGISTERS
CONTROL INTERFACE
OPERAND REGISTER
OPCODE REGISTER
NANO-PROCESSORS
nP
14
18
20
22
24
26
28
32
34

FIG. 1

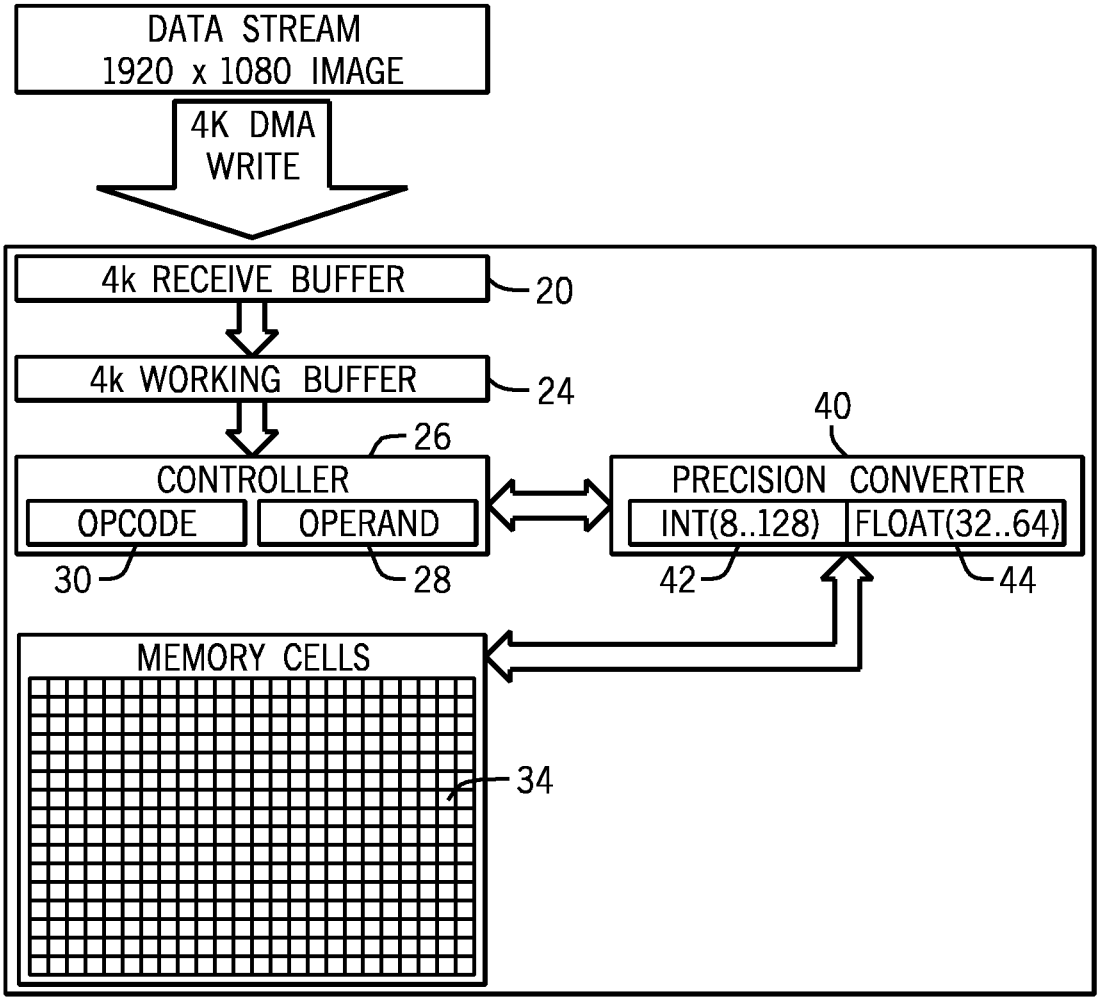
DATA STREAM
1920 x 1080 IMAGE
4K DMA WRITE
4k RECEIVE BUFFER
20
4k WORKING BUFFER
24
26
CONTROLLER
OPCODE
OPERAND
30
28
40
PRECISION CONVERTER
INT(8..128)
FLOAT(32..64)
42
44
MEMORY CELLS
34

