Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2684441 A1 2010/02/04

(21) 2 684 441

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(22) Date de dépôt/Filing Date: 2009/09/22

(41) Mise à la disp. pub./Open to Public Insp.: 2010/02/04

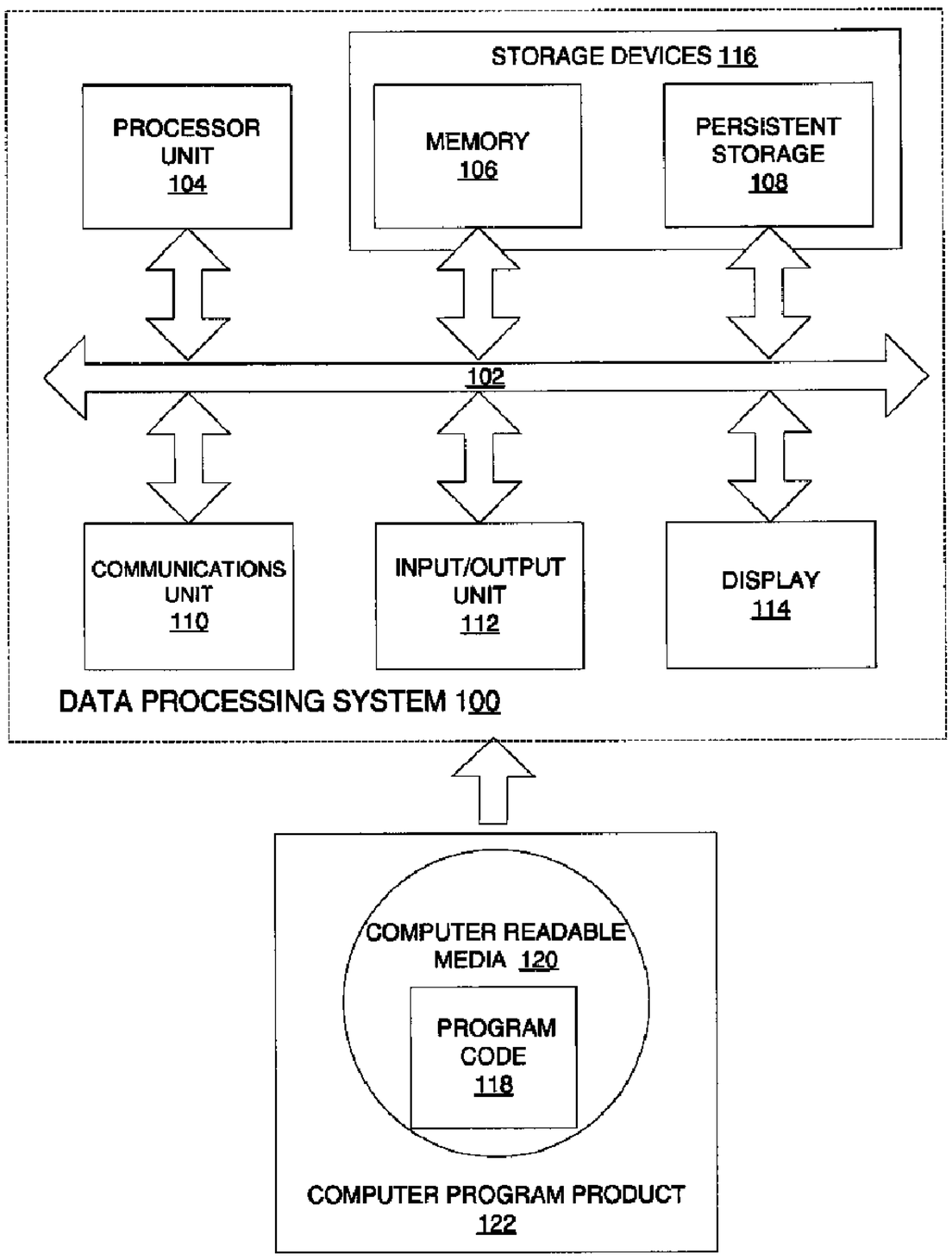
(51) Cl.Int./Int.Cl. *G06F 9/44* (2006.01)

(71) Demandeur/Applicant:
IBM CANADA LIMITED - IBM CANADA LIMITEE, CA

(72) Inventeurs/Inventors:
ARCHAMBAULT, ROCH G., CA;
CUI, SHIMIN, CA;
GAO, YAOQING, CA

(74) Agent: WANG, PETER

(54) Titre : PROPAGATION POSSIBLEMENT CONSTANTE
(54) Title: MAY-CONSTANT PROPAGATION



(57) Abrégé/Abstract:
An illustrative embodiment provides a computer-implemented process for may--constant propagation, obtains a source code, and generates a set of associated data structures from the source code and a set of may-constant data structures. The computer--

(21) 2 684 441

(13) A1

(57) Abrégé(suite)/Abstract(continued):
implemented process identifies a candidate code for may-constant propagation to form an identified candidate code, updates the set of may-constant data structures, and selects an identified candidate code using information in the may-constant data structures, including probability, to form a selected candidate code. The computer-implemented process further identifies a code region associated with the selected candidate code to form an identified code region and modifies the identified code region including the selected candidate code.

ABSTRACT OF THE DISCLOSURE

An illustrative embodiment provides a computer-implemented process for may-constant propagation, obtains a source code, and generates a set of associated data structures from the source code and a set of may-constant data structures. The computer-implemented process identifies a candidate code for may-constant propagation to form an identified candidate code, updates the set of may-constant data structures, and selects an identified candidate code using information in the may-constant data structures, including probability, to form a selected candidate code. The computer-implemented process further identifies a code region associated with the selected candidate code to form an identified code region and modifies the identified code region including the selected candidate code.

MAY-CONSTANT PROPAGATION

BACKGROUND

Statement of Government Rights

This invention was made with Government support under Contract number HR0011-07-9-0002 awarded by the Defense Advanced Research Projects Agency (DARPA). The Government has certain rights in this invention.

1. Technical Field:

[0001] This disclosure relates generally to propagation of constants in a data processing system and more specifically to propagation of may-constants.

2. Description of the Related Art:

[0002] In a data processing system environment programming languages typically provide a mechanism for the propagation of constants as used within computer executable program code. Constants that are assigned to a variable can be propagated through data structures built in memory. Typical data structures include examples such as a call graph and a control flow graph. The propagated constant is substituted at the instance or location of use of the variable.

[0003] The existing technique of constant propagation can be used to trigger the removal of dead code. Constant propagation, in addition, can be used to perform partial evaluation of code segments at compile time.

[0004] Typical existing constant propagation algorithms cannot handle cases such as either intra-procedure or inter-procedure propagation of may-constants in which the variable in use has an unknown value but may have constant values with high probability. For example, with reference to **Figure 1**, a textual representation of an intra-procedure may-constant propagation of a variable of an unknown value is presented. Code snippet **100** represents a portion of computer executable program code or instructions of an intra-procedure may-constant propagation. Statement **102** in a first condition assigns a value of variable "x" equal to "3." Statement **104** representing a "else" portion of the first

condition, assigns the variable "x" equal to another variable "w" that is unknown. Statement **106** further assigns variable "x" to another variable "z" that is also unknown. Statement **108** represents the use of variable "x," however, the value of "x=w" and the value of "x=z" are unknown and cannot be substituted with a propagated constant.

[0005] May-constant propagation is a technique used to propagate a constant through the call graph and control flow graph by ignoring possible kills and re-definitions with low probability. Using a technique of static analysis and dynamic profiling in the example provided, a first determination may provide a probability of 50 percent for variable "x" being assigned "w" while a probability of variable "x" being assigned "z" may be reduced to 25 percent. However, when the probability of "x=w" and "x=z" is determined to be low, variable "x" at statement **108** may have a constant value of "3" with high probability based on the conditions. Since it is a may-constant, compile time folding may not be done, but code versioning with the inserted runtime check can be used to specialize the code for partial evaluation at compile-time. For example, "q=x*x*x" at statement **108** can be replaced by the following code: "`if (x == 3) { q = 27; } else { q = x * x * x; }`".

[0006] With reference to **Figure 2**, a textual representation of an inter-procedure may-constant propagation is presented. Domain or programming language specific information can be used to identify what may-constants should be traced and what kills and re-definitions can be ignored. Code snippet **200** is a code portion comprising a Fortran programming language example. Since an assumed-shape array is rarely relocated or the allocation through memory allocation routine *malloc* rarely fails, the constants for an array descriptor such as *ref. d-t41%* can be propagated within a procedure and across procedures. Code snippet **200** represents a portion of computer executable program code or instructions that may be used in an inter-procedure may-constant propagation. The example code snippet is for an assumed shape array. The assumed shape array is an array that has an unknown shape, but the shape is determined by the actual arguments presented.

[0007] Statements **202** represents statements associated with the actual allocation of the assumed shape array according to the definitions of the previous statements of code snippet **200**. Through static analysis or dynamic profiling, statements **202** may be shown

to have a low probability of being executed. Therefore there is a requirement to provide a more predictable form of may-constant propagation to address scenarios in which variables are unknown.

BRIEF SUMMARY

[0008] According to one embodiment, a computer-implemented process for may-constant propagation is presented. The computer-implemented process obtains a source code, and generates a set of associated data structures from the source code and a set of may-constant data structures. The computer-implemented process identifies a candidate code for may-constant propagation to form identified candidate code, updates the set of may-constant data structures, and selects an identified candidate code using information in the may-constant data structures, including probability, to form a selected candidate code. The computer-implemented process further identifies a code region associated with the selected candidate code to form an identified code region and modifies the identified code region including the selected candidate code.

