(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 3 286 725 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
06.10.2021 Bulletin 2021/40

(21) Application number: **16719164.2**

(22) Date of filing: **04.04.2016**

(51) Int Cl.:
G06T 1/60 *(2006.01)*

(86) International application number:
PCT/US2016/025895

(87) International publication number:
WO 2016/171882 (27.10.2016 Gazette 2016/43)

(54) **SHEET GENERATOR FOR IMAGE PROCESSOR**

SHEETGENERATOR FÜR EINEN BILDPROZESSOR

GÉNÉRATEUR DE "SHEET" POUR UN PROCESSEUR D'IMAGE

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR

(30) Priority: **23.04.2015 US 201514694806**

(43) Date of publication of application:
28.02.2018 Bulletin 2018/09

(73) Proprietor: **Google LLC**
Mountain View, CA 94043 (US)

(72) Inventors:
- **MEIXNER, Albert**
  Mountain View, CA 94043 (US)
- **REDGRAVE, Jason Rupert**
  Mountain View, CA 94043 (US)
- **SHACHAM, Ofer**
  Mountain View, CA 94043 (US)
- **ZHU, Qiuling**
  Mountain View, CA 94043 (US)
- **FINCHELSTEIN, Daniel Frederic**
  Mountain View, CA 94043 (US)

(74) Representative: **Suckling, Andrew Michael et al**
Marks & Clerk LLP
Fletcher House
The Oxford Science Park
Heatley Road
Oxford OX4 4GE (GB)

(56) References cited:
US-A1- 2014 164 737 US-A1- 2015 086 134

- **P H J Van Oosterhout: "OPTIMIZED PIXEL TEMPLATE IMAGE CORRELATOR", Master Thesis, 19 August 1992 (1992-08-19), pages 1-72, XP55279159, Retrieved from the Internet: URL:http://alexandria.tue.nl/extra2/afstverslE/563812.pdf [retrieved on 2016-06-09]**
- **S.G. DYKES ET AL: "Communication and computation patterns of large scale image convolutions on parallel architectures", PARALLEL PROCESSING SYMPOSIUM, 1994. PROCEEDINGS., EIGHTH INTERNATIONA L CANCUN, MEXICO 26-29 APRIL 1994, 1 January 1994 (1994-01-01), pages 926-931, XP055279443, DOI: 10.1109/IPPS.1994.288195 ISBN: 978-0-8186-5602-6**

EP 3 286 725 B1

Description

Field of Invention

[0001] The field of invention pertains generally to image processing, and, more specifically, to a sheet generator for an image processor.

Background

[0002] Image processing typically involves the processing of pixel values that are organized into an array. Here, a spatially organized two dimensional array captures the two dimensional nature of images (additional dimensions may include time (e.g., a sequence of two dimensional images) and data type (e.g., colors). In a typical scenario, the arrayed pixel values are provided by a camera that has generated a still image or a sequence of frames to capture images of motion. Traditional image processors typically fall on either side of two extremes.

[0003] A first extreme performs image processing tasks as software programs executing on a general purpose processor or general purpose-like processor (e.g., a general purpose processor with vector instruction enhancements). Although the first extreme typically provides a highly versatile application software development platform, its use of finer grained data structures combined with the associated overhead (e.g., instruction fetch and decode, handling of on-chip and off-chip data, speculative execution) ultimately results in larger amounts of energy being consumed per unit of data during execution of the program code.

[0004] A second, opposite extreme applies fixed function hardwired circuitry to much larger blocks of data. The use of larger (as opposed to finer grained) blocks of data applied directly to custom designed circuits greatly reduces power consumption per unit of data. However, the use of custom designed fixed function circuitry generally results in a limited set of tasks that the processor is able to perform. As such, the widely versatile programming environment (that is associated with the first extreme) is lacking in the second extreme.

[0005] The master thesis titled "Optimized Pixel Template Image Correlator" by P.H.J Van Oosterhout describes a custom chip named "Optimized Pixel Template Image Correlator (OPTIC)" that implements different kinds of picture processing techniques which are based on graylevel images and which can perform techniques at the speed of 20 megapixels per second. The OPTIC architecture comprises a configurable shift register array containing 64 8 bit registers. These registers are joined together in eight blocks of eight registers and form a shift array. Multiplexers at the beginning of each row allow data from a previous row or data from a data input to be routed to the shift array. This allows a pixel processing window to be configured in four different shapes, 1x64, 2x32, 4x16 or 8x8. While processing an image, the window moves along the image and the pixels inside the window are processed. To this end, image lines comprising the pixels inside the window are stored in a line buffer.

[0006] The paper titled "Communication and Computation Patterns of Large Scale Image Convolutions on Parallel Architectures" presents an analysis of a texture segmentation application containing a 96x96 convolution implemented on distributed memory multicomputers, wherein the parallel algorithms are tailored to each machine's architecture.

[0007] US 2015/086134 proposes a convolution image processor includes a load and store unit, a shift register unit, and a mapping unit. The load and store unit is configured to load and store image pixel data and allow for unaligned access of the image pixel data. The shift register is configured to load and store at least a portion of the image pixel data from the load and store unit and concurrently provide access to each image pixel value in the portion of the image pixel data. The mapping unit is configured to generate a number of shifted versions of image pixel data and corresponding stencil data from the portion of the image pixel data, and concurrently perform one or more operations on each image pixel value in the shifted versions of the portion of the image pixel data and a corresponding stencil value in the corresponding stencil data.

[0008] US 2014/164737 proposes a method for executing instructions on a single-program, multiple-data processor system having a fixed number of execution lanes, including: scheduling a primary instruction for execution with a first wave of multiple data; assigning the first wave to a corresponding primary subset of the execution lanes; scheduling a secondary instruction having a second wave of multiple data, such that the second wave fits in lanes that are unused by the primary subset of lanes; assigning the second wave to a corresponding secondary subset of the lanes; fetching the primary and secondary instructions; configuring the execution lanes such that the primary subset is responsive to the primary instruction and the secondary subset is simultaneously responsive to the secondary instruction; and simultaneously executing the primary and secondary instructions in the execution lanes.

[0009] A technology platform that provides for both highly versatile application software development opportunities combined with improved power efficiency per unit of data remains a desirable yet missing solution.

Summary

[0010] The present invention provides a method according to claim 1; a machine readable storage medium according to claim 10, the machine readable storage medium containing program code that when processed by a processor and/or controller causes such a method to be performed; and an apparatus according to claim 11. According to a further aspect of the present disclosure, a sheet generator circuit is described. The sheet gener-

ator includes electronic circuitry to receive a line group of image data including multiple rows of data from a frame of image data. The multiple rows are sufficient in number to encompass multiple neighboring overlapping stencils. The electronic circuitry is to parse the line group into a smaller sized sheet. The electronic circuitry is to load the sheet into a data computation unit having a two dimensional shift array structure coupled to an array of processors.

[0011] According to yet another aspect of the present disclosure, an apparatus is described having means for receiving a line group of image data including multiple rows of data from a frame of image data. The multiple rows are sufficient in number to encompass multiple neighboring overlapping stencils. The apparatus also includes means for parsing the line group into a smaller sized sheet. The apparatus also includes means for loading the sheet into a two dimensional shift array structure coupled to an array of processors. The apparatus also includes means for executing program code on the array of processors to process the multiple neighboring overlapping stencils over said sheet.

List of Figures

[0012] The following description and accompanying drawings are used to illustrate embodiments of the invention. In the drawings:

Fig. 1 shows an embodiment of an image processor hardware architecture;

Figs. 2a, 2b, 2c, 2d and 2e depict the parsing of image data into a line group, the parsing of a line group into a sheet and the operation performed on a sheet with overlapping stencils;

Fig. 3a shows an embodiment of a stencil processor;

Fig. 3b shows an embodiment of a instruction word of the stencil processor;

Fig. 4 shows an embodiment of a data computation unit within a stencil processor;

Figs. 5a, 5b, 5c, 5d, 5e, 5f, 5g, 5h, 5i, 5j and 5k depict an example of the use of a two-dimensional shift array and an execution lane array to determine a pair of neighboring output pixel values with overlapping stencils;

Fig. 6 shows an embodiment of a unit cell for an integrated execution lane array and two-dimensional shift array;

Fig. 7 pertains to a first operation performed by a sheet generator;

Fig. 8 pertains to a second operation performed by a sheet generator;

Fig. 9 pertains to a third operation performed by a sheet generator;

Fig. 10 pertains to a fourth operation performed by a sheet generator;

Fig. 11 pertains to a fifth operation performed by a sheet generator;

Fig. 12 pertains to a sixth operation performed by a sheet generator;

Fig. 13 shows an embodiment of a sheet generator;

Fig. 14 shows an embodiment of a computing system.

Detailed Description

a. image processor hardware architecture and operation

[0013] Fig. 1 shows an embodiment of an architecture 100 for an image processor implemented in hardware. The image processor may be targeted, for example, by a compiler that converts program code written for a virtual processor within a simulated environment into program code that is actually executed by the hardware processor. As observed in Fig. 1, the architecture 100 includes a plurality of line buffer units 101_1 through 101_M interconnected to a plurality of stencil processor units 102_1 through 102 N and corresponding sheet generator units 103_1 through 103 N through a network 104 (e.g., a network on chip (NOC) including an on chip switch network, an on chip ring network or other kind of network). In an embodiment, any line buffer unit may connect to any sheet generator and corresponding stencil processor through the network 104.

[0014] In an embodiment, program code is compiled and loaded onto a_corresponding stencil processor 102 to perform the image processing operations earlier defined by a software developer (program code may also be loaded onto the stencil processor's associated sheet generator 103, e.g., depending on design and implementation). In at least some instances an image processing pipeline may be realized by loading a first kernel program for a first pipeline stage into a first stencil processor 102_1, loading a second kernel program for a second pipeline stage into a second stencil processor 102_2, etc. where the first kernel performs the functions of the first stage of the pipeline, the second kernel performs the functions of the second stage of the pipeline, etc. and additional control flow methods are installed to pass output image data from one stage of the pipeline to the next stage of the pipeline.

[0015] In other configurations, the image processor may be realized as a parallel machine having two or more stencil processors 102_1, 102_2 operating the same kernel program code. For example, a highly dense and high data rate stream of image data may be processed by spreading frames across multiple stencil processors each of which perform the same function.

[0016] In yet other configurations, essentially any DAG of kernels may be loaded onto the hardware processor by configuring respective stencil processors with their own respective kernel of program code and configuring appropriate control flow hooks into the hardware to direct output images from one kernel to the input of a next kernel in the DAG design.

[0017] As a general flow, frames of image data are received by a macro I/O unit 105 and passed to one or more of the line buffer units 101 on a frame by frame basis. A particular line buffer unit parses its frame of image data into a smaller region of image data, referred to as a "a line group", and then passes the line group through the network 104 to a particular sheet generator. A complete or "full" singular line group may be composed, for example, with the data of multiple contiguous complete rows or columns of a frame (for simplicity the present specification will mainly refer to contiguous rows). The sheet generator further parses the line group of image data into a smaller region of image data, referred to as a "sheet", and presents the sheet to its corresponding stencil processor.

[0018] In the case of an image processing pipeline or a DAG flow having a single input, generally, input frames are directed to the same line buffer unit 101_1 which parses the image data into line groups and directs the line groups to the sheet generator 103_1 whose corresponding stencil processor 102_1 is executing the code of the first kernel in the pipeline/DAG. Upon completion of operations by the stencil processor 102_1 on the line groups it processes, the sheet generator 103_1 sends output line groups to a "downstream" line buffer unit 101_2 (in some use cases the output line group may be sent_back to the same line buffer unit 101_1 that earlier had sent the input line groups).

[0019] One or more "consumer" kernels that represent the next stage/operation in the pipeline/DAG executing on their own respective other sheet generator and stencil processor (e.g., sheet generator 103_2 and stencil processor 102_2) then receive from the downstream line buffer unit 101_2 the image data generated by the first stencil processor 102_1. In this manner, a "producer" kernel operating on a first stencil processor has its output data forwarded to a "consumer" kernel operating on a second stencil processor where the consumer kernel performs the next set of tasks after the producer kernel consistent with the design of the overall pipeline or DAG.

[0020] A stencil processor 102 is designed to simultaneously operate on multiple overlapping stencils of image data. The multiple overlapping stencils and internal hardware processing capacity of the stencil processor effectively determines the size of a sheet. Here, within a stencil processor 102, arrays of execution lanes operate in unison to simultaneously process the image data surface area covered by the multiple overlapping stencils.

[0021] As will be described in more detail below, in various embodiments, sheets of image data are loaded into a two-dimensional register array structure within the stencil processor 102. The use of sheets and the two-dimensional register array structure is believed to effectively provide for power consumption improvements by moving a large amount of data into a large amount of register space as, e.g., a single load operation with processing tasks performed directly on the data immediately thereafter by an execution lane array. Additionally, the use of an execution lane array and corresponding register array provide for different stencil sizes that are easily programmable/configurable.

[0022] Figs. 2a through 2e illustrate at a high level embodiments of both the parsing activity of a line buffer unit 101, the finer grained parsing activity of a sheet generator unit 103 as well as the stencil processing activity of the stencil processor 102 that is coupled to the sheet generator unit 103.

[0023] Fig. 2a depicts an embodiment of an input frame of image data 201. Fig. 2a also depicts an outline of three overlapping stencils 202 (each having a dimension of 3 pixels x 3 pixels) that a stencil processor is designed to operate over. The output pixel that each stencil respectively generates output image data for is highlighted in solid black. For simplicity, the three overlapping stencils 202 are depicted as overlapping only in the vertical direction. It is pertinent to recognize that in actuality a stencil processor may be designed to have overlapping stencils in both the vertical and horizontal directions.

[0024] Because of the vertical overlapping stencils 202 within the stencil processor, as observed in Fig. 2a, there exists a wide band of image data within the frame that a single stencil processor can operate over. As will be discussed in more detail below, in an embodiment, the stencil processors process data within their overlapping stencils in a left to right fashion across the image data (and then repeat for the next set of lines, in top to bottom order). Thus, as the stencil processors continue forward with their operation, the number of solid black output pixel blocks will grow right-wise horizontally. As discussed above, a line buffer unit 101 is responsible for parsing a line group of input image data from an incoming frame that is sufficient for the stencil processors to operate over for an extended number of upcoming cycles. An exemplary depiction of a line group is illustrated as a shaded region 203. In an embodiment, the line buffer unit 101 can comprehend different dynamics for sending/receiving a line group to/from a sheet generator. For example, according to one mode, referred to as "full group", the complete full width lines of image data are passed between a line buffer unit and a sheet generator. According to a second mode, referred to as "virtually tall", a line group is passed initially with a subset of full width rows. The remaining rows are then passed sequentially in smaller (less than full width) pieces.

[0025] With the line group 203 of the input image data having been defined by the line buffer unit and passed to the sheet generator unit, the sheet generator unit further parses the line group into finer sheets that are more precisely fitted to the hardware limitations of the stencil processor. More specifically, as will be described in more detail further below, in an embodiment, each stencil processor consists of a two dimensional shift register array. The two dimensional shift register array essentially shifts image data "beneath" an array of execution lanes where the pattern of the shifting causes each execution lane to operate on data within its own respective stencil (that is,

each execution lane processes on its own stencil of information to generate an output for that stencil). In an embodiment, sheets are surface areas of input image data that "fill" or are otherwise loaded into the two dimensional shift register array.

[0026] As will be described in more detail below, in various embodiments, there are actually multiple layers of two dimensional register data that can be shifted on any cycle. For convenience, much of the present description will simply use the term "two-dimensional shift register" and the like to refer to structures that have one or more such layers of two-dimensional register data that can be shifted.

[0027] Thus, as observed in Fig. 2b, the sheet generator parses an initial sheet 204 from the line group 203 and provides it to the stencil processor (here, the sheet of data corresponds to the shaded region that is generally identified by reference number 204). As observed in Figs. 2c and 2d, the stencil processor operates on the sheet of input image data by effectively moving the overlapping stencils 202 in a left to right fashion over the sheet. As of Fig. 2d, the number of pixels for which an output value could be calculated from the data within the sheet is exhausted (no other pixel positions can have an output value determined from the information within the sheet). For simplicity the border regions of the image have been ignored.

[0028] As observed in Fig. 2e the sheet generator then provides a next sheet 205 for the stencil processor to continue operations on. Note that the initial positions of the stencils as they begin operation on the next sheet is the next progression to the right from the point of exhaustion on the first sheet (as depicted previously in Fig. 2d). With the new sheet 205, the stencils will simply continue moving to the right as the stencil processor operates on the new sheet in the same manner as with the processing of the first sheet.

[0029] Note that there is some overlap between the data of the first sheet 204 and the data of the second sheet 205 owing to the border regions of stencils that surround an output pixel location. The overlap could be handled simply by the sheet generator re-transmitting the overlapping data twice. In alternate implementations, to feed a next sheet to the stencil processor, the sheet generator may proceed to only send new data to the stencil processor and the stencil processor reuses the overlapping data from the previous sheet.

b. stencil processor design and operation

[0030] Fig. 3a shows an embodiment of a stencil processor architecture 300. As observed in Fig. 3a, the stencil processor includes a data computation unit 301, a scalar processor 302 and associated memory 303 and an I/O unit 304. The data computation unit 301 includes an array of execution lanes 305, a two-dimensional shift array structure 306 and separate random access memories 307 associated with specific rows or columns of the array.

[0031] The I/O unit 304 is responsible for loading "input" sheets of data received from the sheet generator into the data computation unit 301 and storing "output" sheets of data from the stencil processor into the sheet generator. In an embodiment the loading of sheet data into the data computation unit 301 entails parsing a received sheet into rows/columns of image data and loading the rows/columns of image data into the two dimensional shift register structure 306 or respective random access memories 307 of the rows/columns of the execution lane array (described in more detail below). If the sheet is initially loaded into memories 307, the individual execution lanes within the execution lane array 305 may then load sheet data into the two-dimensional shift register structure 306 from the random access memories 307 when appropriate (e.g., as a load instruction just prior to operation on the sheet's data). Upon completion of the loading of a sheet of data into the register structure 306 (whether directly from a sheet generator or from memories 307), the execution lanes of the execution lane array 305 operate on the data and eventually "write back" finished data as a sheet directly back to the sheet generator, or, into the random access memories 307. If the later the I/O unit 304 fetches the data from the random access memories 307 to form an output sheet which is then forwarded to the sheet generator.

[0032] The scalar processor 302 includes a program controller 309 that reads the instructions of the stencil processor's program code from scalar memory 303 and issues the instructions to the execution lanes in the execution lane array 305. In an embodiment, a single same instruction is broadcast to all execution lanes within the array 305 to effect a SIMD-like behavior from the data computation unit 301. In an embodiment, the instruction format of the instructions read from scalar memory 303 and issued to the execution lanes of the execution lane array 305 includes a very-long-instruction-word (VLIW) type format that includes more than one opcode per instruction. In a further embodiment, the VLIW format includes both an ALU opcode that directs a mathematical function performed by each execution lane's ALU (which, as described below, in an embodiment may specify more than one traditional ALU operation) and a memory opcode (that directs a memory operation for a specific execution lane or set of execution lanes).

[0033] The term "execution lane" refers to a set of one or more execution units capable of executing an instruction (e.g., logic circuitry that can execute an instruction). An execution lane can, in various embodiments, include more processor-like functionality beyond just execution units, however. For example, besides one or more execution units, an execution lane may also include logic circuitry that decodes a received instruction, or, in the case of more MIMD-like designs, logic circuitry that fetches and decodes an instruction. With respect to MIMD-like approaches, although a centralized program control approach has largely been described herein, a more distributed approach may be implemented in various alter-

native embodiments (e.g., including program code and a program controller within each execution lane of the array 305).

[0034] The combination of an execution lane array 305, program controller 309 and two dimensional shift register structure 306 provides a widely adaptable/configurable hardware platform for a broad range of programmable functions. For example, application software developers are able to program kernels having a wide range of different functional capability as well as dimension (e.g., stencil size) given that the individual execution lanes are able to perform a wide variety of functions and are able to readily access input image data proximate to any output array location.

[0035] Apart from acting as a data store for image data being operated on by the execution lane array 305, the random access memories 307 may also keep one or more look-up tables. In various embodiments one or more scalar look-up tables may also be instantiated within the scalar memory 303.

[0036] A scalar look-up involves passing the same data value from the same look-up table from the same index to each of the execution lanes within the execution lane array 305. In various embodiments, the VLIW instruction format described above is expanded to also include a scalar opcode that directs a look-up operation performed by the scalar processor into a scalar look-up table. The index that is specified for use with the opcode may be an immediate operand or fetched from some other data storage location. Regardless, in an embodiment, a look-up from a scalar look-up table within scalar memory essentially involves broadcasting the same data value to all execution lanes within the execution lane array 305 during the same clock cycle. Additional details concerning use and operation of look-up tables is provided further below.

