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

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2288614 C 2004/05/11

(11)(21) 2 288 614

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(22) Date de dépôt/Filing Date: 1999/11/08

(41) Mise à la disp. pub./Open to Public Insp.: 2001/05/08

(45) Date de délivrance/Issue Date: 2004/05/11

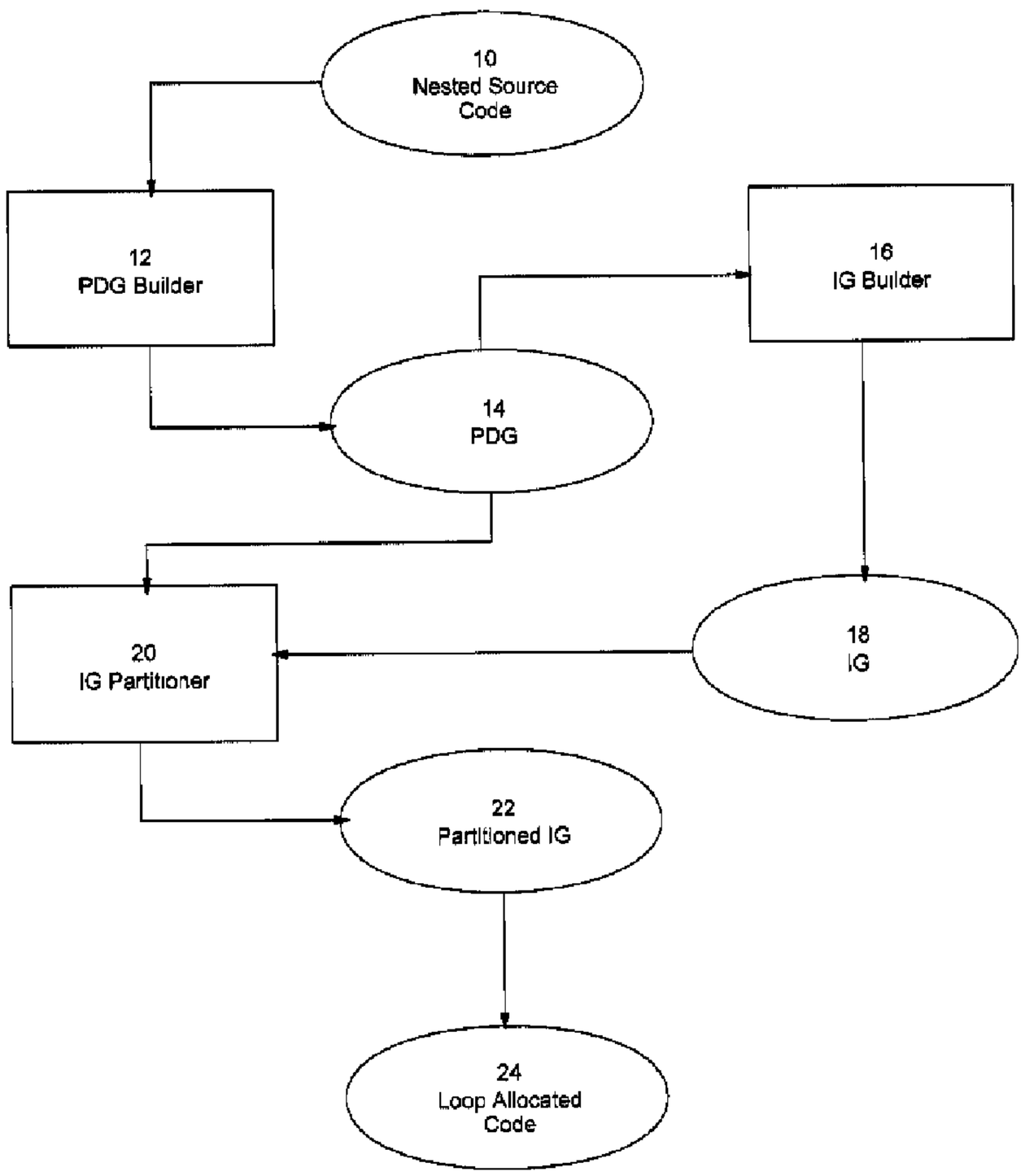
(51) Cl.Int.^{7}/Int.Cl.^{7} G06F 9/45

(72) Inventeurs/Inventors:
BLAINEY, ROBERT JAMES, CA;
ARCHAMBAULT, ROCH GEORGES, CA

(73) Propriétaire/Owner:
IBM CANADA LIMITED-IBM CANADA LIMITEE, CA

(74) Agent: SAUNDERS, RAYMOND H.

(54) Titre : AFFECTATION DE BOUCLES POUR L'OPTIMISATION DE COMPILATEURS
(54) Title: LOOP ALLOCATION FOR OPTIMIZING COMPILERS



(57) Abrégé/Abstract:
Loop allocation for optimizing compilers includes the generation of a program dependence graph for a source code segment. Control dependence graph representations of the nested loops, from innermost to outermost, are generated and data dependence graph representations are generated for each level of nested loop as constrained by the control dependence

CA 2288614 C 2004/05/11

(11)(21) 2 288 614

(13) C

(57) Abrégé(suite)/Abstract(continued):
graph. An interference graph is generated with the nodes of the data dependence graph. Weights are generated for the edges of the interference graph reflecting the affinity between statements represented by the nodes joined by the edges. Nodes in the interference graph are given weights reflecting resource usage by the statements associated with the nodes. The interference graph is partitioned using a profitability test based on the weights of edges and nodes and on a correctness test based on the reachability of nodes in the data dependence graph. Code is emitted based on the partitioned interference graph.

LOOP ALLOCATION FOR OPTIMIZING COMPILERS

ABSTRACT

Loop allocation for optimizing compilers includes the generation of a program dependence graph for a source code segment. Control dependence graph representations of the nested loops, from innermost to outermost, are generated and data dependence graph representations are generated for each level of nested loop as constrained by the control dependence graph. An interference graph is generated with the nodes of the data dependence graph. Weights are generated for the edges of the interference graph reflecting the affinity between statements represented by the nodes joined by the edges. Nodes in the interference graph are given weights reflecting resource usage by the statements associated with the nodes. The interference graph is partitioned using a profitability test based on the weights of edges and nodes and on a correctness test based on the reachability of nodes in the data dependence graph. Code is emitted based on the partitioned interference graph.

LOOP ALLOCATION FOR OPTIMIZING COMPILERS

FIELD OF THE INVENTION

The present invention is directed to an improvement in computing systems and in particular to computer systems which provide for optimized loop code generation in the compilation of computer programs.