[0009] According to another embodiment, a computer program product for may-constant propagation comprises a computer recordable-type media containing computer executable program code stored thereon, the computer executable program code comprises computer executable program code for obtaining a source code, computer executable program code for generating a set of associated data structures from the source code, computer executable program code for generating a set of may-constant data structures, computer executable program code for identifying a candidate code for may-constant propagation to form identified candidate code, computer executable program code for updating the set of may-constant data structures, computer executable program code for selecting an identified candidate code using information in the may-constant data structures, including a probability, to form a selected candidate code, computer executable program code for identifying a code region associated with the selected candidate code to form an identified code region, and computer executable program code for modifying the identified code region including the selected candidate code.

[0010] According to another embodiment, an apparatus for apparatus for may-constant propagation comprises a communications fabric, a memory connected to the communications fabric, wherein the memory contains computer executable program code, a communications unit connected to the communications fabric, an input/output unit connected to the communications fabric, a display connected to the communications fabric, and a processor unit connected to the communications fabric, wherein the processor unit executes the computer executable program code to direct the apparatus to obtain a source code, generate a set of associated data structures from the source code, generate a set of may-constant data structures, identify a candidate code for may-constant propagation to form identified candidate code, update the set of may-constant data structures, select an identified candidate code using information in the may-constant data structures, including a probability, to form a selected candidate code, identify a code region associated with the selected candidate code to form an identified code region, and modify the identified code region including the selected candidate code.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

[0011] For a more complete understanding of this disclosure, reference is now made to the following brief description, taken in conjunction with the accompanying drawings and detailed description, wherein like reference numerals represent like parts.

[0012] Figure 1 is a textual representation of a typical intra-procedure may-constant propagation;

[0013] Figure 2 is a textual representation of a typical inter-procedure may-constant propagation;

[0014] Figure 3 is a block diagram of an exemplary data processing system operable for various embodiments of the disclosure;

[0015] Figure 4; is a block diagram of a compilation system that may be implemented within the data processing system of **Figure 3**, in accordance with various embodiments of the disclosure;

[0016] Figure 5 is a flowchart of a may-constant propagation process of the compilation system of **Figure 4**, in accordance with an embodiment of the disclosure;

[0017] Figure 6 is a textual representation of specialized code replacement for a portion of inter-procedure may-constant propagation of **Figure 2**, in accordance with various embodiments of the disclosure;

[0018] Figure 7a, 7b, 7c are flowcharts of an analysis portion of the may-constant propagation process of **Figure 5**, in accordance with an embodiment of the disclosure; and

[0019] Figure 7d is a flowchart of a code selection portion and a code transformation portion of the may-constant propagation process of **Figure 5**, in accordance with an embodiment of the disclosure.

DETAILED DESCRIPTION

[0020] Although an illustrative implementation of one or more embodiments is provided below, the disclosed systems and/or methods may be implemented using any number of techniques. This disclosure should in no way be limited to the illustrative implementations, drawings, and techniques illustrated below, including the exemplary designs and implementations illustrated and described herein, but may be modified within the scope of the appended claims along with their full scope of equivalents.

[0021] As will be appreciated by one skilled in the art, the present disclosure may be embodied as a system, method or computer program product. Accordingly, the present disclosure may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module," or "system." Furthermore, the present invention may take the form of a computer program product tangibly embodied in any medium of expression with computer usable program code embodied in the medium.

[0022] Computer program code for carrying out operations of the present disclosure may be written in any combination of one or more programming languages, including an object oriented programming language such as Java™, Smalltalk, C++, or the like and conventional procedural programming languages, such as the C or FORTRAN programming language or similar programming languages. Java and all Java-based

trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States, other countries or both. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

[0023] The present disclosure is described below with reference to flowchart illustrations and/or block diagrams of methods, apparatus, systems, and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions.

[0024] These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer program instructions may also be stored in a computer readable medium that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instruction means which implement the function/act specified in the flowchart and/or block diagram block or blocks.

[0025] The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer-implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0026] Turning now to **Figure 3** a block diagram of an exemplary data processing system operable for various embodiments of the disclosure is presented. In this illustrative example, data processing system **300** includes communications fabric **302**, which provides communications between processor unit **304**, memory **306**, persistent storage **308**, communications unit **310**, input/output (I/O) unit **312**, and display **314**.

[0027] Processor unit **304** serves to execute instructions for software that may be loaded into memory **306**. Processor unit **304** may be a set of one or more processors or may be a multi-processor core, depending on the particular implementation. Further, processor unit **304** may be implemented using one or more heterogeneous processor systems in which a main processor is present with secondary processors on a single chip. As another illustrative example, processor unit **304** may be a symmetric multi-processor system containing multiple processors of the same type.

[0028] Memory **306** and persistent storage **308** are examples of storage devices **316**. A storage device is any piece of hardware that is capable of storing information, such as, for example without limitation, data, program code in functional form, and/or other suitable information either on a temporary basis and/or a permanent basis. Memory **306**, in these examples, may be, for example, a random access memory or any other suitable volatile or non-volatile storage device. Persistent storage **308** may take various forms depending on the particular implementation. For example, persistent storage **308** may contain one or more components or devices. For example, persistent storage **308** may be a hard drive, a flash memory, a rewritable optical disk, a rewritable magnetic tape, or some combination of the above. The media used by persistent storage **308** also may be removable. For example, a removable hard drive may be used for persistent storage **308**.

[0029] Communications unit **310,** in these examples, provides for communications with other data processing systems or devices. In these examples, communications unit **310** is a network interface card. Communications unit **310** may provide communications through the use of either or both physical and wireless communications links.

[0030] Input/output unit **312** allows for input and output of data with other devices that may be connected to data processing system **300**. For example, input/output unit **312** may provide a connection for user input through a keyboard, a mouse, and/or some other

suitable input device. Further, input/output unit **312** may send output to a printer. Display **314** provides a mechanism to display information to a user.

[0031] Instructions for the operating system, applications and/or programs may be located in storage devices **316,** which are in communication with processor unit **304** through communications fabric **302.** In these illustrative examples the instructions are in a functional form on persistent storage **308**. These instructions may be loaded into memory **306** for execution by processor unit **304**. The processes of the different embodiments may be performed by processor unit **304** using computer-implemented instructions, which may be located in a memory, such as memory **306**.

[0032] These instructions are referred to as program code, computer usable program code, or computer readable program code that may be read and executed by a processor in processor unit **304**. The program code in the different embodiments may be embodied on different physical or tangible computer readable media, such as memory **306** or persistent storage **308**.

[0033] Program code **318** is located in a functional form on computer readable media **320** that is selectively removable and may be loaded onto or transferred to data processing system **300** for execution by processor unit **304**. Program code **318** and computer readable media **320** form computer program product **322** in these examples. In one example, computer readable media **320** may be in a tangible form, such as, for example, an optical or magnetic disc that is inserted or placed into a drive or other device that is part of persistent storage **308** for transfer onto a storage device, such as a hard drive that is part of persistent storage **308**. In a tangible form, computer readable media **320** also may take the form of a persistent storage, such as a hard drive, a thumb drive, or a flash memory that is connected to data processing system **300**. The tangible form of computer readable media **320** is also referred to as computer recordable storage media. In some instances, computer readable media **320** may not be removable.

[0034] Alternatively, program code **318** may be transferred to data processing system **100** from computer readable media **320** through a communications link to communications unit **310** and/or through a connection to input/output unit **312**. The communications link and/or the connection may be physical or wireless in the illustrative

examples. The computer readable media also may take the form of non-tangible media, such as communications links or wireless transmissions containing the program code.

[0035] In some illustrative embodiments, program code **318** may be downloaded over a network to persistent storage **308** from another device or data processing system for use within data processing system **300**. For instance, program code stored in a computer readable storage medium in a server data processing system may be downloaded over a network from the server to data processing system **300**. The data processing system providing program code **318** may be a server computer, a client computer, or some other device capable of storing and transmitting program code **318.**

[0036] The different components illustrated for data processing system **300** are not meant to provide architectural limitations to the manner in which different embodiments may be implemented. The different illustrative embodiments may be implemented in a data processing system including components in addition to or in place of those illustrated for data processing system **300**. Other components shown in **Figure 3** can be varied from the illustrative examples shown. The different embodiments may be implemented using any hardware device or system capable of executing program code. As one example, the data processing system may include organic components integrated with inorganic components and/or may be comprised entirely of organic components excluding a human being. For example, a storage device may be comprised of an organic semiconductor.

[0037] As another example, a storage device in data processing system **300** may be any hardware apparatus that may store data. Memory **306**, persistent storage **308** and computer readable media **320** are examples of storage devices in a tangible form.

[0038] In another example, a bus system may be used to implement communications fabric **302** and may be comprised of one or more buses, such as a system bus or an input/output bus. Of course, the bus system may be implemented using any suitable type of architecture that provides for a transfer of data between different components or devices attached to the bus system. Additionally, a communications unit may include one or more devices used to transmit and receive data, such as a modem or a network adapter. Further, a memory may be, for example, memory **306** or a cache such as found in an interface and memory controller hub that may be present in communications fabric **302**.

[0039] According to an illustrative embodiment, a computer-implemented process for propagation of may-constants is presented. In one example using data processing system **300** of **Figure 3** the computer-implemented process obtains a source code from storage devices **316**, communications unit **310**, input/output unit **312** or display **314**. Processor unit **304** generates a set of associated data structures from the source code, and a set of may-constant data structures, identifies a candidate code for may-constant propagation to form identified candidate code and updates the set of may-constant data structures contained in memory **306**. Processor unit **304** further selects an identified candidate code using information in the may-constant data structures, including a probability, to form a selected candidate code, identifies a code region associated with the selected candidate code to form an identified code region; and modifying the identified code region including the selected candidate code.

[0040] In an alternative embodiment, program code **318** containing the computer-implemented process may be stored within computer readable media **320** as computer program product **322**. In another illustrative embodiment, the computer-implemented process for propagation of may-constants is presented in an apparatus. The processor unit of the apparatus executes the computer executable program code to direct the apparatus to perform the process.

