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

(54) Title: METHODS AND APPARATUS FOR COMPILING COMPUTER PROGRAMS USING PARTIAL FUNCTION INLINING
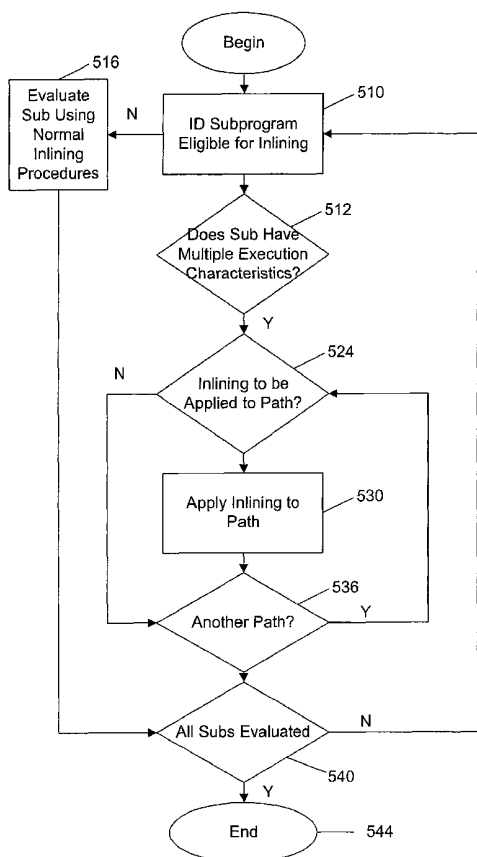
(57) Abstract: A method and system makes inlining decisions that are efficient for subprograms that have significantly varying execution times over a range of variables or execution paths. A subprogram of a computer program is identified and certain execution paths of the subprogram are selectively inlined. The subprogram may be identified based on execution characteristics of the subprogram. The selective inlining of the execution paths may be based on execution characteristics of the paths. The paths may be selectively inlined based on an inline indication associated with an execution path, where the inline indication may be an inline directive. The inline directive may be included as part of a program comment statement. A compiler makes determinations whether to inline a specific execution path of a subprogram by evaluating certain information supplied in conjunction with the path. By supplying information in association with the subprogram path, the compiler may more easily determine the various execution characteristics of the execution paths and may inline or not based on the execution characteristic indication associated with the subprogram execution path.

WO 02/10909 A2

METHODS AND APPARATUS FOR COMPILING
COMPUTER PROGRAMS USING PARTIAL FUNCTION INLINING

I.   FIELD OF THE INVENTION

The present invention relates to optimizing computer code during

compilation, and more particularly, to partial function inlining during compilation.

5   II.   BACKGROUND OF THE INVENTION

Modern computer program languages are based on modular design

models, where computer code is written in small, modular units or subprograms

that define certain objects or functions. These subprograms may be called (i.e.

invoked) wherever desired in the program by a simple reference to the

10   subprogram. While modular program design is an effective programming

technique, additional overhead may be introduced during the execution stage

after the program is compiled. An overhead penalty is introduced when a

subprogram is called frequently but the execution time of the subprogram is

small relative to the time required to call the subprogram.

15   Most compilers are equipped with various optimization routines that

determine how to represent the original source code in an efficient executable

form, such as reducing the size and required execution time of various calls to

the subprogram. One well known optimization technique is referred to as

inlining. Inlining techniques replace the subprogram calls at the various locations

20   in the computer program with the lines of code that define the subprogram.

Inlining is typically performed when a subprogram is called many times in a

program and when the execution time of the subprogram is small compared with the time necessary to set-up for and call the subprogram.

Inlining provides performance improvements for various reasons. First, the subprogram linkage is removed, including the code to save and restore registers,

5   allocate stack space, and the branch itself. Second, the code surrounding the call site can be improved, since the call itself, which may be a barrier for some optimization procedures, is no longer present. By removing the call site, it is also possible to perform better instruction scheduling, register allocation, etc. Third, the subprogram code that is substituted for the call can be optimized for the

10   specific call context.

The decision to inline a subprogram may be determined by the compiler automatically or assisted by user directed inlining. With user directed inlining, the programmer specifies which subprograms should be inlined. The compiler then attempts to inline the subprograms chosen by the user at each of its call sites.

15   When automatic inlining is used, the compiler determines which subprograms should be inlined by following a set of inline optimization rules. However, the typical rules implemented by a compiler do not account for subprograms that exhibit varying execution characteristics due to the range of variables or arguments over which the subprogram operates. With the varying arguments,

20   the subprogram's actual run-time may be substantially influenced depending on the argument received. For example, within a subprogram various execution paths may be taken based on the argument received. In some cases, the path taken is shorter and faster than others and the path may be taken more

frequently. However, because subprograms are inlined based on the execution time of the subprogram as a whole, the disparate execution times and disparate frequency of execution of the different paths are not accounted for very well in optimization techniques.

5   One method used to help the compiler determine whether to inline based on different variables is referred to as profiling. When using profiling, the computer program is executed at compile time using different data scenarios to determine how programs will perform (profiling) before producing the final compiled code. The use of profiling information typically requires at least two

10   passes to compile the program. One pass is performed then the compiled program is executed to generate the profiling information, and the other pass performs the automatic inlining based on the profiling information. The compiler's determination of whether to inline a subprogram that has been profiled is typically based on the number of times a subprogram is called and the

15   execution time of the subprogram. While using profiling to determine whether to inline a subprogram is beneficial in some cases, it does not solve the optimization problem introduced by subprograms that exhibit significantly different execution or run-time characteristics based on the arguments used or execution paths taken in the subprograms. That is, profiling only provides a

20   guess as to the best way to inline based on the data sets used to perform the profiling. Hence, the compiler's decision whether to inline may be correct or efficient for some scenarios but costly for others.

