# RDP BITMAP HASH ACCELERATION USING SIMD INSTRUCTIONS

## BACKGROUND OF THE INVENTION

**[0001]** Although computers were once isolated and had minimal or little interaction with other computers, today's computers interact with a wide variety of other computers through communications networks, such as Local Area Networks (LANs) and Wide Area Networks (WANs). With the wide-spread growth of the INTERNET™, connectivity between computers is becoming more important and has opened up many new applications and technologies. The growth of large-scale networks, and the wide-spread availability of low-cost personal computers, has fundamentally changed the way that many people work, interact, communicate, and play.

**[0002]** One increasing popular form of networking may generally be referred to as virtual computing systems, which can use protocols such as Remote Desktop Protocol (RDP), Independent Computing Architecture (ICA), and others to share a desktop and other applications with a remote client. Such computing systems typically transmit the keyboard presses and mouse clicks or selections from the client to a server, relaying the screen updates back in the other direction over a network connection (e.g., the INTERNET). As such, the user has the experience as if their machine is operating as part of a LAN, when in reality the client device is only sent screenshots of the applications as they appear on the server side.

**[0003]** Two common techniques to send graphics data to a client are sending graphic primitives and other operations, which tell a sub-routine on the client side what and how to draw something, and sending a bitmap image to the client for display. When sequences of primitives are too complex, it may sometimes make more sense to send a bitmap representation that can more simply be displayed, rather than the potentially long sequence of other more complicated primitive operations. However, it may be too expensive to continually send full bitmap representations of the screen because of the limitations of most bit stream compressors as well as limited network bandwidth.

**[0004]** To alleviate these issues, a frame that is being sent to a client (such as an application window) may be subdivided into tiles. Those tiles are then cached on the client side, and when a tile is repeated between two bitmaps, rather than re-sending the client the tile, the server sends an instruction for the client to display the cached tile. This may greatly reduce the bandwidth costs of a RDP session, especially where tiles are frequently repeated. However, in doing this, processing resources must then be devoted to caching tiles.

**[0005]** Further, the tiling algorithm is often implemented in such a way as to maximize the chances of a cache hit. Where a tile is smaller, it has a better chance that it will be used twice (either within that frame or in a future frame). There is often a minimum useful tile size as well, because where a tile is too small, only a small benefit is received from a cache hit between two tiles.

**[0005]** These RDP bitmap caching algorithms and detecting of the difference between tiles ("tile differencing") are critically important to reducing the bandwidth of a RDP display stream to levels that are acceptable for transmission over a LAN, WAN or wireless local area network (wLAN). These caching algorithms typically trade-off processing time (frequently of the central processing unit (CPU)) on a server in exchange for a decreased amount of bandwidth required for that server to transmit the information to a client across a network.

**[0006]** One of the major processing costs of RDP bitmap caching is the hash algorithm used – an algorithm that transforms the larger image data into a smaller data that may be used as an index to a sorted data structure, such as an array or a tree. Some hashing algorithms implement a cipher block chaining (CBC) algorithm, or a variation upon a CBC algorithm. However, this processing time used on the hashing algorithm can inhibit the scalability of the server, since all available processing resources may be used by RDP sessions before any other resource – such as the server's network bandwidth – becomes exhausted. This processing time also increases the time required to encode an image frame, the rate at which these frames may be produced and sent to a client (the frame-rate (FPS)).

**[0007]** Increasing the speed of the hashing algorithm with current parallel processors is difficult, because the CBC hash algorithm is typically serial, which does not lend itself well to parallel processing, such as on a single instruction, multiple data (SIMD) processor.

**[0008]** There exist a class of processors known as vector processors that have SIMD instructions in their instruction set architecture (ISA). Streaming SIMD extensions (SSE) such as the SSE 4.2 instructions in some INTEL™ x86 ISA processors, like the NEHALEM™ processor are a form of these SIMD instructions. These processors are able to speed up processing of certain types of data because they can operate on a large chunk of data at once. For instance, where an image is being processed, instead of operating on a single pixel at a time, a SIMD processor may operate on several pixels in parallel with a single instruction. Not only does this improve the performance of processing the instruction itself, but it may decrease the time spent fetching data from memory.