[0041] With reference to **Figure 4,** a block diagram of a compilation system that may be implemented within the data processing system of **Figure 3**, in accordance with various embodiments of the disclosure is presented. Compilation system **400** is an example of a compilation system for propagation of may-constants comprising compiler **402**, input in the form of source code **404** and output in the form of compiled code **406**.

[0042] Compiler **402** is further composed of a number of components comprising data structures representing call graph **408**, control flow graph **410**, dominator tree **412**, post dominator tree **414**, static single assignment (SSA) graph **416**, node analyzer **418**, universal may-constant table **420**, may-constant def table **422**, may-constant use table **424**, code selector **426** and code modifier **428**. A first set of data structures of call graph **408**, control flow graph **410**, dominator tree **412**, post dominator tree **414**, static single assignment (SSA) graph **416** are created by compiler **402** from the input of source code

404 and further processed during phases of compilation. Representations of these forms are well known to those skilled in the art.

[0043] Additional data structures of universal may-constant table **420**, may-constant def table **422**, may-constant use table **424** are used by node analyzer **418** of compiler **402** when analyzing the data created during generation of the first set of data structures. Code selector **426** further processes the results of node analyzer **418** to determine and identify candidate code. The candidate code may be complete procedures or programs or code portions thereof. Code modifier **428** performs modifications including code versioning or cloning type transformations as needed on the identified candidate code.

[0044] Compiler **402** performs three basic operations with regard to may-constant propagation. An initial data collection and analysis phase is representative of node analyzer **418**. Code selector **426** represents a candidate code selection phase and code modifier **428** represents a code transformation phase. Although shown as separate phases, an implementation may choose to collect the operations into one or more units without change to the overall functional capability.

[0045] Universal may-constant table **420** consists of a set of one or more entries. Each global or universal may-constant table entry contains information including function id, argument position, constant kind, such as a value parameter, reference parameter, or global variable, constant value with offset, probability, and call graph edge index. The content of universal may-constant table **420** represents characteristics including constant information, of a program or code portion and provides a global perspective with respect to variables and values of the program.

[0046] May-constant def table **422** contains a set of one or more entries. Each may-constant def table entry contains information including information of a static single assignment (SSA) def node, constant value with offset, basic block index, and probability. A probability is associated with a constant value. May-constant def table **422** comprises entries describing function parameters of the functions comprising the code portions being analyzed. The def tables typically include assignments.

[0047] May-constant use table **424** also contains a set of one or more entries: Each may-constant use table entry contains information including the information of the static single

assignment (SSA) use node. The entries comprise information regarding arguments for the functions of the code being analyzed. The use tables typically include references.

[0048] While the examples describe use of tables for the additional data structures of universal may-constant table **420**, may-constant def table **422**, may-constant use table **424**, the data structures of the examples are not limited to tabular formats and may be implemented in other forms including arrays, lists, and linked lists.

[0049] The probability values, contained with entries of universal may-constant table **420** and may-constant def table **422,** are based on conditions similar to those involved in branch predictions. Probability assignment may be made on speculation that an execution flow may occur. For example, an initial probability may be 50 percent that one of two branches will be taken. In a next iteration, conditions may have changed to reduce the probability to 25 percent. The resulting calculated probability may then be viewed as the product of the two choices, yielding a probability of slightly more than 10 percent. Probability may be determined based on instrumentation of the code as when profiling code and performing execution tracing. This activity may be fairly precise but the overhead is usually significant and therefore reduces the capability to measure many variables.

[0050] Probability values within the tables provide an additional measure of confidence when selecting a may-constant as a candidate for propagation. For example, may-constants may be ranked according to probability to aid in the selection process. Importance of a candidate may be directly related to the probability value assigned. Therefore code segments containing may-constants with low probability can be ignored while focusing on more important high probability entries.

[0051] Enhancements to compiler **402** enable may-constant propagation to optimize application performance by propagating a high-probability constant through the call graph (inter procedure) and control flow graph (intra procedure). Ignoring possible kills and re-definitions with low probability propagates the high-probability constant. A combination of static analysis, dynamic profiling information, and language and domain specific information may be used. The may-constant information collected guides compiler **402** to perform code transformation or modification including code versioning and cloning to specialize the code for better performance.

[0052] With reference to **Figure 5,** a flowchart of a may-constant propagation process of the compilation system of **Figure 4**, in accordance with one embodiment of the disclosure is presented. Process **500** is an example of a may-constant propagation process as used within compiler **402** of **Figure 4**. Process **500** provides a capability to evaluate unknown constants in a more predictable manner. For example, the capabilities of process **500** are provided through code analysis and speculative processing of a may-constant value according to an assigned probability and modify associated code to enhance the performance of the code including selected may-constant propagation candidates.

[0053] Process **500** starts (step **502**) and obtains a source code (step **504**). Source code is provided as input to the compiler in a form specific to the programming language in use. Having obtained a source code input process **500** generates a set of associated data structures from the source code (step **506**). The set of associated data structures includes structures comprising a call graph that is typically generated for inter procedure propagation; while a control flow graph is used for inter procedure propagation. Additional structures including dominator and post-dominator trees as well as static single assignment (SSA) graph may be created from the source code input.

[0054] Generate a set of may-constant data structures for use with the source code is performed (step **508**). The may-constant data structures as used in the previous illustrative embodiment include a global or universal may-constant table, a may-constant def table and a may-constant use table. The first two data structures include a probability value associated with each may-constant entry. Identify a candidate code for may-constant propagation to form an identified candidate code is performed (step **510**). The candidate code is a portion of the previously obtained input. The portion may be a code segment comprising a function, a loop, a procedure, or a complete program or other block of related code.

[0055] Initially all code may be viewed as candidate code. Candidate code may be further refined to include code containing important variables, with a further focus on key variables. For example, in a loop instance an induction variable may be typically selected. In the previous example of the shaped array, variables associated with the size of shaping of the array would be of importance. Candidate code is determined by a

number of factors including a probability value. Code having a higher probability value associated with a may-constant will be selected as a candidate over code having a lower probability value. A threshold value may be set for probability values to cause all "low" probability entries to be ignored. The selection of higher probability entries allows code modification to focus on fewer more important code segments or portions.

[0056] Process **500** then analyzes the identified candidate code (step **512**). Code analysis is performed to update the set of may-constant data structures (step **514**). Updates include updating of indices, offsets or other positional values that may have changed. Select an identified candidate code using information in the may-constant data structures including probability to form a selected candidate code is performed (step **516**). Information from the may-constant data structures is used to refine the choices of candidates. For example, may-constants selected for processing may be ranked or selected according to probability values.

[0057] Having a selected candidate, identify a code region associated with the selected candidate code to form an identified code region (step **518**). Identify a modification for the identified code region including the selected candidate code is performed (step **520**). Modification choices typically include code versioning and cloning transformations. Modify the identified code region including the selected candidate code is performed (step **522**) with process **500** terminating thereafter (step **524**). Typically modification includes transformation of the code region by code versioning or cloning as appropriate for the scope may-constant and of the code region.

[0058] With reference to **Figure 6**, a textual representation of specialized code replacement for a portion of inter-procedure may-constant propagation of **Figure 2**, in accordance with various embodiments of the disclosure is presented. Using the example of code snippet **200** of **Figure 2** that represents a portion of computer executable program code or instructions that may be used in an inter-procedure may-constant propagation. Statements **202** may be shown to have a low probability of being executed. Statements **202** may be replaced as a result of using process **500** of **Figure 5**.

[0059] The collection and analysis, code selection and modification provides code segment **600** as a replacement for statements **202** of **Figure 2**. The resulting code

segment **600** when combined with the remaining code of code snippet **200** of **Figure 2** comprises fewer statements and typically more predictable results.

[0060] With reference to **Figures 7a, 7b**, and **7c**, flowcharts of an analysis portion of the may-constant propagation process of **Figure 5**, in accordance with an embodiment of the disclosure are presented. Process **700** represents a more detailed view of a code collection and analysis process within the overview provided in process **500** of **Figure 5**.

[0061] Process **700** starts (step **702**) and creates a set of data structures (step **704**). The data structures typically represent structures used to collect information from the source code supplied as input. A set of structures including graphs and may-constant particular tables are created as in the examples of **Figure 4**. In this example, traverse the call graph in a top down order is performed (step **706**). Traverse function entry PHI definitions is performed (step **708**). Identify a variable definition on entry from a function entry PHI node to form an identified function entry variable (step **710**). The PHI function is a set of one or more statements of a function, generated by the compiler, specifying a definition of a variable dependent on flow control at a location. Initialization of the values of parameters and global variables is performed on entry to the function. Upward exposed variables such as value parameters, reference parameters and global variables can be identified by function entry PHI nodes.

[0062] Process **700** determines, using the universal may-constant table, whether the identified function entry variable is a def candidate for may-constant propagation (step **712**). The information in the universal may-constant table is used to analyze incoming call edges. When a determination is made that the identified function is a def candidate for may-constant propagation, a "yes" is obtained. When a determination is made that the identified function is not a def candidate for may-constant propagation, a "no" is obtained. When a "no" is obtained from step **712**, process **700** skips ahead to step **722**.

[0063] When a "yes" is obtained in step **712**, add an entry with a combined probability to the may-constant def table (step **714**). Determine whether a static single assignment (SSA) use occurs for the PHI def node of the identified function entry variable is performed (step **716**). When no static single assignment use occurs a "no" result is obtained. When a static single assignment use occurs a "yes" result is obtained. When a "no" result is obtained in step **716**, process **700** skips to step **722**.

[0064] When a "yes" is obtained in step **716**, add an entry to the may-constant use table for each static single assignment use for the identified function entry variable is performed (step **718**). Determine whether more function entry PHI definitions exist is performed (step **720**). When a determination is made that more function entry PHI definitions exist a "yes" result is obtained. When a determination is made that no more function entry PHI definitions exist a "no" result is obtained.