FIG. 2

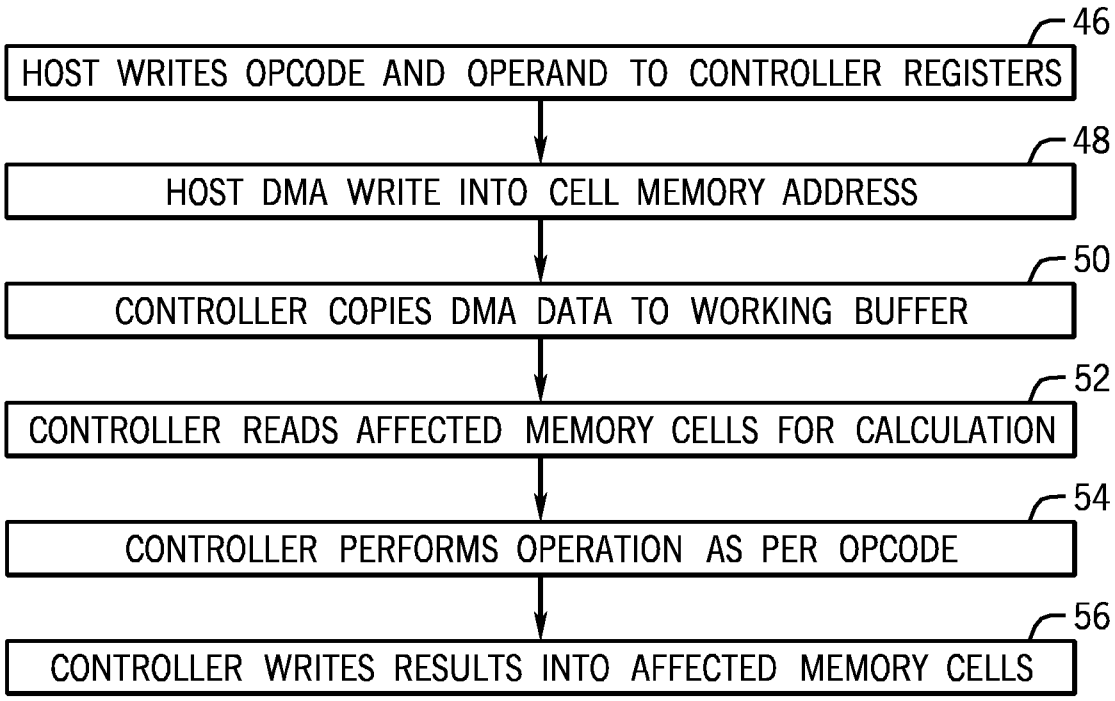
46
HOST WRITES OPCODE AND OPERAND TO CONTROLLER REGISTERS
48
HOST DMA WRITE INTO CELL MEMORY ADDRESS
50
CONTROLLER COPIES DMA DATA TO WORKING BUFFER
52
CONTROLLER READS AFFECTED MEMORY CELLS FOR CALCULATION
54
CONTROLLER PERFORMS OPERATION AS PER OPCODE
56
CONTROLLER WRITES RESULTS INTO AFFECTED MEMORY CELLS