[0037] Fig. 3b summarizes the VLIW instruction word embodiments(s) discussed above. As observed in Fig. 3b, the VLIW instruction word format includes fields for three separate instructions: 1) a scalar instruction 351 that is executed by the scalar processor; 2) an ALU instruction 352 that is broadcasted and executed in SIMD fashion by the respective ALUs within the execution lane array; and, 3) a memory instruction 353 that is broadcasted and executed in a partial SIMD fashion (e.g., if execution lanes along a same row in the execution lane array share a same random access memory, then one execution lane from each of the different rows actually execute the instruction (the format of the memory instruction 353 may include an operand that identifies which execution lane from each row executes the instruction)

[0038] A field 354 for one or more immediate operands is also included. Which of the instructions 351, 352, 353 use which immediate operand information may be identified in the instruction format. Each of instructions 351, 352, 353 also include their own respective input operand and resultant information (e.g., local registers for ALU operations and a local register and a memory address for memory access instructions). In an embodiment, the scalar instruction 351 is executed by the scalar processor before the execution lanes within the execution lane array execute either of the other to instructions 352, 353. That is, the execution of the VLIW word includes a first cycle upon which the scalar instruction 351 is executed followed by a second cycle upon with the other instructions 352, 353 may be executed (note that in various embodiments instructions 352 and 353 may be executed in parallel).

[0039] In an embodiment, the scalar instructions executed by the scalar processor include commands issued to the sheet generator to load/store sheets from/into the memories or 2D shift register of the data computation unit. Here, the sheet generator's operation can be dependent on the operation of the line buffer unit or other variables that prevent pre-runtime comprehension of the number of cycles it will take the sheet generator to complete any command issued by the scalar processor. As such, in an embodiment, any VLIW word whose scalar instruction 351 corresponds to or otherwise causes a command to be issued to the sheet generator also includes no-operation (NOOP) instructions in the other two instruction field 352, 353. The program code then enters a loop of NOOP instructions for instruction fields 352, 353 until the sheet generator completes its load/store to/from the data computation unit. Here, upon issuing a command to the sheet generator, the scalar processor may set a bit of an interlock register that the sheet generator resets upon completion of the command. During the NOOP loop the scalar processor monitors the bit of the interlock bit. When the scalar processor detects that the sheet generator has completed its command normal execution begins again.

[0040] Fig. 4 shows an embodiment of a data computation component 401. As observed in Fig. 4, the data computation component 401 includes an array of execution lanes 405 that are logically positioned "above" a two-dimensional shift register array structure 406. As discussed above, in various embodiments, a sheet of image data provided by a sheet generator is loaded into the two-dimensional shift register 406. The execution lanes then operate on the sheet data from the register structure 406.

[0041] The execution lane array 405 and shift register structure 406 are fixed in position relative to one another. However, the data within the shift register array 406 shifts in a strategic and coordinated fashion to cause each execution lane in the execution lane array to process a different stencil within the data. As such, each execution lane determines the output image value for a different pixel in the output sheet being generated. From the architecture of Fig. 4 it should be clear that overlapping stencils are not only arranged vertically but also horizontally as the execution lane array 405 includes vertically adjacent execution lanes as well as horizontally adjacent execution lanes.

[0042] Some notable architectural features of the data computation unit 401 include the shift register structure

406 having wider dimensions than the execution lane array 405. That is, there is a "halo" of registers 409 outside the execution lane array 405. Although the halo 409 is shown to exist on two sides of the execution lane array, depending on implementation, the halo may exist on less (one) or more (three or four) sides of the execution lane array 405. The halo 405 serves to provide "spill-over" space for data that spills outside the bounds of the execution lane array 405 as the data is shifting "beneath" the execution lanes 405. As a simple case, a 5x5 stencil centered on the right edge of the execution lane array 405 will need four halo register locations further to the right when the stencil's leftmost pixels are processed. For ease of drawing, Fig. 4 shows the registers of the right side of the halo as only having horizontal shift connections and registers of the bottom side of the halo as only having vertical shift connections when, in a nominal embodiment, registers on either side (right, bottom) would have both horizontal and vertical connections.

[0043] Additional spill-over room is provided by random access memories 407 that are coupled to each row and/or each column in the array, or portions thereof (E.g., a random access memory may be assigned to a "region" of the execution lane array that spans 4 execution lanes row wise and 2 execution lanes column wise. For simplicity the remainder of the application will refer mainly to row and/or column based allocation schemes). Here, if an execution lane's kernel operations require it to process pixel values outside of the two-dimensional shift register array 406 (which some image processing routines may require) the plane of image data is able to further spill-over, e.g., from the halo region 409 into random access memory 407. For example, consider a 6X6 stencil where the hardware includes a halo region of only four storage elements to the right of an execution lane on the right edge of the execution lane array. In this case, the data would need to be shifted further to the right off the right edge of the halo 409 to fully process the stencil. Data that is shifted outside the halo region 409 would then spill-over to random access memory 407. Other applications of the random access memories 407 and the stencil processor of Fig. 3 are provided further below.

[0044] Figs. 5a through 5k demonstrate a working example of the manner in which image data is shifted within the two dimensional shift register array "beneath" the execution lane array as alluded to above. As observed in Fig. 5a, the data contents of the two dimensional shift array are depicted in a first array 507 and the execution lane array is depicted by a frame 505. Also, two neighboring execution lanes 510 within the execution lane array are simplistically depicted. In this simplistic depiction 510, each execution lane includes a register R1 that can accept data from the shift register, accept data from an ALU output (e.g., to behave as an accumulator across cycles), or write output data into an output destination.

[0045] Each execution lane also has available, in a local register R2, the contents "beneath" it in the two dimensional shift array. Thus, R1 is a physical register of the execution lane while R2 is a physical register of the two dimensional shift register array. The execution lane includes an ALU that can operate on operands provided by R1 and/or R2. As will be described in more detail further below, in an embodiment the shift register is actually implemented with multiple (a "depth" of) storage/register elements per array location but the shifting activity is limited to one plane of storage elements (e.g., only one plane of storage elements can shift per cycle). Figs. 5a through 5k depict one of these deeper register locations as being used to store the resultant X from the respective execution lanes. For illustrative ease the deeper resultant register is drawn alongside rather than beneath its counterpart register R2.

[0046] Figs. 5a through 5k focus on the calculation of two stencils whose central position is aligned with the pair of execution lane positions 511 depicted within the execution lane array. For ease of illustration, the pair of execution lanes 510 are drawn as horizontal neighbors when in fact, according to the following example, they are vertical neighbors.

[0047] As observed initially in Fig. 5a, the execution lanes are centered on their central stencil locations. Fig. 5b shows the object code executed by both execution lanes. As observed in Fig. 5b the program code of both execution lanes causes the data within the shift register array to shift down one position and shift right one position. This aligns both execution lanes to the upper left hand corner of their respective stencils. The program code then causes the data that is located (in R2) in their respective locations to be loaded into R1.

[0048] As observed in Fig. 5c the program code next causes the pair of execution lanes to shift the data within the shift register array one unit to the left which causes the value to the right of each execution lane's respective position to be shifted into each execution lane' position. The value in R1 (previous value) is then added with the new value that has shifted into the execution lane's position (in R2). The resultant is written into R1. As observed in Fig. 5d the same process as described above for Fig. 5c is repeated which causes the resultant R1 to now include the value A+B+C in the upper execution lane and F+G+H in the lower execution lane. At this point both execution lanes have processed the upper row of their respective stencils. Note the spill-over into a halo region on the left side of the execution lane array (if one exists on the left hand side) or into random access memory if a halo region does not exist on the left hand side of the execution lane array.

[0049] As observed in Fig. 5e, the program code next causes the data within the shift register array to shift one unit up which causes both execution lanes to be aligned with the right edge of the middle row of their respective stencils. Register R1 of both execution lanes currently includes the summation of the stencil's top row and the middle row's rightmost value. Figs. 5f and 5g demonstrate continued progress moving leftwise across the middle row of both execution lane's stencils. The accu-

mulative addition continues such that at the end of processing of Fig. 5g both execution lanes include the summation of the values of the top row and the middle row of their respective stencils.

[0050] Fig. 5h shows another shift to align each execution lane with its corresponding stencil's lowest row. Figs. 5i and 5j show continued shifting to complete processing over the course of both execution lanes' stencils. Fig. 5k shows additional shifting to align each execution lane with its correct position in the data array and write the resultant thereto.

[0051] In the example of Figs 5a-5k note that the object code for the shift operations may include an instruction format that identifies the direction and magnitude of the shift expressed in (X, Y) coordinates. For example, the object code for a shift up by one location may be expressed in object code as SHIFT 0, +1. As another example, a shift to the right by one location may expressed in object code as SHIFT +1, 0. In various embodiments shifts of larger magnitude may also be specified in object code (e.g., SHIFT 0, +2). Here, if the 2D shift register hardware only supports shifts by one location per cycle, the instruction may be interpreted by the machine to require multiple cycle execution, or, the 2D shift register hardware may be designed to support shifts by more than one location per cycle. Embodiments of the later are described in more detail further below.

[0052] Fig.6 shows another, more detailed depiction of the unit cell for the array execution lane and shift register structure (registers in the halo region do not include a corresponding execution lane). The execution lane and the register space associated with each location in the execution lane array is, in an embodiment, implemented by instantiating the circuitry observed in Fig.6 at each node of the execution lane array. As observed in Fig.6, the unit cell includes an execution lane 601 coupled to a register file 602 consisting of four registers R2 through R5. During any cycle, the execution lane 601 may read from or write to any of registers R1 through R5. For instructions requiring two input operands the execution lane may retrieve both of operands from any of R1 through R5.

[0053] In an embodiment, the two dimensional shift register structure is implemented by permitting, during a single cycle, the contents of any of (only) one of registers R2 through R4 to be shifted "out" to one of its neighbor's register files through output multiplexer 603, and, having the contents of any of (only) one of registers R2 through R4 replaced with content that is shifted "in" from a corresponding one if its neighbors through input multiplexers 604 such that shifts between neighbors are in a same direction (e.g., all execution lanes shift left, all execution lanes shift right, etc.). Although it may be common for a same register to have its contents shifted out and replaced with content that is shifted in on a same cycle, the multiplexer arrangement 603, 604 permits for different shift source and shift target registers within a same register file during a same cycle.

[0054] As depicted in Fig.6 note that during a shift sequence an execution lane will shift content out from its register file 602 to each of its left, right, top and bottom neighbors. In conjunction with the same shift sequence, the execution lane will also shift content into its register file from a particular one of its left, right, top and bottom neighbors. Again, the shift out target and shift in source should be consistent with a same shift direction for all execution lanes (e.g., if the shift out is to the right neighbor, the shift in should be from the left neighbor).

[0055] Although in one embodiment the content of only one register is permitted to be shifted per execution lane per cycle, other embodiments may permit the content of more than one register to be shifted in/out. For example, the content of two registers may be shifted out/in during a same cycle if a second instance of the multiplexer circuitry 603, 604 observed in Fig.6 is incorporated into the design of Fig.6. Of course, in embodiments where the content of only one register is permitted to be shifted per cycle, shifts from multiple registers may take place between mathematical operations by consuming more clock cycles for shifts between mathematical operations (e.g., the contents of two registers may be shifted between math ops by consuming two shift ops between the math ops).

[0056] If less than all the content of an execution lane's register files are shifted out during a shift sequence note that the content of the non shifted out registers of each execution lane remain in place (do not shift). As such, any non shifted content that is not replaced with shifted in content persists local to the execution lane across the shifting cycle. The memory unit ("M") observed in each execution lane is used to load/store data from/to the random access memory space that is associated with the execution lane's row and/or column within the execution lane array. Here, the M unit acts as a standard M unit in that it is often used to load/store data that cannot be loaded/stored from/to the execution lane's own register space. In various embodiments, the primary operation of the M unit is to write data from a local register into memory, and, read data from memory and write it into a local register.

[0057] With respect to the ISA opcodes supported by the ALU unit of the hardware execution lane 601, in various embodiments, the mathematical opcodes supported by the hardware ALU are integrally tied with (e.g., substantially the same as) the mathematical opcodes supported by a virtual execution lane (e.g., ADD, SUB, MOV, MUL, MAD, ABS, DIV, SHL, SHR, MIN/MAX, SEL, AND, OR, XOR, NOT). As described just above, memory access instructions can be executed by the execution lane 601 to fetch/store data from/to their associated random access memory. Additionally the hardware execution lane 601 supports shift op instructions (right, left, up, down) to shift data within the two dimensional shift register structure. As described above, program control instructions are largely executed by the scalar processor of the stencil processor.

c. sheet generator operation and design

[0058] Figs. 7-12 pertain to special considerations and/or operations of the sheet generator. As described above, a sheet generator is responsible for generating sheets of information for processing by a corresponding stencil processor. In order to impose wide versatility/programmability into the design of the overall processor, the sheet generator in some circumstances may need to perform additional operations in preparing an input sheet beyond just parsing appropriate sections from a received line group.

[0059] For example, in some cases the program code will call for simultaneously processing multiple channels of a same image. For example many video images have a red (R) channel, a blue (B) channel and green (G) channel. In an embodiment the sheet generator is implemented with a processor having associated memory and program code that executes out of the memory.

[0060] As observed in Fig. 7, in response to a need detected from the application software that the kernel will simultaneously process data from different channels (which may have been hinted at from a compiler) the program code executed by the sheet generator will proceed to form separate sheets along different "planes" (i.e., form a different sheet from each channel) and load them together into the data computation unit. That is, the sheet generator will generate an R sheet, a B sheet and a G sheet for a same section of the array and load all three sheets into the computation unit. The execution lanes within the execution lane array are then free to operate on the R, G and B sheets as needed (e.g., by storing an R sheet in one layer of the register file, a G sheet in the another layer of the register file and a B sheet in yet another layer of the register file).

[0061] Fig. 8 pertains to sheet generation for multi-dimensional input images. Here, although many input images are in the form of a simple array, in some cases each location of the array will correspond to a multi-dimensional data construct. As an illustrative example, Fig. 8 depicts an image where each array location contains 27 different values that correspond to different segments of 3x3x3 cube. Here, where each array location has a multi-dimensional data construct, the sheet generator will "unroll" the input array to form a separate sheet for each data construct dimension. Thus, as seen in Fig. 8, the sheet generator will generate 27 sheets (one for each cube segment) where each array location of each sheet across all the sheets contains a scalar value (one cube segment). The 27 sheets are then loaded into the stencil processor. The program code executed by the execution lanes within the execution lane array then operate on the 27 sheets with an understanding of the manner in which the multi-dimensional input array has been unrolled.

[0062] Fig. 9 pertains to a technique used to permit the execution lanes within the execution lane array to handle different data bit widths. Here, as is understood in the art, greater dynamic range is achieved by increasing the bit width of the data values (a 16 bit value can express values with greater dynamic range than an 8 bit value can). In an embodiment, the stencil processors are expected to operate on images having different bit widths such as 8, 16 or 32 bit pixel values. As such, according to one approach, the execution lanes themselves are 32 bit machines in the sense that the execution lanes internally can handle 32 bit operands.

[0063] However, to decrease the size and complexity of the two dimensional shift register, the individual storage elements of the registers within each execution lane's register file are limited to 8 bits. In the case of 8 bit image data there is no issue because an entire sheet of data can fit in one register of the register file. By contrast, in the case of 16 or 32 bit operands, the sheet generator generates multiple sheets to appropriately express the input operand data set.

[0064] For example, as depicted in Fig. 9 in the case of 16 bit input operands the sheet generator will generate a HI half sheet and a LO half sheet. The HI half sheet contains the upper 8 bits of each data item at the correct array location. The LO half sheet contains the lower 8 bits of each data item at the correct array location. 16 bit operations are then performed by loading both sheets into the stencil processor and informing the execution lane hardware (e.g., via an immediate value in the program code) that 16 bit operation is to take place. Here, as just one possible mode of operation, both the HI and LO sheets are loaded in two different registers of each execution lanes register file.

[0065] The execution lane units are able to internally construct the correct operands by first reading from one of the register file locations and appending the data therein with the data read from another of the register file locations. Similarly, in the write direction, the execution lane units will have to perform two writes. Specifically, a first write of the lower 8 bits to a first register of the register file containing the LO sheet and then a second write of the upper 8 bits to a second register of the register file containing the HI sheet.

[0066] Recall from the discussion of Fig. 12 that in various embodiment shifts the content of only one register is permitted to be shifted per cycle. As such, in order to move 16 bit data values around the two dimensional shift register structure, two cycles are consumed per shift sequence (between math ops) rather than one cycle in the case of 8 bit data values. That is, in the nominal case of 8 bit data values, all data can be shifted between locations in a single cycle. By contrast in the case of 16 bit data values, two 8 bit values have to be shifted per shift register shift operation (the HI half sheet and the LO half sheet). In an embodiment, in the case of 32 bits, the same principles apply except that four sheets are created to represent the entire image data rather than two sheets. Likewise, as many as four cycles may need to be consumed per shift sequence.

[0067] Fig. 10 pertains to situations where the image processor "up-samples" the input image data from a low-

er density resolution to a higher density resolution. Here, the stencil processors are responsible for generating more output values per unit area of an image than the input image contains. The sheet generator handles the up-sampling problem by repeating a same data value across a sheet such that the sheet data value density corresponds to the up-sampled (higher density) output image. That is, for example in the case where the output execution lane array density corresponds to 4:1 up-sampling in view of the density of the input image (four output pixels for every input pixel), as observed in Fig. 10, the sheet generator manufactures a sheet with four identical values for every input value.

[0068] Fig. 11 pertains to the reverse situation of "down-sampling". In the case of down-sampling, the sheet generator will generate more sheets than for a lower density input image. Specifically, if the input image has a factor of S higher resolution in one (e.g., X) direction and a factor of T higher resolution in the other (e.g., Y) direction, the sheet generator will generate S*T sheets from an initial more dense initial sheet. This effectively assigns more input pixels to any particular output pixel.

[0069] Fig. 12 pertains to situations where the mathematical operations performed by the execution lanes within the execution lane array require a larger surface area of image data than the size of the two-dimensional shift register structure. As observed in Fig. 12, the sheet to be loaded into the two-dimensional shift register structure for processing corresponds to the shaded region 1201 of an input frame. The mathematical operations that will calculate output values for array locations within the shaded area, however, require values within the frame that is bounded by the dashed border 1202 observed in Fig. 12. Thus, there exists a large "support region" outside the surface area of the two-dimensional shift register structure that will be included in the operations.

[0070] Under these conditions the sheet generator will not only load a sheet corresponding to the shaded region 1201 into the stencil processor but will also load the three (unshaded) neighboring sheets into the data computation unit. The program code executed by the execution lanes will call in and move out sheets to/from random access memory as needed and/or store some or all of the sheets in the deeper registers of the two dimensional shift register array.

[0071] Fig. 13 provides an embodiment of the hardware design 1300 for the sheet generator. As observed in Fig. 13, in an embodiment, the sheet generator is implemented as a computing system having a processor/controller 1301 that executes program code stored in memory 1302 to perform sheet generator tasks such as any of the tasks described above with respect to Figs. 7-12. The sheet generator also includes an I/O unit 1303 for receiving/sending line groups from/to the network and receiving/sending sheets from/to the sheet generator's associated stencil processor.

[0072] A pertinent feature of the sheet generator is its configuration space 1304 which may be implemented as separate register space within the sheet generator (as depicted in Fig. 13), within the processor/controller 1301 and/or within memory 1302. The configuration space 1304 lends itself to the wide adaptability and programmability of the overall platform. Here, settings made in the configuration space 1304 may include, e.g., pertinent image features and dimensions such as frame size, line group size, sheet size, input image pixel resolution, output image pixel resolution, etc. The program code within memory 1302 then uses the information within configuration space as input variables to effect correct operation on correctly sized sheets, etc.

[0073] Alternatively or in some combination, the wide adaptability and programmability of the overall platform may be realized by loading custom program code into memory 1302 for a particular application and/or image dimension(s). Here, for example, a compiler may be able to make easy reference to the X, Y coordinates of the position relative addressing scheme and/or any of frame size and line group size to easily determine sheet sizes, sheet boundaries, etc and customize generic program code templates into software programs that are specific to the image processing task at hand. Likewise, any such translation and practical use of the relative positioning or other image dimensions may be entered into configuration space 1304 where program code existent on the sheet generator makes determinations of sheet boundaries, sheet sizes, etc.

d. implementation embodiments

[0074] It is pertinent to point out that the various image processor architecture features described above are not necessarily limited to image processing in the traditional sense and therefore may be applied to other applications that may (or may not) cause the image processor to be re-characterized. For example, if any of the various image processor architecture features described above were to be used in the creation and/or generation and/or rendering of animation as opposed to the processing of actual camera images, the image processor may be characterized as a graphics processing unit. Additionally, the image processor architectural features described above may be applied to other technical applications such as video processing, vision processing, image recognition and/or machine learning. Applied in this manner, the image processor may be integrated with (e.g., as a co-processor to) a more general purpose processor (e.g., that is or is part of a CPU of computing system), or, may be a stand alone processor within a computing system.