[0065] When a "yes" is obtained in step **720**, process **700** loops back to perform step **710**. When a "no" is obtained in step **720**, traverse a dominator tree of a control flow graph in a top down direction is performed (step **722**). The dominator tree is one of the data structures that were initialized when a set of data structures was created in step **704**. The dominator tree comprises block entries for each PHI definition.

[0066] With reference to **Figure 7c**, process **700** continues to traverse a basic block entry PHI definition (step **724**). During the traversal an analysis of all reaching definitions of all PHI uses of the PHI definition is performed to determine whether a PHI definition for a variable on basic block entry to the basic block is a candidate for a may-constant propagation (step **726**). When a determination is made that the PHI definition for a variable on basic block entry to the basic block is a candidate for a may-constant propagation a "yes" result is obtained. When a determination is made that the PHI definition for a variable on basic block entry to the basic block is not a candidate for a may-constant propagation a "no" result is obtained.

[0067] When a "no" result is obtained in step **726**, process **700** skips to step **740**. When a "yes" is obtained in step **726**, add an entry for the basic block entry variable with a combined probability to the may-constant def table (step **728**). Determine whether a static single assignment (SSA) use occurs for the PHI def node of the basic block entry variable is performed (step **730**). When a determination is made that no static single assignment use occurs a "no" result is obtained. When a determination is made that a static single assignment use occurs a "yes" result is obtained. When a "no" result is obtained in step **730**, process **700** skips to step **740**. When a "yes" is obtained in step **730**, determine whether the static single assignment (SSA) use is a PHI function use occurs (step **732**). When a determination is made that the static single assignment (SSA)

use is a PHI function use a "yes" result is obtained. When a determination is made that the static single assignment (SSA) use is not a PHI function use, a "no" result occurs.

[0068] When a "no" result is obtained in step **732** process **700** skips to step **740**. When a "yes" result occurs in step **732**, add an entry to the may-constant use table for the PHI function (step **734**). Process **700** determines whether more basic block entry PHI definitions exist (step **736**). When a determination is made that more basic block entry PHI definitions exist, a "yes" result is obtained. When a determination is made that no more basic block entry PHI definitions exist, a "no" result is obtained. When a "no" result is obtained in step **736**, process **700** loops back to perform step **726**. When a "yes" is obtained in step **736**, traverse the statement trees in the basic block is performed (step **738**).

[0069] With reference to **Figure 7c**, process **700** determines whether a statement tree assigns a constant or a list of constants to a variable (step **740**). When a determination is made that a statement tree assigns a constant or a list of constants to a variable, a "yes" result is obtained. When a determination is made that a statement tree does not assign a constant or a list of constants to a variable, a "no" result is obtained.

[0070] When a "no" result is obtained in step **740**, process **700** skips to step **752**. When a "yes" is obtained in step **740**, process **700** adds an entry with a value of the constant with 100 percent probability to the may-constant def table (step **742**). Determine whether the statement tree contains a function call is performed (step **746**). When the statement tree contains a function call a "yes" result is obtained. When the statement tree does not contain a function call, a "no" result is obtained. When a "no" result is obtained in step **746**, process **700** skips to step **752**.

[0071] When a "yes" result is obtained in step **740**, determine whether static single assignment (SSA) use occurs for a parameter or a global variable at a call site is performed (step **748**). When a determination is made that a static single assignment (SSA) use occurs for a parameter or a global variable at a call site, a "yes" result is obtained. When a determination is made that a static single assignment (SSA) use does not occur for a parameter or a global variable at a call site, a "no" result is obtained. When a "no" result is obtained in step **748**, process **700** skips to step **752**. When a "yes"

result is obtained in step **748**, process **700** adds an entry with a value of the constant or the may-constant of a reaching def into the universal may-constant table (step **750**).

[0072] Determine whether there are more statements in a basic block is performed (step **752**). When a determination is made that there are more statements in a basic block, a "yes" result is obtained. When a determination is made that there are no more statements in a basic block, a "no" result is obtained. When a "yes" result is obtained in step **752**, process **700** loops back to step **740** to process the remaining statements. When a "no" result is obtained in step **752**, determine whether there are more basic blocks is performed (step **754**). When a determination is made that there are more basic blocks, a "yes" result is obtained. When a determination is made that there are no more basic blocks, a "no" result is obtained. When a "yes" is obtained in step **754**, process **700** loops back to step **724**.

[0073] When a "no" is obtained in step **754**, determine whether there are more call graph nodes in the call graph is performed (step **756**). When a determination is made that there are more call graph nodes in the call graph, a "yes" result is obtained. When a determination is made that there are no more call graph nodes in the call graph, a "no" result is obtained. When a "yes" result is obtained in step **756**, process **700** loops back to step **708**. When a "no" is obtained in step **756**, traverse the may–constant use table is performed (step **758**). Identify a candidate set of code to form an identified candidate is performed (step **760**). The identified candidate is a portion of code including use of the may-constants.

[0074] With reference to **Figure 7d**, a flowchart of a code selection portion and a code transformation portion of the may-constant propagation process of **Figure 5**, in accordance with an embodiment of the disclosure is presented. **Figure 7d** is also a continuation of the process **700** of **Figure 7c**.

[0075] Candidates as previously described include variables or constants of interest or deemed to be important. Continuing from step **760** of **Figure 7c**, process **700** performs identify a code region of the identified candidate to form an identified code region (step **762**). The code region is associated with or contains the identified candidate code. The region may comprise a function, a procedure or a program or portions thereof. Having identified the code region, determine whether to apply code versioning (step **764**). Code

versioning is typically applied to regions such as a version of a loop. When a determination is made to apply code versioning a "yes" is obtained. When a determination is made to not apply code versioning a "no" result is obtained. When a "no" result is obtained in step **764**, process **700** skips to step **768**. When a "yes" result is obtained in step **764**, apply code versioning to the identified candidate and identified code region (step **766**). Determine whether to apply cloning transformation is performed (step **768**).

[0076] When a determination is made to apply cloning transformation, a "yes" result is obtained. When a determination is made to not apply cloning transformation, a "no" result is obtained. When a "no" is obtained in step **768**, process **700** terminates (step **772**). When a "yes" result is obtained, apply cloning transformation to the identified candidate and identified code region is performed (step **770**) with process **700** terminating thereafter (step **772**). Cloning transformation is typically applied to a whole procedure.

[0077] Operations **704** through **756** are representative of data structure creation and node analysis elements **408** through **418** of **Figure 4**. Operations **760** and **762** are representative of code selector **426** of **Figure 4**. Operations **764** through **770** are representative of code modifier **428** of **Figure 4**.

[0078] An illustrative embodiment thus provides a capability in the form of a computer-implemented process for may-constant propagation that uses information in may-constant data structures including probability information to aid in selection of candidates for may-constant propagation. The computer-implemented process obtains a source code, and generates a set of associated data structures from the source code and a set of may-constant data structures. The computer-implemented process identifies a candidate code for may-constant propagation to form identified candidate code, updates the set of may-constant data structures, and selects an identified candidate code using information in the may-constant data structures, including probability, to form a selected candidate code. The computer-implemented process further identifies a code region associated with the selected candidate code to form an identified code region and modifies the identified code region including the selected candidate code.

[0079] The flowchart and block diagrams in the figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing a specified logical function. It should also be noted that, in some alternative implementations, the functions noted in the block might occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

[0080] The corresponding structures, materials, acts, and equivalents of all means or step plus function elements in the claims below are intended to include any structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The description of the present invention has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the invention. The embodiment was chosen and described in order to best explain the principles of the invention and the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.

[00100]The invention can take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment containing both hardware and software elements. In a preferred embodiment, the invention is implemented in software, which includes but is not limited to firmware, resident software, microcode, and other software media that may be recognized by one skilled in the art.

[00101]It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media, such as digital and analog communications links, wired or wireless communications links using transmission forms, such as, for example, radio frequency and light wave transmissions. The computer readable media may take the form of coded formats that are decoded for actual use in a particular data processing system.

[00102]A data processing system suitable for storing and/or executing program code will include at least one processor coupled directly or indirectly to memory elements through a system bus. The memory elements can include local memory employed during actual execution of the program code, bulk storage, and cache memories which provide temporary storage of at least some program code in order to reduce the number of times code must be retrieved from bulk storage during execution.

[00103]Input/output or I/O devices (including but not limited to keyboards, displays, pointing devices, etc.) can be coupled to the system either directly or through intervening I/O controllers.

[00104]Network adapters may also be coupled to the system to enable the data processing system to become coupled to other data processing systems or remote printers or storage devices through intervening private or public networks. Modems, cable modems, and Ethernet cards are just a few of the currently available types of network adapters.

[00105]The description of the present invention has been presented for purposes of illustration and description, and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others

of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.

CLAIMS:

What is claimed is:

1. A computer-implemented process for may-constant propagation, the computer-implemented process comprising:

obtaining a source code;

generating a set of associated data structures from the source code;

generating a set of may-constant data structures;

identifying a candidate code for may-constant propagation to form identified candidate code;

updating the set of may-constant data structures;

selecting an identified candidate code using information in the may-constant data structures, including probability, to form a selected candidate code;

identifying a code region associated with the selected candidate code to form an identified code region; and

modifying the identified code region including the selected candidate code.

2. The computer-implemented process of claim 1 wherein identifying a candidate code for may-constant propagation to form identified candidate code further comprises:

traversing the may-constant use table.

3. The computer-implemented process of claim 1 wherein modifying the identified code region including the selected candidate code further comprises:

determining whether to apply code versioning;

responsive to a determination to apply code versioning, applying code versioning to the identified candidate and identified code region;

determining whether to apply cloning transformation; and

responsive to a determination to apply cloning transformation, applying cloning transformation to the identified candidate and identified code region.

4. The computer-implemented process of claim 1wherein identifying a candidate code for may-constant propagation to form identified candidate code further comprises:

traversing a call graph in a top down direction;

traversing function entry PHI definitions;

identifying a variable definition on entry to a function from a function entry PHI def node to form an identified function entry variable; and

determining, using the universal-may-constant table, whether the identified function entry variable is a def candidate for a may-constant propagation.