**[0009]** While SIMD instructions offer opportunities for improving the performance of some types of processes, such as processing image data for compression, the algorithms and techniques required to implement the process are considerably more difficult than with a non-vector processor. Special attention must be paid to data flow, and to organizing data in such a manner that it may be operated on in parallel. To that end, there would be a benefit from new techniques to increase the parallelism in hashing operations on RDP tiles.

**SUMMARY**

**[0011]** The present techniques for acceleration make use of parallel processing by computing the hash algorithm in parallel on a multiple of bits as the serial algorithm operates, and then combining these separately computed hashes to determine a hash for the image. This produces a large reduction in CPU time required for processing, without increasing the collision hit rate – the rate at which two different inputs to a hash algorithm will produce the same output – only a small amount. In many RDP tile hashing situations, this minor increase in the collision hit rate is acceptable.

**[0010]** The present techniques are useful in scenarios beyond RDP tile classification, so long as they may benefit from an increase in hashing speed while accepting some increase in the collision hit rate. This collision hit rate increase is very minor, approximately (100/2^192)%.

**[0011]** In example embodiments of the present disclosure, a method, system and computer readable storage medium are provided for RDP bitmap hash algorithm acceleration using SIMD instructions.

**[0012]** The present techniques offer a significant gain for RDP in two cases. First, a CBC hash algorithm or CBC-type hash algorithm may be used to compute a hash of a bitmap tile. If the bitmap tile is encountered again, the hash function will produce the same cache keys and the server can save bandwidth by avoiding resending the bitmap.

**[0013]** Second, in the case where RDP is running using pure screen scraping (extracting the image data from the display output of another program), this same hash function may used to compute hashes of the image tiles on each new frame to perform a quick comparison of which tiles have changed frame to frame.

**[0014]** Both of these cases mean that the hash function is typically one of the highest CPU costs of the RDP stack and since RDP is in some cases CPU limited, reducing CPU cost may allow one to increase the throughput of RDP.

**[0015]** It can be appreciated by one of skill in the art that one or more various aspects of the disclosure may include but are not limited to circuitry and/or programming for effecting the herein-referenced aspects of the present disclosure; the circuitry and/or programming can be virtually any combination of hardware, software, and/or firmware configured to effect the herein-referenced aspects depending upon the design choices of the system designer.

**[0016]** The foregoing is a summary and thus contains, by necessity, simplifications, generalizations and omissions of detail. Those skilled in the art will appreciate that the summary is illustrative only and is not intended to be in any way limiting.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0017]** The systems, methods, and computer readable media for hash acceleration in accordance with this specification are further described with reference to the accompanying drawings in which:

**[0018]** FIG. 1 illustrates an exemplary general purpose computing environment in which in which the hash acceleration described herein may be embodied.

**[0019]** FIG. 2 illustrates exemplary operational procedures for hash acceleration.

**[0020]** FIG. 3 illustrates a client and server communicating via a remote desktop protocol (RDP) that utilizes the above accelerated tile differencing techniques.

## DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

**[0021]** FIG. 1 is a block diagram of a general purpose computing device in which the techniques described herein may be employed. The computing system environment 120 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the presently disclosed subject matter. Neither should the computing environment 120 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 120. In some embodiments the various depicted computing elements may include circuitry configured to instantiate specific aspects of the present disclosure. For example, the term circuitry used in the disclosure can include specialized hardware components configured to perform function(s) by firmware or switches. In other examples embodiments the term circuitry can include a general purpose processing unit, memory, etc., configured by software instructions that embody logic operable to perform function(s). In example embodiments where circuitry includes a combination of hardware and software, an implementer may write source code embodying logic and the source code can be compiled into machine readable code that can be

processed by the general purpose processing unit. Since one skilled in the art can appreciate that the state of the art has evolved to a point where there is little difference between hardware, software, or a combination of hardware/software, the selection of hardware versus software to effectuate specific functions is a design choice left to an implementer. More specifically, one of skill in the art can appreciate that a software process can be transformed into an equivalent hardware structure, and a hardware structure can itself be transformed into an equivalent software process. Thus, the selection of a hardware implementation versus a software implementation is one of design choice and left to the implementer.