BACKGROUND OF THE INVENTION

Optimizing compilers permit efficient object code to be emitted given a particular piece of source code to be compiled. Source code which includes loops is typically the subject of optimization in compilers. For a given segment of source code containing loops and for a given target machine micro architecture, cache geometry and parallel processing capability, the loop allocation of an optimizing compiler will be used to attempt to determine a collection of object code loop nests which will give efficient execution at an acceptable compilation-time cost.

Loop allocation optimization found in known compilers typically relies upon a set of ordered loop allocation transformations, as well as optimizations for data locality and parallelism. For example, loop source code may be optimized by emitting source code which minimizes off-chip access when the loop object code is executed. Another optimization for loop source code is to emit object code which may be executed in parallel by a multi-processor machine.

Typically, prior art optimizing compilers which carry out loop allocation include loop distribution early in the set of transformations, followed by parallelism and data locality transformations and finish with loop fusion and array contraction as a cleanup phase.

In prior art optimizing compilers, nested loops are optimized on a loop-by-loop basis. A prior art approach to optimizing sibling loops is to merge such nests. This approach is described by Sarkar, V. and Gao, G.R., "Optimization of Array Accesses by Collective Loop Transformations," 5th International Conference on Supercomputing, Cologne, Germany, June 1991, pp. 194-205. This

art approach involves a profitability and correctness test for the merger of the sibling loops. The optimization determines first if fusion of the sibling loops is desirable (a profitability analysis). Another prior art approach to loop optimization is to first distribute the loop code, to then optimize the distributed code and then to fuse the code after optimization.

Each of the above approaches to optimization involves optimizations of the loop code independent of, or following, loop distribution steps. Where nested loops are optimized on a loop-by-loop basis, optimizing which may be possible due to relationships between code in different nested loops may be missed. Similarly, where the loop code is distributed, optimized and then fused, the optimization is carried out on distributed portions of the code and interrelationships between those sections of code may not be considered in the optimization.

It is therefore desirable to have a computer system which carries out the loop allocation in an optimized compiler without accomplishing the loop distribution step at an early point in the sequence of loop transformations.

SUMMARY OF THE INVENTION

According to one aspect of the present invention, there is provided an improved system for the optimization of loop code compilation.

According to another aspect of the present invention, there is provided a computer program product storing computer readable program code, said computer readable program code for compiling a source code segment, said computer program product, when executed by a computer, adapts said computer to generate a program dependence graph for the source code segment, the program dependence graph comprising a control dependence graph and a data dependence graph, each of the control dependence graph and the data dependence graph comprising nodes, each node in the data dependence graph being associated with one or more statements in the source code segment, generate an interference graph from the data dependence graph, comprising instruction means for deriving nodes for the interference graph from the nodes in the data dependence graph, the nodes in the

interference graph thereby each being associated with one or more statements in the source code segment, generate a node weight for each node in the interference graph, each node having a node weight reflecting the resource usage for the one or more statements associated with the node, generate edges for the interference graph, each edge connecting a pair of nodes in the interference graph, generate an associated edge weight for each edge reflecting the desirability of maintaining the one or more statements associated with each of the pair of nodes connected by the edge within the same loop, partition the interference graph into subgraphs based on the edge weights and the node weights of the interference graph, and emit code conforming to the partitioned interference graph.

According to another aspect of the present invention, there is provided the above computer program product in which the instruction means for partitioning the interference graph comprises instruction means for first conducting a profitability test to select a pair of nodes in the interference graph, instruction means for then conducting a correctness test on the selected pair, and instruction means for merging the selected pair of nodes into a coalesced node where the correctness test is satisfied for the selected pair.

According to another aspect of the present invention, there is provided the above computer program product in which the instruction means for conducting a profitability test comprises instruction means for selecting the pair of nodes in the interference graph having the highest associated edge weight, the selected pair of nodes having a sum of node weights lower than a pre-defined resource limit for a target machine for the compiler.

According to another aspect of the present invention, there is provided the above computer program product in which the instruction means for conducting a correctness test comprises instruction means for comparing the selected pair of nodes in the interference graph with nodes in the interference graph corresponding to nodes in the data dependence graph defined to be reachable by the data dependence graph from those nodes in the data dependence graph reachable from the nodes in the data dependence graph corresponding to the selected pair of nodes in the interference graph.

According to another aspect of the present invention, there is provided the above computer program product in which the instruction means for comparing nodes comprises instruction means for defining a test set by generating a merged reachability set by taking the union of the nodes reachable from the selected pair of nodes, and removing the selected pair of nodes, and taking the union of the nodes reachable from the merged reachability set, and comparing the intersection of the test set with the union of the pair of selected nodes with the null set.

According to another aspect of the present invention, there is provided the above computer program product in which each node in the interference graph has an associated reachability vector representing which nodes in the interference graph are reachable from the node and in which set operations to determine reachability of nodes are carried out using the reachability vectors.

According to another aspect of the present invention, there is provided the above computer program product in which the instruction means for generating a program dependence graph comprises instruction means for ordering the generation of the program dependence graph from an innermost level of nested loops in the source code segment to an outermost level of nested loops in the source code segment.

According to another aspect of the present invention, there is provided the above computer program product in which the instruction means for generating a program dependence graph comprises instruction means for generating the control dependence graph for a level of nested loops in the source code segment and for then generating corresponding nodes for the data dependence graph for the said level of nested loops, the corresponding nodes in the data dependence graph being defined by constraints determined from the control dependence graph.

According to another aspect of the present invention, there is provided the above computer program product in which the instruction means for generating a program dependence graph comprises instruction means for determining pi blocks in the segment of source code and in which pi blocks are maintained in the generation of the program dependence graph from an inner level of nested

loops to a parent level of nested loops in the source code segment.

According to another aspect of the present invention, there is provided the above computer program product in which the means for emitting code comprises instruction means for generating optimized object code including one or more techniques selected from the set of techniques comprising scalar expansion, usage of temporary storage, array contraction, and strip mining.

According to another aspect of the present invention, there is provided the above computer program product of in which the instruction means for emitting code comprises means for generating optimized object code by optimizing for parallelism and for data locality.