5. The computer-implemented process of claim 4 wherein updating the set of may-constant data structures further comprises:

responsive to a determination that the identified function entry variable is a def candidate for a may-constant propagation, adding an entry with a combined probability to the may-constant def table for the identified function entry variable;

determining whether a static single assignment use occurs for the PHI def node of the identified function entry variable; and

responsive to a determination that a static single assignment use occurs, adding an entry to the may-constant use table for each static single assignment use for the identified function entry variable.

6. The computer-implemented process of claim 5, wherein updating the set of may-constant data structures further comprises:

determining whether more function entry PHI definitions exist;

responsive to a determination that no more function entry PHI definitions exist, traversing a dominator tree of a control flow graph in a top down direction;

traversing a basic block entry PHI definition;

determining whether a PHI definition for a variable on entry to the basic block is a candidate for a may-constant propagation;

responsive to a determination that a PHI definition for a variable on entry to the basic block is a candidate for a may-constant propagation, adding an entry for the

variable on entry to the basic block, with a combined probability, to the may-constant def table;

determining whether a static single assignment use occurs for the PHI def node of the variable on entry to the basic block; and

responsive to a determination that a static single assignment use occurs, determining whether the static single assignment use is a PHI function use;

responsive to a determination that a PHI function use occurs, adding an entry to the may-constant use table for the PHI function;

determining whether more basic block entry PHI definitions exist; and

responsive to a determination that no more basic block entry PHI definitions exist; traversing the statement trees in the basic block.

7. The computer-implemented process of claim 6, wherein updating the set of may-constant data structures further comprises:

determining whether a statement tree, of the created data structures, assigns a constant or a list of constants to a variable;

responsive to a determination that the statement tree assigns a constant or a list of constants to a variable, adding an entry with 100 percent probability to the may-constant def table;

determining whether the statement tree contains a function call;

responsive to a determination that the statement tree contains a function call, determining whether static single assignment use occurs for a parameter or a global variable at a call site;

responsive to a determination that a static single assignment use occurs for a parameter or a global variable at the call site, adding an entry with a value of the constant or may–constant of a reaching def into the universal may-constant table; and

responsive to a determination that there are no more statements in the basic block, no more basic blocks and no more call graph nodes in the call graph, traversing the may-constant use table.

8. A computer program product for may-constant propagation, the computer program product comprising:

a computer recordable-type media containing computer executable program code stored thereon, the computer executable program code comprising:

computer executable program code for obtaining a source code;

computer executable program code for generating a set of associated data structures from the source code;

computer executable program code for generating a set of may-constant data structures;

computer executable program code for identifying a candidate code for may-constant propagation to form identified candidate code;

computer executable program code for updating the set of may-constant data structures;

computer executable program code for selecting an identified candidate code using information in the may-constant data structures, including probability, to form a selected candidate code;

computer executable program code for identifying a code region associated with the selected candidate code to form an identified code region; and

computer executable program code for modifying the identified code region including the selected candidate code.

9. The computer program product of claim 8 wherein computer executable program code for identifying a candidate code for may-constant propagation to form identified candidate code further comprises:

computer executable program code for traversing the may-constant use table.

10. The computer program product of claim 8 wherein computer executable program code for modifying the identified code region including the selected candidate code further comprises:

computer executable program code for determining whether to apply code versioning;

computer executable program code responsive to a determination to apply code versioning, for applying code versioning to the identified candidate and identified code region;

computer executable program code for determining whether to apply cloning transformation; and

computer executable program code responsive to a determination to apply cloning transformation, for applying cloning transformation to the identified candidate and identified code region.

11. The computer program product of claim 8wherein computer executable program code for identifying a candidate code for may-constant propagation to form identified candidate code further comprises:

computer executable program code for traversing a call graph in a top down direction;

computer executable program code for traversing function entry PHI definitions;

computer executable program code for identifying a variable definition on entry to a function from a function entry PHI def node to form an identified function entry variable; and

computer executable program code for determining, using the universal may-constant table, whether the identified function entry variable is a def candidate for a may-constant propagation.

12. The computer program product of claim 11 wherein computer executable program code for updating the set of may-constant data structures further comprises:

computer executable program code responsive to a determination that the identified function entry variable is a def candidate for a may-constant propagation, for adding an entry with a combined probability to the may-constant def table for the identified function entry variable;

computer executable program code for determining whether a static single assignment use occurs for the PHI def node of the identified function entry variable; and

computer executable program code responsive to a determination that a static single assignment use occurs, for adding an entry to the may-constant use table for each static single assignment use for the identified function entry variable.

13. The computer program product of claim 12, wherein computer executable program code for updating the set of may-constant data structures further comprises:

computer executable program code for determining whether more function entry PHI definitions exist;

computer executable program code responsive to a determination that no more function entry PHI definitions exist, for traversing a dominator tree of a control flow graph in a top down direction;

computer executable program code for traversing a basic block entry PHI definition;

computer executable program code for determining whether a PHI definition for a variable on entry to the basic block is a candidate for a may-constant propagation;

computer executable program code responsive to a determination that a PHI definition for a variable on entry to the basic block is a candidate for a may-constant propagation, for adding an entry for the variable on entry to the basic block, with a combined probability, to the may-constant def table;

computer executable program code for determining whether a static single assignment use occurs for the PHI def node of the variable on entry to the basic block;

computer executable program code responsive to a determination that a static single assignment use occurs, for determining whether the static single assignment use is a PHI function use;

computer executable program code responsive to a determination that a PHI function use occurs, for adding an entry to the may-constant use table for the PHI function;

computer executable program code for determining whether more basic block entry PHI definitions exist; and

computer executable program code responsive to a determination that no more basic block entry PHI definitions exist; for traversing the statement trees in the basic block.

14. The computer program product of claim 13, wherein computer executable program code for updating the set of may-constant data structures further comprises:

computer executable program code for determining whether a statement tree, of the created data structures, assigns a constant or a list of constants to a variable;

computer executable program code responsive to a determination that the statement tree assigns a constant or a list of constants to a variable, for adding an entry with 100 percent probability to the may-constant def table;

computer executable program code for determining whether the statement tree contains a function call;

computer executable program code responsive to a determination that the statement tree contains a function call, for determining whether static single assignment use occurs for a parameter or a global variable at a call site;

computer executable program code responsive to a determination that a static single assignment use occurs for a parameter or a global variable at the call site, for adding an entry with a value of the constant or may–constant of a reaching def into the universal may-constant table; and

computer executable program code responsive to a determination that there are no more statements in the basic block, no more basic blocks and no more call graph nodes in the call graph, for traversing the may-constant use table.

15. An apparatus for may-constant propagation, the apparatus comprising:

a communications fabric;

a memory connected to the communications fabric, wherein the memory contains computer executable program code;

a communications unit connected to the communications fabric;

an input/output unit connected to the communications fabric;

a display connected to the communications fabric; and

a processor unit connected to the communications fabric, wherein the processor unit executes the computer executable program code to direct the apparatus to:

obtain a source code;

generate a set of associated data structures from the source code;

generate a set of may-constant data structures;

identify a candidate code for may-constant propagation to form identified candidate code;

update the set of may-constant data structures;

select an identified candidate code using information in the may-constant data structures, including probability, to form a selected candidate code;

identify a code region associated with the selected candidate code to form an identified code region; and

modify the identified code region including the selected candidate code.

16. The apparatus of claim 15 wherein the processor unit executes the computer executable program code to modify the identified code region including the selected candidate code further comprises to direct the apparatus to:

determine whether to apply code versioning;

responsive to a determination to apply code versioning, apply code versioning to the identified candidate and identified code region;

determine whether to apply cloning transformation; and

responsive to a determination to apply cloning transformation, apply cloning transformation to the identified candidate and identified code region.

17. The apparatus of claim 15 wherein the processor unit executes the computer executable program code to identify a candidate code for may-constant propagation to form identified candidate code further comprises to direct the apparatus to:

traverse a call graph in a top down direction;

traverse function entry PHI definitions;

identify a variable definition on entry to a function from a function entry PHI def node to form an identified function entry variable; and

determine, using the universal-may-constant table, whether the identified function entry variable is a def candidate for a may-constant propagation.

18. The apparatus of claim 17 wherein the processor unit executes the computer executable program code to update the set of may-constant data structures further comprises to direct the apparatus to:

responsive to a determination that the identified function entry variable is a def candidate for a may-constant propagation, add an entry with a combined probability to the may-constant def table for the identified function entry variable;

determine whether a static single assignment use occurs for the PHI def node of the identified function entry variable; and

responsive to a determination that a static single assignment use occurs, add an entry to the may-constant use table for each static single assignment use for the identified function entry variable.

19. The apparatus of claim 18, wherein the processor unit executes the computer executable program code to update the set of may-constant data structures further comprises to direct the apparatus to:

determine whether more function entry PHI definitions exist;

responsive to a determination that no more function entry PHI definitions exist, traverse a dominator tree of a control flow graph in a top down direction;

traverse a basic block entry PHI definition;

determine whether a PHI definition for a variable on entry to the basic block is a candidate for a may-constant propagation;

responsive to a determination that a PHI definition for a variable on entry to the basic block is a candidate for a may-constant propagation, add an entry for the variable on entry to the basic block, with a combined probability, to the may-constant def table;

determine whether a static single assignment use occurs for the PHI def node of the variable on entry to the basic block; and

responsive to a determination that a static single assignment use occurs, determine whether the static single assignment use is a PHI function use;

responsive to a determination that a PHI function use occurs, add an entry to the may-constant use table for the PHI function;

determine whether more basic block entry PHI definitions exist; and

responsive to a determination that no more basic block entry PHI definitions exist; traverse the statement trees in the basic block.