**[0022]** Computer 141 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by computer 141 and includes both volatile and nonvolatile media, removable and non-removable media. The system memory 122 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 123 and random access memory (RAM) 160. A basic input/output system 124 (BIOS), containing the basic routines that help to transfer information between elements within computer 141, such as during start-up, is typically stored in ROM 123. RAM 160 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 159. By way of example, and not limitation, FIG. 1 illustrates operating system 125, application programs 126, other program modules 127, and program data 128.

**[0023]** The computer 141 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 138 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 139 that reads from or writes to a removable, nonvolatile magnetic disk 154, and an optical disk drive 140 that reads from or writes to a removable, nonvolatile optical disk 153 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 138 is typically connected to the system bus 121 through an non-removable memory interface such as interface 134, and magnetic disk drive 139 and optical disk drive 140 are typically connected to the system bus 121 by a removable memory interface, such as interface 135.

**[0024]** The drives and their associated computer storage media discussed above and illustrated in FIG. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 141. In FIG. 1, for example, hard disk drive 138 is illustrated as storing operating system 158, application programs 157, other program modules 156, and program data 155. Note that these components can either be the same as or different from operating system 125, application programs 126, other program modules 127, and program data 128. Operating system 158, application programs 157, other program modules 156, and program data 155 are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 141 through input devices such as a keyboard 151 and pointing device 152, commonly referred to as a mouse, trackball or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 159 through a user input interface 136 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 142 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 132. In addition to the monitor, computers may also include other peripheral output devices such as speakers 144 and printer 143, which may be connected through a output peripheral interface 133.

**[0025]** The computer 141 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 146. The remote computer 146 may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 141, although only a memory storage device 147 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 145 and a wide area network (WAN) 149, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

**[0026]** When used in a LAN networking environment, the computer 141 is connected to the LAN 145 through a network interface or adapter 137. When used in a WAN networking environment, the computer 141 typically includes a modem 150 or other means for establishing communications over the WAN 149, such as the Internet. The modem 150, which may be internal or external, may be connected to the system bus 121

via the user input interface 136, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 141, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates remote application programs 148 as residing on memory device 147. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

**[0027]** FIG. 2 depicts exemplary operational procedures for hash acceleration. A serial hashing process may take one DWORD (double word; in many system architectures this is 32 bits) of image data at a time, starting at the front of the image and working to the back. It will compute one key for the image, which is updated based on each new DWORD of the image that is processed. In contrast, the present exemplary operating procedures operate on a plurality of DWORDS at a time. In an embodiment where a DWORD comprises 32 bits and the processor may operate upon 128 bits simultaneously, this may be four DWORDS of data. The procedures keep four partial keys for the image – one corresponding to each DWORD operated upon in parallel – and those four partial keys may be combined once all of the image has been processed to produce a final key.

**[0028]** Optional operation 202 depicts selecting a first random number and a second random number, the first random number differing from the second random number.

**[0031]** Optional operation 204 depicts setting each of a checksum, a datum, a first key and a second key equal to zero. In any embodiment, the checksum, datum, first key and second key equal zero before the present operations begin, so it is not necessary to set them to zero at this point. Where the techniques use these four numbers to keep running sums, they may be initialized by setting them to zero. The checksum and the datum may be used as internal variables that are used to determine the first key and the second key. The first key and the second key may comprise a big key for an image, the big key later used to look up the image or a reference thereto, such as for a server to look up whether it has sent a client the image, or for the client that receives the big key in lieu of an image it has already received, to look up the image itself. It is faster to calculate the big key as these two components – the first key and the second key – and then combine them than it is to calculate a monolithic big key.