Thus, there is a need for a system and method that enables a compiler to make inlining decisions that are efficient for subprograms that have significantly varying execution times over a range of variables or execution paths.

III.   SUMMARY OF THE INVENTION

5   Methods and systems consistent with the present invention enable a compiler to make inlining decisions that are efficient for subprograms that have significantly varying execution times over a range of variables or execution paths.

In one aspect consistent with the present invention, a subprogram of a computer program is identified and certain execution paths of the subprogram

10   are selectively inlined. The subprogram may be identified based on execution characteristics of the subprogram and the selective inlining of the execution paths may be based on execution characteristics of the paths. These execution characteristics may be based on the execution time for the paths and/or on the frequency of execution of the paths. The paths may be selectively inlined based

15   on an inline indication associated with an execution path, where the inline indication may be an inline directive. The inline directive may be included as part of a program comment statement. The selective inlining of the paths may be determined using information profiles associated with the execution path.

In another aspect of the present invention, a subprogram is identified that

20   operates in a first manner when operands passed to the subprogram fall within a first range of values and that operates in a second manner when operands passed to the subprogram fall within a second range of values. Subprogram

statements that cause the subprogram to operate in the first manner are replaced with expanded code.

A compiler makes determinations whether to inline a specific execution path of a subprogram by evaluating certain information supplied in conjunction

5   with the path. By supplying information in association with the subprogram path, the compiler may more easily determine the various execution characteristics of the execution paths. Subprograms may be programmed to indicate or may be associated with the range of variables or parameters that cause the subprogram to exhibit a specific execution characteristic. When the compiler encounters an

10   indication of a subprogram path that may exhibit one of multiple execution characteristics, the compiler makes the decision whether to inline or not based on the execution characteristic indication associated with the subprogram call.

IV.   **BRIEF DESCRIPTION OF THE DRAWINGS**

Fig. 1 is a block diagram of a compiler system consistent with the present

15   invention;

Fig. 2 is a detailed drawing of a compiler consistent with the present invention;

Fig. 3 is a block diagram of modules associated with optimization procedures consistent with the present invention.

20   Fig. 4 is a diagram of the process of creating computer code consistent with the present invention; and

Fig. 5 is a diagram of the process for inlining that is consistent with the present invention.

V.   DETAILED DESCRIPTION OF THE INVENTION

Referring to the figures, a detailed description of the preferred embodiments of the present invention is described. A system or method operating consistent with the present invention identifies a subprogram that has a first and a second execution characteristic and replaces a first portion of the subprogram with expanded code that exhibits the first execution characteristic while leaving intact a second portion of the subprogram that exhibits the second execution characteristic. The first execution characteristic may be exhibited when operands passed to the subprogram fall within a first range of values and the second execution characteristic may be exhibited when operands passed to the subprogram fall within a second range of values. Such systems and methods are particularly useful in a compiler for selectively inlining portions of identified subprogram calls of a particular subprogram based on various execution characteristics of the subprogram. The term subprogram generally refers to user defined or predefined computer program routines or functions designed to carry out a desired task and expanded code generally refers to replacement computer program code that more explicitly defines the steps of an operation than the code it replaced.

Within a subprogram various execution paths may be taken based on the arguments received, and in some cases, the paths that may be taken vary in complexity, execution time, and frequency of traversal.   In prior systems, because subprograms are typically inlined based on the execution time and frequency of execution of the subprogram as a whole, the path dependent

execution times and frequency of execution are not accounted for very well in

optimization techniques. However, a system or method operating consistent with

the present invention enables various execution paths of a subprogram to be

evaluated separately for consideration for inlining. In an embodiment consistent

5   with the present invention, directives may be included in the various execution

paths within a subprogram to indicate that the path should be inlined or

considered for inlining. A compiler consistent with the present invention

recognizes the directive as an indication to consider the program instructions of

the particular execution path for inlining. As a result, various branches within a

10   subprogram may be selectively inlined according to the specifications or

characteristics of a particular execution path or branch.

Referring to Fig. 1, a compiler system 10 consistent with the present

invention operating in a computer system 14 is illustrated. It should be

appreciated that certain components of the computer system 14 are not

15   illustrated because such components are not necessary for an understanding of

the present invention. The computer system 14 includes a central processing

unit (CPU) 18, memory module 20, input/output ports 24, and a computer system

bus 28. The computer system bus 28 communicates data and signals among

the components of the computer system 14.

20   The memory module 20 is representative of random access memory, read

only memory and other memory elements used for storage and processing in the

computer system 14. The memory module 20 includes source code 30 of a

program to be compiled, a compiler 32, intermediate code 36 (without inlining),

intermediate code 38 (with inlining), and the assembly code 40. As known to those skilled in the art, CPU 18 executes compiler 32 in a manner consistent with the present invention. A computer program represented by the source code 30 is first converted to the intermediate code 36 by the compiler 32 prior to the

5   compiler 32 applying optimization procedures. The compiler 32 inlines selected portions of subprograms, consistent with the present invention, to produce intermediate code 38. The intermediate code 38 is then optimized to produce the assembly code 40. The compiler 32 consistent with the present invention is adapted to selectively inline portions of subprograms that exhibit various

10   execution characteristics. Particularly, selected execution branches of a subprogram are inlined based on the branches' execution characteristics.