20. The computer-implemented process of claim 19, wherein the processor unit executes the computer executable program code to update the set of may-constant data structures further comprises to direct the apparatus to:

determine whether a statement tree, of the created data structures, assigns a constant or a list of constants to a variable;

responsive to a determination that the statement tree assigns a constant or a list of constants to a variable, add an entry with 100 percent probability to the may-constant def table;

determine whether the statement tree contains a function call;

responsive to a determination that the statement tree contains a function call, determine whether static single assignment use occurs for a parameter or a global variable at a call site;

responsive to a determination that a static single assignment use occurs for a parameter or a global variable at the call site, add an entry with a value of the constant or may–constant of a reaching def into the universal may-constant table; and

responsive to a determination that there are no more statements in the basic block, no more basic blocks and no more call graph nodes in the call graph, traversing the may-constant use table.

FIG. 1

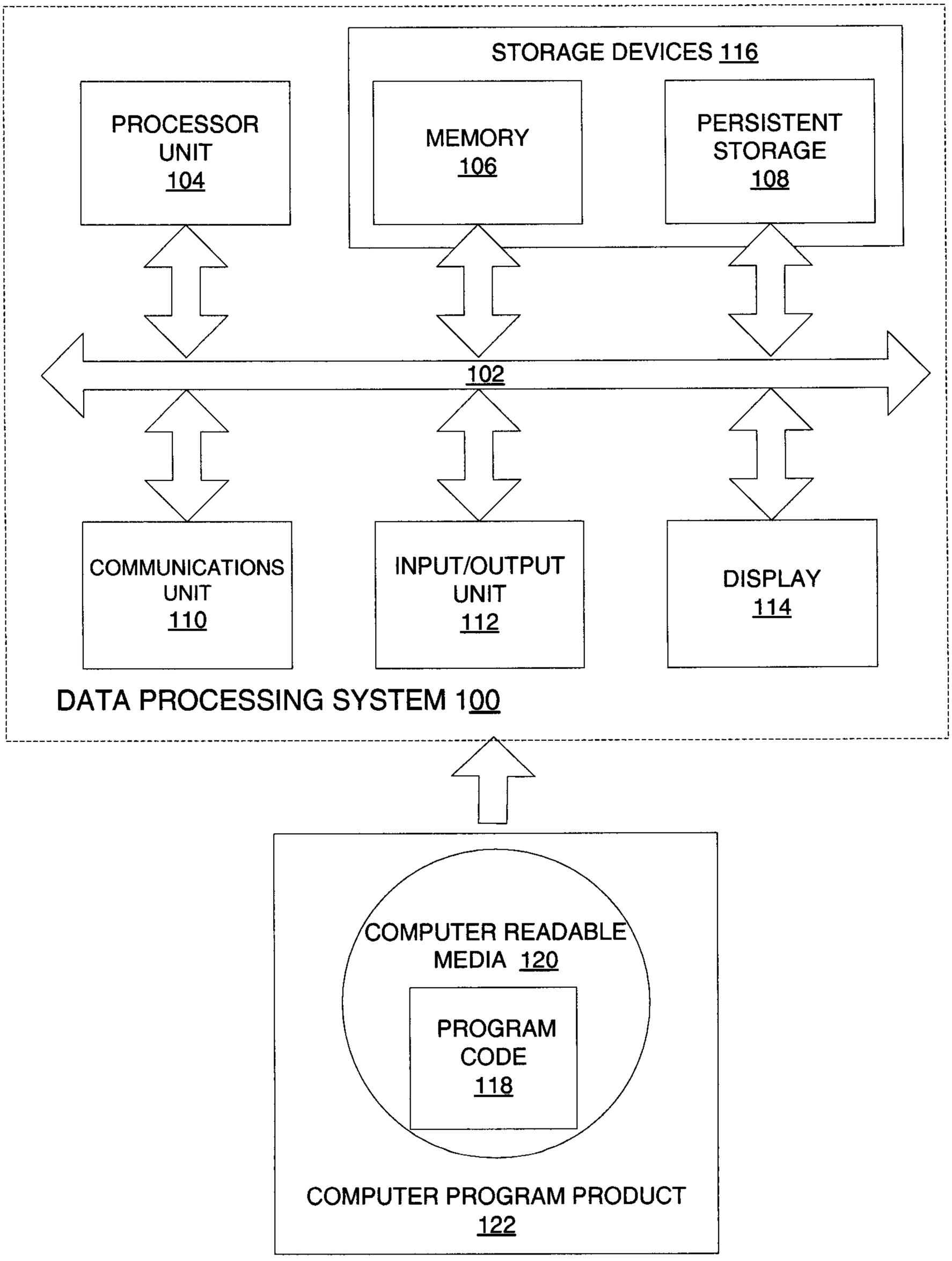
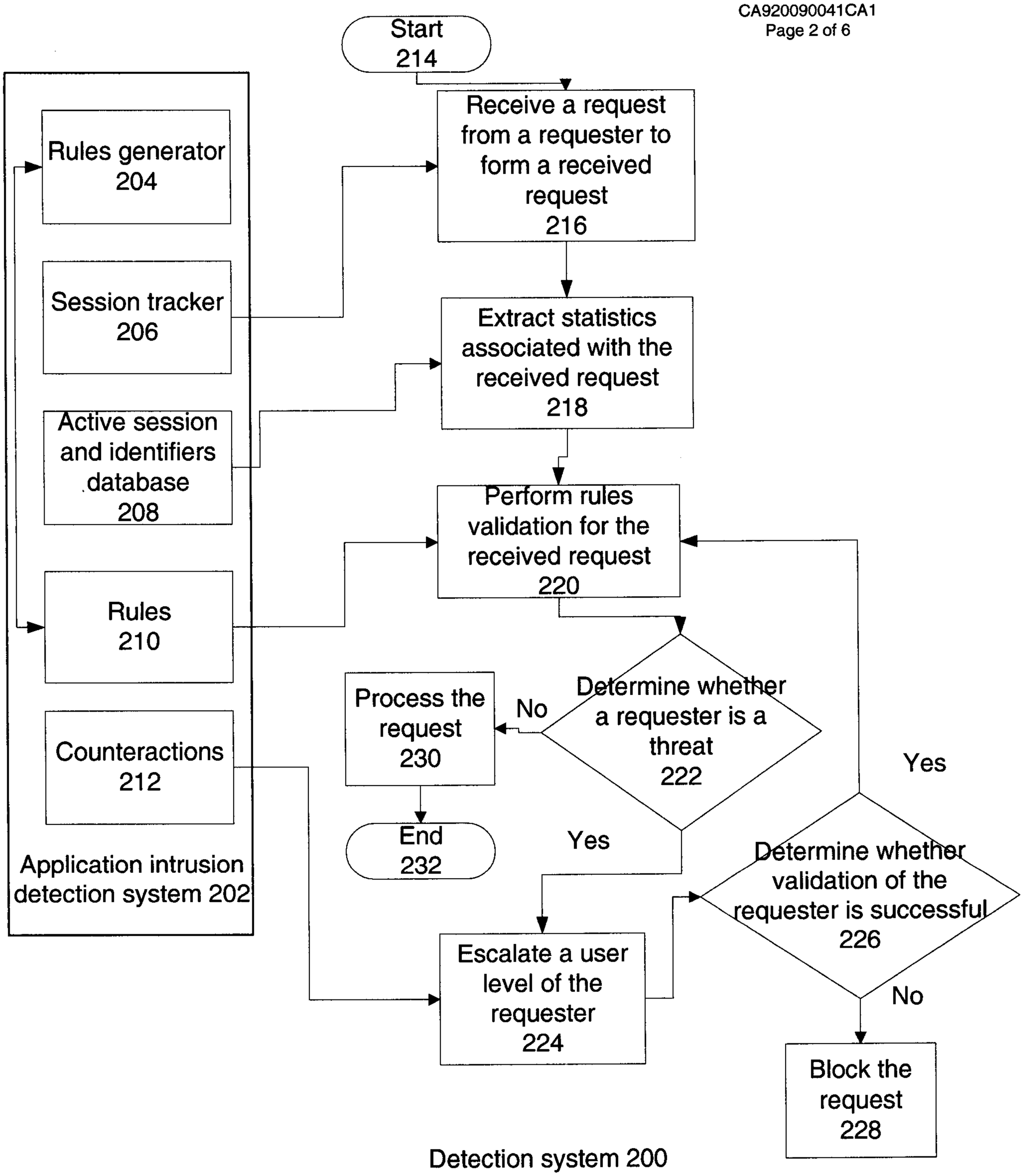

STORAGE DEVICES 116
PROCESSOR UNIT 104
MEMORY 106
PERSISTENT STORAGE 108
102
COMMUNICATIONS UNIT 110
INPUT/OUTPUT UNIT 112
DISPLAY 114
DATA PROCESSING SYSTEM 100
COMPUTER READABLE MEDIA 120
PROGRAM CODE 118
COMPUTER PROGRAM PRODUCT 122