**[0029]** Operation 206 depicts grouping the image into at least one group, each group comprising a number of bits of the image equal to the bit width. A processor may operate on more than one bit of data at a time. For instance, a 128-bit processor may operate on 128 bits of data at a time. This processor then has a "bit width" of 128 bits. Those 128

bits need not belong to the same data structure, such as one that represent a 128-bit number. Rather, that 128 bits may comprise multiple pieces of data, such as four discrete 32 bit-integers, or eight discrete 16-bit integers. In such a situation, instructions exist such that when an operation is performed on a group of four 32-bit integers, the result of that operation (such as a multiple, or a logical shift) does not overflow into an adjoining integer in the 128 bits.

**[0030]** In an embodiment, the image comprises a screen scrape. A screen scrape comprises taking data from the display output of another program. Since this data was intended by the other program to be displayed on an output device, it typically lacks documentation or structure to aid in parsing it.

**[0031]** In an embodiment, the image comprises a tile in a Remote Desktop Protocol (RDP), the tile comprising a bitmap. A tile may comprise a subset of a frame, a frame typically comprising an application window. In a typical embodiment, a frame is divided into a plurality of rectangular tiles, and only those tiles that have changed need to be sent to the client.

**[0035]** In an embodiment, where the image is unaligned on an alignment boundary, the first bits of the image prior to the first bit boundary are hashed with a serial cipher block chaining (CBC) algorithm to produce a pre-first key and a pre-second key. For instance, where the present operations operate on 128 bits in parallel, and the image is unaligned on a 128-bit boundary, with 56 bits of the image occurring before the first 128-bit boundary in the image, those 56 bits are hashed with a serial CBC algorithm, and the resulting pre-first key and pre-second key are combined with the resulting first key and second key, respectively, of the following operations via a serial CBC algorithm.

**[0032]** Operation 208 depicts, for each group, performing a hash comprising: setting the checksum equal to the checksum plus the group; setting the datum to equal the sum of the group and the datum multiplied by the first random number and summed with the second random number; setting the first key to equal the first key modified by the datum; and setting the second key to equal the second key modified by the datum.

**[0033]** In an embodiment, the first key is modified by the datum by performing an exclusive-or operation on the first key and the datum, and the second key is modified by the datum by performing an exclusive-or operation on the second key and the datum. An exclusive or is a bitwise operation where the result of the two inputs is 1 where one and only one of the inputs is also a 1. For instance, where "^" denotes the exclusive OR operation, 0^0=0, 0^1=1, 1^0=1, and 1^1=0.

**[0034]** The image may be thought of as having a start and an end, and in an embodiment, the operations performed for each group are first performed for the group nearest to the start for which the operations performed for each group have not already been performed. That is, the technique proceeds sequentially through the image, taking one group of bits at a time and processing it before doing the same for the next group of bits, until the end of the image is reached.

**[0035]** In an embodiment, the hash comprises a cipher block chaining (CBC) hash.

**[0036]** Optional operation 210 depicts setting a DWORD first key based on each DWORD size of bits of the first key; and setting a DWORD second key based on each DWORD size of bits of the second key.

**[0037]** Where the first key and the second key each comprise fewer bits than the bit width of the processor, partial keys are calculated. When all of the image has been processed, these partial keys are then used to determine the full first key and the second key. Take the embodiment where the first key and the second key each comprise 32 bits and the bit width is 128 bits. In this embodiment, 128 bits of the image are processed simultaneously, producing four 32-bit partial first keys and four 32-bit partial second keys. When the entire image has been processed, then each of those four partial keys is used to determine the full key, or DWORD first key and DWORD second key.

**[0038]** In an embodiment, the four partial first keys are used to determine the DWORD first key by exlusive-or'ing (XORing) them together. Where the four partial first keys are expressed as Key1[0], Key1[1], Key1[2], and Key1[3], this operation may logically be expressed as Key1[0] ^ Key1[1] ^ Key1[2] ^ Key1[3].