FIG. 3

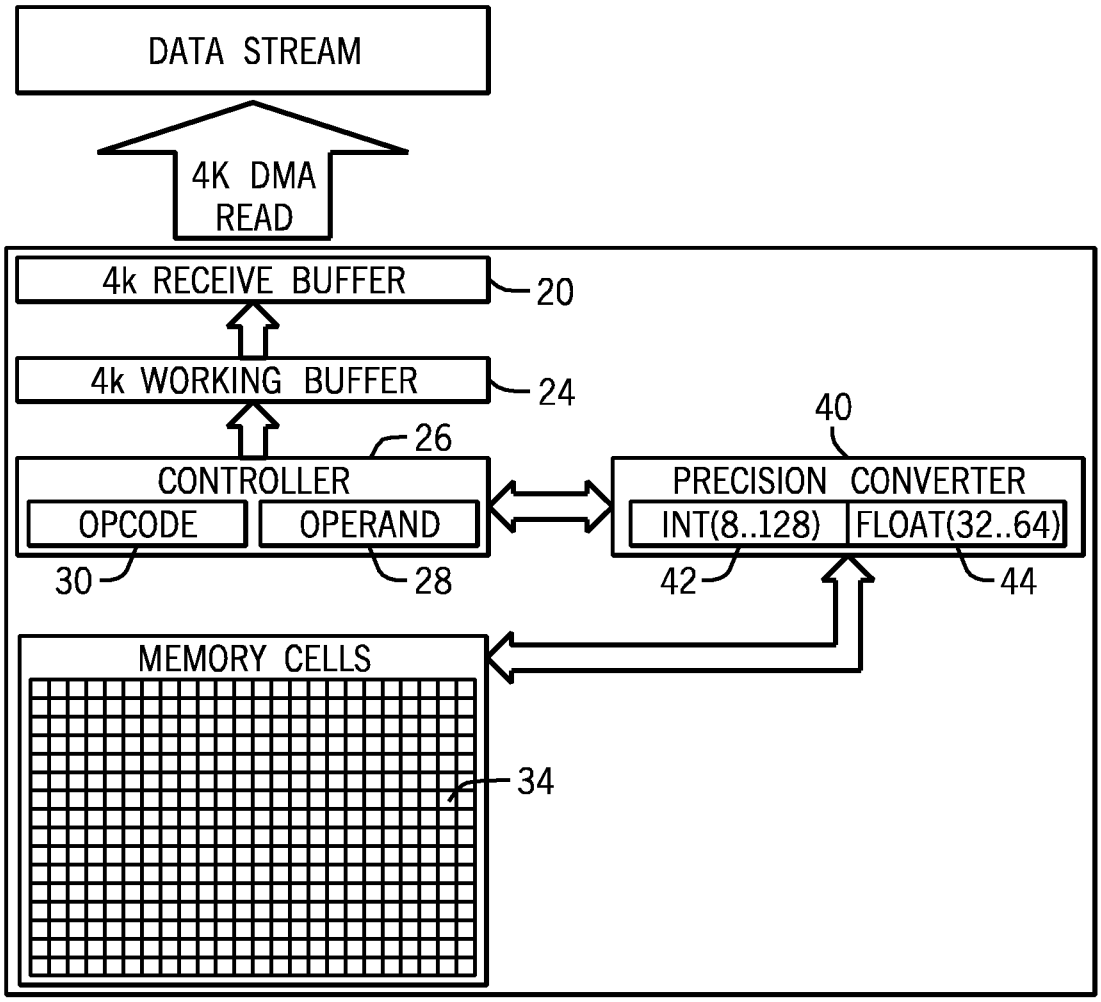
DATA STREAM
4K DMA READ
4k RECEIVE BUFFER
20
4k WORKING BUFFER
24
26
CONTROLLER
OPCODE
OPERAND
30
28
40
PRECISION CONVERTER
INT(8..128)
FLOAT(32..64)
42
44
MEMORY CELLS
34