FIG. 2



Detection system 200

FIG. 3

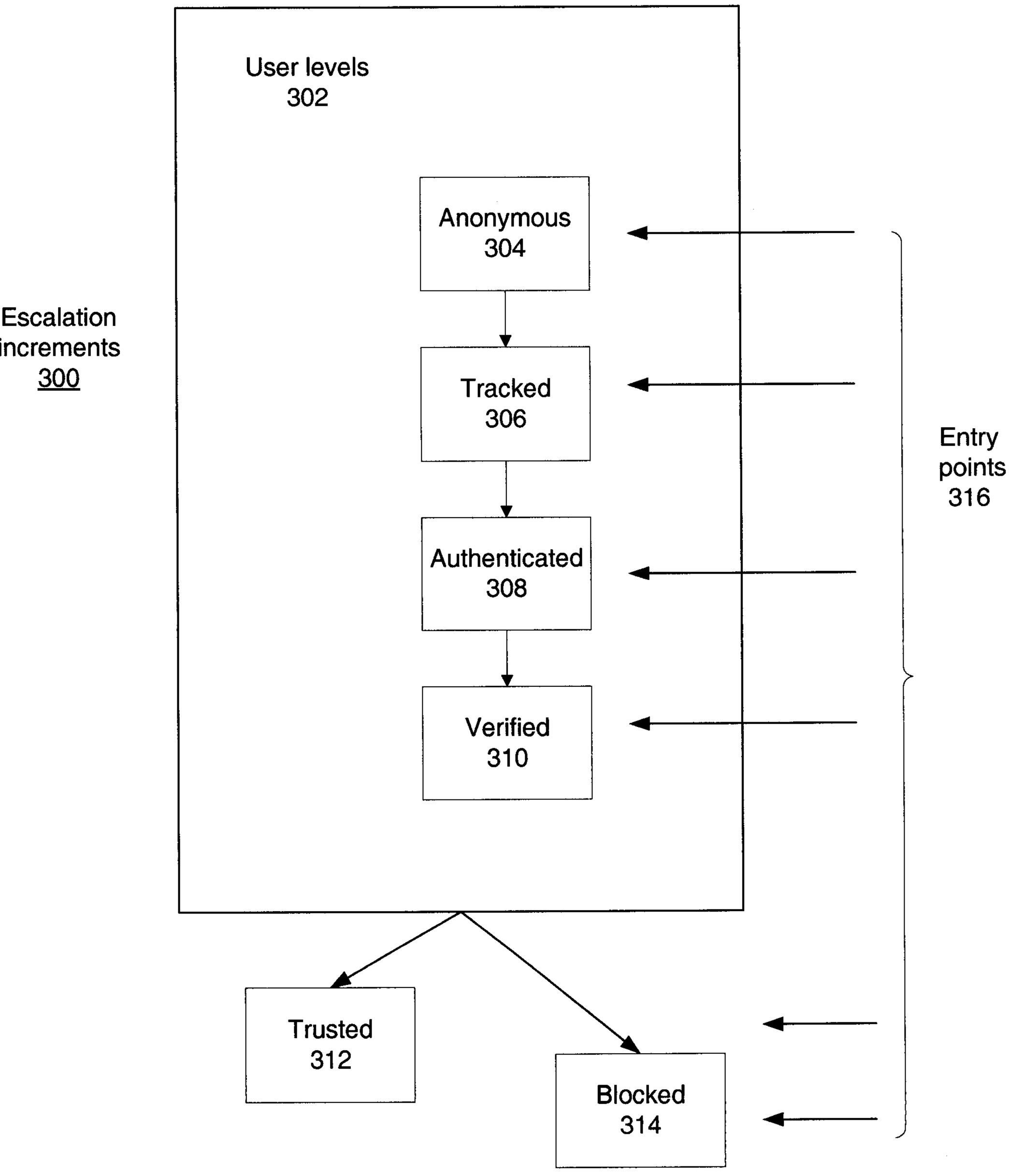

User levels
302
Escalation increments
300
Anonymous
304
Tracked
306
Authenticated
308
Verified
310
Trusted
312
Blocked
314
Entry points
316

FIG. 4

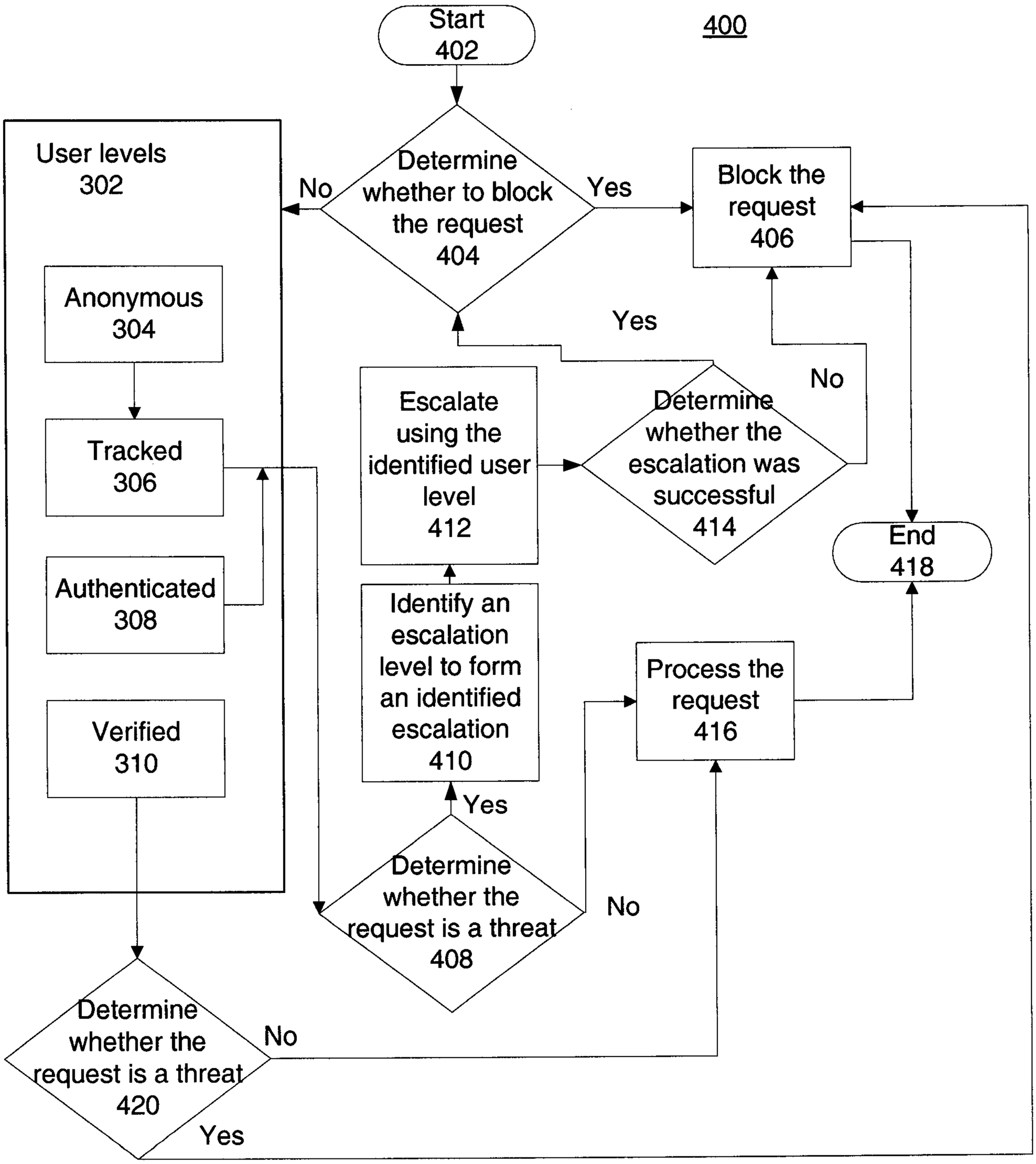

400
Start
402
User levels
302
Anonymous
304
Tracked
306
Authenticated
308
Verified
310
Determine whether to block the request
404
No
Yes
Block the request
406
Yes
No
Escalate using the identified user level
412
Determine whether the escalation was successful
414
End
418
Identify an escalation level to form an identified escalation
410
Yes
Determine whether the request is a threat
408
No
Process the request
416
Determine whether the request is a threat
420
No
Yes