        Referring to Fig. 2, a more detailed view of the compiler 32 is illustrated. The compiler 32 includes several subcomponents: an internal representation translator unit 210 that produces the intermediate code 36 from the source code

15   30, optimization procedures 220, and an assembler 240. The internal representation of the source code 30 is subjected to an array of optimization procedures 220 that are modeled into the compiler 32. The optimization procedures 220 are modeled consistent with the present invention to recognize directives included as part of an execution path of a subprogram. The directives

20   are used in association with execution paths to indicate that a path of a subprogram is to be given special consideration for inlining as discussed herein. The optimization procedures 220 produce the inlined intermediate code 38, which is provided to the assembler 240 to produce the assembly code 40. It

should be appreciated that optimization procedures, other than those discussed

herein, that are well known in the art may be used in combination with an

embodiment consistent with the present invention.

Referring to Fig. 3, a more detailed view of the optimization procedures

5   220 is illustrated. The optimization procedures 220 may include an inline

eligibility module 310, an inline profitability module 320, and a profiling module

330. The inline eligibility module 310 determines the portions of the source code

that are eligible for inlining by identifying subprograms using inlining eligibility

rules as known to those skilled in the art. After a program module is identified as

10   eligible for inlining, the profitability of inlining each identified subprogram is

determined by the profitability module 320. Generally speaking, the profitability

module 320 determines whether execution time will be saved by inlining

subprograms and inlines the subprograms after making the determination. The

profitability module 320 estimates the profitability of inlining based on rules

15   encoded into the profitability module 320, by assessing user inline directives,

and/or by invoking the profiling module 330.

When a developer includes a inline directive in a branch or execution path

of a subprogram, consistent with the present invention, the inline profitability

module 320 considers the identified path separately for inlining from other paths

20   of the subprogram or other instructions or operations of the subprogram. For

example, an inline directive may indicate to the compiler that it is likely that a

particular conditional execution statement, such as an "if" statement or branch,

will be taken. The compiler 32 uses this directive to determine whether to inline.

Consequently, the inline profitability module 320 may inline an identified path of a

subprogram that has a different characteristic than other paths of the subroutine.

It should be appreciated by those skilled in the art that procedures for

determining profitability are well known and therefore, are not discussed in detail

5   herein, as generally discussed in U.S. Patent No. 5,740,443.

As discussed above, a compiler may profile a program by compiling the

code and executing the compiled code with different sets of data to determine

the best way to finally compile the code. When profiling is desired, an option to

profile the code may be selected to invoke the profiling module 330. When

10   profiling is used, the inline profitability module 320 and the profiling module 330

implement a two stage compiling process. In the first stage, the program is

compiled and run a number of times using different scenarios or data. The

subprograms identified as possessing multiple execution characteristics have

identified portions (e.g. including an inline directive along a selected path or

15   paths of the subprogram) of the subprogram evaluated individually, with

corresponding profiling information. Inline directives are program statements that

provide hints to aid the compiler in making the decision of whether to inline a

particular segment of code.

The profiling module 330 may base profiling considerations on a single set

20   of profiling information and/or execution characteristics exhibited by certain paths

or portions of the subprogram in view of the various data sets designated for a

specific path or portion. The profiling module 330 collects and records

information on how many branches of or within the subprogram were taken and

how long it took the branches of the subprogram to run. In the second stage, the program is recompiled using the recorded information to determine whether to inline a subprogram or portion of the subprogram based on the gathered data. Subprograms or portions thereof that where not frequently utilized and/or had

5   relatively long execution times are not inlined and those that were frequently utilized and had relatively short execution times are inlined.

As discussed above, a system operating consistent with the present invention identifies a subprogram that has multiple execution characteristics and inlines certain segments of the subprogram based on the execution

10   characteristics of the subprogram. For example, in a first scenario, some subprograms' operations are straightforward or not very time consuming when the operands sent to the subprogram fall within a normal range of values. In another scenario, the operands sent to the subprogram fall outside the normal operating ranges for the subprogram and special processing is used to

15   accomplish the desired task. The special processing can include error trapping, error recovery, or it may require alternative computational methods. Systems and methods operating consistent with the present invention treat the invocation of a subprogram for operands that produce normal processing and operands that produce special processing as distinct cases for consideration for inlining. The

20   normal case processing is considered for inlining according to the general inlining procedures specified for the compiler and special case processing, which frequently occupies the bulk of the subprogram, is not likely to be inlined.

Many subprograms, such as the mathematical sine function, have
different execution characteristics based on the variables or arguments over
which it operates. In a computer program, for angles (Θ) < $\pi/8$, a sine
subprogram executes in a relatively fast manner compared to the time required

5   to call the sine subprogram in a computer program. However, the sine
subprogram executes in a relatively slower manner compared to the time
required to call the sine function for angles (Θ) $\geq$ $\pi/8$. In conventional compilers,
the compiler would assess whether the sine subprogram, in general, takes a long
time to execute based on conventional profitability analysis and would either

10   inline the entire sine subprogram based on the determined profitability. This
results in optimization of sine subprograms in certain situations and not others.
However, a system or method operating consistent with the present invention
distinguishes between the various execution characteristics of subprograms,
such as a sine subprogram, and selectively inlines portions of the sine

15   subprogram based on the execution characteristic that a branch of the sine
subprogram will likely exhibit.

Thus, in a system or method operating consistent with the present
invention, since inlining the sine function is profitable for angles < $\pi/8$, the
compiler 32 inlines code along the sine subprogram path that receive angles (Θ)

20   < $\pi/8$ and does not inline code along the sine subprogram path that receive
angles (Θ) > $\pi/8$. Consequently, subprograms are partially inlined based on the
execution characteristic of a particular branch.