[0075] The hardware design embodiments discussed above may be embodied within a semiconductor chip and/or as a description of a circuit design for eventual targeting toward a semiconductor manufacturing process. In the case of the later, such circuit descriptions may take of the form of higher/behavioral level circuit descriptions (e.g., a VHDL description) or lower level circuit description (e.g., a register transfer level (RTL) description,

transistor level description or mask description) or various combinations thereof. Circuit descriptions are typically embodied on a computer readable storage medium (such as a CD-ROM or other type of storage technology).

[0076] From the preceding sections is pertinent to recognize that an image processor as described above may be embodied in hardware on a computer system (e.g., as part of a handheld device's System on Chip (SOC) that processes data from the handheld device's camera). In cases where the image processor is embodied as a hardware circuit, note that the image data that is processed by the image processor may be received directly from a camera. Here, the image processor may be part of a discrete camera, or, part of a computing system having an integrated camera. In the case of the later the image data may be received directly from the camera or from the computing system's system memory (e.g., the camera sends its image data to system memory rather than the image processor). Note also that many of the features described in the preceding sections may be applicable to a graphics processor unit (which renders animation).

[0077] Fig. 14 provides an exemplary depiction of a computing system. Many of the components of the computing system described below are applicable to a computing system having an integrated camera and associated image processor (e.g., a handheld device such as a smartphone or tablet computer). Those of ordinary skill will be able to easily delineate between the two.

[0078] As observed in Fig. 14, the basic computing system may include a central processing unit 1401 (which may include, e.g., a plurality of general purpose processing cores 1415_1 through 1415 N and a main memory controller 1417 disposed on a multi-core processor or applications processor), system memory 1402, a display 1403 (e.g., touchscreen, flat-panel), a local wired point-to-point link (e.g., USB) interface 1404, various network I/O functions 1405 (such as an Ethernet interface and/or cellular modem subsystem), a wireless local area network (e.g., WiFi) interface 1406, a wireless point-to-point link (e.g., Bluetooth) interface 1407 and a Global Positioning System interface 1408, various sensors 1409_1 through 1409 N, one or more cameras 1410, a battery 1414, a power management control unit 1412, a speaker and microphone 1413 and an audio coder/decoder 1414.

[0079] An applications processor or multi-core processor 1450 may include one or more general purpose processing cores 1415 within its CPU 1401, one or more graphical processing units 1416, a memory management function 1417 (e.g., a memory controller), an I/O control function 1418 and an image processing unit 1419. The general purpose processing cores 1415 typically execute the operating system and application software of the computing system. The graphics processing units 1416 typically execute graphics intensive functions to, e.g., generate graphics information that is presented on the display 1403. The memory control function 1417 interfaces with the system memory 1402 to write/read data to/from system memory 1402. The power management control unit 1412 generally controls the power consumption of the system 1400.

[0080] The image processing unit 1419 may be implemented according to any of the image processing unit embodiments described at length above in the preceding sections. Alternatively or in combination, the IPU 1419 may be coupled to either or both of the GPU 1416 and CPU 1401 as a co-processor thereof. Additionally, in various embodiments, the GPU 1416 may be implemented with any of the image processor features described at length above.

[0081] Each of the touchscreen display 1403, the communication interfaces 1404 - 1407, the GPS interface 1408, the sensors 1409, the camera 1410, and the speaker/microphone codec 1413, 1414 all can be viewed as various forms of I/O (input and/or output) relative to the overall computing system including, where appropriate, an integrated peripheral device as well (e.g., the one or more cameras 1410). Depending on implementation, various ones of these I/O components may be integrated on the applications processor/multi-core processor 1450 or may be located off the die or outside the package of the applications processor/multi-core processor 1450.

[0082] In an embodiment one or more cameras 1410 includes a depth camera capable of measuring depth between the camera and an object in its field of view. Application software, operating system software, device driver software and/or firmware executing on a general purpose CPU core (or other functional block having an instruction execution pipeline to execute program code) of an applications processor or other processor may perform any of the functions described above.

[0083] Embodiments of the invention may include various processes as set forth above. The processes may be embodied in machine-executable instructions. The instructions can be used to cause a general-purpose or special-purpose processor to perform certain processes. Alternatively, these processes may be performed by specific hardware components that contain hardwired logic for performing the processes, or by any combination of programmed computer components and custom hardware components.

[0084] Elements of the present invention may also be provided as a machine-readable medium for storing the machine-executable instructions. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs, and magneto-optical disks, FLASH memory, ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, propagation media or other type of media/machine-readable medium suitable for storing electronic instructions. For example, the present invention may be downloaded as a computer program which may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection).

[0085] In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto within the scope of the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

Claims

1. A method, comprising:

receiving, by a sheet generator (103_1) and from a line buffer (101_1 ... 101_M), a line group (203) of image data including multiple rows of data from a frame (201) of image data, said multiple rows being sufficient in number to encompass multiple neighboring overlapping stencils (202);
parsing, by the sheet generator (103_1), said line group (203) into a plurality of smaller sized sheets (204, 205); and, for each sheet of the plurality of sheets:

loading, by the sheet generator (103_1), said sheet (204, 205) into a stencil processor (102-1) associated with the sheet generator and comprising a two dimensional shift array structure (306, 406, 602) coupled to an array (305, 405) of execution lanes (510, 601), the execution lane array (305,405) and the shift array structure (306, 406, 602) being fixed in position relative to one another;
executing program code on said array (305, 405) of execution lanes (510, 601) to process said multiple neighboring overlapping stencils (202) over said sheet (204, 205), wherein each execution lane (510, 601) processes its own stencil to generate an output value for that stencil, and wherein the output values of the execution lanes constitute respective pixel values in an output sheet of image data; and
receiving, by the sheet generator (103_1), the output sheet of image data from the associated stencil processor (102_1).

2. The method of claim 1 wherein said neighboring overlapping stencils (202) extend in both horizontal and vertical directions.

3. The method of claim 1 further comprising:

parsing line groups (203) from more than one channel;
parsing more than one sheet (204, 205) from the more than one channel to create at least two sheets from different channels;
loading said at least two sheets into said two dimensional shift array structure (306, 406, 602);
executing program code on said array (305, 405) of execution lanes (510, 601) to process said multiple neighboring overlapping stencils (202) concurrently over said at least two sheets;
wherein the two dimensional shift array structure (306,406, 602) comprises a plurality of layers, and wherein loading said at least two sheets into said two dimensional shift array structure (306,406, 602) comprises loading a first sheet having data from a first channel of the more than one channel into a first layer of the plurality of layers and loading a second sheet having data from a second channel of the more than one channel into a second layer of the plurality of layers.

4. The method of claim 1 further comprising:

parsing a line group (203) whose frame (201) is structured as an array having a multi-dimensional feature at an array location of the frame (201);
parsing the line group (203) into multiple sheets (204, 205) such that there exists a different sheet for each of the different dimensions of the feature.

5. The method of claim 1 wherein the register array (306, 406, 602) contains registers of a first bit width that is smaller than a second bit width at which the execution lanes (510, 601) are able to process data, the method further comprising:

receiving a line group (203) whose data values have the second bit width;
creating first and second sheets (204, 205) from the line group, the first sheet having values of the first size and containing an upper portion of the data values, the second sheet having values of the first size containing a lower portion of the data values;
loading the first and second sheets into the shift register array (306, 406, 602);
concurrently processing the first and second sheets with the array (305, 405) of execution lanes (510, 601) to process the data at the second bit width.

6. The method of claim 1 wherein the method further comprises one of:

effecting up-sampling by replicating a data value from a location of the line group (203) multiple times over at a corresponding location on the

sheet (204, 205); and
effecting down-sampling by creating a second output sheet from the sheet (204, 205), the second output sheet having a density of pixel values that is less than the density of pixel values of the sheet (204, 205), wherein, no data is lost when comparing the content of the line group (203) that is mapped to the sheet (204, 205) and the second output sheet.

7. The method of claim 1 wherein the method further comprises:

recognizing that processing over the stencils (202) requires more image surface area than the sheet (204, 205);
parsing at least one more sheet from the line group (203) that neighbors the sheet; and,
loading the at least one more sheet into storage space coupled to the execution lanes (510, 601).

8. The method of claim 7 wherein the storage space is the two-dimensional shift array (306, 406, 602).

9. The method of claim 1 wherein the method further comprises performing the parsing by executing program code on a processor and/or controller (1301).

10. A machine readable storage medium containing program code that when processed by a processor and/or controller causes a method according to any of claims 1 to 9 to be performed.

11. An apparatus (1400), comprising:
a sheet generator circuit (103, 1300) comprising electronic circuitry to

receive, from a line buffer (101_1 ... 101_M), a line group (203) of image data including multiple rows of data from a frame (201) of image data, said multiple rows being sufficient in number to encompass multiple neighboring overlapping stencils (202);
parse said line group into a smaller sized sheet (204, 205);
load said sheet (204, 205) into a data computation unit (301, 401) of an associated stencil processor (102_1), the data computation unit having a two dimensional shift array structure (306, 406, 602) coupled to an array (305, 405) of execution lanes (510, 601), the execution lane array (305,405) and the shift array structure (306, 406, 602) being fixed in position relative to one another, the stencil processor processing said multiple neighboring overlapping stencils (202) over said sheet (204, 205), wherein each execution lane (510, 601) is configured to process its own stencil to generate an output value for that stencil, and wherein the output values of the execution lanes constitute respective values in an output sheet of image data; and
receive the output sheet of image data from the associated stencil processor (102_1).

12. The apparatus (1400) of claim 11 wherein said neighboring overlapping stencils (202) extend in both horizontal and vertical directions.

13. The apparatus (1400) of claim 11 wherein said electronic circuitry is further to:

parse line groups (203) from more than one channel;
parse more than one sheet (204, 205) from the more than one channel to create at least two sheets from different channels;
load said at least two sheets into said data computation unit (301, 401).

14. The apparatus (1400) of claim 11 wherein the electronic circuitry is to:

receive a line group (203) whose data values have the second bit width;
create first and second sheets from the line group (203), the first sheet having values of the first size and containing an upper portion of the data values, the second sheet having values of the first size containing a lower portion of the data values;
load the first and second sheets into the data computation unit (301, 401).

15. The apparatus (1400) of claim 11 wherein the electronic circuitry is one of:

to effect up-sampling by replicating a data value from a location of the line group (203) multiple times over at a corresponding location on the sheet (204, 205); and
to effect down-sampling by creating a second output sheet from the sheet (204, 205), the second output sheet having a density of pixel values that is less than the density of pixel values of the sheet (204, 205), wherein, no data is lost when comparing the content of the line group (203) that is mapped to the sheet (204, 205) and the second output sheet.

Patentansprüche

1. Verfahren, umfassend:

Empfangen einer Zeilengruppe (203) von Bilddaten, die mehrere Zeilen von Daten aus einem

Rahmen (201) von Bilddaten einschließt, durch einen Blatterzeuger (103_1) und von einem Zeilenpuffer (101_1 ... 101_M), wobei die mehreren Zeilen zahlenmäßig ausreichend sind, um mehrere benachbarte überlappende Schablonen (202) zu umschließen;
Parsen der Zeilengruppe (203) durch den Blatterzeuger (103_1) in eine Vielzahl von Blättern geringerer Größe (204, 205); und, für jedes Blatt der Vielzahl von Blättern:

Laden des Blatts (204, 205) durch den Blatterzeuger (103_1) in einen Schablonenprozessor (102_1), der dem Blatterzeuger zugeordnet ist und eine zweidimensionale Verschiebe-Arraystruktur (306, 406, 602) umfasst, die mit einem Array (305, 405) von Ausführungsbahnen (510, 601) gekoppelt ist, wobei das Ausführungsbahn-Array (305, 405) und die Verschiebe-Arraystruktur (306, 406, 602) in ihrer Position relativ zueinander festgelegt sind;
Ausführen von Programmcode auf dem Array (305, 405) von Ausführungsbahnen (510, 601), um die mehreren benachbarten überlappenden Schablonen (202) auf dem Blatt (204, 205) zu verarbeiten, wobei jede Ausführungsbahn (510, 601) ihre eigene Schablone verarbeitet, um einen Ausgabewert für diese Schablone zu erzeugen, und wobei die Ausgabewerte der Ausführungsbahnen jeweilige Pixelwerte in einem Ausgabeblatt aus Bilddaten bilden; und
Empfangen des Ausgabeblatts von Bilddaten durch den Blatterzeuger (103_1) von dem zugeordneten Schablonenprozessor (102_1).

2. Verfahren nach Anspruch 1, wobei die benachbarten überlappenden Schablonen (202) sich sowohl in horizontaler als auch in vertikaler Richtung erstrecken.

3. Verfahren nach Anspruch 1, ferner umfassend:

Parsen von Zeilengruppen (203) aus mehr als einem Kanal;
Parsen von mehr als einem Blatt (204, 205) aus dem mehr als einen Kanal, um mindestens zwei Blätter aus unterschiedlichen Kanälen zu erzeugen;
Laden der mindestens zwei Blätter in die zweidimensionale Verschiebe-Arraystruktur (306, 406, 602);
Ausführen von Programmcode auf dem Array (305, 405) von Ausführungsbahnen (510, 601), um die mehreren benachbarten überlappenden Schablonen (202) gleichzeitig auf den mindestens zwei Blättern zu verarbeiten;
wobei die zweidimensionale Verschiebe-Arraystruktur (306, 406, 602) eine Vielzahl von Schichten umfasst, und wobei das Laden der mindestens zwei Blätter in die zweidimensionale Verschiebe-Arraystruktur (306, 406, 602), das Laden eines ersten Blatts mit Daten aus einem ersten Kanal des mehr als einen Kanals in eine erste Schicht der Vielzahl von Schichten und das Laden eines zweiten Blatts mit Daten aus einem zweiten Kanal des mehr als einen Kanals in eine zweite Schicht der Vielzahl von Schichten, umfasst.

4. Verfahren nach Anspruch 1, ferner umfassend:

Parsen einer Zeilengruppe (203), deren Rahmen (201) als ein Array strukturiert ist, das an einer Array-Speicherstelle des Rahmens (201) ein mehrdimensionales Merkmal aufweist;
Parsen der Zeilengruppe (203) in mehrere Blätter (204, 205), sodass für jede der unterschiedlichen Dimensionen des Merkmals ein anderes Blatt existiert.

5. Verfahren nach Anspruch 1, wobei das Register-Array (306, 406, 602) Register mit einer ersten Bitbreite enthält, die kleiner ist als eine zweite Bitbreite, bei der die Ausführungsbahnen (510, 601) Daten verarbeiten können, wobei das Verfahren ferner umfasst:

Empfangen einer Zeilengruppe (203), deren Datenwerte die zweite Bitbreite aufweisen;
Erzeugen eines ersten und eines zweiten Blattes (204, 205) aus der Zeilengruppe, wobei das erste Blatt Werte der ersten Größe aufweist und einen oberen Abschnitt der Datenwerte enthält, wobei das zweite Blatt Werte der ersten Größe aufweist und einen unteren Abschnitt der Datenwerte enthält;
Laden des ersten und des zweiten Blatts in das Schieberegister-Array (306, 406, 602);
gleichzeitiges Verarbeiten des ersten und des zweiten Blatts mit dem Array (305, 405) von Ausführungsbahnen (510, 601), um die Daten mit der zweiten Bitbreite zu verarbeiten.

6. Verfahren nach Anspruch 1, wobei das Verfahren ferner eines von Folgendem umfasst:

Bewirken einer Abtastratenerhöhung durch mehrmaliges Kopieren eines Datenwerts von einer Speicherstelle der Zeilengruppe (203) über eine entsprechende Speicherstelle auf dem Blatt (204, 205); und
Bewirken einer Abtastratenverringerung durch Erzeugen eines zweiten Ausgabeblatts aus dem Blatt (204, 205), wobei das zweite Ausgabeblatt eine Pixelwertdichte aufweist, die geringer als

die Pixelwertdichte des Blatts (204, 205) ist, wobei keine Daten verloren gehen, wenn der Inhalt der Zeilengruppe (203), die auf das Blatt (204, 205) und das zweite Ausgabeblatt abgebildet wird, verglichen wird.

7. Verfahren nach Anspruch 1, wobei das Verfahren ferner umfasst:

Erkennen, dass die Verarbeitung auf den Schablonen (202) mehr Bildfläche als das Blatt (204, 205) benötigt;
Parsen mindestens eines weiteren Blatts aus der Zeilengruppe (203), das an das Blatt angrenzt; und
Laden des mindestens einen weiteren Blatts in einen Speicherbereich, der mit den Ausführungsbahnen (510, 601) gekoppelt ist.

8. Verfahren nach Anspruch 7, wobei der Speicherbereich das zweidimensionale Verschiebe-Array (306, 406, 602) ist.

9. Verfahren nach Anspruch 1, wobei das Verfahren ferner umfasst: das Durchführen des Parsens durch Ausführen von Programmcode auf einem Prozessor und/oder Controller (1301).

10. Maschinenlesbares Speichermedium, welches Programmcode enthält, der, wenn er durch einen Prozessor und/oder Controller verarbeitet wird, bewirkt, dass ein Verfahren nach einem der Ansprüche 1 bis 9 durchgeführt wird.

11. Vorrichtung (1400), umfassend:
eine Blatterzeugerschaltung (103, 1300), die eine elektronische Schaltung umfasst, um:

von einem Zeilenpuffer (101_1 ... 101_M) eine Zeilengruppe (203) von Bilddaten zu empfangen, die mehrere Zeilen von Daten aus einem Rahmen (201) von Bilddaten einschließt, wobei die mehreren Zeilen zahlenmäßig ausreichend sind, um mehrere benachbarte überlappende Schablonen (202) zu umschließen;
die Zeilengruppe in ein Blatt kleinerer Größe (204, 205) zu parsen;
das Blatt (204, 205) in eine Datenberechnungseinheit (301, 401) eines zugeordneten Schablonenprozessors (102_1) zu laden, wobei die Datenberechnungseinheit eine zweidimensionale Verschiebe-Arraystruktur (306, 406, 602) aufweist, die mit einem Array (305, 405) von Ausführungsbahnen (510, 601) gekoppelt ist, wobei das Ausführungsbahn-Array (305, 405) und die Verschiebe-Arraystruktur (306, 406, 602) in ihrer Position relativ zueinander festgelegt sind, wobei der Schablonenprozessor die mehreren benachbarten überlappenden Schablonen (202) auf dem Blatt (204, 205) verarbeitet, wobei jede Ausführungsbahn (510, 601) dafür konfiguriert ist, ihre eigene Schablone zu verarbeiten, um einen Ausgabewert für diese Schablone zu erzeugen, und wobei die Ausgabewerte der Ausführungsbahnen jeweilige Werte in einem Ausgabeblatt von Bilddaten bilden; und
Empfangen des Ausgabeblatts aus Bilddaten von dem zugeordneten Schablonenprozessor (102_1).

12. Vorrichtung (1400) nach Anspruch 11, wobei die benachbarten überlappenden Schablonen (202) sich sowohl in horizontaler als auch in vertikaler Richtung erstrecken.

13. Vorrichtung (1400) nach Anspruch 11, wobei die elektronische Schaltung ferner dazu dient:

Zeilengruppen (203) aus mehr als einem Kanal zu parsen;
mehr als ein Blatt (204, 205) aus dem mehr als einen Kanal zu parsen, um mindestens zwei Blätter aus unterschiedlichen Kanälen zu erzeugen;
die mindestens zwei Blätter in die Datenberechnungseinheit (301, 401) zu laden.

14. Vorrichtung (1400) nach Anspruch 11, wobei die elektronische Schaltung dazu dient:

eine Zeilengruppe (203) zu empfangen, deren Datenwerte die zweite Bitbreite aufweisen;
ein erstes und ein zweites Blatt aus der Zeilengruppe (203) zu erzeugen, wobei das erste Blatt Werte der ersten Größe aufweist und einen oberen Abschnitt der Datenwerte enthält, wobei das zweite Blatt Werte der ersten Größe aufweist und einen unteren Abschnitt der Datenwerte enthält;
das erste und das zweite Blatt in die Datenberechnungseinheit (301, 401) zu laden.

15. Vorrichtung (1400) nach Anspruch 11, wobei die elektronische Schaltung zu einem von Folgendem dient:

eine Abtastratenerhöhung durch mehrmaliges Kopieren eines Datenwerts von einer Speicherstelle der Zeilengruppe (203) über eine entsprechende Speicherstelle auf dem Blatt (204, 205) zu bewirken; und
eine Abtastratenverringerung durch Erzeugen eines zweiten Ausgabeblatts aus dem Blatt (204, 205) zu bewirken, wobei das zweite Ausgabeblatt eine Pixelwertdichte aufweist, die geringer als die Pixelwertdichte des Blatts (204,

205) ist, wobei keine Daten verloren gehen, wenn der Inhalt der Zeilengruppe (203), die auf das Blatt (204, 205) und das zweite Ausgabeblatt abgebildet wird, verglichen wird.