According to another aspect of the present invention, there is provided a method for compiling a source code segment, the method comprising the following steps:

1. generating a program dependence graph for the source code segment, the program dependence graph comprising a control dependence graph and a data dependence graph, each of the control dependence graph and the data dependence graph comprising nodes, each node in the data dependence graph being associated with one or more statements in the source code segment,

2. causing a computer to generate an interference graph from the data dependence graph, by

deriving nodes for the interference graph from the nodes in the data dependence graph, the nodes in the interference graph thereby each being associated with one or more statements in the source code segment and

generating a node weight for each node in the interference graph, each node having a node weight reflecting the resource usage for the one or more statements associated with the node,

generating edges for the interference graph, each edge connecting a pair of nodes in the interference graph,

generating an associated edge weight for each edge reflecting the desirability of maintaining the one or more statements associated with each of the pair of nodes connected by the edge within the same loop,

3. partitioning the interference graph into subgraphs based on the edge weights and the node weights of the interference graph, and

4. emitting code conforming to the partitioned interference graph.

According to another aspect of the present invention, there is provided the above method in which the step of partitioning the interference graph comprises the steps of

1. conducting a profitability test to select a pair of nodes in the interference graph,

2. conducting a correctness test on the selected pair, and

3. merging the selected pair of nodes into a coalesced node where the correctness test is satisfied for the selected pair.

According to another aspect of the present invention, there is provided the above method in which the step of conducting the profitability test comprises selecting the pair of nodes in the interference graph having the highest associated edge weight, the selected pair of nodes having a sum of node weights lower than a pre-defined resource limit for a target machine for the compiler.

According to another aspect of the present invention, there is provided the above method in which conducting a correctness test comprises the steps of comparing the selected pair of nodes in the interference graph with nodes in the interference graph corresponding to nodes in the data dependence graph defined to be reachable by the data dependence graph from those nodes in the data dependence graph reachable from the nodes in the data dependence graph corresponding to the

selected pair of nodes in the interference graph.

According to another aspect of the present invention, there is provided a computer program product tangibly embodying a program of instructions executable by a computer to perform the above method steps.

According to another aspect of the present invention, there is provided a system for compilation of a source code segment. The system has means to generate a program dependence graph for the source code segment. The program dependence graph includes a control dependence graph and a data dependence graph. Each of the control dependence graph and the data dependence graph have nodes, each node in the data dependence graph is associated with one or more statements in the source code segment. There is also means to generate an interference graph from the data dependence graph, with means for deriving nodes for the interference graph from the nodes in the data dependence graph. The nodes in the interference graph are thereby each associated with one or more statements in the source code segment. There is means for generating a node weight for each node in the interference graph, each node having a node weight reflecting the resource usage for the one or more statements associated with the node. There is also means for generating edges for the interference graph, each edge connecting a pair of nodes in the interference graph, with means for generating an associated edge weight for each edge reflecting the desirability of maintaining the one or more statements associated with each of the pair of nodes connected by the edge within the same loop. There is also provided means for partitioning the interference graph into subgraphs based on the edge weights and the node weights of the interference graph, and means for emitting code conforming to the partitioned interference graph.

According to another aspect of the present invention, there is provided the above system in which the means for partitioning the interference graph comprises means for first conducting a profitability test to select a pair of nodes in the interference graph, means for then conducting a correctness test on the selected pair, and means for merging the selected pair of nodes into a coalesced node where the correctness test is satisfied for the selected pair.

According to another aspect of the present invention, there is provided the above system in which the means for conducting a profitability test comprises means for selecting the pair of nodes in the interference graph having the highest associated edge weight, the selected pair of nodes having a sum of node weights lower than a pre-defined resource limit for a target machine for the compiler.

According to another aspect of the present invention, there is provided the above system in which the means for conducting a correctness test comprises means for comparing the selected pair of nodes in the interference graph with nodes in the interference graph corresponding to nodes in the data dependence graph defined to be reachable by the data dependence graph from those nodes in the data dependence graph reachable from the nodes in the data dependence graph corresponding to the selected pair of nodes in the interference graph.

According to another aspect of the present invention, there is provided the above system in which the means for comparing nodes comprises means for defining a test set by generating a merged reachability set by taking the union of the nodes reachable from the selected pair of nodes, and removing the selected pair of nodes, and taking the union of the nodes reachable from the merged reachability set, and comparing the intersection of the test set with the union of the pair of selected nodes with the null set.

According to another aspect of the present invention, there is provided the above system in which each node in the interference graph has an associated reachability vector representing which nodes in the interference graph are reachable from the node and in which set operations to determine reachability of nodes are carried out using the reachability vectors.

Advantages of the present invention include improvements in optimization across loops and nests of loops. In addition, the manipulation of graph representations of the code permits application of known heuristic-based graph partitioning techniques to the loop allocation compilation.

BRIEF DESCRIPTION OF THE DRAWINGS

The preferred embodiment of the invention is shown in the drawings, wherein:

Figure 1 is a block diagram illustrating the process flow of the optimizing compiler of the preferred embodiment.

Figure 2 is a graph diagram illustrating an example partitioned interference graph as generated by the preferred embodiment.

Figure 3 is a graph diagram illustrating an example data dependence graph as generated by the preferred embodiment.

In the drawings, the preferred embodiment of the invention is illustrated by way of example. It is to be expressly understood that the description and drawings are only for the purpose of illustration and as an aid to understanding, and are not intended as a definition of the limits of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The system of the preferred embodiment is illustrated in Figure 1 which shows the flow of the loop allocation of the preferred embodiment in a block schematic diagram. Nested source code 10 is used by PDG (program dependence graph) builder 12 to create the program dependence graph shown as PDG 14. An interference graph (IG) is then created by IG builder 16 and is shown in Figure 1 as IG 18. The next step in the loop allocation of the preferred embodiment is IG partitioning 20 which takes as input both PDG 14 and IG 18 to create partitioned IG 22. This last partitioned graph, shown in Figure 1 as partitioned IG 22 is used to define loop allocated code 24.

As is described in more detail below, approaching loop allocation in an optimizing compiler by using the transformation ordering of the preferred embodiment results in natural optimization within and across loop nests. As the description of the preferred embodiment sets out, the preferred embodiment accomplishes an early maximal loop fusion. The compiler of the preferred embodiment also incorporates known techniques for generating optimized object code. These include scalar expansion, usage of temporary storage including registers (through the application of temporary vector allocation), array contraction and strip-mining. In the preferred embodiment, these techniques

are used before transformations for parallelism and data locality are performed. The description set out below does not describe such techniques in detail as they are known to those skilled in the art and may be utilized in conjunction with the loop allocation of the preferred embodiment as will be understood by those skilled in the art.