The following is an example of source code used to illustrate an implementation consistent with the present invention. It should be appreciated that this source code example is not intended to represent a specific source code (high level) language but instead represents the general type logic statements

5   that may be implemented in various computer program languages, such as Java, C, Fortran, Pascal, or other high level language and does not limit the invention to any specific computer language. (Java is a registered trademark of Sun Microsystems Corporation.) The following represents a subprogram to compute the "sine" function:

10   If (Θ < $\pi/8$) then

        c$dir INLINE PATH

        {compute sine with a quick formula}

        else

        {compute sine with long process}

15   end if

        In the subprogram code illustrated above, the sine subprogram has two execution paths: one path that computes sine using a quick formula when Θ < $\pi/8$ (the short path) and another path that computes sine using a long formula

20   when Θ $\geq$ $\pi/8$ (the long path). In this example, the calculation of sine along the short path may be considered a first execution characteristic of the subprogram and the calculation of sine along the long path may be considered a second execution characteristic of the subprogram. The computation of sine along the

short path takes less time than the computation of sine along the long path. A directive "c$dir INLINE PATH" is included along the short path to identify the short path as having a certain execution characteristic. The "c" indicates the language following is a program comment and the "$dir" indicates that the

5   compiler is to interpret this comment as a special directive. In this example, the directive is named "INLINE PATH" and indicates to the compiler that this path or branch of the subprogram is to be inlined. By using the comment "c" syntax, if this code is evaluated by a compiler that does not have the logic consistent with the present invention modeled therein, the compiler treats the statement as a

10   program comment and does not perform any action with respect to the statement. Without an element to distinguish execution paths, such as the directive statement discussed above, a compiler would not be able to distinguish whether one path of subprogram was shorter/faster and would not know whether the path is frequently or infrequently executed.

15   When the compiler 32 encounters a path of a subprogram that includes a directive specified in accordance with the present invention, the compiler 32 inlines or considers for inlining the code along the short path. When profiling is used, a compiler consistent with the present invention may evaluate data collected after testing different sets of data to determine execution paths that

20   were taken frequently and executed quickly as opposed to only evaluating the entire subprogram as in conventional compilers.

It should be appreciated that the sine subprogram discussed is intended only as an example of a subprogram that may be evaluated and inlined

consistent with the principles of the present invention and that other

subprograms with different execution characteristics may be inlined consistent

the principles of the present invention. Another example of a subprogram that

can be inlined consistent with the present invention is the mathematical tangent

5   function, which processes normally for a certain range of variables but requires

special case processing for another range of variables.

Referring to Figs. 4-5, flow diagrams consistent with an embodiment of the

present invention are illustrated. Fig 4. is a flow diagram of the process used in

developing source code that is consistent with the present invention. Such

10   source code enables a compiler to identify the range of variables that are

associated with a specific execution characteristic of a subprogram. A

subprogram is identified that exhibits varying execution characteristics based on

the range of variables that cause the different execution characteristics (step

402). After identifying the ranges, the specific ranges of variables identifying

15   execution characteristics are associated with the particular conditional branch of

the subprogram that causes the associated execution characteristic to be

exhibited (step 410). The subprogram path or paths that are to be given special

consideration, such as for inlining, are identified and a directive consistent with

the present invention is included along the path to identify the path that is to be

20   given special consideration for inlining (step 430).

Referring to Fig. 5, a flow diagram of the processes for inlining computer

code consistent with the present invention is shown. Since the inlining process is

the same for various compilation systems, the discussion associated with Fig. 5

describes the processes that occur during the second pass of compilation when

profiling is used and describes the inlining processes when a single compilation

stage compiler is used. After optimization processing begins, the compiler

identifies a subprogram that is eligible for inlining (510). The compiler

5   determines whether the subprogram has multiple execution characteristics (step

512). If the subprogram does not have multiple execution characteristics, the

subprogram is evaluated using normal inlining procedures (step 516). The

compiler determines that the subprogram does not have multiple execution

characteristics if an inline directive statement is not associated with the

10   subprogram. The compiler then determines whether additional subprograms are

to be evaluated for inlining (step 540). If all subprograms have been evaluated,

the process ends (step 544). If all subprograms have not been evaluated, the

process continues (step 510).

If the compiler determines that a subprogram has multiple execution

15   characteristics (i.e. a directive consistent with the present invention is associated

with a particular execution path of the subprogram) (step 512), the compiler

considers the indicated execution path for inlining separately from the

subprogram as a whole (step 524). If the compiler determines that inlining

should not be applied to the execution path (step 524), the process checks for

20   other paths to be evaluated, and if no other paths exist (step 536) within the

subprogram, other subprograms, if any more exist, are evaluated (step 510). If

the compiler determines that inline processing is to occur (step 524), the

subprogram execution path is inlined (step 530). If there are no other execution

paths in subprogram (step 536), the process determines whether other

subprograms are to be evaluated (step 540). If there is another execution path

of the subprogram to be evaluated (step 536), the process determines whether

an inline directive is associated with the branch (step 524). If an inline directive

5   is associated with the execution path, the execution path will be given special

consideration for inlining (step 524), as discussed herein.

In summary, a compiler consistent with the present invention makes

determinations as to whether to inline a specific call to a subprogram by

evaluating certain information supplied in conjunction with the subprogram call.