FIG. 5a

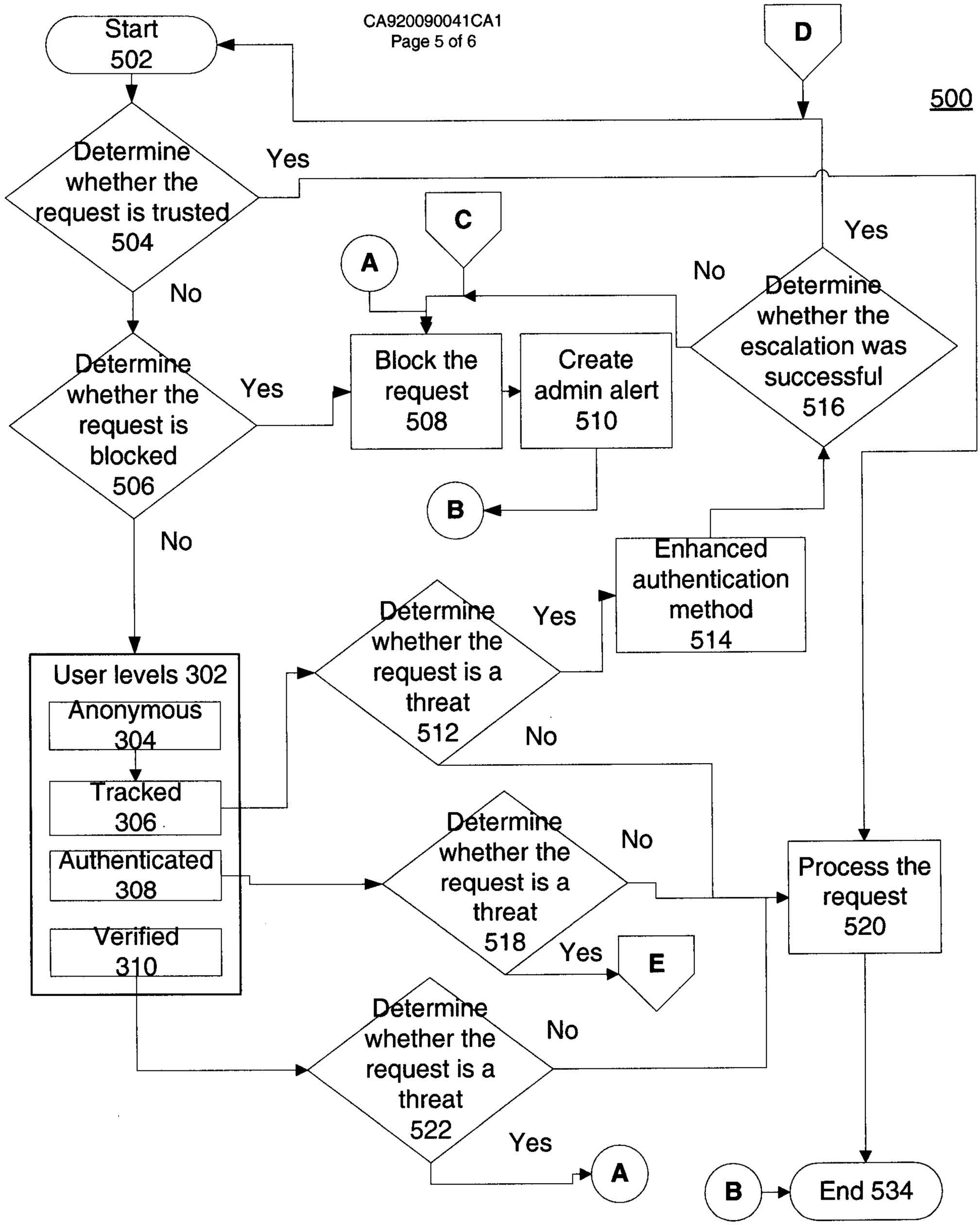
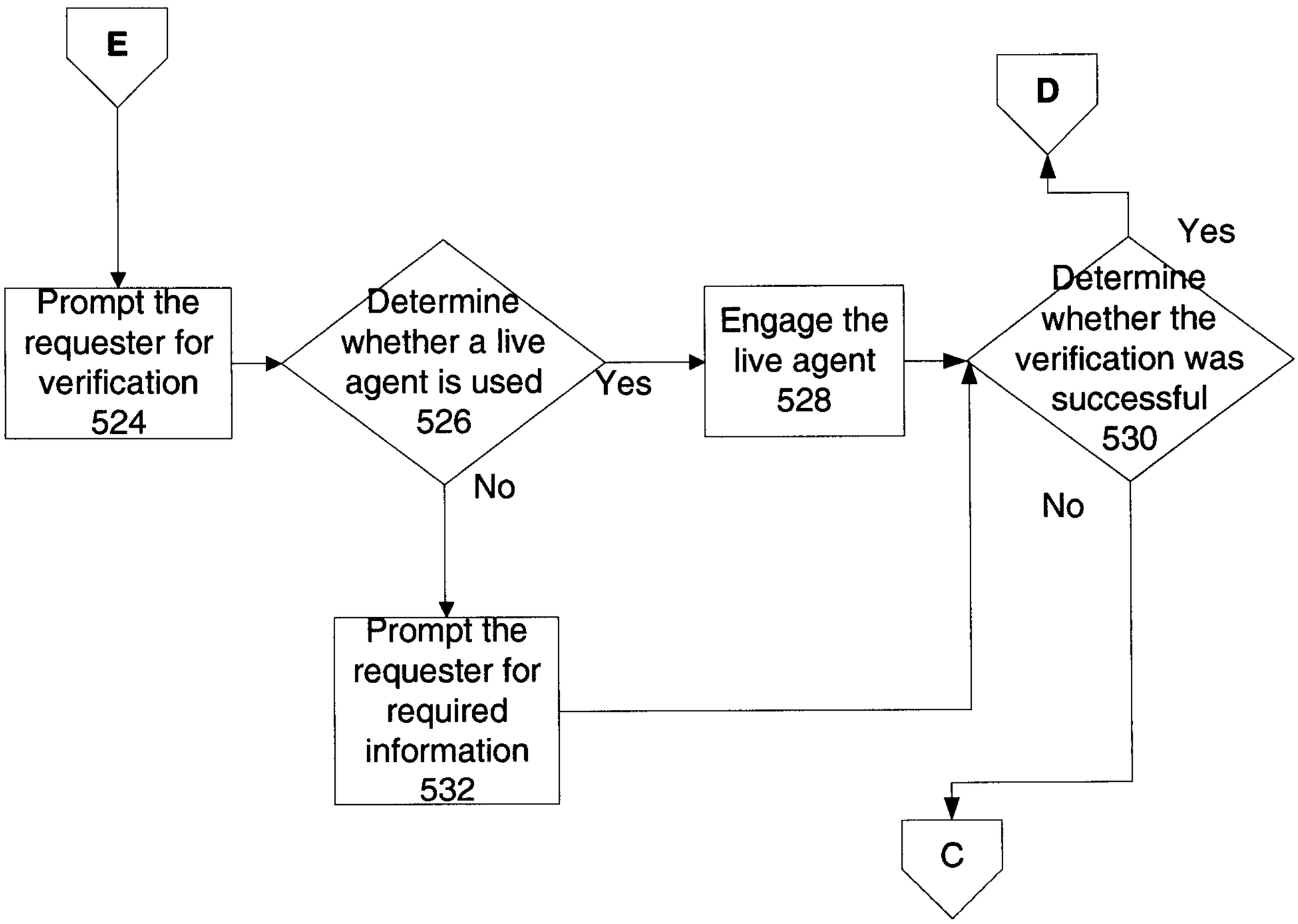
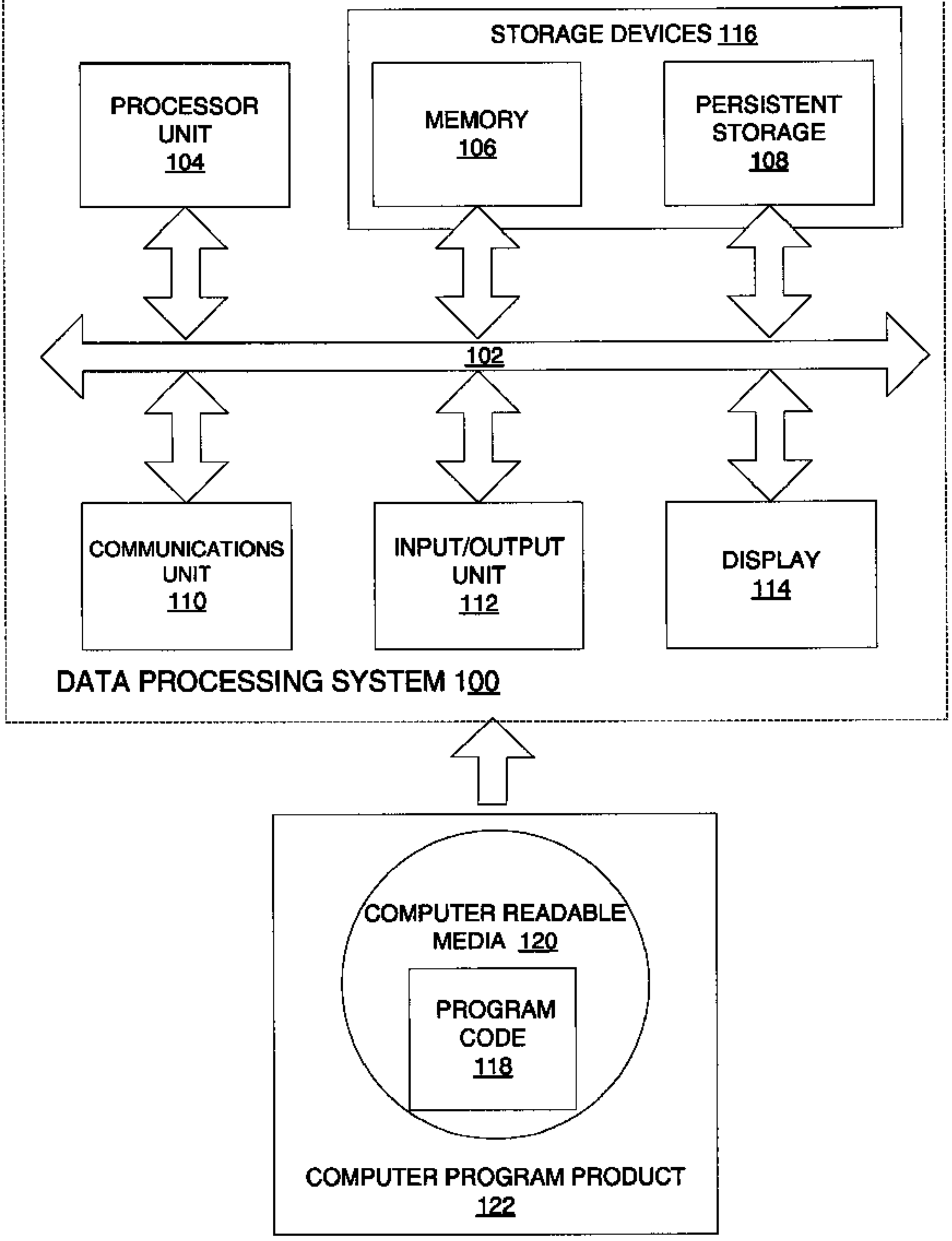

500
Start
502
Determine whether the request is trusted
504
Yes
No
Determine whether the request is blocked
506
Yes
No
Block the request
508
Create admin alert
510
A
C
B
D
E
Determine whether the escalation was successful
516
Yes
No
Enhanced authentication method
514
Determine whether the request is a threat
512
Yes
No
User levels 302
Anonymous
304
Tracked
306
Authenticated
308
Verified
310
Determine whether the request is a threat
518
No
Yes
Determine whether the request is a threat
522
No
Yes
Process the request
520
B
End 534

FIG. 5b


E
Prompt the requester for verification 524
Determine whether a live agent is used 526
Yes
No
Engage the live agent 528
D
Yes
Determine whether the verification was successful 530
No
Prompt the requester for required information 532
C


STORAGE DEVICES 116
PROCESSOR UNIT 104
MEMORY 106
PERSISTENT STORAGE 108
102
COMMUNICATIONS UNIT 110
INPUT/OUTPUT UNIT 112
DISPLAY 114
DATA PROCESSING SYSTEM 100
COMPUTER READABLE MEDIA 120
PROGRAM CODE 118
COMPUTER PROGRAM PRODUCT 122