As indicated in Figure 1, the loop allocation of the preferred embodiment creates PDG 14 from nested source code 10. PDG builder 12 starts with the innermost nested loop and moves outwards. The construction of such a PDG is known in the art and is described, for example, in Ferrante, J., Ottenstein, K. and Warren, J., "The program dependence graph and its use in optimization," ACM Trans. Program. Lang. Syst., 1987, pp. 319-349. Although referred to as a single graph, there are two components to a program dependence graph such as PDG 14 shown in Figure 1: a control dependence graph (CDG) and a data dependence graph (DDG). In the CDG, the nodes are defined to be blocks of code (branch-free segments of the code) and edges are ordering constraints for the blocks (for example, evaluation of conditions). In the DDG, the nodes are individual statements (or as set out below, aggregate statements specified by the CDG) and the edges are data dependencies (flow dependence, anti-dependence and output dependence).

A program dependence graph (PDG) represents semantic information for a source code segment and is used in optimizing compilers to determine whether a transformation involving a reordering of code is permissible. In the preferred embodiment, the CDG for a given PDG (or portion of PDG) is generated first. Because the CDG defines required control dependencies, the CDG is used to create aggregate blocks of statements in nodes in the DDG. Where the CDG requires that statements are maintained together, they are included together in the nodes of the DDG.

The PDG is calculated starting at the innermost nest of loops in the code to be represented by the PDG. The PDG computes π blocks which are subgraphs of the DDG in which each node is reachable from each other node (strongly connected regions). In the generation of the PDG, π blocks are contributed up to the next level of the graph, when the parent loop is being processed. Once the top level of the PDG is created, then a partitioning, or interference, graph is built, as is described below.

As is shown in Figure 1, once PDG14 is created for source code 10, the interference graph IG 18 is created by IG builder 16. The graph of IG 18 is induced from the nodes of the DDG of PDG 14. The interference graph takes the nodes from the DDG of PDG 14, but not the edges. Thus IG 14 has nodes corresponding to the statements of nested source code 10. IG 18 is an undirected graph (the edges of the graph have no direction). Edges for IG 18 are created by IG builder 16 by giving a weight for each edge between nodes based on the desirability of keeping the two statements (or blocks of statements), represented by the two nodes at the ends of the edge, in the same loop. In addition, each node in IG 18 is given a weight by IG builder 16. This weight is determined by a heuristic in IG builder 16 which determines the resources used by the statement corresponding to the node in IG 18. For example, the number of linear streams in memory, or the number of registers used by the statement will affect the weight given to the node for that statement in IG 18, as determined by the heuristic of IG builder 16, for the given target machine.

An example of the determination of the weight for an edge between two nodes in IG 18 is shown for the following two statements:

A(i) = B(i) + C(i)

D(i)=B(i) + E(i)

Because B(i) is a data stream shared by the two statements there will be an affinity between the two statements (there will be efficiencies achieved by keeping the two statements in the same loop) and the weight of the edge between the two nodes representing the two statements will be adjusted by IG builder 16, accordingly. Other factors will be apparent to those skilled in the art as being relevant to the weights accorded to the edges in IG 18. For example, complimentary use of execution resources such as the balanced use of different pipelines in a target machine, may be included by IG builder 16 in determining the weights in IG 18.

As is shown in Figure 1, the nodes of IG 18 are then partitioned, or merged, into super nodes in partitioned IG 22. This is also referred to as a coalescing of nodes in IG 18. It will be apparent to

those skilled in the art that partitioned IG 22 may be implemented as a modified IG 18, rather than a separate data structure, as is shown in the block diagram of Figure 1.

Figure 2 is a graph diagram showing an example interference graph which has been partitioned or merged into a partitioned interference graph (as shown in Figure 1 as partitioned interference graph 22). The interference graph of Figure 2 has nodes 30, 32, 34, 36, 38 which correspond to statements A, B, C, D, E respectively. The interference graph in Figure 2 has weights assigned to the edges between nodes 30, 32 (a weight of 5), between nodes 32, 34 (15) , between nodes 32, 36 (10), between nodes 34, 36 (2) and between nodes 36, 38 (12). In addition, each of the nodes has been assigned a weight (for example 300 for node 30). The graph of Figure 2 is partitioned by selecting the edges of the graph in order of their relative weights. In the example of Figure 2, the nodes 32, 34 have the edge between them with the highest weight (15). The graph is therefore partitioned to create a merged node 40 comprising nodes 32,34. The next partition is performed on nodes 36, 38, which share an edge with the next highest weight, to create supernode 42.

The partitioning of an interference graph in the compiler of the preferred embodiment also includes a calculation based on the combination of the node weights in the merged node. For a given target machine, the loop allocation of the preferred embodiment has a specified resource limit or limits. This is reflected in a weight (which may be implemented as a vector of weights) which is used as a limit in the merging or coalescing of nodes into supernodes in the interference graph.

The resource limit for a merged node will be used to determine the desirability of merging two nodes which have been identified as candidates for merger due to the affinity weight found on the edge between the two nodes in the interference graph. If the resource limit is exceeded by the merged node's weight then the merge is not desirable. The resource limit is therefore used in a profitability test for the merger of nodes in the graph. Thus even where there is an affinity for statements being kept in the same loop, the profitability test encompasses a check to determine if keeping the statements in the same loop will result in the resource limit being exceeded for the target machine.

In the example of Figure 2, the supernode 42 has a merged weight of 600. If the resource limit for the target machine is 550, for example, supernode 42 will not be created.

The edge affinity weight and the resource limit test for merger of nodes in the interference graph are measures of the profitability of merging the nodes in the graph. This is a mechanism for determining whether the statements represented by the nodes in the graph are to be kept in the same loop in the emitted code. If the profitability test is passed, the next test carried out in the loop allocation of the preferred embodiment is to determine the correctness of the merger of the nodes.

In the preferred embodiment, the correctness test for the merger of two nodes is carried out by reference to the reachability of the nodes in the DDG. The DDG is a directed graph and may be used to determine whether the merger of the nodes will violate any of the constraints which are reflected in the DDG. The reachability of the nodes in the DDG reflects constraints on ordering of the statements which are subject to the proposed merger in the interference graph.

The calculation of reachability is carried out in the preferred embodiment by determining a reachability bit vector for each node in the DDG. An example DDG is shown in the graph of Figure 3. Nodes 50, 52, 54, 56, 58 in Figure 3 represent statements A, B, C, D, E, respectively. The universe for the bit vector is therefore the set A, B, C, D, E. The reachability of node 52 (statement B) is the set B, C, D, E, in the example of Figure 3. The reachability bit vector for a selected node is a bit vector which has five bits, each bit corresponding to the reachability of the corresponding node in the graph from the selected node. For example, the reachability bit vector for node 52 is (0 ,1, 1, 1, 1), representing the fact that node 52 (statement B) is reachable from nodes 52, 54, 56, 58, but not from node 50 (in terms of statements, it is reachable from statements B, C, D, E, but not from A).