10   By supplying information in association with the subprogram call, the compiler

may more easily determine the various execution characteristics of the execution

paths of a subprogram. For many subprograms, the source code developer

knows or can determine that certain predefined or developer defined

subprograms exhibit different characteristics based on the different variables

15   operated on by the subprogram. Thus, subprograms may be programmed to

indicate or may be associated with the range of variables or parameters that

cause the subprogram to exhibit a specific execution characteristic. When the

compiler encounters an indication of a subprogram path that may exhibit one of

multiple characteristics of the subprogram, the compiler makes the decision

20   whether to inline or not based on the execution characteristic indication

associated with the subprogram call.

By processing subprograms in a manner consistent with the present

invention, two disadvantages of normal inlining are solved. First, since only a

small portion of the subprogram is inlined, it is much less likely that the

executable code produced as a result of the inlining will grow to unacceptable

bounds. Second, the optimization is performed more efficiently since less code

will be inlined.

5   It should be appreciated by those skilled in the art that the present

invention may be used in various compilers or stages of compilation that perform

optimization. For example, a system or method consistent with the present

invention may be used for optimization as described herein when compiling

source code to intermediate code, such as Java byte codes. Additionally, a

10   system or method consistent with the present invention may be used for

optimization when byte codes are converted to object code.

It should be understood by those skilled in the art that various changes

and modifications may be made to the described embodiments and principles,

and equivalents may be substituted for elements without departing from the

15   scope of the invention. Modifications may be made to adapt a particular

element, technique, or implementation to the teachings of the present invention

without departing from the scope of the invention. It should be appreciated that

steps for performing processes consistent with the present invention may be

reordered. Steps may also be removed or added without departing from the

20   scope of the present invention. Although the described implementation is

discussed specifically in terms of software, the invention may be implemented as

a combination of hardware and software. Additionally, although aspects of the

present invention are described as being stored in memory, one skilled in the art

will appreciate that these aspects can also be stored on other types of computer-

readable media, such as secondary storage devices, like hard disks, floppy

disks, or CD-ROM; a carrier wave from the Internet; or other forms of RAM or

ROM. Therefore, the described embodiments should be taken as illustrative and

5   not restrictive, of the invention defined by the following claims and their scope of

equivalents.

VI.   WHAT IS CLAIMED

1.   A computer-implemented method for inlining code of a computer program, comprising:

identifying a subprogram of the computer program; and

selectively inlining computer code of certain execution paths of the subprogram.

2.   The method of claim 1 comprising identifying the subprogram based on execution characteristics of the subprogram.

3.   The method of claim 1 wherein said step of selectively inlining is based on execution characteristics of the execution paths.

4.   The method of claim 3 wherein the execution characteristics are based on the execution time for the paths.

5.   The method of claim 4 wherein the execution characteristics are based on the frequency of execution of the paths.

6.   The method of claim 1 wherein the step of selectively inlining is based on an inline indication associated with an execution path.

7.   The method of claim 6 wherein the inline indication is an inline directive.

8.   The method of claim 7 wherein the inline directive is included as part of a program comment statement.

9.   The method of claim 1 further comprising determining whether to selectively inline the execution paths using information profiles associated with the execution path.

10.   A computer program compiler for inlining computer program code, comprising:

a subprogram identification module that identifies subprograms of the computer code; and

5   a path identification module that selectively inlines computer code of certain execution paths of the subprogram.

11.   The apparatus of claim 10 wherein the path identification module selectively inlines based on an inline indication associated with an execution path.

10   12.   The apparatus of claim 11 wherein the inline indication is an inline directive.

13.   The method of claim 12 wherein the inline directive is included as part of a program comment statement.

14.   A computer-implemented method of determining whether to replace

15   subprogram code of a computer program, comprising the steps of:

identifying a subprogram that has a first and a second execution characteristic;

replacing a portion of the subprogram that exhibits the first execution characteristic with program instructions that explicitly define the operations of the

20   first execution characteristic; and

leaving intact a second portion of the subprogram that exhibits the second execution characteristic.

15. The method of claim 14 wherein said second execution characteristic is an atypical characteristic.

16. The method of claim 15 wherein said atypical characteristic is an execution time duration.

5   17. The method of claim 16 wherein said execution time duration exceeds a predetermined threshold.

18. The method of claim 17 wherein said first execution characteristic is a typical execution characteristic.

19. The method of claim 17 wherein said first and second execution

10 characteristics are execution time durations.

20. The method of claim 19 wherein the first and second execution characteristics are based on arguments operated on by the subprogram.

21. The method of claim 20 wherein the first and second execution characteristics are invoked based on conditional execution computer statements

15 associated with the characteristics.

22. The method of claim 11 wherein the first and second execution characteristics are invoked based on conditional execution computer statements associated with the characteristics.

23. A computer readable medium for inlining computer program code,

20 which when executed by a computer, performs the steps of:

identifying a subprogram that has a plurality of execution characteristics;

inlining only a selected portion of the subprogram that corresponds to one of the execution characteristics.

24.   The medium of claim 23 wherein the selected portion is defined by a selected path of a plurality of execution paths that may be executed by the subprogram.

25.   The medium of claim 24 further comprising determining the selected path by identifying a directive associated with the path.

26.   The medium of claim 25 wherein the directive is included in the selected path.

27.   The medium of claim 23 wherein the execution characteristics are identified by evaluating a conditional execution statement associated with a subprogram call.

28.   A computer-implemented method of replacing subprogram code in a computer system, comprising the steps of:

identifying a subprogram that operates in a first manner when operands passed to the subprogram fall within a first range of values and that operates in a second manner when operands passed to the subprogram fall within a second range of values; and

replacing subprogram statements that cause the subprogram to operate in the first manner with expanded code.