**[0043]** In an embodiment, the four partial first keys are used to determine the DWORD first key by combining them using a serial CBC hash algorithm. In an embodiment, Key1[0] is CBC hashed with Key1[1], the result of which is CBC hashed with Key1[2], the result of which is CBC hashed with Key1[3].

**[0044]** Optional operation 212 depicts, when the image size divided by the bit width has a remainder number of bits other than zero, hashing the remainder number of bits with a serial algorithm to produce a post-first key and a post-second key, determining the first key based on the post first-key, and determining the second key based on the post-second key. Where the image has a number of bits that is evenly divisible by the bit width, a series of parallel operations as described in the above operations may take place on every bit of the image. However, where the image has a number of bits that is not evenly divisible by the bit width, a conventional serial hashing of those final bits may take place,

as opposed to the above parallel hashing. The resulting post-first key and post-second key may be combined with the first key and the second key, such as by setting the first key to the result of performing a serial CBC hash on the first key and the post-first key, and by setting the second key to the result of performing a serial CBC hash on the second key and the post-second key.

**[0045]** In an embodiment, the image has an end, and no part of the image is nearer the end than the remainder number of bits. That is, the remainder bits will be considered to be the final bits of the image, and they shall be processed in serial fashion after all proceeding bits have been processed in parallel fashion, a bit width number of bits at a time.

**[0046]** In an embodiment, the first key and the second key that are calculated from this serial processing are combined with the first key and the second key calculated from the above parallel processing, such as through a serial CBC hash of the respective first keys, as well as a serial CBC hash of the respective second keys, an XOR operation of the respective first keys, as well as the respective second keys.

**[0039]** Operation 214 depicts storing a big key comprising the first key and the second key in a big key store. Once hashed, the image may be stored at a memory address and in a data structure, such as a tree, and the datum is then used to access the image at the location of the memory address. For instance, where an array memory structure is used, the datum may serve as an index to that array, and the image is stored at the datum-index of the array.

**[0048]** Where each key comprises 32 bits, the big key comprises 64 bits, so there exist 2^64 possible values for a big key. In an embodiment, while the possibility of a collision (two different image tiles correspond to the same big key) the odds of that occurring are so small (approximately 1 in 2^64 as between any two big keys) that the processing resources required to detect collisions greatly outweigh the benefit of detecting one and identifying the correct tile image that collisions are ignored and an incorrect image may be used.

**[0040]** Operation 216 depicts encoding the image. In an embodiment, this comprises encoding the image via run-length encoding (RLE). In an embodiment, this may comprise encoding different portions of the image with different codecs. For instance, where part of the image must be displayed in lossless quality, such as if it is a medical x-ray used by a physician, it may be encoded with a lossless codec. Another part of the image may be black text on a white background, and encoding it with a RLE algorithm would reduce its space by the most amount (as well as be lossless). In this case, these two codecs may be used on the appropriate parts of the image. In an embodiment, the color channels of the

image are separated – a RGBA image is separated into its individual red (R), green (G), blue (B) and alpha (A) color channels, and each of those channels is separately encoded.

**[0041]** Operation 218 depicts sending the image to a client computer across a communications network. This is not necessary where the image has already been sent to the client, and the client caches images that it has received. In this case, the client may be sent the big key. The client may maintain a hash table of received images similar to the hash table maintained by the apparatus performing the present operations. In this situation, when it receives the big key, the client may use the big key as an index to its hash structure, receive the corresponding image, and process that image as it would if it had received the image rather than the datum.

**[0042]** FIG. 3 illustrates a client and server communicating via a remote desktop protocol (RDP) that utilizes the above accelerated tile differencing techniques. A server 302 comprises a RDP server 304, a key database 306, and at least one bitmap image 308. The image 308 is divided into at least one tile 310, each tile comprising a bitmap. The server 302 communicates with a client 314 across a communications network 312. The client 314 comprises a RDP client 316, and a tile database 318, and is connected to a display device 320.