Using a bit vector to represent the reachability of nodes in the DDG permits operations to be carried out on the bit vectors in a single operation for each of the 32 nodes in the graph for many implementing computers. For example, a 32-bit machine is capable of storing a DDG reachability bit vector for a graph of up to 32 nodes.

The correctness test for merging two subgraphs in an interference graph may be described using the following steps, where R(n) is the set of nodes reachable from node n:

Let G1 and G2 be two subgraphs of the interference graph

Let $S = (R(G1) \cup R(G2)) - (G1 \cup G2)$

Let $T = \cup_{i \in S} R(i)$ (the union over i in S:R(i))

The merger will be correct if $T \cap (G1 \cup G2) == \varnothing$. The above steps will determine whether the proposed merger will result in a violation of the constraint of the DDG by determining whether the proposed merger (G1 ∪ G2) will result in nodes being reachable by the merged node (set T) which are in the merged set (G1 ∪ G2). Because reachability in the DDG is an indication of a semantic constraint that the statement of a reachable node follow a statement of a given node, if $T \cap (G1 \cup G2)$ is not null, the merged node will result in statements in the merged node being constrained to follow the merged node, a contradictory conclusion which indicates that the constraints of the DDG are violated by the proposed merged supernode (or subgraph).

After the partitioning or coalescing of subgraphs, the reachability vectors are updated as follows:

GNEW = G1 ∪ G2 (where G1 and G2 are coalesced.)

$R(GNEW) = R(G1) \cup R(G2)$

For each other subgraph D,

if $R(D) \cap (G1 \cup G2) != \varnothing$,

then

$R(D) = R(D) \cup R(GNEW)$

The above operations are carried out by the preferred embodiment to ensure that the reachability vectors are maintained correctly following the partitioning of the interference graph. As is described above, the use of reachability vectors provides a correctness test which defines whether a merger of nodes in the interference graph which has been identified as profitable is in fact permitted by the

constraints represented in the DDG.

Following the graph partitioning set out above, the optimizing compiler of the preferred embodiment uses known techniques to emit optimized code, such as scalar expansion, usage of temporary storage, array contraction and strip-mining. The generation of both the PDG and the interference graph for a given segment of source code, and then the coalescing of subgraphs in the interference graph permit the optimization to occur across loops and nests of loops. In addition, the manipulation of graph representations of the code permits application of known heuristic-based graph partitioning techniques to the loop allocation compilation.

The detailed descriptions may have been presented in terms of program procedures executed on a computer or network of computers. These procedural descriptions and representations are the means used by those skilled in the art to most effectively convey the substance of their work to others skilled in the art. They may be implemented in hardware or software, or a combination of the two.

A procedure is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. These steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, objects, attributes or the like. It should be noted, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

Further, the manipulations performed are often referred to in terms, such as adding or comparing, which are commonly associated with mental operations performed by a human operator. No such capability of a human operator is necessary, or desirable in most cases, in any of the operations described herein which form part of the present invention; the operations are machine operations. Useful machines for performing the operations of the present invention include general purpose

digital computers or similar devices.

Each step of the method may be executed on any general computer, such as a mainframe computer, personal computer or the like and pursuant to one or more, or a part of one or more, program modules or objects generated from any programming language, such as C++, Java, Fortran or the like. And still further, each step, or a file or object or the like implementing each step, may be executed by special purpose hardware or a circuit module designed for that purpose.

In the case of diagrams depicted herein, they are provided by way of example. There may be variations to these diagrams or the steps (or operations) described herein without departing from the spirit of the invention. For instance, in certain cases, the steps may be performed in differing order, or steps may be added, deleted or modified. All of these variations are considered to comprise part of the present invention as recited in the appended claims.

The invention may be implemented as an article of manufacture comprising a computer usable medium having computer readable program code means therein for executing the method steps of the invention, a program storage device readable by a machine, tangibly embodying a program of instructions executable by a machine to perform the method steps of the invention, or a computer program product. Such an article of manufacture, program storage device or computer program product may include, but is not limited to, CD-ROMs, diskettes, tapes, hard drives, computer RAM or ROM and/or the electronic, magnetic, optical, biological or other similar embodiment of the program. Indeed, the article of manufacture, program storage device or computer program product may include any solid or fluid transmission medium, magnetic or optical, or the like, for storing or transmitting signals readable by a machine for controlling the operation of a general or special purpose programmable computer according to the method of the invention and/or to structure its components in accordance with a system of the invention.

The invention may also be implemented in a system. A system may comprise a computer that includes a processor and a memory device and optionally, a storage device, an output device such

as a video display and/or an input device such as a keyboard or computer mouse. Moreover, a system may comprise an interconnected network of computers. Computers may equally be in stand-alone form (such as the traditional desktop personal computer) or integrated into another apparatus (such a cellular telephone). The system may be specially constructed for the required purposes to perform, for example, the method steps of the invention or it may comprise one or more general purpose computers as selectively activated or reconfigured by a computer program in accordance with the teachings herein stored in the computer(s). The procedures presented herein are not inherently related to a particular computer system or other apparatus. The required structure for a variety of these systems will appear from the description given.

While this invention has been described in relation to preferred embodiments, it will be understood by those skilled in the art that changes in the details of construction, arrangement of parts, compositions, processes, structures and materials selection may be made without departing from the spirit and scope of this invention. Many modifications and variations are possible in light of the above teaching. Thus, it should be understood that the above described embodiments have been provided by way of example rather than as a limitation and that the specification and drawing(s) are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A computer program product storing computer readable program code, said computer readable program code for compiling a source code segment, said computer program product, when executed by a computer, adapts said computer to:

generate a program dependence graph for the source code segment, the program dependence graph comprising a control dependence graph and a data dependence graph, each of the control dependence graph and the data dependence graph comprising nodes, each node in the data dependence graph being associated with one or more statements in the source code segment,

generate an interference graph from the data dependence graph, comprising instruction means for deriving nodes for the interference graph from the nodes in the data dependence graph, the nodes in the interference graph thereby each being associated with one or more statements in the source code segment,

generate a node weight for each node in the interference graph, each node having a node weight reflecting the resource usage for the one or more statements associated with the node,

generate edges for the interference graph, each edge connecting a pair of nodes in the interference graph,

generate an associated edge weight for each edge reflecting the desirability of maintaining the one or more statements associated with each of the pair of nodes connected by the edge within the same loop,

partition the interference graph into subgraphs based on the edge weights and the node weights of the interference graph, and

emit code conforming to the partitioned interference graph.