29.   A computer-implemented method for compiling a program containing at least one reference to a subprogram from source code to object code, wherein the subprogram has alternative paths of execution, the method comprising:

a) determining the execution path of the subprogram to be executed when the reference to the subprogram is encountered during compilation of the program; and

b) replacing the reference to the subprogram in the program with at least a portion of the subprogram corresponding to the execution path of the subprogram to be executed when the reference to the subprogram is encountered during compilation of the program.

30.   The method of claim 29 wherein determining the execution path of the subprogram to be executed when the reference to the subprogram is encountered during compilation of the program, comprises:

analyzing a set of input parameters that may be provided to the subprogram during execution.

31.   The method of claim 29, further comprising:

determining whether the subprogram contains a command signaling a preference for optimizing compilation of the program by performing steps a) and b).

32.   The method of claim 29, further comprising:

converting the program from source code to object code.

33.   A computer-implemented method for compiling a program containing at least one reference to a subprogram from source code to object code, wherein the subprogram has alternative paths of execution, the method comprising:

determining the execution path of the subprogram to be executed when the reference to the subprogram is encountered during compilation of the program; and

replacing the reference to the subprogram with only a portion of the subprogram corresponding to the execution path of the subprogram to be executed when the reference to the subprogram is encountered during compilation of the program.

34.   The method of claim 33 wherein determining the execution path of the subprogram to be executed when the reference to the subprogram is encountered during execution of the program, comprises:

analyzing a set of input parameters that may be provided to the subprogram during execution.

35.   An apparatus having a processor and a memory containing programs for inlining code of a computer program which when executed using the processor perform steps comprising:

identifying a subprogram of the computer program; and

selectively inlining computer code of certain execution paths of the subprogram.

36.   The apparatus of claim 35 wherein said subprogram is identified based on execution characteristics of the subprogram.

37.   The apparatus of claim 35 wherein said computer code is selectively inlined is based on execution characteristics of the execution paths.

38.   The apparatus of claim 37 wherein the execution characteristics are based on the execution time for the paths.

39.   The apparatus of claim 37 wherein the execution characteristics are based on the frequency of execution of the paths.

5   40.   The apparatus of claim 35 wherein the computer code is selectively inlined based on an inline indication associated with an execution path.

41.   The apparatus of claim 40 wherein the inline indication is an inline directive.

42.   The apparatus of claim 41 wherein the inline directive is included as

10   part of a program comment statement.

43.   The apparatus of claim 35 further comprising using information profiles associated with the execution path to determine whether to selectively inline the execution paths.

44.   An apparatus having a processor and a memory containing

15   programs for determining whether to replace subprogram code of a computer program which when executed using the processor perform steps comprising:

identifying a subprogram that has a first and a second execution characteristic;

replacing a portion of the subprogram that exhibits the first execution

20   characteristic with program instructions that explicitly define the operations of the first execution characteristic; and

leaving intact a second portion of the subprogram that exhibits the second execution characteristic.

45.   The apparatus of claim 44 wherein said second execution

characteristic is an atypical characteristic.

46.   The apparatus of claim 45 wherein said atypical characteristic is an

execution time duration.

5   47.   The apparatus of claim 46 wherein said execution time duration

exceeds a predetermined threshold.

48.   The apparatus of claim 47 wherein said first execution

characteristic is a typical execution characteristic.

49.   The apparatus of claim 47 wherein said first and second execution

10   characteristics are execution time durations.

50.   The apparatus of claim 49 wherein the first and second execution

characteristics are based on arguments operated on by the subprogram.

51.   The apparatus of claim 50 wherein the first and second execution

characteristics are invoked based on conditional execution computer statements

15   associated with the characteristics.

52.   The apparatus of claim 51 wherein the first and second execution

characteristics are invoked based on conditional execution computer statements

associated with the characteristics.

53.   An apparatus having a processor and a memory containing

20   programs for replacing subprogram code in a computer system which when

executed using the processor perform steps comprising, comprising the steps of:

identifying a subprogram that operates in a first manner when operands

passed to the subprogram fall within a first range of values and that operates in a