Revendications

1. Procédé, comprenant :

la réception, par un générateur de feuille (103_1) et à partir d'un tampon de ligne (101_1 ... 101_M), d'un groupe de ligne (203) de données d'images incluant plusieurs rangées de données provenant d'un cadre (201) de données d'image, lesdites plusieurs rangées étant en nombre suffisant pour comprendre plusieurs pochoirs superposés voisins (202) ;
l'analyse, par le générateur de feuille (103_1), dudit groupe de ligne (203) en une pluralité de feuilles de taille inférieure (204, 205) ; et, pour chaque feuille de la pluralité de feuilles :

le chargement, par le générateur de feuilles (103_1), de ladite feuille (204, 205) dans un processeur de pochoir (102-1) associé au générateur de feuille et comprenant une structure de réseau de déplacement en deux dimensions (306, 406, 602) couplé à un réseau (305, 405) de voies d'exécution (510, 601), le réseau de voie d'exécution (305, 405) et la structure de réseau de déplacement (306, 406, 602) étant fixés en position l'un par rapport à l'autre ;
l'exécution d'un code programme sur ledit réseau (305, 405) de voies d'exécution (510, 601) pour traiter lesdits plusieurs pochoirs superposés voisins (202) sur ladite feuille (204, 205), dans lequel chaque voie d'exécution (510, 601) traite son propre pochoir pour générer une valeur de sortie pour ce pochoir, et dans lequel les valeurs de sortie des voies d'exécution constituent des valeurs de pixels respectives dans une feuille de sortie de données d'image ; et
la réception, par le générateur de feuille (103_1), de la feuille de sortie de données d'image à partir du processeur de pochoir associé (102_1).

2. Procédé selon la revendication 1, dans lequel lesdits pochoirs superposés voisins (202) s'étendent dans les directions horizontale et verticale.

3. Procédé selon la revendication 1, comprenant en outre :

l'analyse de groupes de lignes (203) à partir de plus d'un canal ;
l'analyse de plus d'une feuille (204, 205) à partir de plus d'un canal pour créer au moins deux feuilles de canaux différents ;
le chargement desdites au moins deux feuilles dans ladite structure de réseau de déplacement en deux dimensions (306, 406, 602) ;
l'exécution de code programme sur ledit réseau (305, 405) de voies d'exécution (510, 601) pour traiter lesdits plusieurs pochoirs superposés voisins (202) simultanément sur lesdites au moins deux feuilles ;
dans lequel la structure de réseau de déplacement en deux dimensions (306, 406, 602) comprend une pluralité de couches, et dans lequel le chargement desdites au moins deux couches dans ladite structure de réseau de déplacement en deux dimensions (306, 406, 602) comprend le chargement d'une première feuille présentant des données d'un premier canal du plus d'un canal dans une première couche de la pluralité de couches et le chargement d'une seconde feuille présentant des données provenant d'un second canal du plus d'un canal dans une seconde couche de la pluralité de couches.

4. Procédé selon la revendication 1, comprenant en outre :

l'analyse d'un groupe de ligne (203) dont le cadre (201) est structuré comme un réseau présentant une caractéristique pluridimensionnelle à un emplacement de réseau du cadre (201) ;
l'analyse du groupe de ligne (203) dans plusieurs feuilles (204, 205) de sorte qu'il existe une feuille différente pour chacune des différentes dimensions de la caractéristique.

5. Procédé selon la revendication 1, dans lequel le réseau de registre (306, 406, 602) contient des registres d'une première largeur binaire qui est inférieure à une seconde largeur binaire à laquelle les voies d'exécution (510, 601) sont capables de traiter des données, le procédé comprenant en outre :

la réception d'un groupe de ligne (203) dont les valeurs de données présentent la seconde largeur binaire ;
la création de première et seconde feuilles (204, 205) à partir du groupe de ligne, la première feuille présentant des valeurs de la première taille et contenant une partie supérieure des valeurs de données, la seconde feuille présentant des valeurs de la première taille contenant une partie inférieure des valeurs de données ;
le chargement des première et seconde feuilles dans le réseau de registre de changement (306, 406, 602) ;

le traitement simultané des première et seconde feuilles avec le réseau (305, 405) des voies d'exécution (510, 601) pour traiter les données à la seconde largeur binaire.

6. Procédé selon la revendication 1, dans lequel le procédé comprend en outre une parmi :

la réalisation d'un échantillonnage ascendant en reproduisant une valeur de donnée d'un emplacement du groupe de ligne (203) plusieurs fois à un emplacement correspondant sur la feuille (204, 205) ; et
la réalisation d'un échantillonnage descendant en créant une seconde feuille de sortie à partir de la feuille (204, 205), la seconde feuille de sortie présentant une densité de valeurs de pixel qui est inférieure à la densité de valeurs de pixels de la feuille (204, 205), dans lequel aucune donnée n'est perdue si l'on compare le contenu du groupe de ligne (203) qui est cartographié sur la feuille (204, 205) et la seconde feuille de sortie.

7. Procédé selon la revendication 1, dans lequel le procédé comprend en outre :

la reconnaissance que le traitement sur les pochoirs (202) nécessite davantage de superficie d'image que la feuille (204, 205) ;
l'analyse d'au moins une feuille supplémentaire du groupe de ligne (203) qui avoisine la feuille ; et
le chargement de l'au moins une feuille supplémentaire dans un espace de stockage raccordé aux voies d'exécution (510, 601).

8. Procédé selon la revendication 7, dans lequel l'espace de stockage est le réseau de déplacement en deux dimensions (306, 406, 602).

9. Procédé selon la revendication 1, dans lequel le procédé comprend en outre la réalisation de l'analyse en exécutant un code programme sur un processeur et/ou un système de commande (1301).

10. Support de stockage lisible sur une machine contenant un code programme qui, lorsqu'il est traité par un processeur et/ou un système de commande provoque la réalisation d'un procédé selon l'une quelconque des revendications 1 à 9.

11. Appareil (1400), comprenant :
un circuit générateur de feuille (103, 1300) comprenant un circuit électronique pour :

recevoir, d'un tampon de ligne (101_1... 101_M), un groupe de lignes (203) de données d'images incluant plusieurs rangées de données provenant d'un cadre (201) de données d'image, lesdites plusieurs rangées étant suffisantes en nombre pour comprendre plusieurs pochoirs superposés avoisinants (202) ;
analyser ledit groupe de ligne en une feuille de taille inférieure (204, 205) ;
charger ladite feuille (204, 205) dans une unité de calcul de données (301, 401) d'un processeur de pochoir associé (102_1), l'unité de calcul de données présentant une structure de réseau de déplacement en deux dimensions (306, 406, 602) couplée à un réseau (305, 405) de voies d'exécution (510, 601), le réseau de voies d'exécution (305, 405) et la structure de réseau de déplacement (306, 406, 602) étant fixés en position l'un par rapport à l'autre, le processeur de pochoirs traitant lesdits plusieurs pochoirs superposés avoisinants (202) sur ladite feuille (204, 205), dans lequel chaque voie d'exécution (510, 601) est configurée pour traiter son propre pochoir pour générer une valeur de sortie pour ce pochoir, et dans lequel les valeurs de sortie des voies d'exécution constituent des valeurs respectives dans une feuille de sortie de données d'image ; et
recevoir la feuille de sortie de données d'image du processeur de pochoirs associé (102_1).

12. Appareil (1400) selon la revendication 11, dans lequel lesdits pochoirs superposés avoisinants (202) s'étendent à la fois dans les directions horizontale et verticale.

13. Appareil (1400) selon la revendication 11, dans lequel ledit circuit électronique vise en outre à :

analyser des groupes de ligne (203) à partir de plus d'un canal ;
analyser plus d'une feuille (204, 205) à partir de plus d'un canal pour créer au moins deux feuilles de canaux différents ;
charger lesdites au moins deux feuilles dans ladite unité de calcul de données (301, 401).

14. Appareil (1400) selon la revendication 11, dans lequel le circuit électronique vise à :

recevoir un groupe de lignes (203) dont les valeurs de données présentent la seconde largeur binaire ;
créer des première et seconde feuilles à partir du groupe de ligne (203), la première feuille présentant des valeurs de la première taille et contenant une partie supérieure des valeurs de données, la seconde feuille présentant des valeurs de la première taille contenant une partie inférieure des valeurs de données ;

charger les première et seconde feuilles dans l'unité de calcul de données (301, 401).

15. Appareil (1400) selon la revendication 11, dans lequel le circuit électronique est destiné à effectuer une opération parmi de :

la réalisation d'un échantillonnage ascendant en reproduisant une valeur de données à partir d'un emplacement du groupe de ligne (203) plusieurs fois à un emplacement correspondant sur la feuille (204, 205) ; et
la réalisation d'un échantillonnage descendant en créant une seconde feuille de sortie à partir de la feuille (204, 205), la seconde feuille de sortie présentant une densité de valeurs de pixels inférieure à la densité de valeurs de pixel de la feuille (204, 205), dans lequel, aucune donnée n'est perdue si l'on compare le contenu du groupe de ligne (203) qui est cartographié sur la feuille (204, 205) et la seconde feuille de sortie.

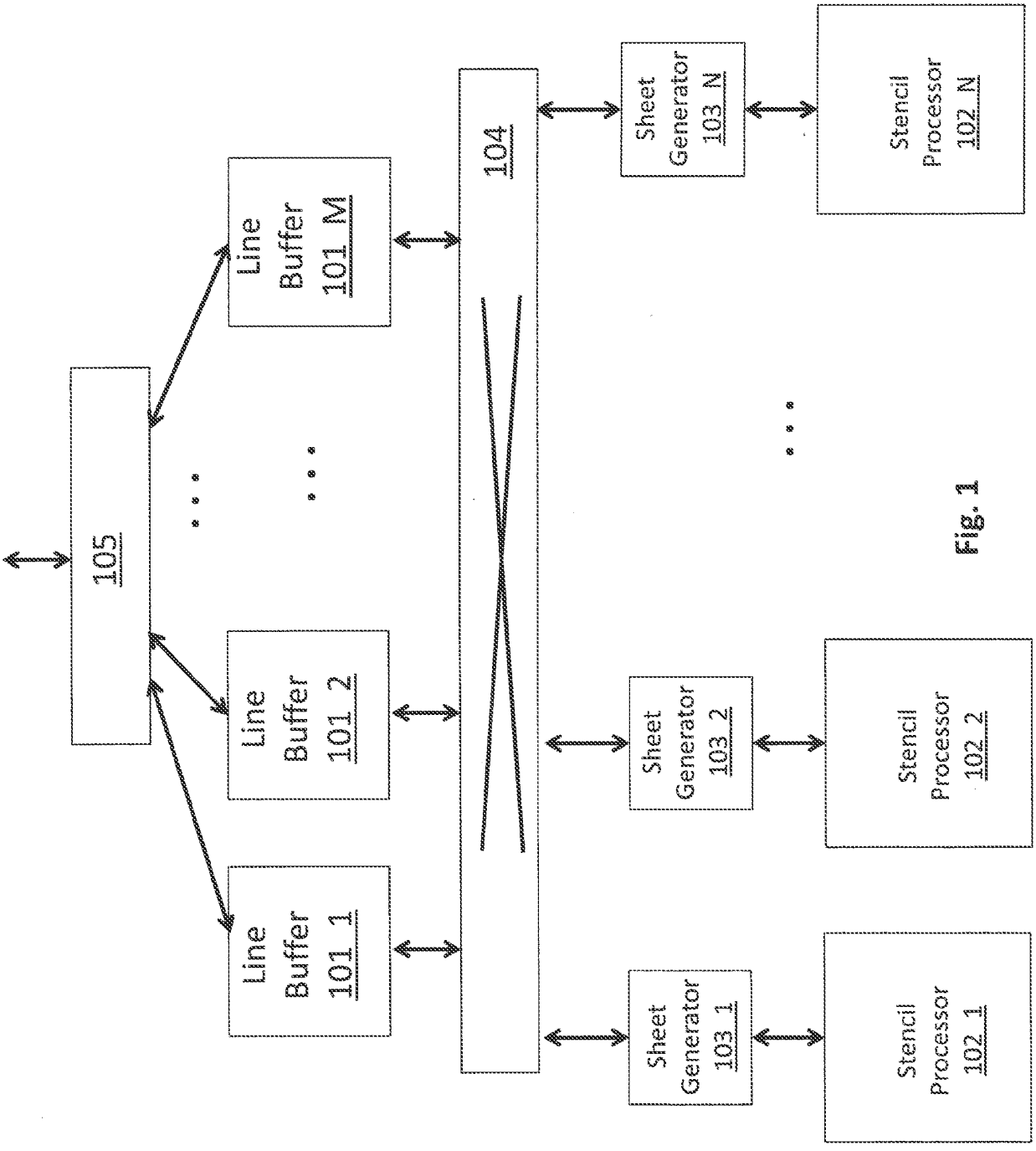

105
Line Buffer 101_1
Line Buffer 101_2
Line Buffer 101_M
104
Sheet Generator 103_1
Sheet Generator 103_2
Sheet Generator 103_N
Stencil Processor 102_1
Stencil Processor 102_2
Stencil Processor 102_N