2. The computer program product of claim 1 wherein said instructions adapting said computer to partition the interference graph comprises instructions for adapting said computer to conduct a profitability test to select a pair of nodes in the interference graph; conduct a correctness test on the selected pair; and merge the selected pair of nodes into a coalesced node where the

correctness test is satisfied for the selected pair.

3. The computer program product of claim 2 wherein said instructions for adapting said computer to conduct a profitability test comprises instructions for adapting said computer to select the pair of nodes in the interference graph having the highest associated edge weight, the selected pair of nodes having a sum of node weights lower than a pre-defined resource limit for a target machine for the compiler.

4. The computer program product of claim 2 or claim 3 wherein the instructions for adapting said computer to conduct a correctness test comprises instructions for adapting said computer to:

compare the selected pair of nodes in the interference graph with nodes in the interference graph corresponding to nodes in the data dependence graph defined to be reachable by the data dependence graph from those nodes in the data dependence graph reachable from the nodes in the data dependence graph corresponding to the selected pair of nodes in the interference graph.

5. The computer program product of claim 4 wherein instructions for adapting said computer to compare nodes comprises instructions for adapting said computer to define a test set by generating a merged reachability set by taking the union of the nodes reachable from the selected pair of nodes, and removing the selected pair of nodes, and taking the union of the nodes reachable from the merged reachability set, and comparing the intersection of the test set with the union of the pair of selected nodes with the null set.

6. The computer program product of claim 5 in which each node in the interference graph has an associated reachability vector representing which nodes in the interference graph are reachable from the node and in which set operations to determine reachability of nodes are carried out using the reachability vectors.

7. The computer program product of claim 6 wherein instructions for adapting said computer to generate a program dependence graph comprises instructions for adapting said computer to generate the control dependence graph for a level of nested loops in the source code segment and generate corresponding nodes for the data dependence graph for the said level of nested loops, the corresponding nodes in the data dependence graph being defined by constraints determined from the control dependence graph.

8. The computer program product of any one of claims 1 to 6 wherein instructions for adapting said computer to generate a program dependence graph comprises instructions for adapting said computer to order the generation of the program dependence graph from an innermost level of nested loops in the source code segment to an outermost level of nested loops in the source code segment.

9. The computer program product of claim 8 wherein instructions for adapting said computer to generate a program dependence graph comprises instructions for adapting said computer to determine pi blocks in the segment of source code and in which pi blocks are maintained in the generation of the program dependence graph from an inner level of nested loops to a parent level of nested loops in the source code segment.

10. The computer program product of claim 9 wherein instructions for adapting said computer to emit code comprises instructions for adapting said computer to generate optimized object code by optimizing for parallelism and for data locality.

11. The computer program product of any one of claims 1 to 10 wherein instructions for adapting said computer to emit code comprises instructions for adapting said computer to generate optimized object code including one or more techniques selected from the set of techniques comprising scalar expansion, usage of temporary storage, array contraction, and strip mining.

12. A loop allocation optimizing compiler comprising,

means for generating a program dependence graph for a source code segment, the program dependence graph comprising a control dependence graph and a data dependence graph, each of the control dependence graph and the data dependence graph comprising nodes, each node in the data dependence graph being associated with one or more statements in the source code segment.

means for generating an interference graph from the data dependence graph, comprising means for deriving nodes for the interference graph from the nodes in the data dependence graph, the nodes in the interference graph thereby each being associated with one or more statements in the source code segment.

means for generating a node weight for each node in the interference graph, each node having a node weight reflecting the resource usage for the one or more statements associated with the node,

means for generating edges for the interference graph, each edge connecting a pair of nodes in the interference graph,

means for generating an associated edge weight for each edge reflecting the desirability of maintaining the one or more statements associated with each of the pair of nodes connected by the edge within the same loop.

means for partitioning the interference graph into subgraphs based on the edge weights and the node weights of the interference graph, comprising means for first conducting a profitability test to select a pair of nodes in the interference graph, means for then conducting a correctness test on the selected pair, and means for merging the selected pair of nodes into a coalesced node where the correctness test is satisfied for the selected