**[0043]** In a RDP session between the client 314 and the server 302, the server sends the client image information across the network 312, corresponding to processing that the server 302 is executing. For instance, the client 314 may have a RDP session where it executes a text editor on the server. The client 314 sends commands to the server 302, such as inputting a series of characters into the text editor at the current editing position, or opening a new file. Those commands are processed on the server 302, and the resulting display output is sent back to the client 314 for display on display device 320. In such an embodiment, the image 308 may comprise the text editor's application window at a given point in time. If the user is adding new text near the bottom of the application window, then the top portion of the application window may not change in the near future. So, the image 308 may be divided into tiles 310, and as a plurality of images 308 is sent to the client 314 over time, each image 308 sent as a plurality of tiles 310, only those tiles 310 that are different from all preceding tiles 310 need be sent. The client 314 may cache previously received tiles 310 in a tile cache 318, and where a tile 310 is repeated, the server 302 may send the client 314 an indication of that tile rather than the tile 308 itself. That first tile that is now cached in the tile cache 318 need not be in the same location as the repeated tile. For instance, with a new document in a text editing session, most tiles

will be nothing but whitespace, so one pure white tile may be used multiple times for all of those pure white tiles.

**[0044]** The first time the server 302 receives a request for an image 308, it will take the first tile 310, and hash it to determine a big key. Where the server 306 hashes the tile 310 according to the techniques depicted in FIG. 2, the key may comprise the first key and the second key. It will then use the RDP server 304 to send the tile to the RDP client 316, along with an indication of where in the image the tile 310 belongs, and will also store the big key to the tile in the key database 306. The key database may comprise a variety of data structures for storing integers, such as a tree.

**[0045]** For each successive tile 310, the server 302 will determine a big key for it, and then check this against the key database 306. Where there is no match in the key database 306, indicating that a tile representing the same image has not previously been sent to the client 314, the sever 302 will send the tile to the client 314 and store the big key to it in the key database 306, as before. The client receives the tile at the RDP client 316, and caches it in the tile cache 318, along with the big key. The big key may be used as a key to locate the image in the tile cache 318, such as by serving as an index to a hash table.

**[0046]** Where there is a match to this tile in the key database 306, that means that the server 302 has previously sent a tile representing the same image as this tile. So, rather than send the client 314 the tile 310, to save network resources, the server 302 will send the client 314 the corresponding big key, which comprise a smaller amount of data. Where the first key and the second key each comprise 32 bits of data, the big key will comprise 64 bits. The client 314 receives this key and datum at the RDP client 316, and the RDP client 314 uses the big key to locate the corresponding tile in the tile cache 320.

**[0047]** Whether the RDP client 314 received the tile itself or received the corresponding big key and looked up the tile in the tile cache 318, it now has the tile. Along with the tile or the big key, the RDP client 316 will receive an indication of which part of the image 308 the tile 310 comprises. The RDP client 316 will then display the tile 310 in the appropriate part of the image on client display device 320.