Fig. 1

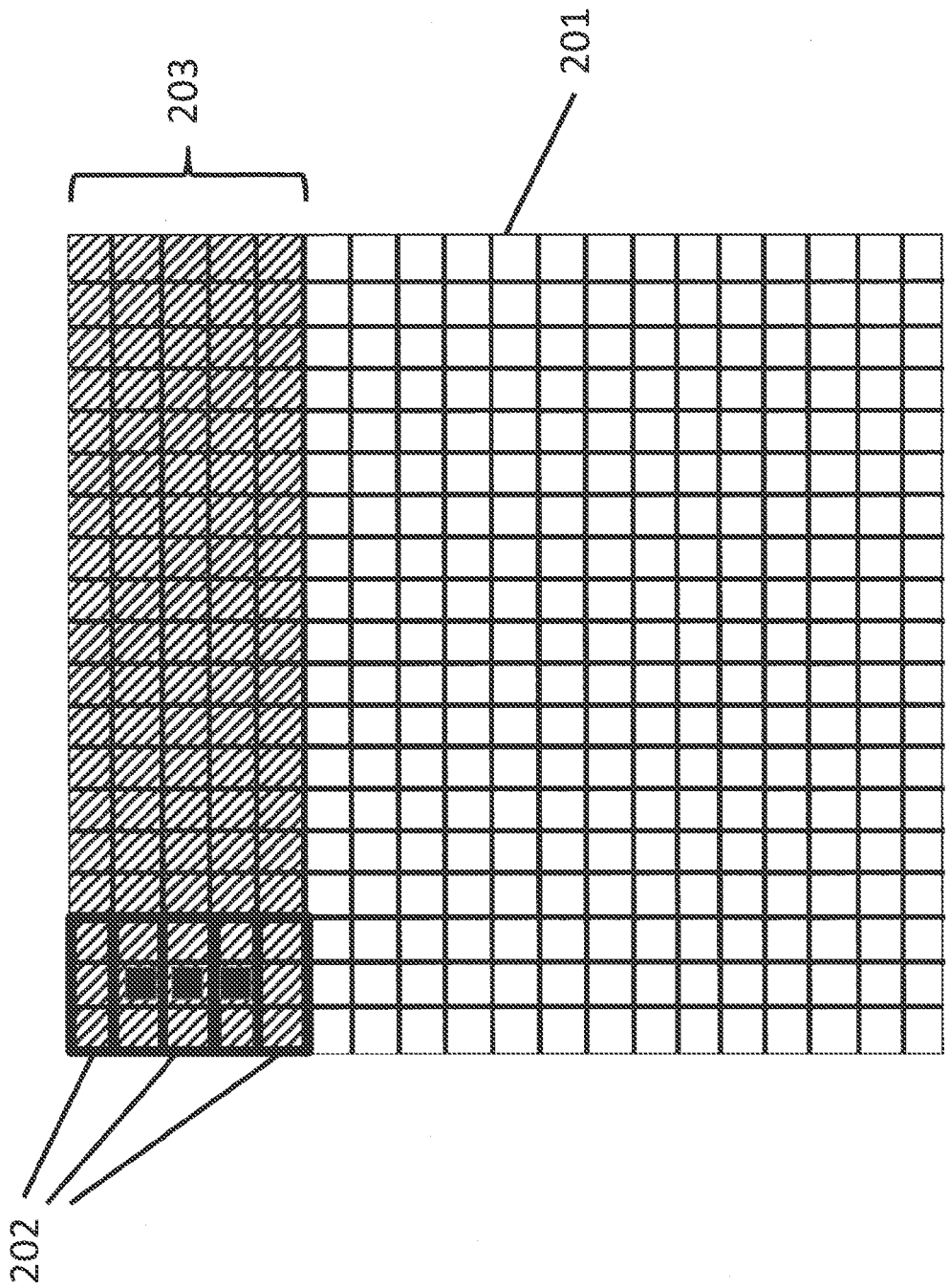


Fig. 2a

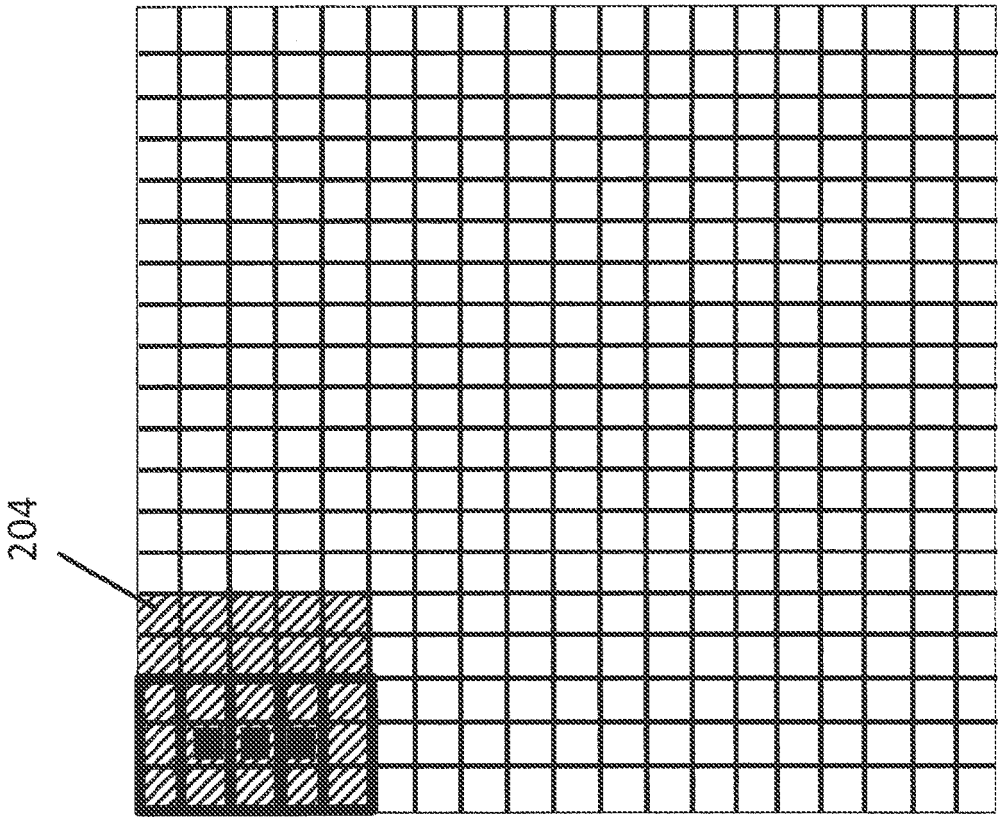


Fig. 2b

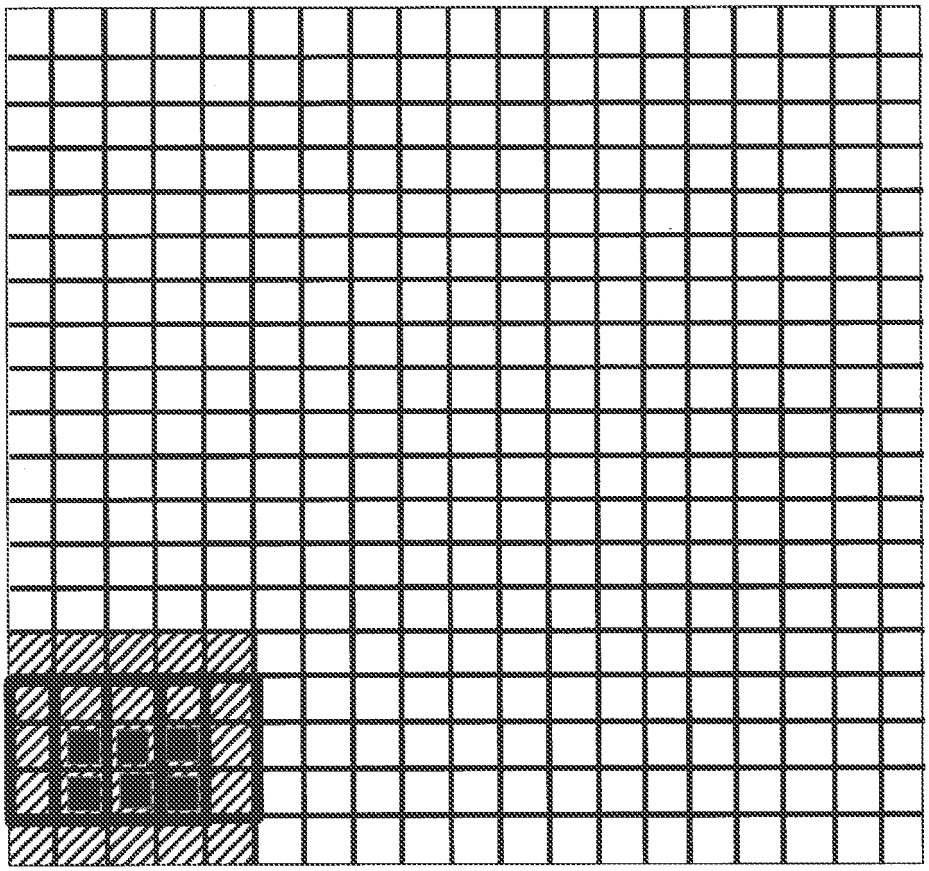


Fig. 2C

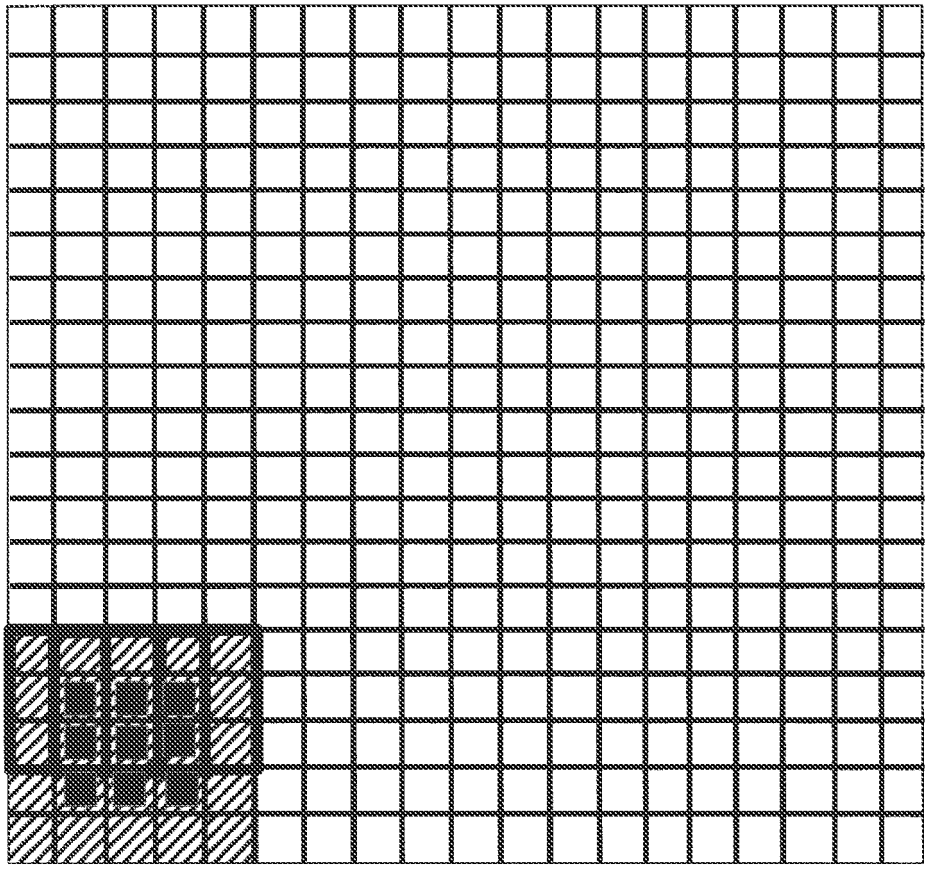


Fig. 2d

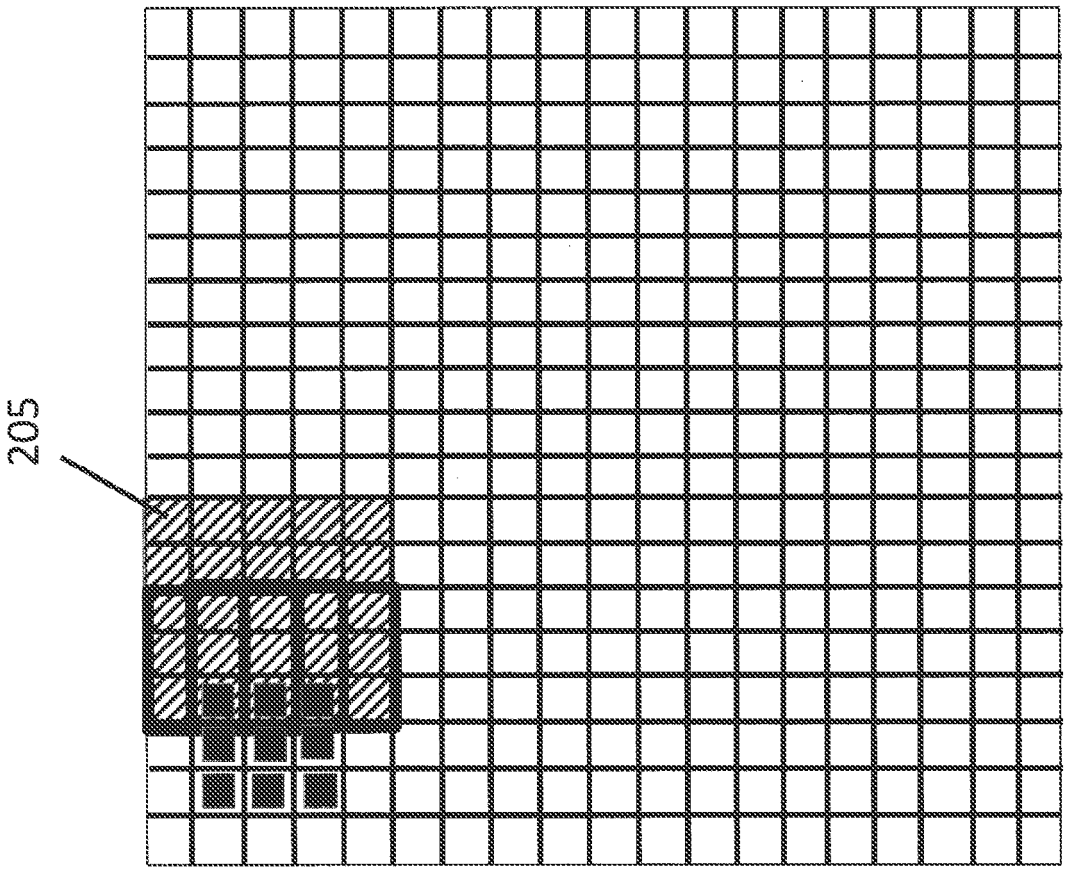


Fig. 2e

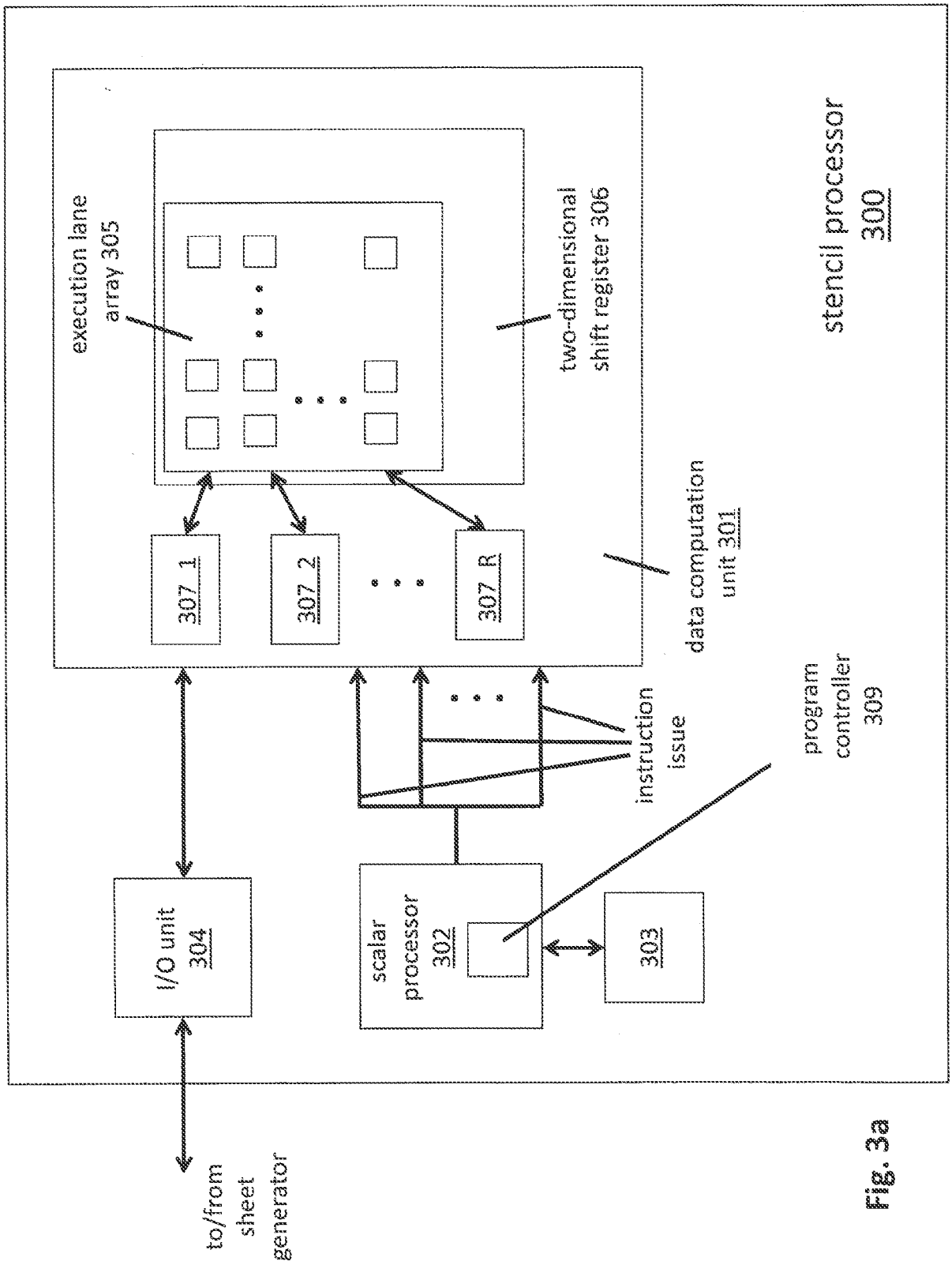

stencil processor
300
execution lane
array 305
two-dimensional
shift register 306
307_1
307_2
307_R
data computation
unit 301
program
controller
309
instruction
issue
I/O unit
304
scalar
processor
302
303
to/from
sheet
generator

Fig. 3a

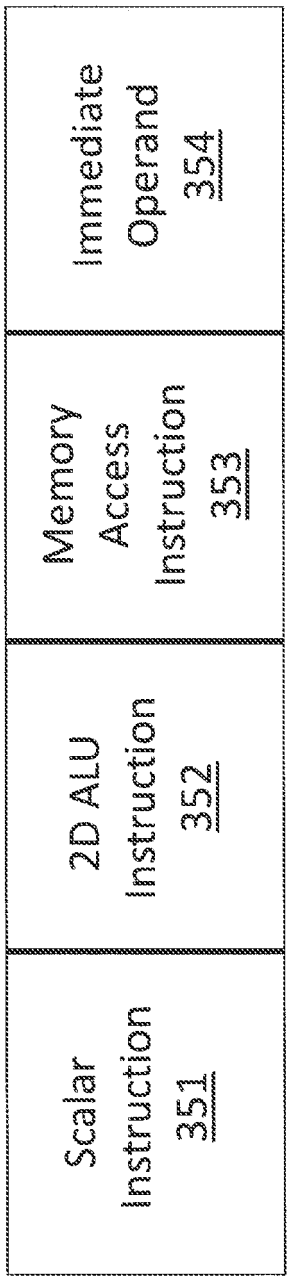

Scalar Instruction 351
2D ALU Instruction 352
Memory Access Instruction 353
Immediate Operand 354

Fig. 3b

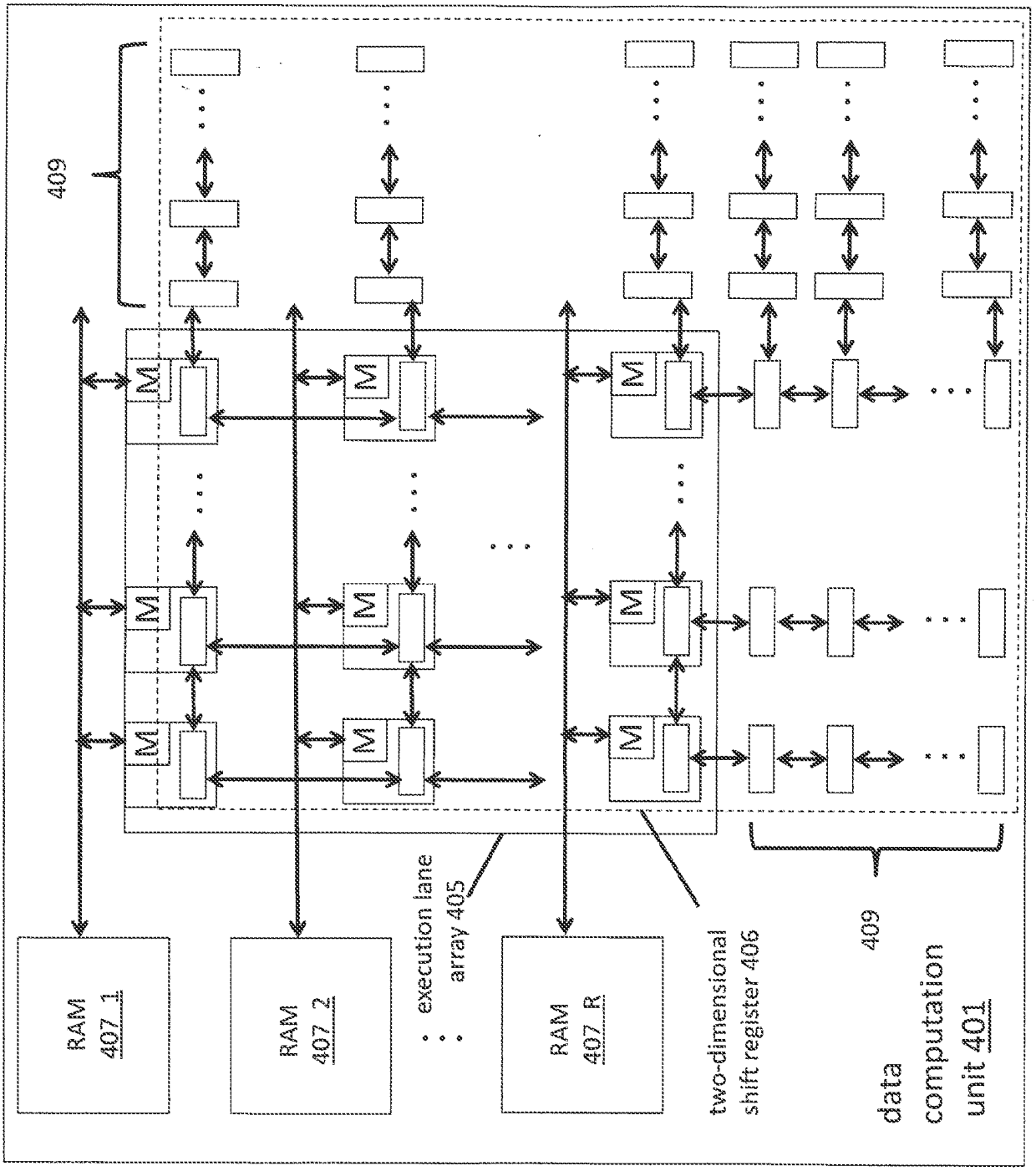

409
M
RAM
407_1
RAM
407_2
RAM
407_R
execution lane
array 405
two-dimensional
shift register 406
409
data
computation
unit 401

Fig. 4

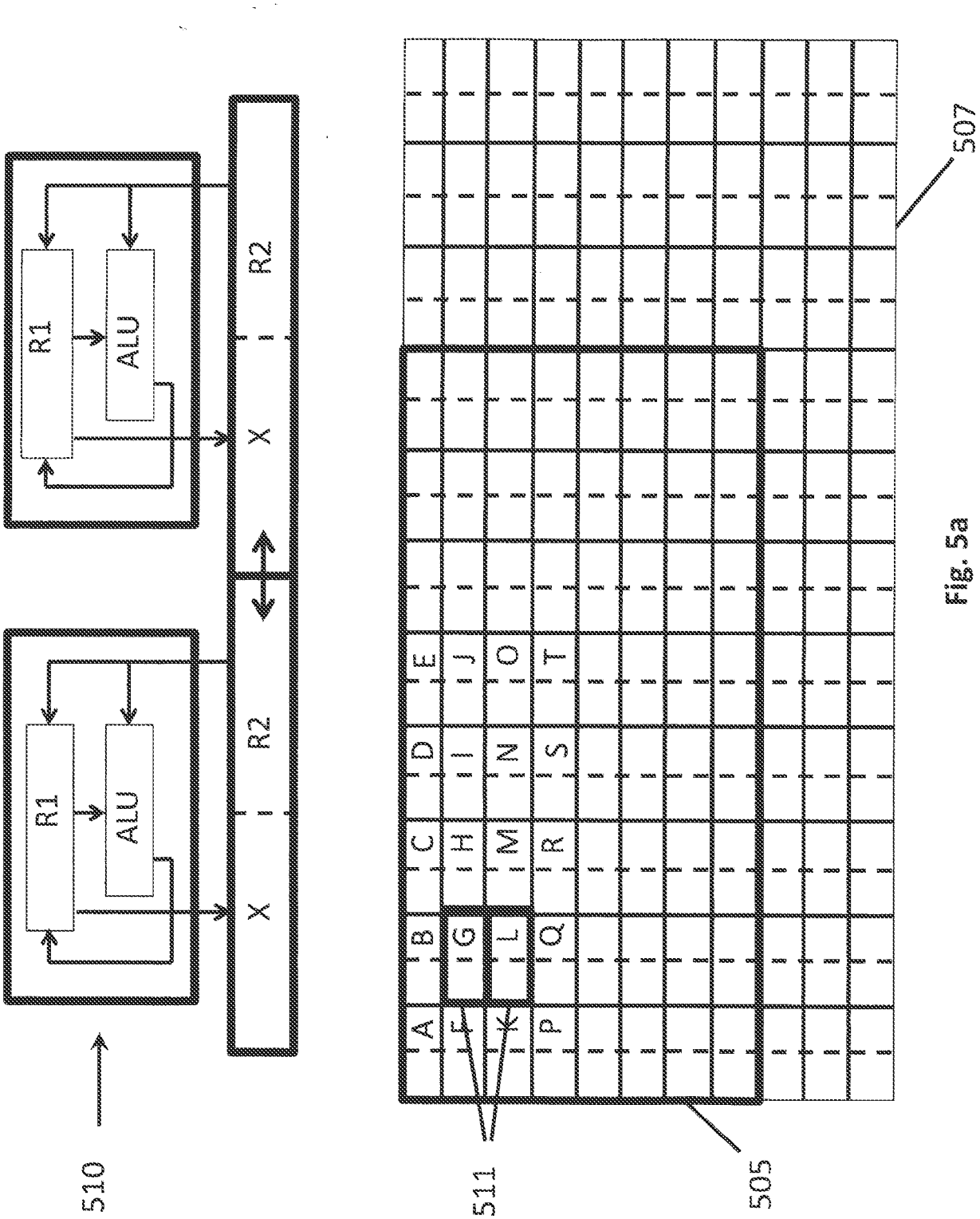


Fig. 5a

```
// thread X1: process stencil 1//
shift down 1
shift right 1
load // put A in R1
```

```
// thread X2: process stencil 2//
shift down 1
shift right 1
load // put F in R1
```