FIG. 4

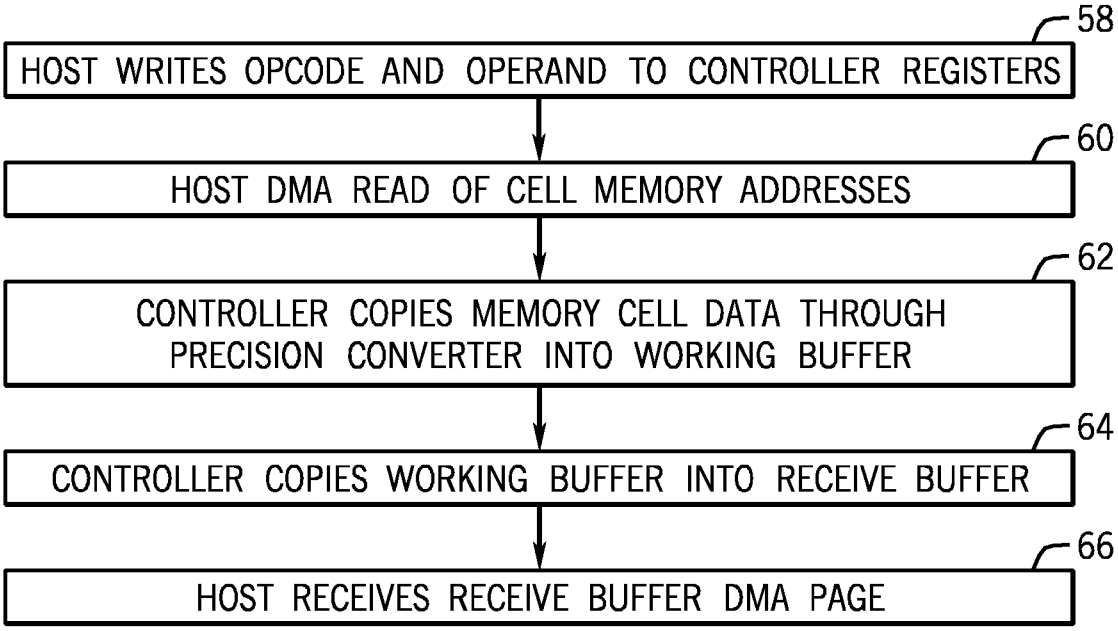

FIG. 5

INPUT IMAGE

x–1, y–1	x, y–1	x+1, y–1
x–1, y	x, y	x+1, y
x–1, y+1	x, y+1	x+1, y+1

CONVOLUTION

a	b	c
d	e	f
g	h	i

$$a(x-1, y-1)+b(x, y-1)+c(x+1, y-1)+$$
$$d(x-1, y)+e(x, y)+f(x+1, y)+$$
$$g(x-1, y+1)+h(x, y+1)+i(x+1, y+1)=$$
NEW OUTPUT PIXEL VALUE FOR x, y