Figure 1

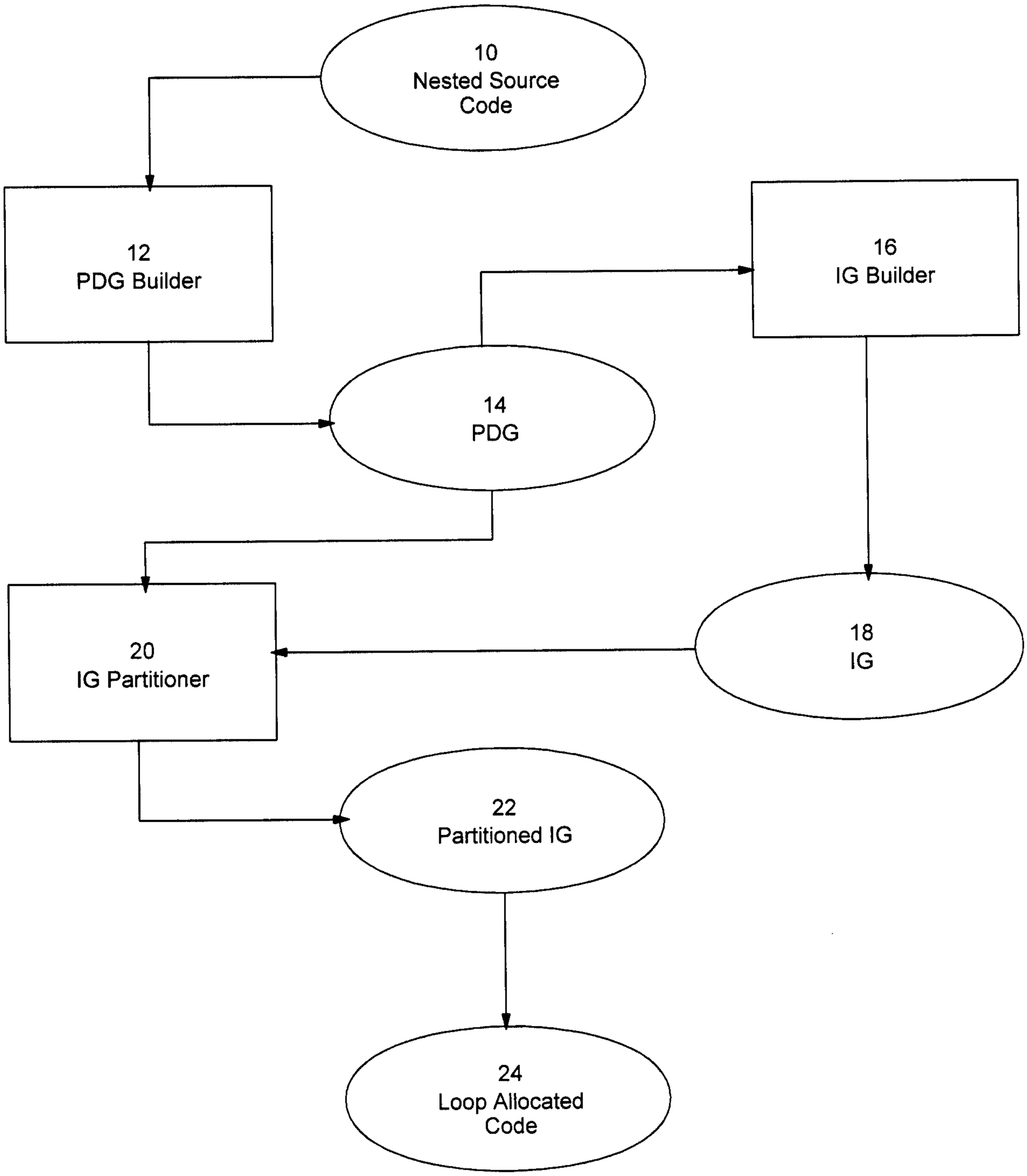

10
Nested Source Code
12
PDG Builder
14
PDG
16
IG Builder
18
IG
20
IG Partitioner
22
Partitioned IG
24
Loop Allocated Code