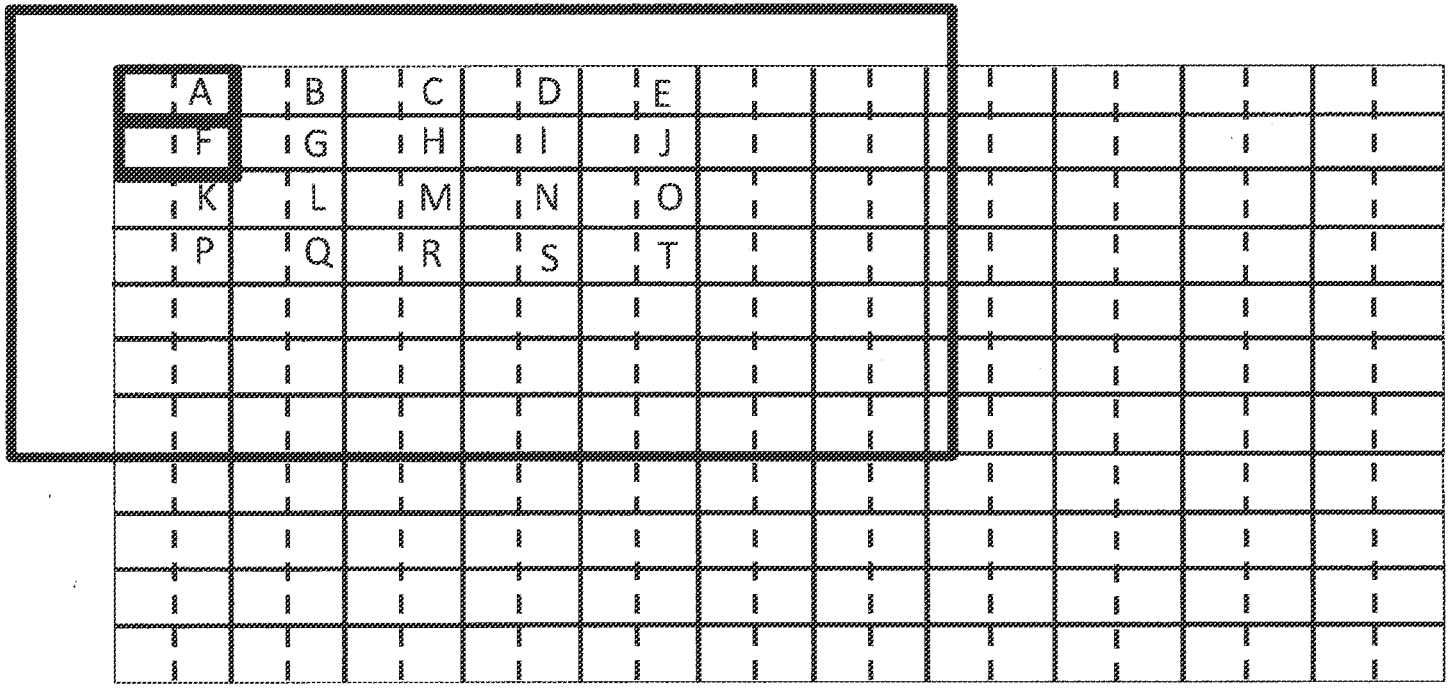


Fig. 5b

```
// thread X1: process stencil 1//
shift down 1
shift right 1
load // put A in R1
shift left 1
R1<= ADD R1, R2 // add A, B
```

```
// thread X2: process stencil 2//
shift down 1
shift right 1
load // put F in R1
shift left 1
R1<= ADD R1, R2 // add F, G
```

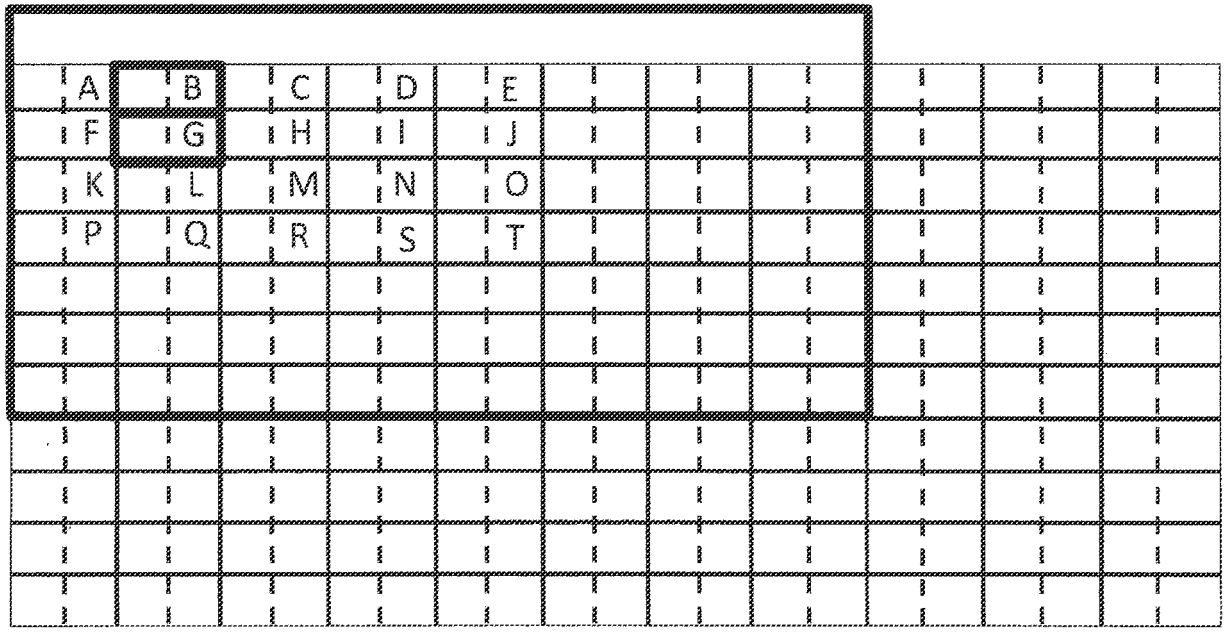


Fig. 5c

EP 3 286 725 B1

```
// thread X1: process stencil 1//
. . .
R1<= ADD R1, R2 // add A, B
shift left 1
R1<= ADD R1, R2 // add C
```

```
// thread X2: process stencil 2//
. . .
R1<= ADD R1, R2 // add F, G
shift left 1
R1<= ADD R1, R2 // add H
```

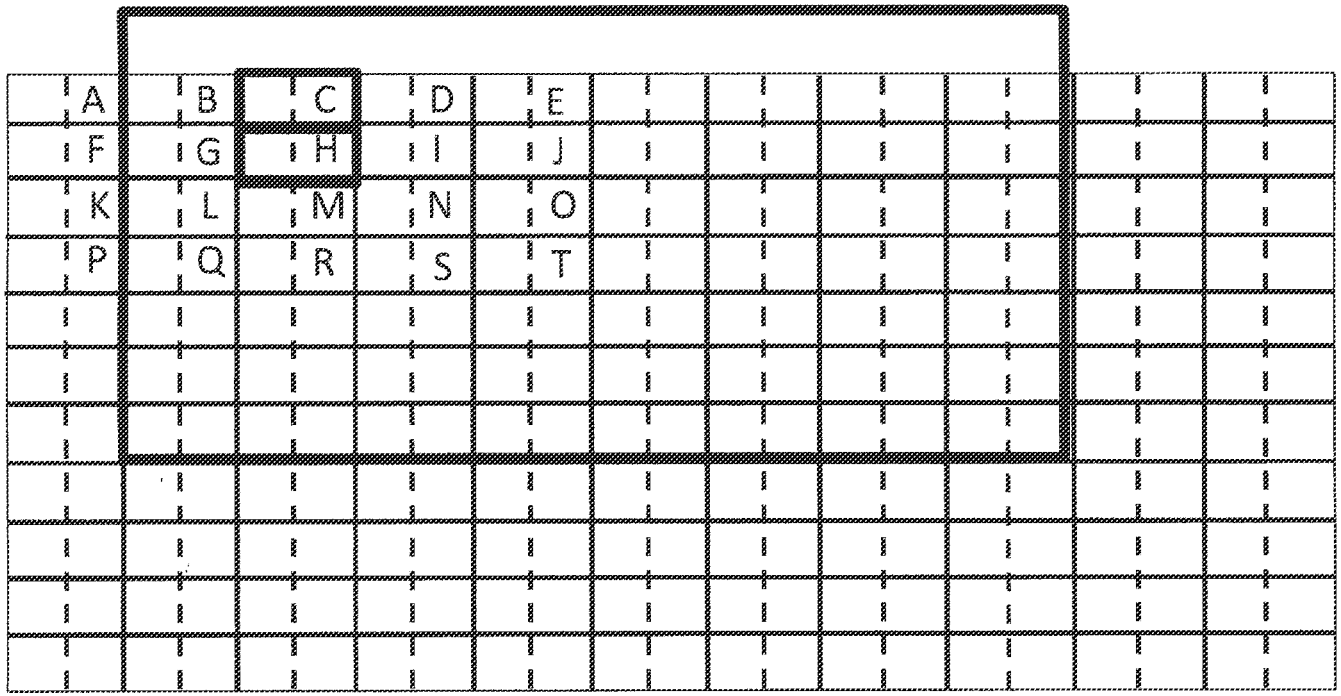


Fig. 5d

EP 3 286 725 B1

```
// thread X1: process stencil 1//
. . .
R1<= ADD R1, R2 // add A, B
shift left 1
R1<= ADD R1, R2 // add C
shift up 1
R1<= ADD R1, R2 // add H
```

```
// thread X2: process stencil 2//
. . .
R1<= ADD R1, R2 // add F, G
shift left 1
R1<= ADD R1, R2 // add H
shift up 1
R1<= ADD R1, R2 // add M
```

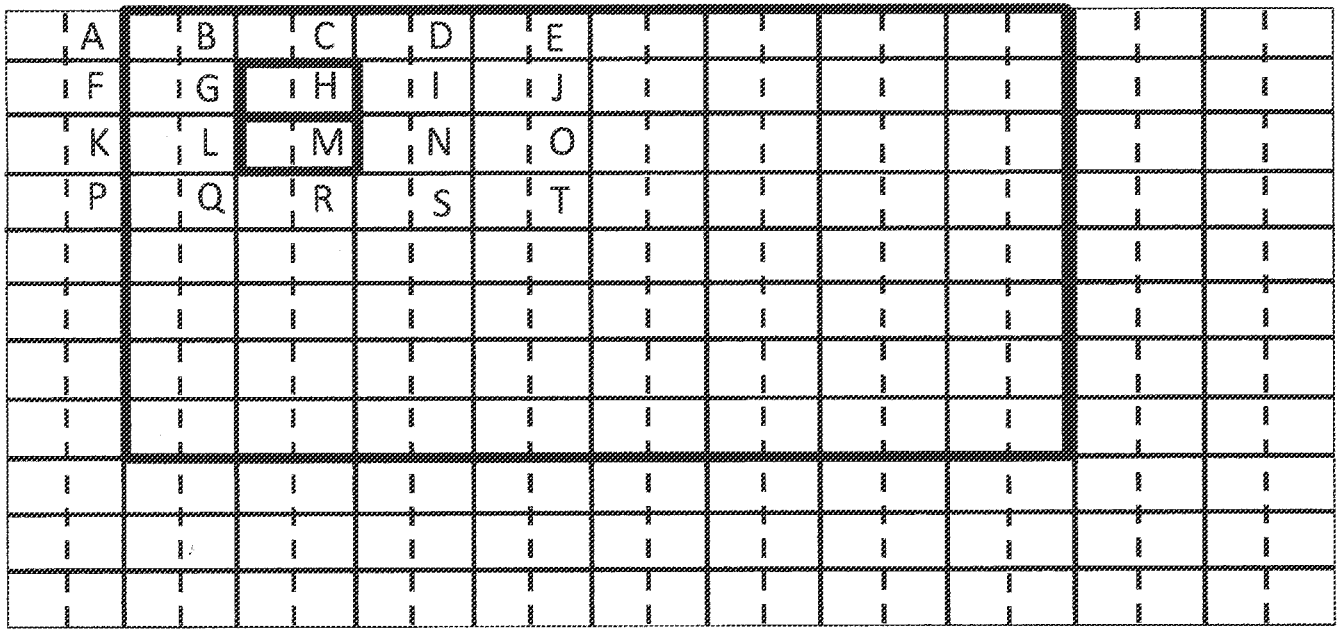


Fig. 5e

EP 3 286 725 B1

```
// thread X1: process stencil 1//
. . .
shift right 1
R1<= ADD R1, R2 // ADD G
```

```
// thread X2: process stencil 2//
. . .
shift right 1
R1<= ADD R1, R2 // ADD L
```

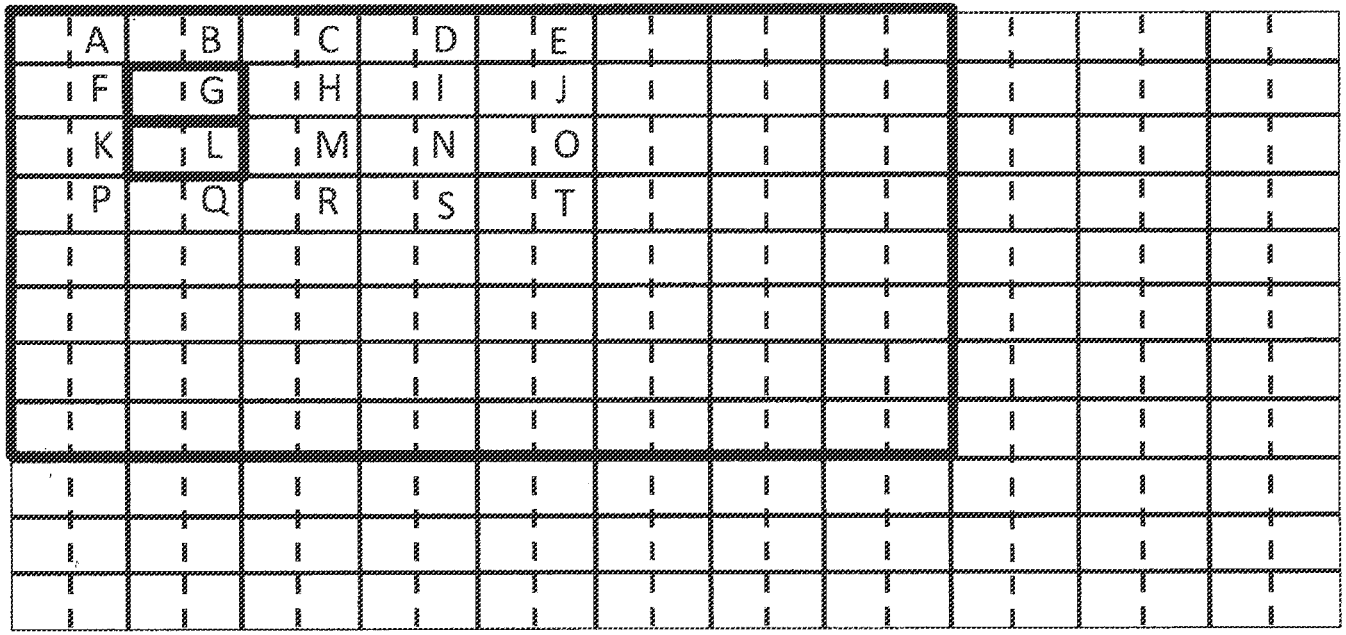


Fig. 5f

EP 3 286 725 B1

```
// thread X1: process stencil 1//
. . .
shift right 1
R1<= ADD R1, R2 // ADD F
```

```
// thread X2: process stencil 2//
. . .
shift right 1
R1<= ADD R1, R2 // ADD K
```

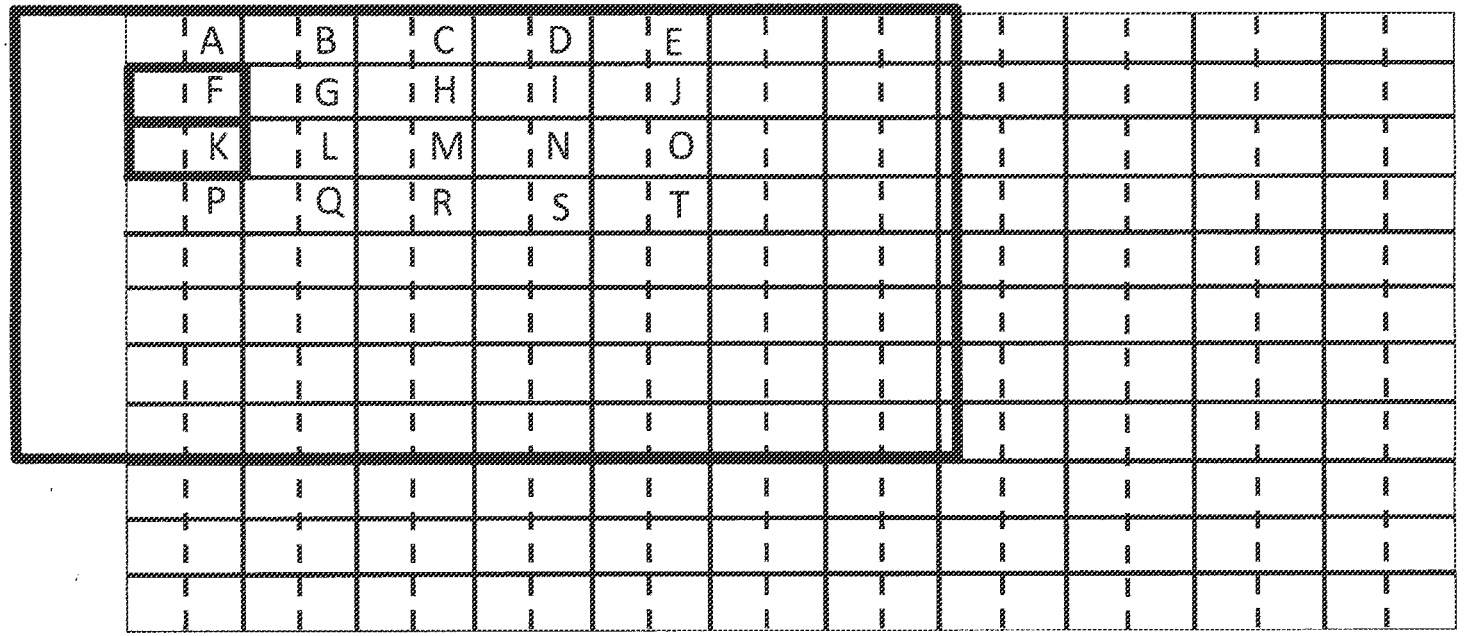


Fig. 5g

EP 3 286 725 B1

```
// thread X1: process stencil 1//
. . .
shift up 1
R1<= ADD R1, R2 // ADD K
```

```
// thread X2: process stencil 2//
. . .
shift up 1
R1<= ADD R1, R2 // ADD P
```

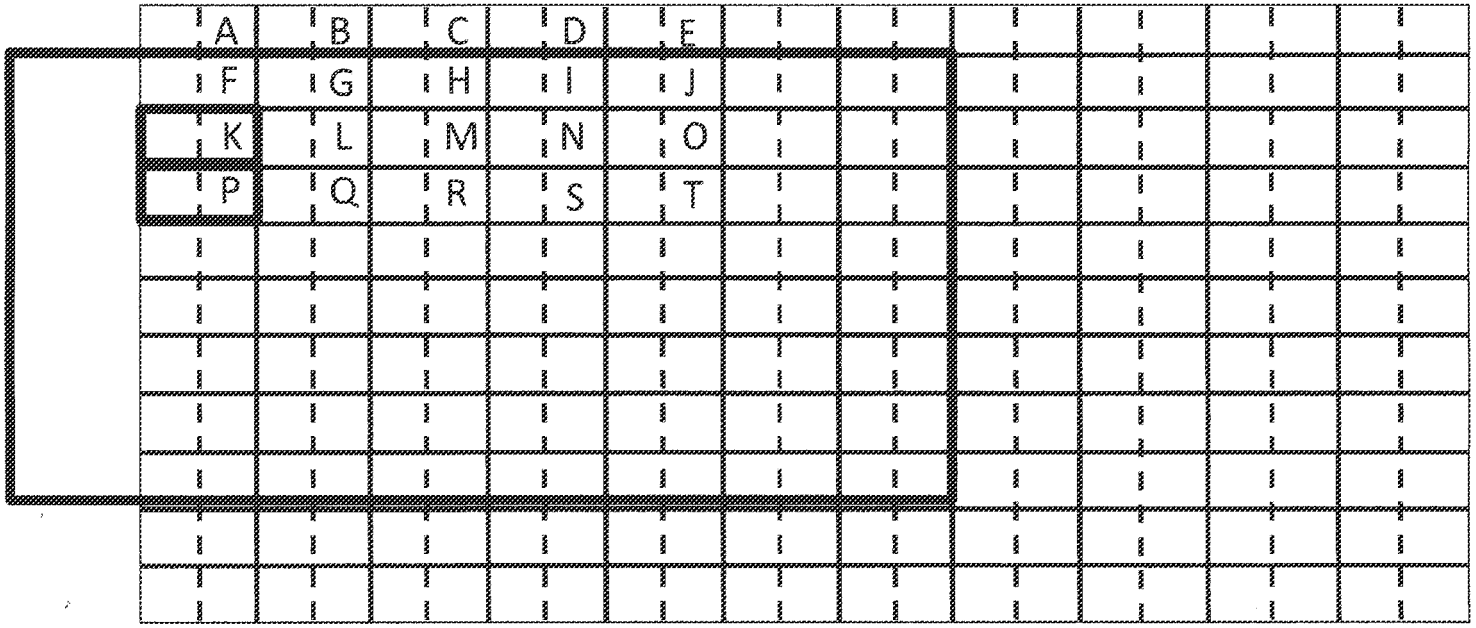


Fig. 5h

EP 3 286 725 B1

```
// thread X1: process stencil 1//
. . .
shift left 1
R1<= ADD R1, R2 // ADD L
```

```
// thread X2: process stencil 2//
. . .
shift left 1
R1<= ADD R1, R2 // ADD Q
```