FIG. 6

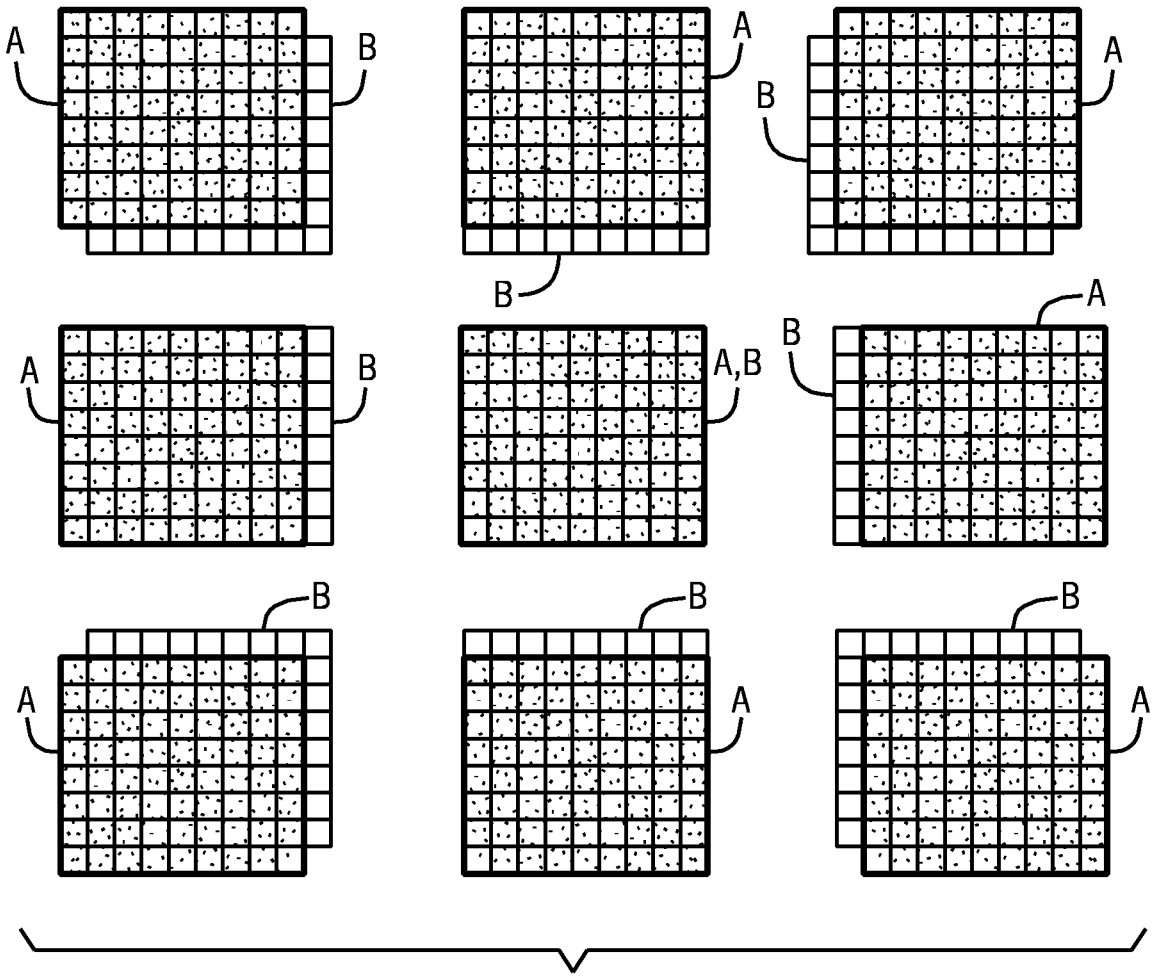

FIG. 7

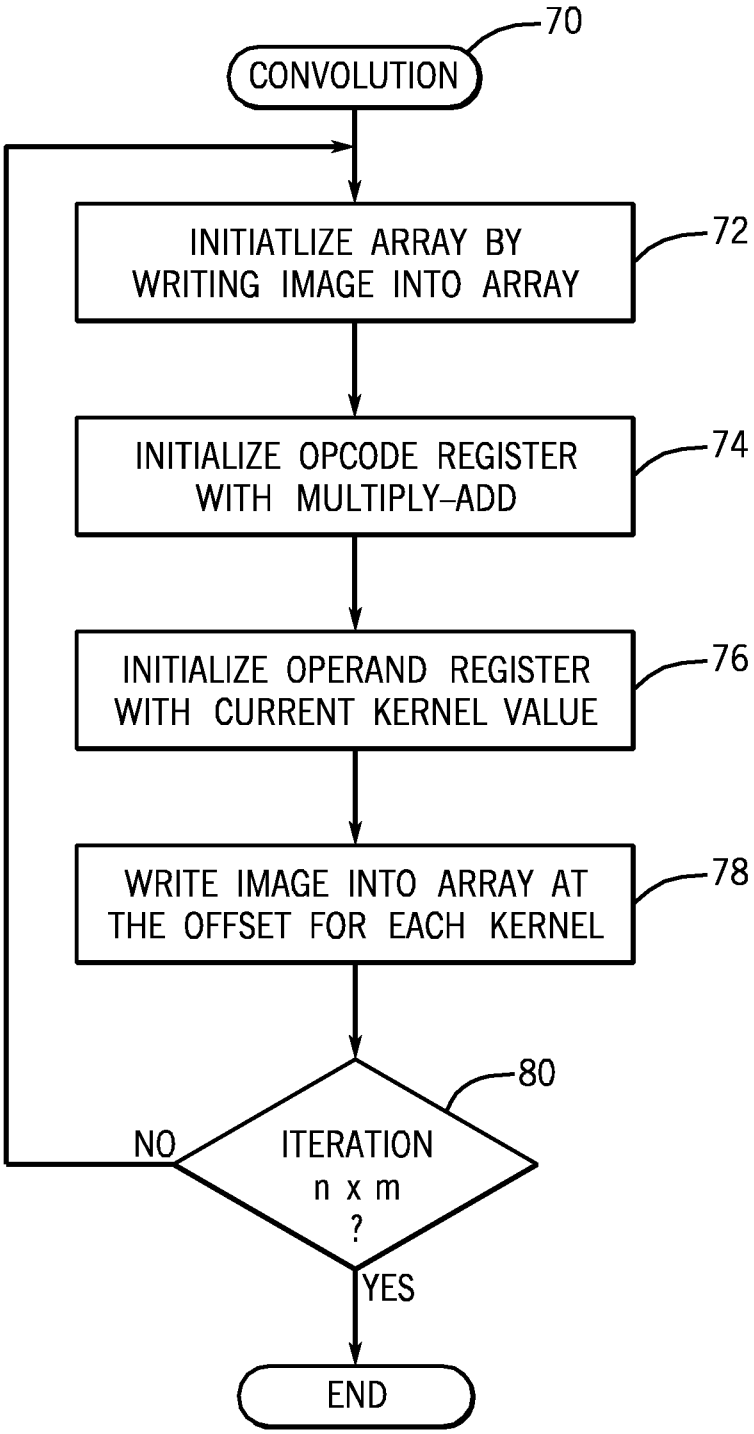
70
CONVOLUTION
INITIATLIZE ARRAY BY WRITING IMAGE INTO ARRAY
72
INITIALIZE OPCODE REGISTER WITH MULTIPLY–ADD
74
INITIALIZE OPERAND REGISTER WITH CURRENT KERNEL VALUE
76
WRITE IMAGE INTO ARRAY AT THE OFFSET FOR EACH KERNEL
78
80
NO
ITERATION n x m ?
YES
END