Figure 2

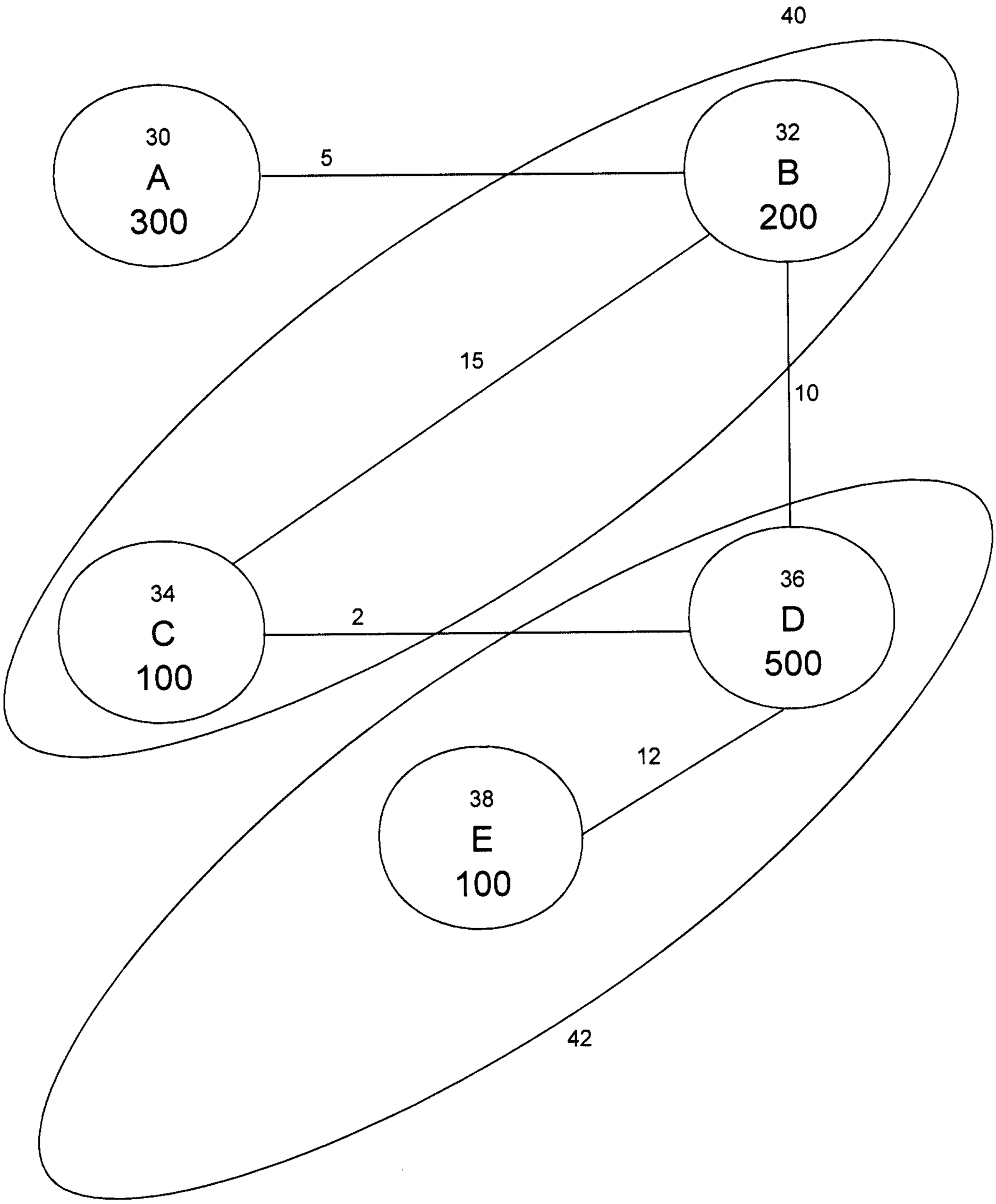
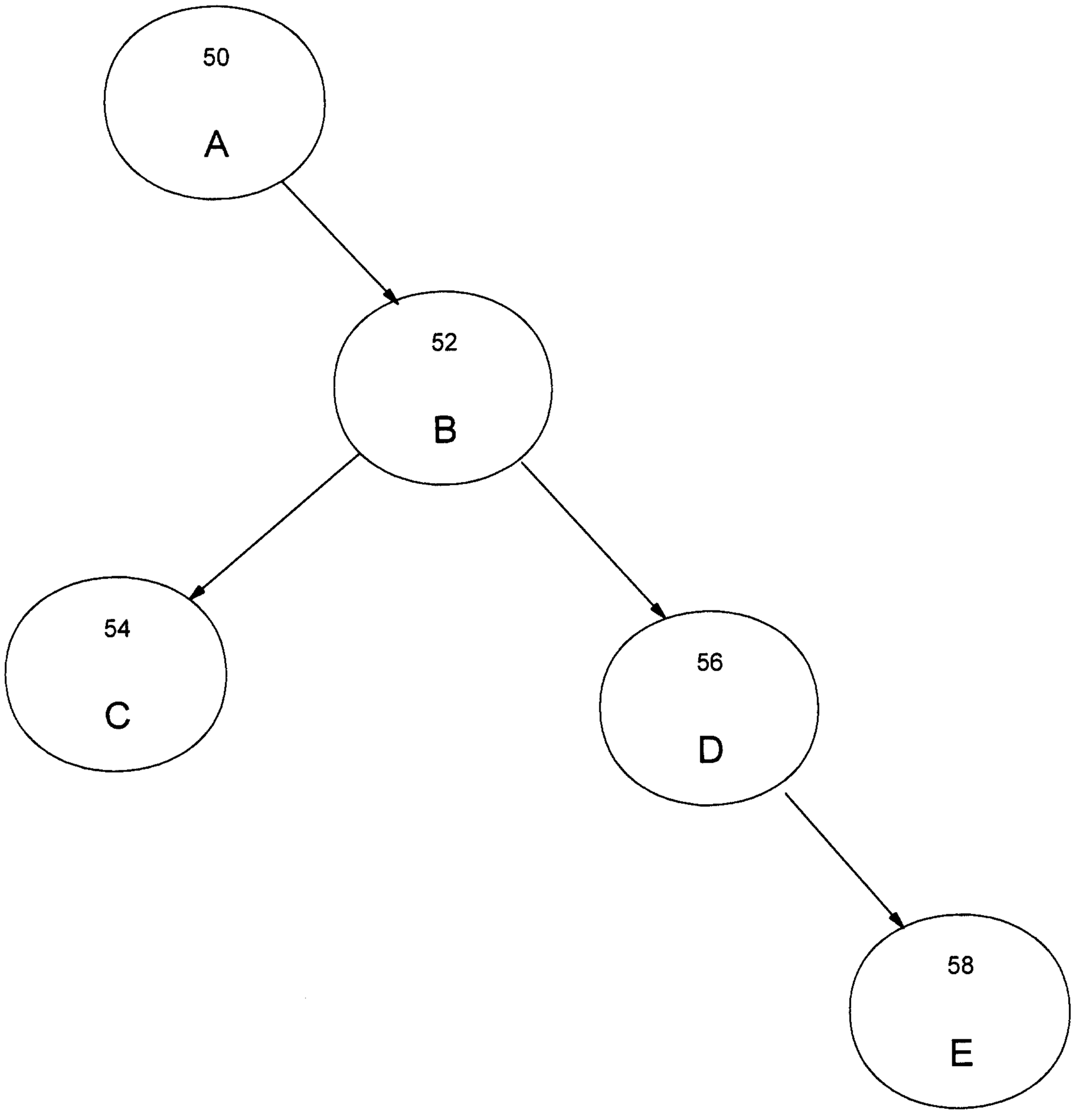
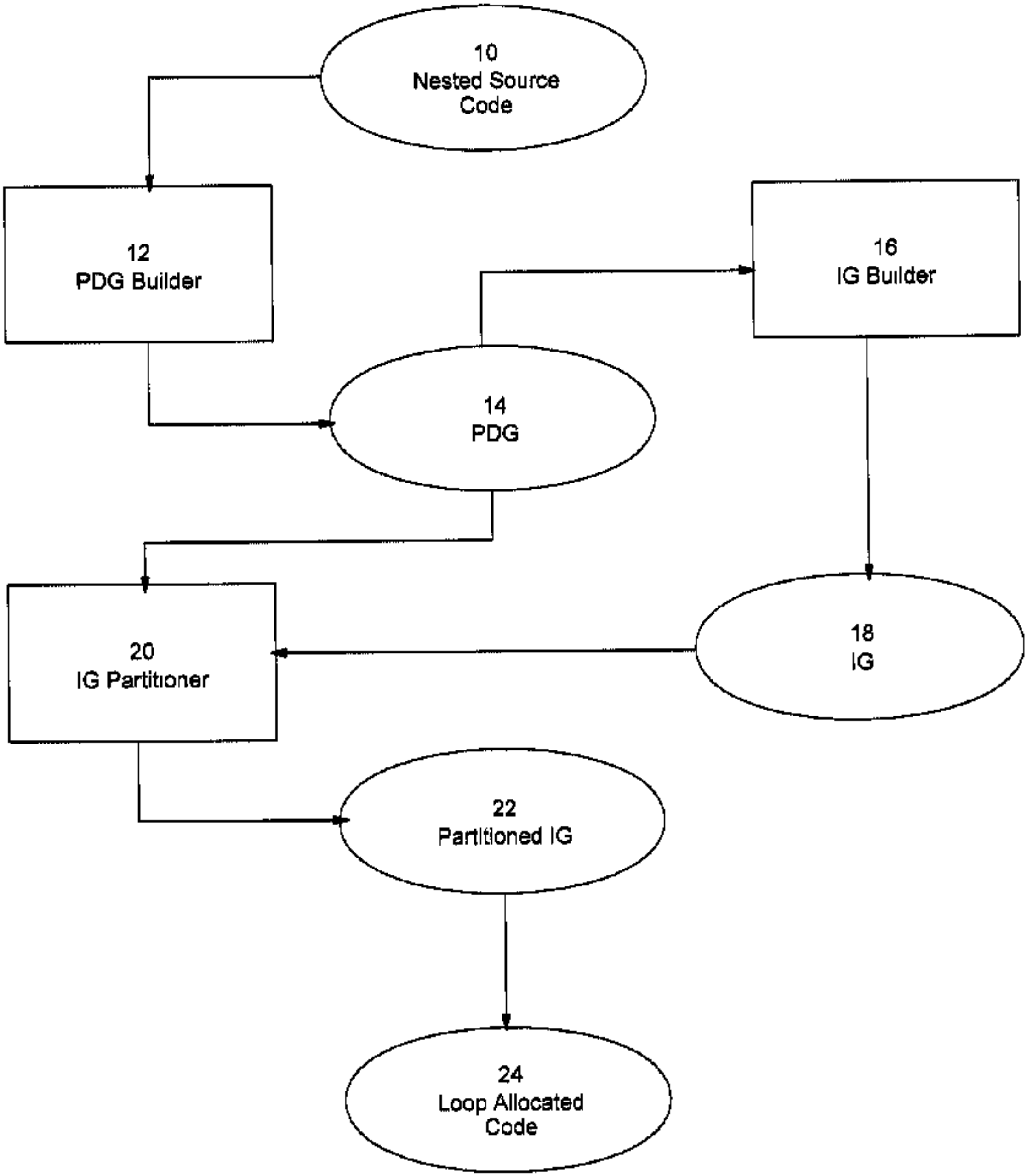


Figure 3




10
Nested Source
Code
12
PDG Builder
14
PDG
16
IG Builder
18
IG
20
IG Partitioner
22
Partitioned IG
24
Loop Allocated
Code