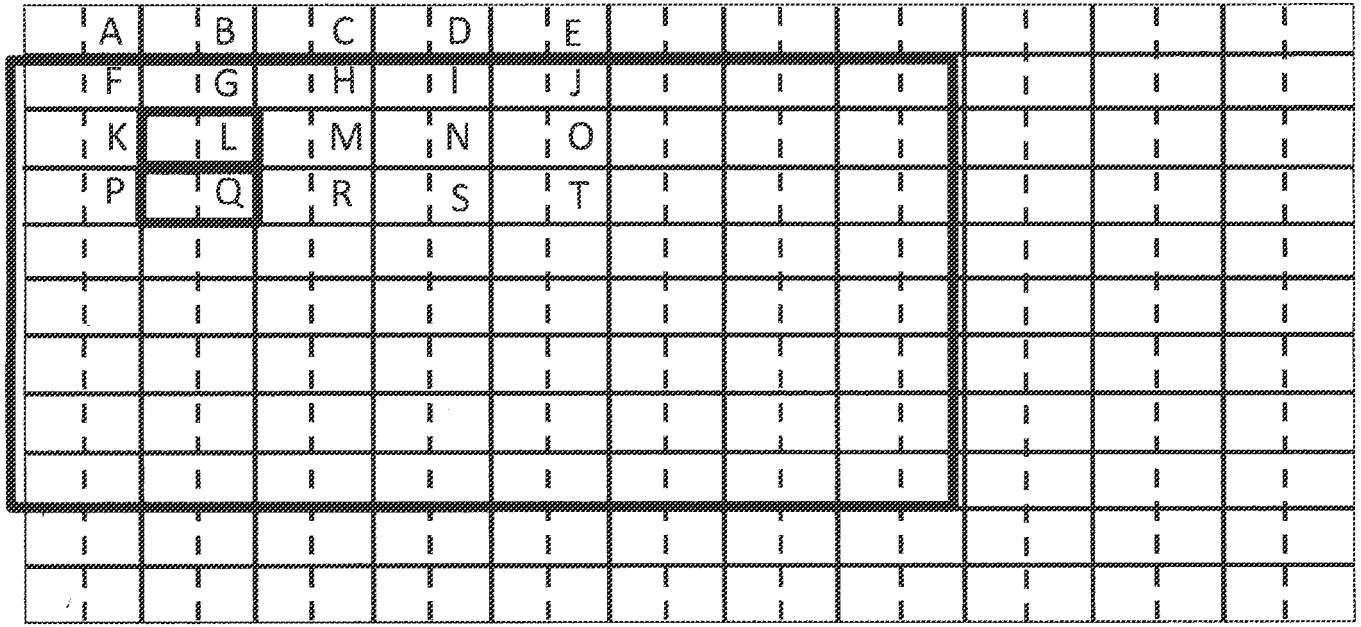


Fig. 5i

EP 3 286 725 B1

```
// thread X1: process stencil 1//
. . .
shift left 1
R1<= ADD R1, R2 // ADD M
R1<= DIV R1, 9
```

```
// thread X2: process stencil 2//
. . .
shift left 1
R1<= ADD R1, R2 // ADD R
R1<= DIV R1, 9
```

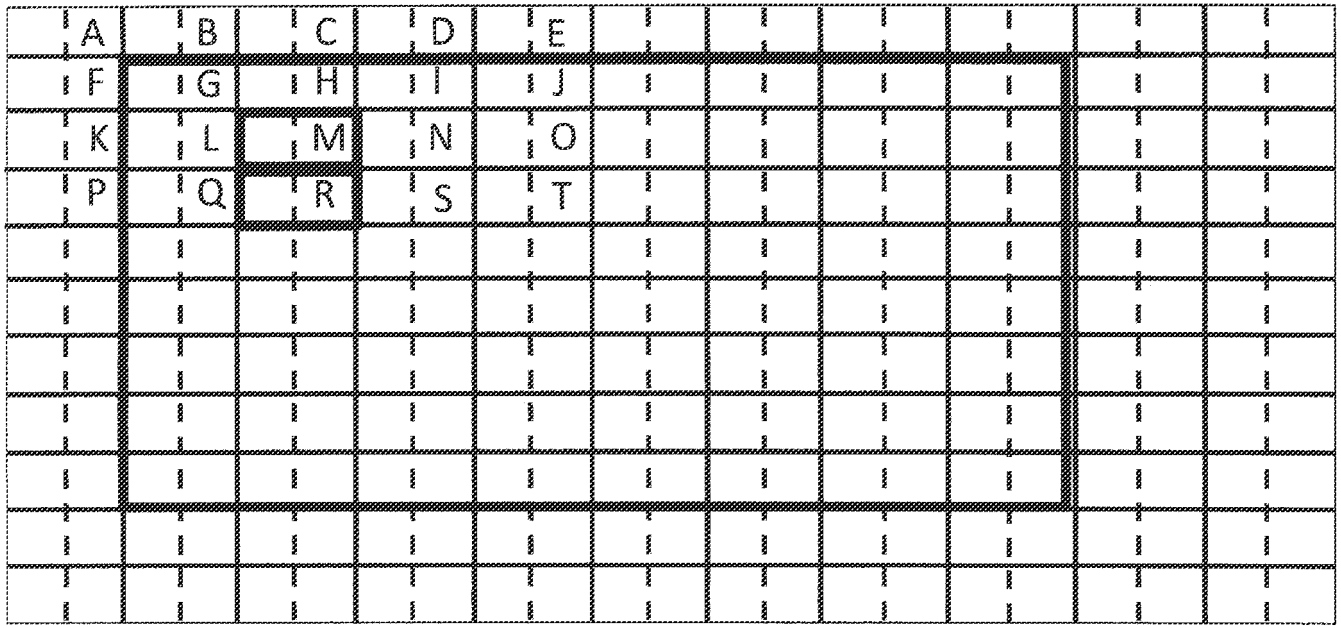


Fig. 5j

```
// thread X1: process stencil 1//
. . .
shift right 1
shift down 1
X<= store R1
```

```
// thread X2: process stencil 2//
. . .
shift right 1
shift down 1
X<=store R1
```

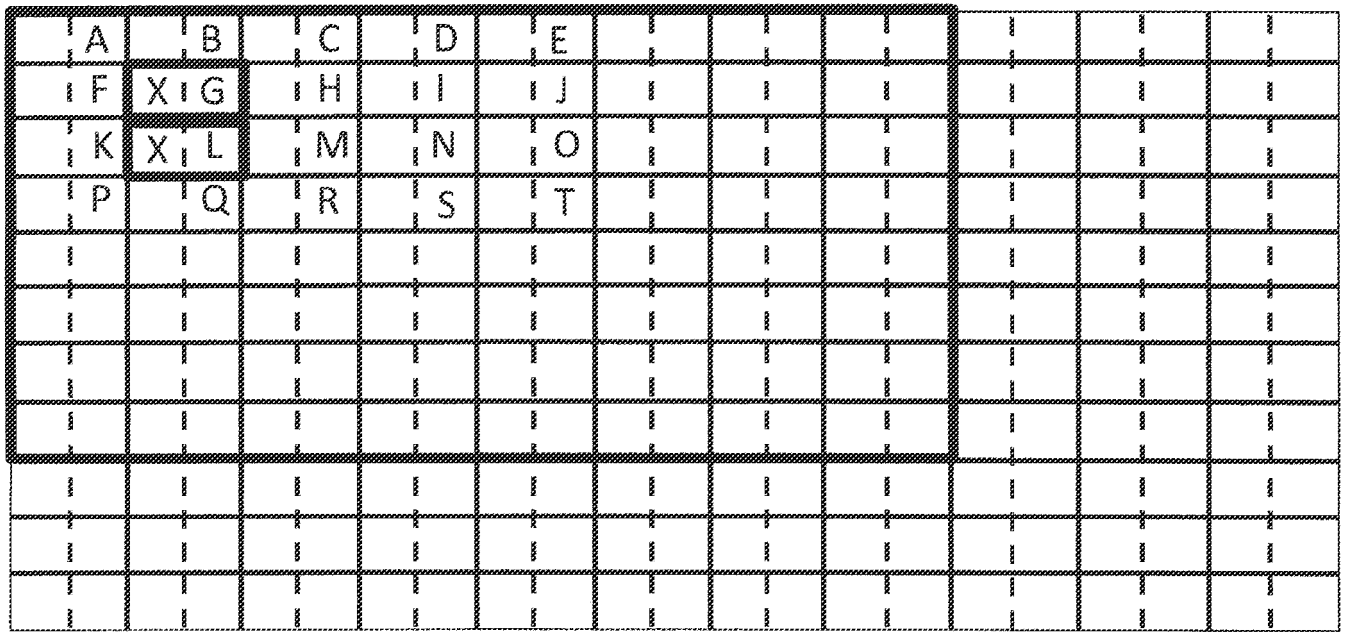


Fig. 5k

EP 3 286 725 B1

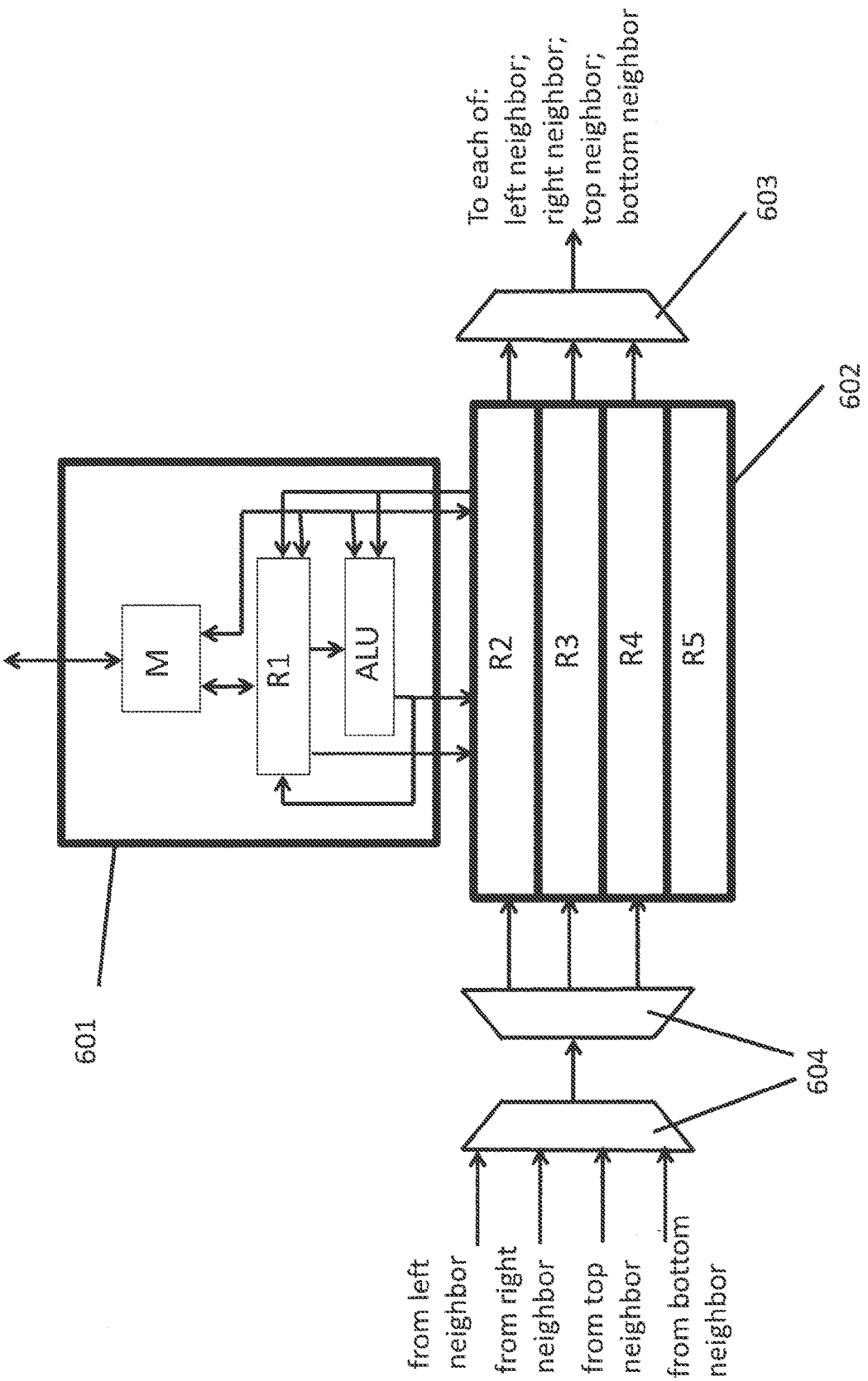

To each of:
left neighbor;
right neighbor;
top neighbor;
bottom neighbor
603
602
M
R1
ALU
R2
R3
R4
R5
601
604
from left neighbor
from right neighbor
from top neighbor
from bottom neighbor

Fig. 6

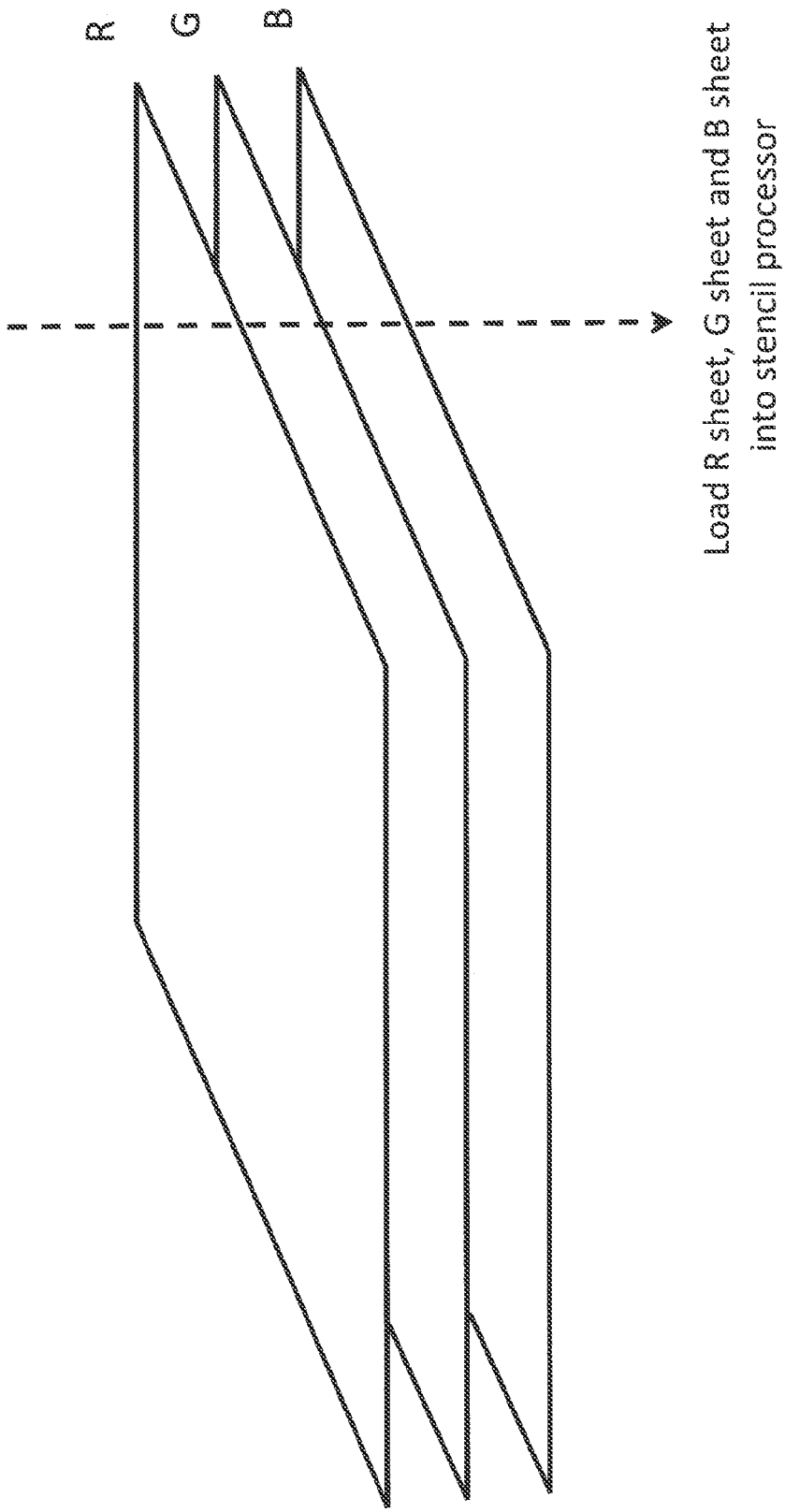
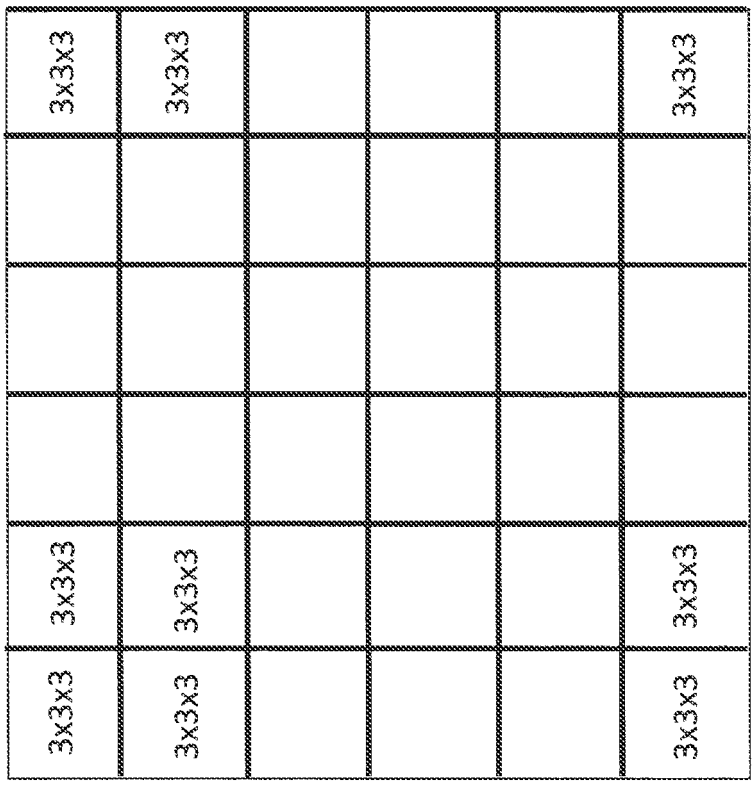