second manner when operands passed to the subprogram fall within a second

range of values; and

  replacing subprogram statements that cause the subprogram to operate in
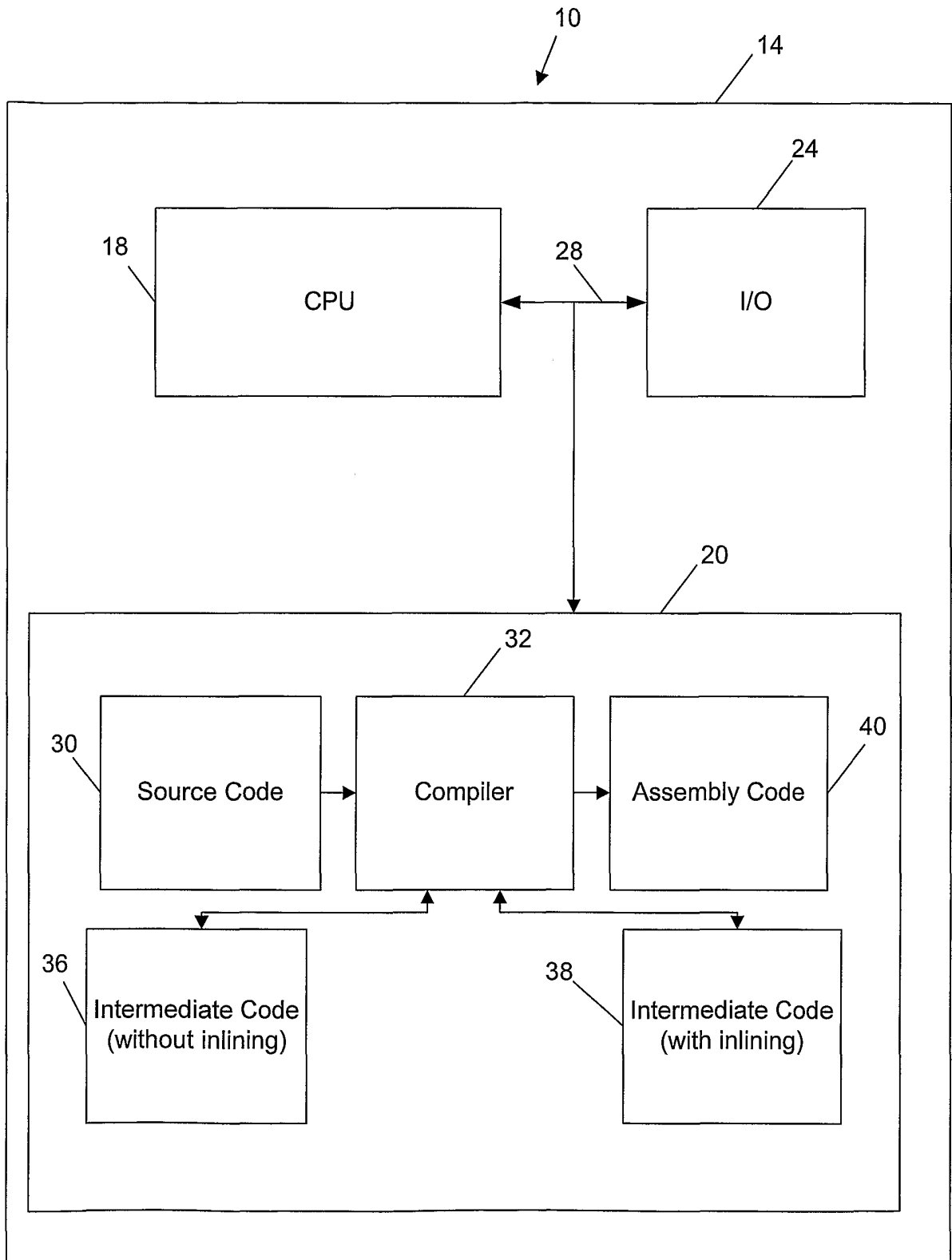
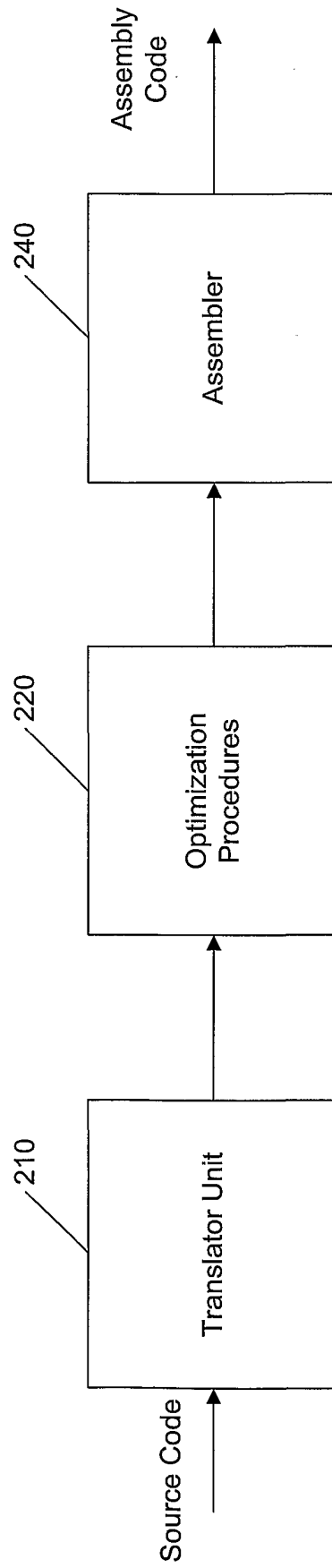the first manner with expanded code.