FIG. 8

Streaming Convolution (3x3 example)

Area Decomposition Transform: 3x3 Convolution Example, Sobel Horizontal Gradient

P1 = [x–1, y–1]	P2 = [x, y–1]	P3 = [x+1, y–1]
P4 = [x–1, y]	P5 = [x, y]	P6 = [x+1, y]
P7 = [x–1, y+1]	P8 = [x, y+1]	P9 = [x+1, y+1]

K1 = –1	K2 = –2	K3 = –1
K4 = 0	K5 = 0	K6 = 0
K7 = 0	K8 = 0	K9 = 0

FIG. 9

SET OPCODE REGISTER: MADD instruction, precision = 16-bits unsigned
Multiply WORKING BUFFER w/OPERAND, ADD to CELL, STORE in CELL

SET OPERAND REGISTER: K1 coefficient value
WRITE ENTIRE IMAGE Image Offset = P1 ((MADD each memory CELL with OPERAND)

SET OPERAND REGISTER: K2 coefficient value
WRITE ENTIRE IMAGE Image Offset = P2 (MADD each memory CELL with OPERAND)

SET OPERAND REGISTER: K3 coefficient value
WRITE ENTIRE IMAGE Image Offset = P3 (MADD each memory CELL with OPERAND)

SET OPERAND REGISTER: K4 coefficient value
WRITE ENTIRE IMAGE Image Offset = P4 (MADD each memory CELL with OPERAND)

SET OPERAND REGISTER: K5 coefficient value
WRITE ENTIRE IMAGE Image Offset = P5 (MADD each memory CELL with OPERAND)

SET OPERAND REGISTER: K6 coefficient value
WRITE ENTIRE IMAGE Image Offset = P6 (MADD each memory CELL with OPERAND)

SET OPERAND REGISTER: K7 coefficient value
WRITE ENTIRE IMAGE Image Offset = P7 (MADD each memory CELL with OPERAND)

SET OPERAND REGISTER: K8 coefficient value
WRITE ENTIRE IMAGE Image Offset = P8 (MADD each memory CELL with OPERAND)

SET OPERAND REGISTER: K9 coefficient value
WRITE ENTIRE IMAGE Image Offset = P9 (MADD each memory CELL with OPERAND)

SET OPCODE REGISTER DIVIDE instruction, precision = 8-bits unsigned
Divide each CELL by OPERAND, STORE result in CELL
SET OPERAND REGISTER: 9 (divide each pixel by 9)
WRITE ENTIRE IMAGE Image Offset = 0

CONVOLUTION FINISHED

Streaming Mathematical Binary Morphology (3x3 example)

P1 = [x–1, y–1]	P2 = [x, y–1]	P3 = [x+1, y–1]
P4 = [x–1, y]	P5 = [x, y]	P6 = [x+1, y]
P7 = [x–1, y+1]	P8 = [x, y+1]	P9 = [x+1, y+1]

K1 = 0	K2 = 1	K3 = 0
K4 = 1	K5 = 0	K6 = 1
K7 = 0	K8 = 1	K9 = 0

FIG. 10

Area Decomposition Transform: 3x3 Dilation Erosion Example

*NOTE: This example assumes:

1. That the source image is prepared as a binary image with only black & white pixels, where black = 0 and white = 1
2. That coefficients (K1 .. K9] equal to zero (0) do not need to be written into CELLS, this is an optimization so we only need to write K2, K4, K6 and K8 into cells.