Fig. 7


3x3x3
3x3x3
3x3x3
3x3x3
3x3x3
3x3x3

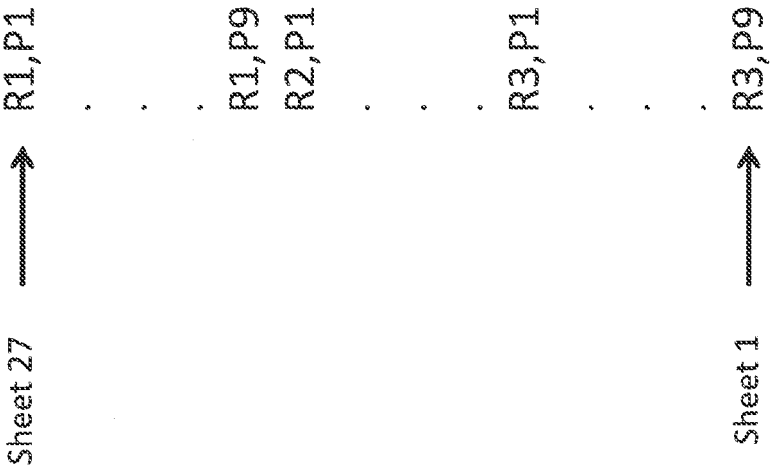

Sheet 27
R1,P1
R1,P9
R2,P1
R3,P1
R3,P9
Sheet 1

Fig. 8

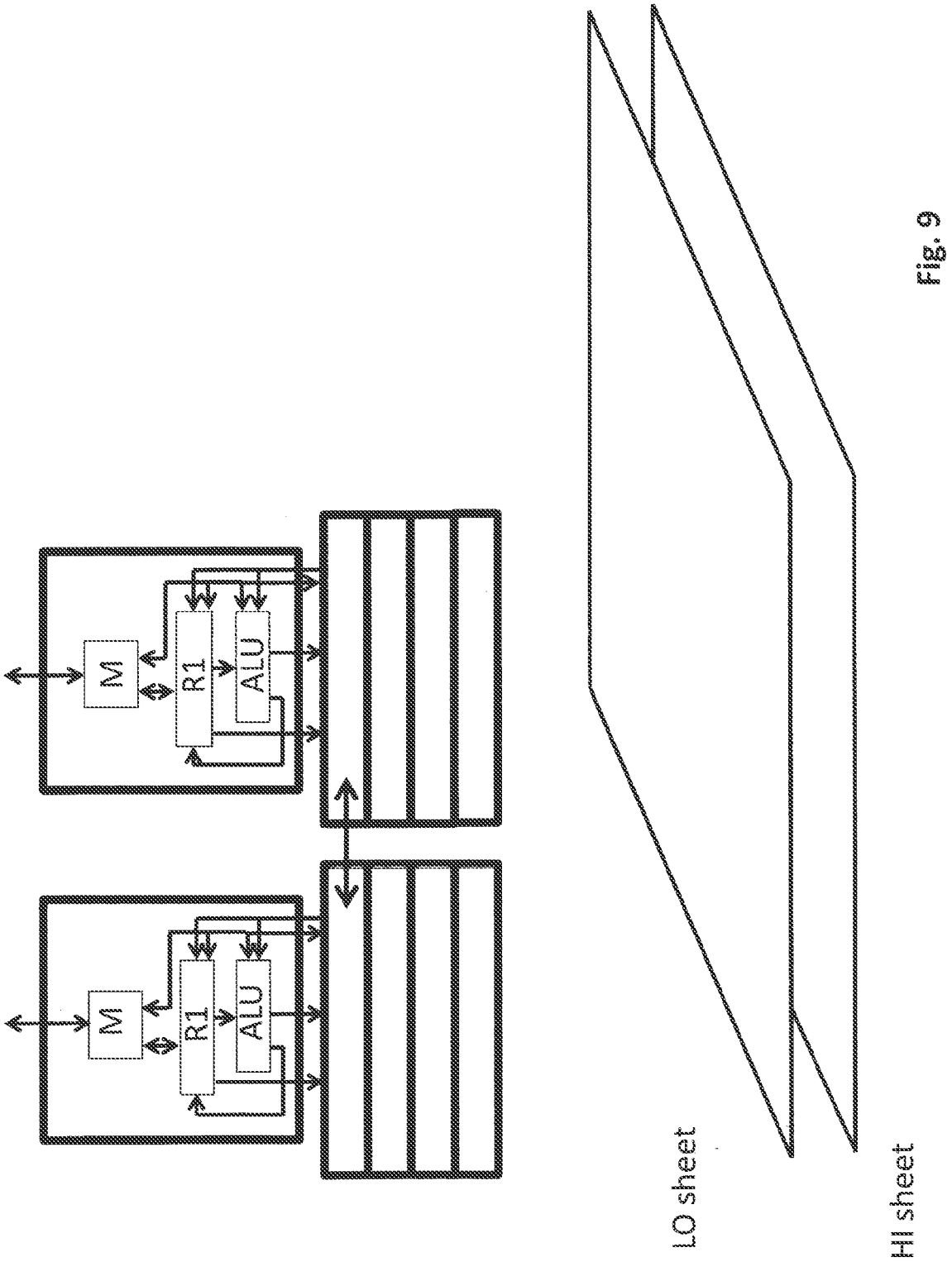
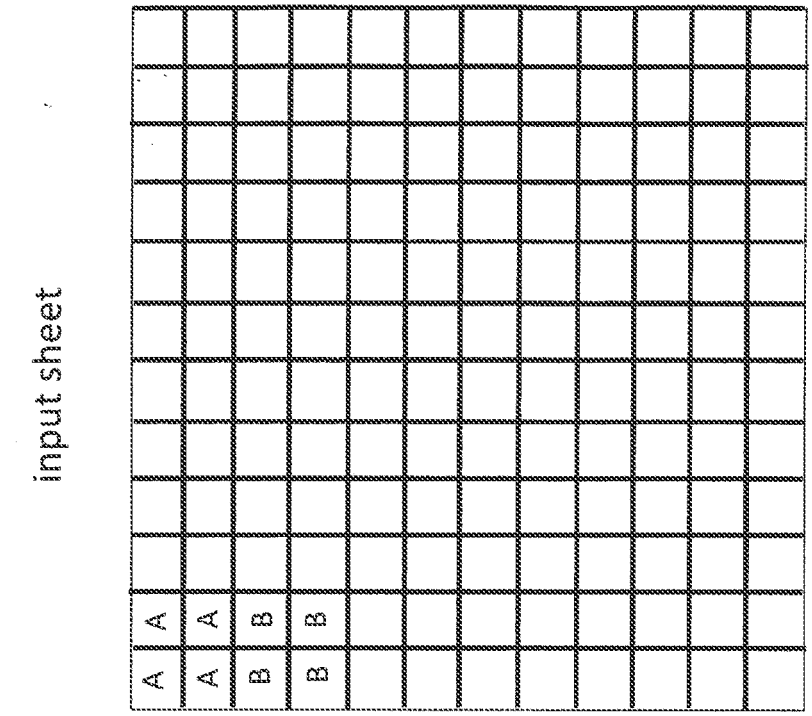


Fig. 9



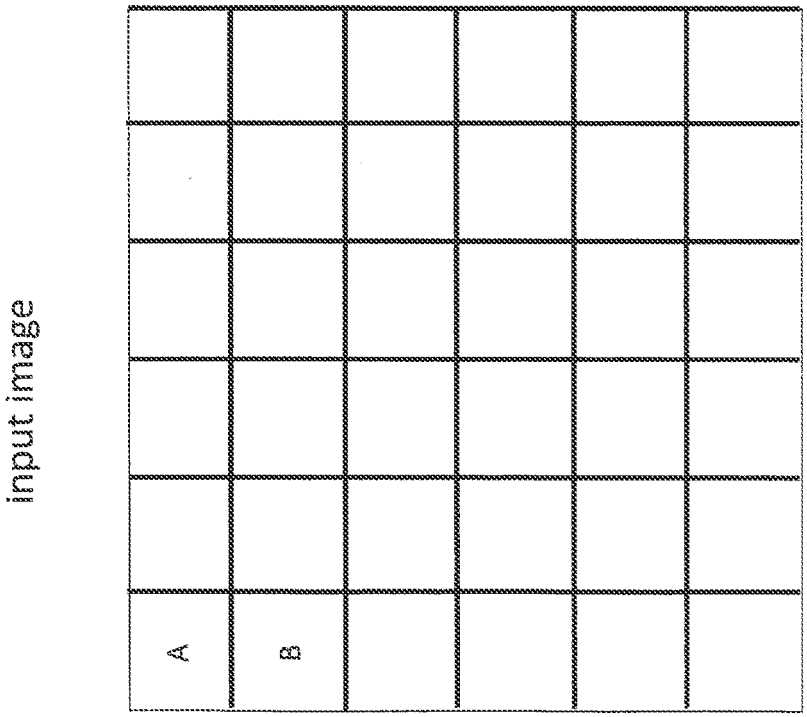


Fig. 10

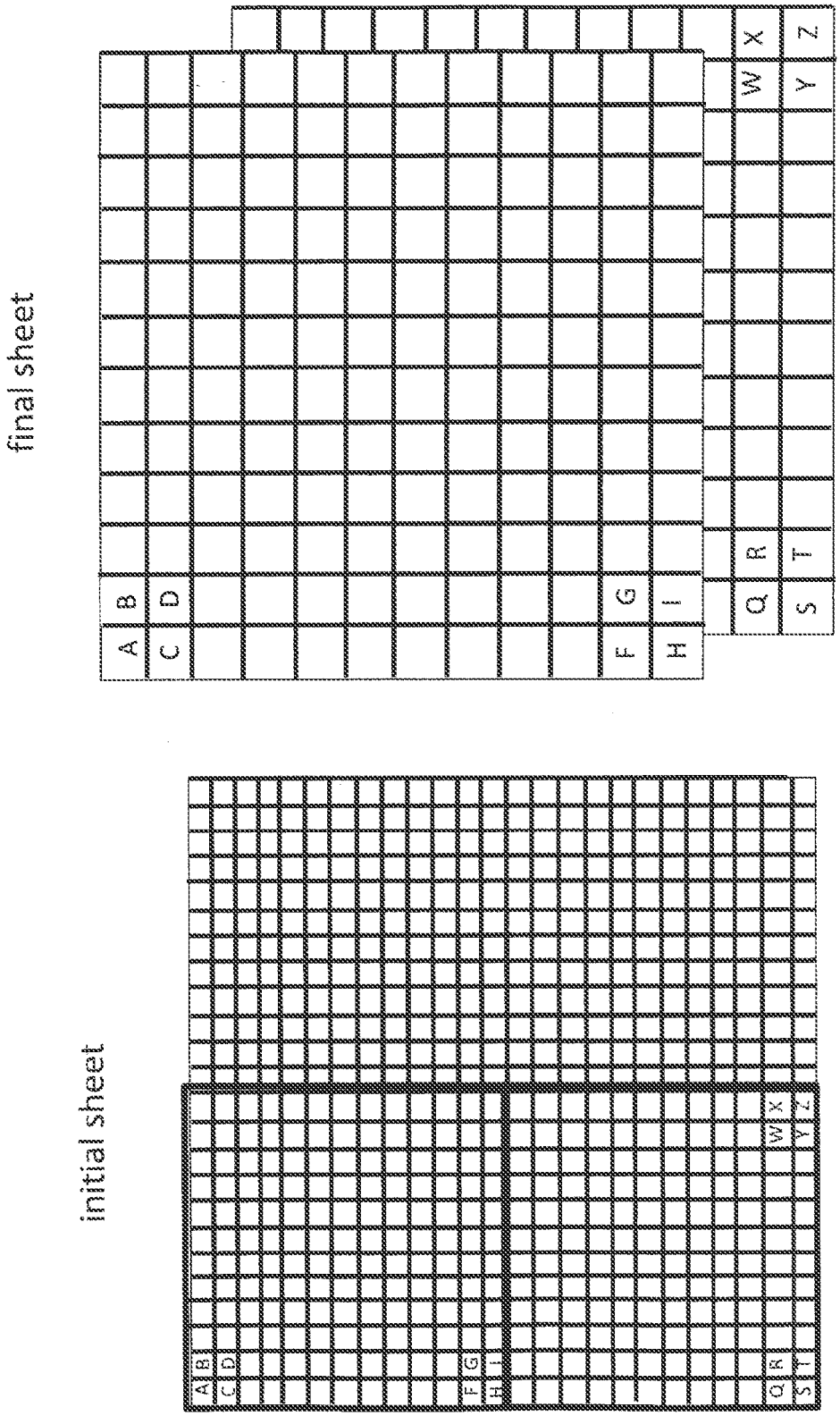

final sheet
A
B
C
D
F
G
H
I
Q
R
S
T
W
X
Y
Z
initial sheet
A
B
C
D
F
G
H
I
Q
R
S
T
W
X
Y
Z

Fig. 11

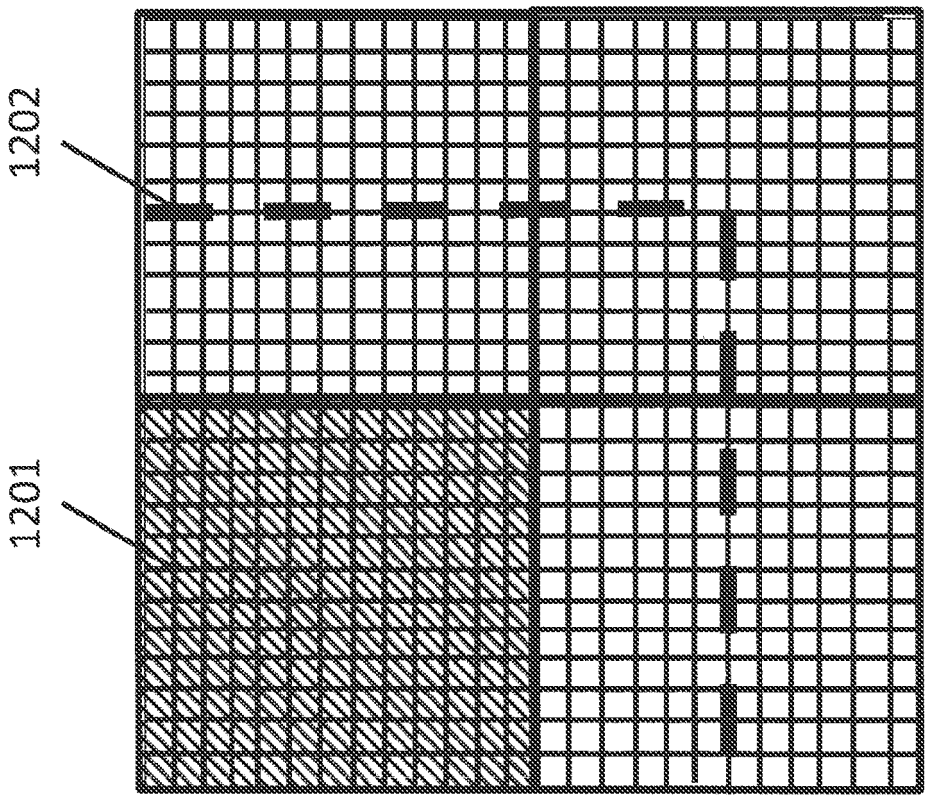


Fig. 12

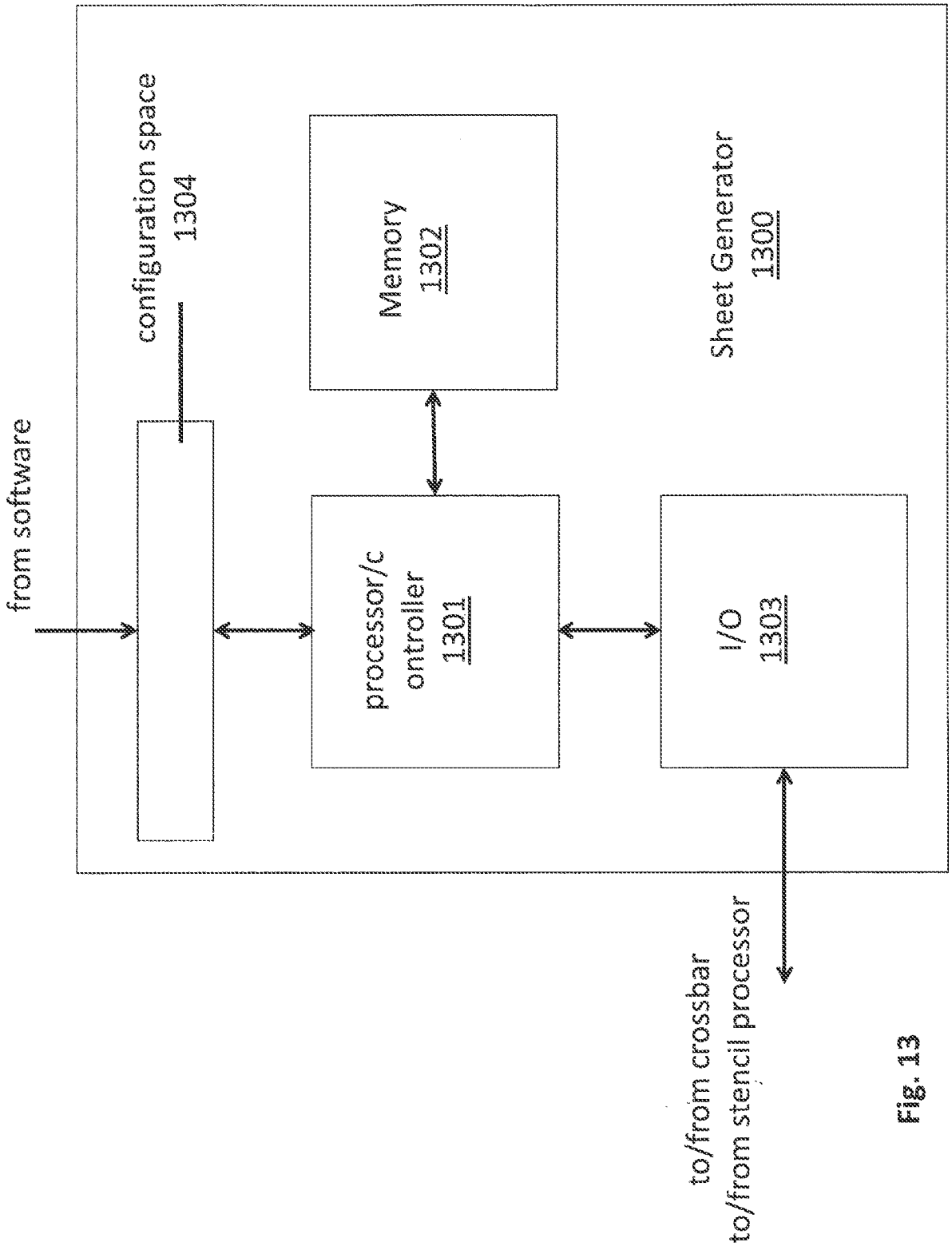

Sheet Generator
1300
configuration space
1304
Memory
1302
from software
processor/c
ontroller
1301
I/O
1303
to/from crossbar
to/from stencil processor

Fig. 13

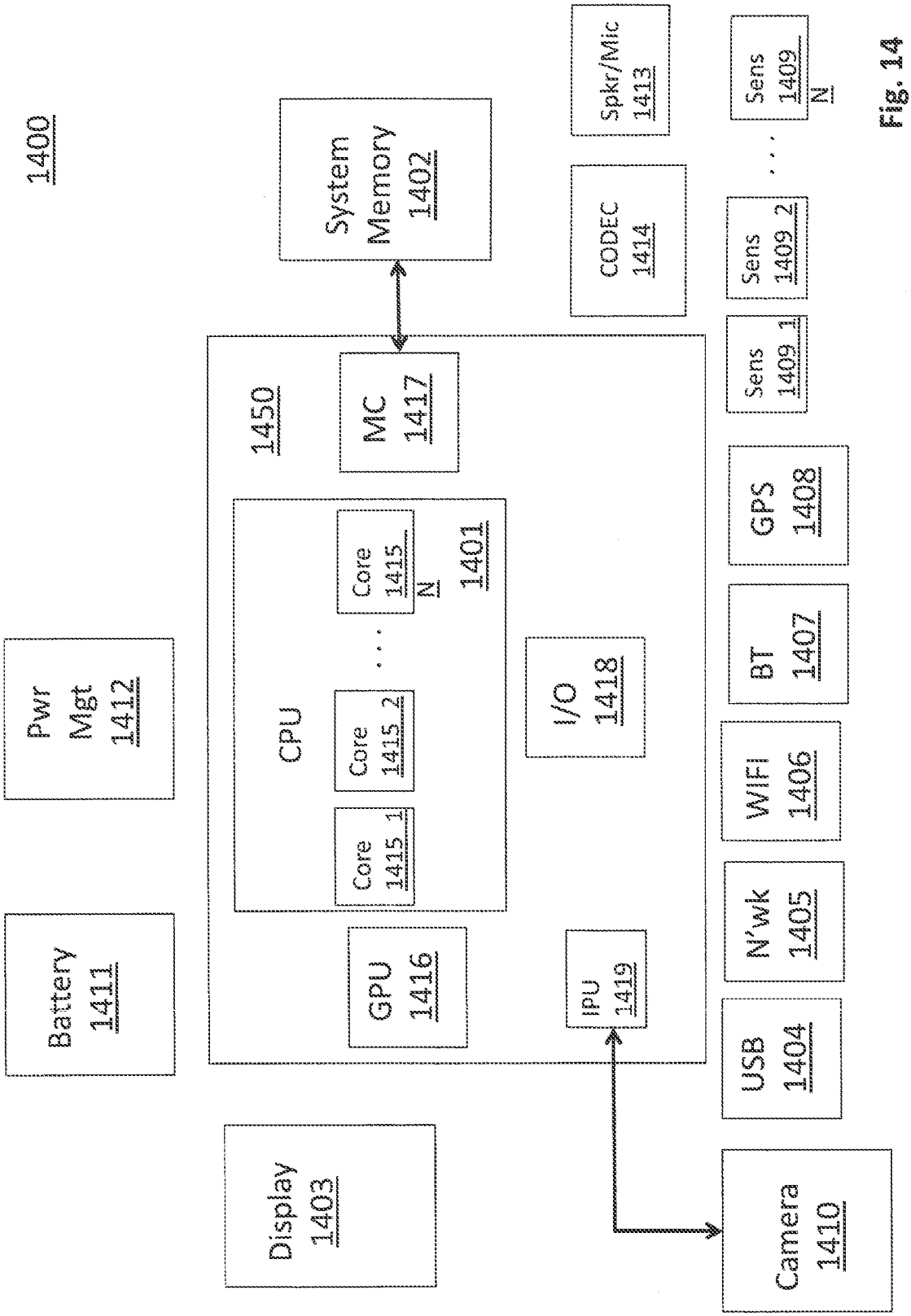

1400
System Memory 1402
Spkr/Mic 1413
Sens 1409 N
CODEC 1414
Sens 1409 2
Sens 1409 1
1450
MC 1417
GPS 1408
CPU
Core 1415 N
Core 1415 2
Core 1415 1
1401
BT 1407
I/O 1418
Pwr Mgt 1412
WIFI 1406
Battery 1411
N'wk 1405
GPU 1416
IPU 1419
USB 1404
Display 1403
Camera 1410

Fig. 14

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- US 2015086134 A **[0007]**
- US 2014164737 A **[0008]**