**Sample Instructions**

**[0057]** The following comprises sample pseudo-code in a high level programming language C-type syntax that, when executed on a vector processor will operate on an image in the manner of an embodiment of the present disclosure, similar to as described in the detailed description of FIG. 2.

```
__inline void __fastcall NextCBC64_SSE(
        CBC64Context *pContext,
```

```
        UINT32 *pData,
        unsigned NumDWORDBlocks)
{

    UINT32 Checksum;
    register UINT32 Datum;
    register UINT32 Key1;
    register UINT32 Key2;
    UINT32 numQuadBlocks;

    const UINT32 CBC_AB[2] = { CBC_b, CBC_bXORa };
    const UINT32 CBC_CD[2] = { CBC_d, CBC_dXORc };

    if (NumDWORDBlocks != 0) {
        Checksum = pContext->Checksum;
        Datum = pContext->Datum;
        Key1 = pContext->Key1;
        Key2 = pContext->Key2;

        INT iterStart = NumDWORDBlocks % 8;

         if (iterStart) {
            NumDWORDBlocks -= iterStart;
            do {
                        Checksum += *pData;
                        Datum = CBC_RandomOddAlpha * (*pData + Datum)+\

      CBC_RandomBeta;
                        Key1 ^= Datum;
                        Key1 = (Key1 << 1)^(CBC_CD[(Key1&0x80000000) >>
31]);
                        Key2 ^= Datum;
                        Key2 = (Key2 << 1)^(CBC_AB[(Key2&0x80000000) >>
31]);
                  pData++;
                  iterStart--;
            } while (iterStart);
        }

        __m128i xmmRawData;
        __m128i xmmChecksum;
        __m128i xmmDatum;
        __m128i xmmCBC_RandomAlpha;
        __m128i xmmCBC_RandomBeta;
        __m128i xmm1;
        __m128i xmmKey1;
        __m128i xmmKey2;

        xmmChecksum = _mm_set_epi32((int)Checksum, (int)Checksum,
(int)Checksum, (int)Checksum);
        xmmCBC_RandomAlpha = _mm_set_epi32((int)CBC_RandomOddAlpha,
          (int)CBC_RandomOddAlpha, (int)CBC_RandomOddAlpha,
(int)CBC_RandomOddAlpha);
        xmmCBC_RandomBeta = _mm_set_epi32((int)CBC_RandomBeta,
(int)CBC_RandomBeta, (int)CBC_RandomBeta, (int)CBC_RandomBeta);
        xmmDatum = _mm_setzero_si128();

        xmmKey1 = _mm_set_epi32((int)Key1, (int)Key1+CBC_RandomOddAlpha,
(int)Key1+CBC_RandomBeta, (int)Key1-CBC_RandomBeta);
        xmmKey2 = _mm_set_epi32((int)Key2, (int)Key2+CBC_RandomOddAlpha,
(int)Key2+CBC_RandomBeta, (int)Key2-CBC_RandomBeta);

        PUINT32 pDataEnd = pData +numQuadBlocks*8;

        while (pData < pDataEnd) {
              xmmRawData = _mm_load_si128((__m128i*)pData);
              pData += 4;

              xmmChecksum = _mm_add_epi32(xmmRawData, xmmChecksum);
              xmm1 = _mm_add_epi32(xmmRawData, xmmDatum);
```

```
                xmm1 = _mm_madd_epi16(xmm1, xmmCBC_RandomAlpha);
                xmmDatum = _mm_add_epi32(xmm1, xmmCBC_RandomBeta);
                xmmKey1 = _mm_xor_si128(xmmKey1, xmmDatum);
                xmmKey2 = _mm_xor_si128(xmmKey2, xmmDatum);
                xmmRawData = _mm_load_si128((__m128i*)pData);
                pData += 4;

                xmmChecksum = _mm_add_epi32(xmmRawData, xmmChecksum);

                xmm1 = _mm_add_epi32(xmmRawData, xmmDatum);
                xmm1 = _mm_madd_epi16(xmm1, xmmCBC_RandomAlpha);
                xmmDatum = _mm_add_epi32(xmm1, xmmCBC_RandomBeta);

                xmmKey1 = _mm_xor_si128(xmmKey1, xmmDatum);
                xmmKey2 = _mm_xor_si128(xmmKey2, xmmDatum);

                }

        PUINT32 pQuadKey1 = (PUINT32)&xmmKey1;
        PUINT32 pQuadKey2 = (PUINT32)&xmmKey2;
        PUINT32 pQuadDatum = (PUINT32)&xmmDatum;

        #define BLEND_QUAD(quad) (quad[0] ^ quad[1] ^ quad[2] ^ quad[3])

        pContext->Checksum = Checksum;
        pContext->Datum = Datum ^ BLEND_QUAD(pQuadDatum);
        pContext->Key1 = Key1 ^ BLEND_QUAD(pQuadKey1);
        pContext->Key2 = Key2 ^ BLEND_QUAD(pQuadKey2);
        }
}
```