SET OPCODE REGISTER: ANDADD instruction, precision = 8-bits unsigned
AND WORKING BUFFER w/OPERAND and CELL, STORE in CELL

SET OPERAND REGISTER: K2 coefficient value
WRITE ENTIRE IMAGE Image Offset = P1 ((ANDADD each memory CELL with OPERAND)
SET OPERAND REGISTER: K4 coefficient value
WRITE ENTIRE IMAGE Image Offset = P2 (ANDADD each memory CELL with OPERAND)
SET OPERAND REGISTER: K6 coefficient value
WRITE ENTIRE IMAGE Image Offset = P3 (ANDADD each memory CELL with OPERAND)
SET OPERAND REGISTER: K8 coefficient value
WRITE ENTIRE IMAGE Image Offset = P4 (ANDADD each memory CELL with OPERAND)
DILATION FINISHED

Streaming MIN filter (3x3 example)

P1 = [x–1, y–1]	P2 = [x, y–1]	P3 = [x+1, y–1]
P4 = [x–1, y]	P5 = [x, y]	P6 = [x+1, y]
P7 = [x–1, y+1]	P8 = [x, y+1]	P9 = [x+1, y+1]

K1 = 0	K2 = 1	K3 = 0
K4 = 1	K5 = 0	K6 = 1
K7 = 0	K8 = 1	K9 = 0

Area Decomposition Transform: 3x3 MIN Filter Example

SET OPCODE REGISTER: MINADD instruction, precision = 8-bits unsigned
Minimum value of WORKING BUFFER and CELL -> CELL
SET OPERAND REGISTER: DON'T CARE
WRITE ENTIRE IMAGE Image Offset = P1 MINADD of WORKING BUFFER and CELL -> CELL
WRITE ENTIRE IMAGE Image Offset = MINADD of WORKING BUFFER and CELL -> CELL
WRITE ENTIRE IMAGE Image Offset = P3 MINADD of WORKING BUFFER and CELL -> CELL
WRITE ENTIRE IMAGE Image Offset = MINADD of WORKING BUFFER and CELL -> CELL
WRITE ENTIRE IMAGE Image Offset = MINADD of WORKING BUFFER and CELL -> CELL
WRITE ENTIRE IMAGE Image Offset = P6 MINADD of WORKING BUFFER and CELL -> CELL
WRITE ENTIRE IMAGE Image Offset = MINADD of WORKING BUFFER and CELL -> CELL
WRITE ENTIRE IMAGE Image Offset = P8 MINADD of WORKING BUFFER and CELL -> CELL
WRITE ENTIRE IMAGE Image Offset = P9 MINADD of WORKING BUFFER and CELL -> CELL
MIN FILTER FINISHED

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2011/068067

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/30(2006.01)i, G06F 9/44(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 9/30; G06K 9/36; H04B 1/66; G06K 9/00; G06K 9/60

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) & Keywords: convert, multi, dimension, series, one.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 7844123 B2 (TAKAKURA, K. et al.) 30 November 2010 See column 13, lines 1-39, claim 1, and figures 1-2, 3a.	1,2,5,8,11,12,15 ,16,17,20 3-4,6-7,9-10,13-14 ,18-19
X A	US 04922544A A (STANSFIELD, P.W. et al.) 01 May 1990 See column 4, lines 11-65, claim 2, and figures 2,4.	1,2,11,12,16,17 3-10,13-15,18-20
X A	US 06130967A A (LEE, S.J. et al.) 10 October 2000 See column 7, lines 30-48, claim 1, and figure 6.	1,11,16 2-10,12-15,17-20
A	US 05841890A A (KRASKE, W.F.) 24 November 1998 See column 11, lines 26-37, claim 1, and figure 9.	1-20

☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
14 SEPTEMBER 2012 (14.09.2012)	**17 SEPTEMBER 2012 (17.09.2012)**

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea

Facsimile No. 82-42-472-7140

Authorized officer

Hwang, Seung Hee

Telephone No. 82-42-481-5749

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2011/068067

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 7844123 B2	30.11.2010	JP 04-772607 B2	01.07.2011
		JP 2008-017405 A	24.01.2008
		JP 4772607 B2	14.09.2011
		US 2008-0008248 A1	10.01.2008
US 04922544A A	01.05.1990	EP 0260139 A1	16.03.1988
		EP 0260139 B1	13.11.1991
		JP 02-509243 B2	16.04.1996
		JP 2509243 B2	19.06.1996
		JP 63-114372 A	19.05.1988
US 06130967A A	10.10.2000	WO 99-01809 A2	14.01.1999
US 05841890A A	24.11.1998	WO 97-42592 A1	13.11.1997