5

FIG. 1
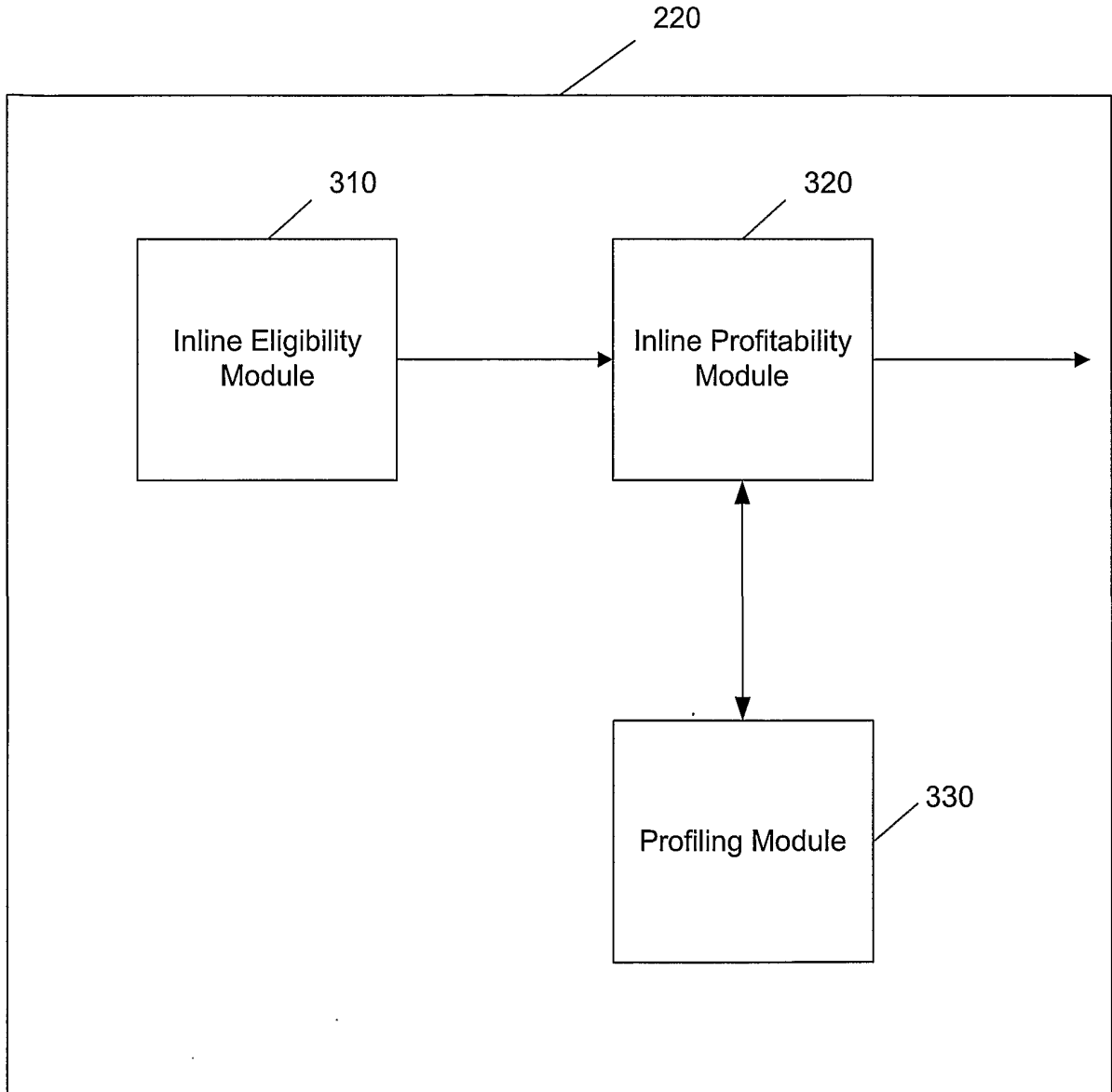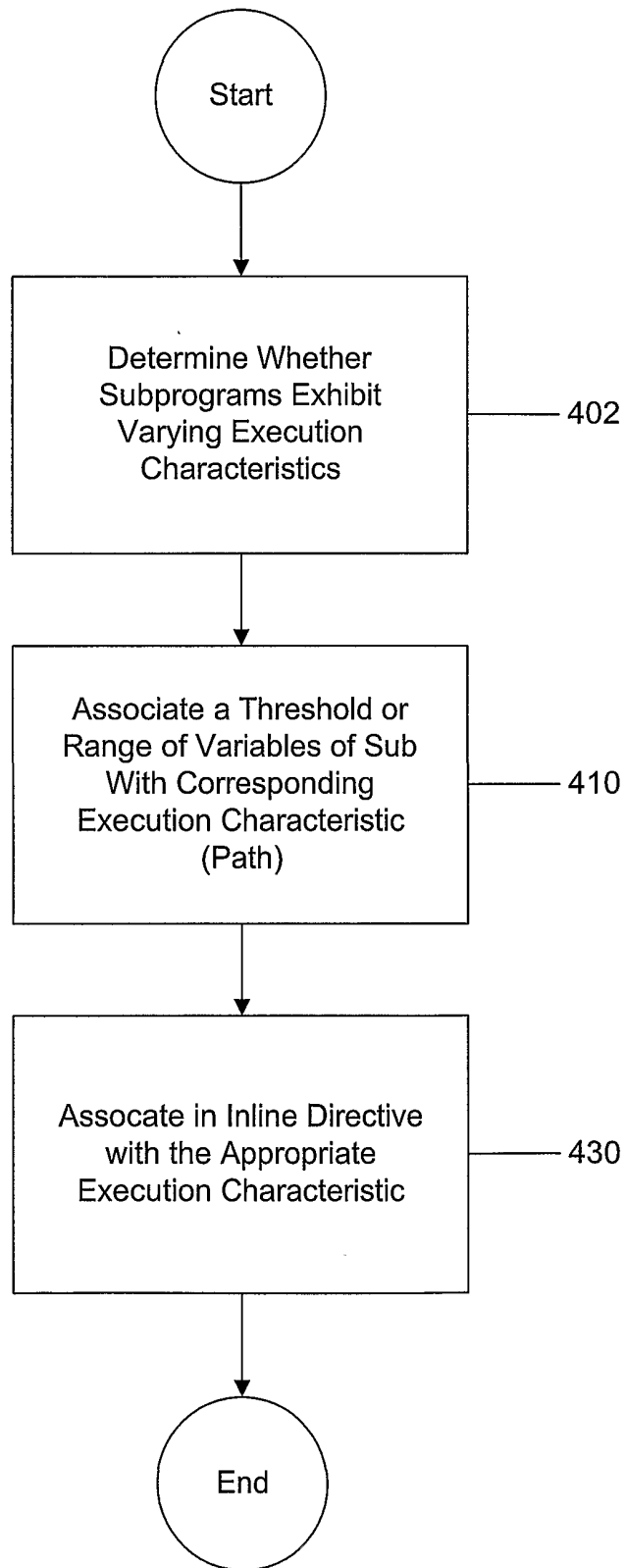
FIG. 2

FIG. 3

FIG. 4

FIG. 5