## Conclusion

**[0048]** While the present disclosure has been described in connection with the preferred aspects, as illustrated in the various figures, it is understood that other similar aspects may be used or modifications and additions may be made to the described aspects for performing the same function of the present disclosure without deviating there from. Therefore, the present disclosure should not be limited to any single aspect, but rather construed in breadth and scope in accordance with the appended claims. For example, the various procedures described herein may be implemented with hardware or software, or a combination of both. Thus, the methods and apparatus of the disclosed embodiments, or certain aspects or portions thereof, may take the form of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other machine-readable storage medium. When the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus configured for practicing the disclosed embodiments. In addition to the specific implementations explicitly set forth herein, other aspects and implementations will be apparent to those skilled in the art from consideration of the specification disclosed herein. It is intended that the specification and illustrated implementations be considered as examples only.

What is Claimed:

1. A method for classifying an image on a parallel processing system (314) having a bit width, the method involving a first random number and a second random number, the first random number differing from the second random number, and a checksum, a datum, a first key and a second key, comprising:

grouping the image into at least one group, each group comprising a number of bits of the image equal to the bit width (206);

for each group, performing a hash comprising (208):

setting the checksum equal to the checksum plus the group;

setting the datum to equal the sum of the group and the datum multiplied by the first random number and summed with the second random number;

setting the first key to equal the first key modified by the datum; and

setting the second key to equal the second key modified by the datum; and

storing a big key comprising the first key and the second key in a big key store (216).

2. The method of claim 1, further comprising:

sending the image to a client computer across a communications network.

3. The method of claim 2, wherein sending the image to a client computer across a communications network occurs after:

determining that the big key does not match any other big key in the big key store;

4. The method of claim 3, further comprising:

sending the client the big key.

5. The method of claim 4, further comprising:

sending the client computer an indication to use the big key to retrieve a second image from a client hash, and to process the second image as if the client had received it instead of the indication.

6. The method of claim 2, further comprising:

encoding the image before sending the image to the client computer.

7. The method of claim 1, further comprising:

setting each of a checksum, a datum, a first key and a second key equal to zero before performing operations on each group.

8. The method of claim 1, wherein the first key is modified by the datum by performing an exclusive-or operation on the first key and the datum, and wherein the second key is modified by the datum by performing an exclusive-or operation on the second key and the datum.

9. The method of claim 1, wherein the image has a start and an end, and the operations performed for each group are first performed for the group nearest to the start for which the operations performed for each group have not already been performed.

10. The method of claim 1, further comprising:

when the image size divided by the bit width has a remainder number of bits other than zero, hashing the remainder number of bits with a serial cipher block chaining (CBC) algorithm to produce a post-first key and a post-second key, determining the first key based on the post-first key, and determining the second key based on the post-second key.

11. The method of claim 5, wherein the image has an end, the reminder number of bits and no part of the image is nearer the end than the remainder number of bits.

12. The method of claim 1, wherein the hash comprises a cipher block chaining (CBC) variant.

13. The method of claim 1, wherein the image comprises a screen scrape.

14. The method of claim 1, wherein the image comprises a tile in a Remote Desktop Protocol (RDP).

15. A system for classifying an image on a parallel processing system having a bit width, the method involving a first random number and a second random number, the first random number differing from the second random number, and a checksum, a datum, a first key and a second key, comprising:

a processor (159);

circuitry for grouping the image into at least one group, each group comprising a number of bits of the image equal to the bit width (206);

circuitry for, for each group, performing a hash comprising (208):

calculating a running reference to the image, the running reference comprising the checksum, the first key, the second key, and the datum, and calculated based on the running reference, the group, the datum, the first random number and the second random number; and

circuitry for storing a big key comprising the first key and the second key in a big key store (214).