Title: SYSTEMS AND METHODS FOR REGISTER ALLOCATION

Abstract: System and methods are provided for register allocation. An original code block and a target code block associated with a branch of an execution loop are determined. An original allocation of a plurality of physical registers to one or more original variables associated with the original code block is detected. A target allocation of the plurality of physical registers to one or more target variables associated with the target code block is determined. One or more temporary registers are selected from the plurality of physical registers at least in part on the original allocation and the target allocation. The original allocation is changed to the target allocation using the selected temporary registers. The instructions are executed using one or more processors.

FIG. 1
CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This disclosure claims priority to and benefit from U.S. Provisional Patent Application No. 61/892,638, filed on October 18, 2013, the entirety of which is incorporated herein by reference.

FIELD

[0002] The technology described in this patent document relates generally to program execution and more particularly to register allocation for program execution.

BACKGROUND

[0003] Just-in-time compilation (JIT), also known as dynamic translation, is compilation done during execution of a program or an application at run time, rather than prior to execution. Often JIT involves translation/compilation to native codes for a machine (e.g., a computer), which are then executed directly by one or more processors of the machine.

[0004] JIT compilers may be used for generating native codes for Java programs. Java programs may be run on any machine on which a Java virtual machine (JVM) is executing. A Java program includes a plurality of object classes, and each object class can have zero or more methods. When a Java program is executed, the methods of the object classes are invoked and executed.

[0005] A java method may be executed in one of two ways. One way is for the JVM to execute the method in an interpretive manner. For example, before a Java program is executed, source codes of the program (e.g., the source codes of the methods) is broken down into Java
bytecodes. At runtime, the Java interpreter of the JVM interprets the bytecodes of a method, which is a relatively slow process. As an alternative, the JVM may choose not to interpret a method, but rather compile the bytecodes of the method down into native codes, e.g., using a JIT compiler at runtime. The JVM then causes the native codes to be executed directly by one or more processors.

[006] Local variables associated with the execution of native codes generated by a JIT compiler often need to be loaded from memory, which is a time consuming process. One way to improve the execution of the native codes generated by the JIT compiler is to maintain often-used local variables in physical registers (e.g., hardware registers) as much as possible. Register allocation is an important optimization process for improving efficiency. For example, a linear scan algorithm can be implemented for register allocation. The linear scan algorithm involves assigning registers to variables in a single linear scan over live intervals of all variables in a program. A live interval of a variable corresponds to a range of instructions starting at a defining instruction and ending at an instruction where the variable is used for the last time. If the live intervals of two variables overlap, the variables cannot reside in a same physical register.

SUMMARY

[007] In accordance with the teachings described herein, system and methods are provided for register allocation. An original code block and a target code block associated with a branch of an execution loop are determined. An original allocation of a plurality of physical registers to one or more original variables associated with the original code block is detected. A target allocation of the plurality of physical registers to one or more target variables associated with the target code block is determined. One or more temporary registers are selected from the plurality of physical registers based at least in part on the original allocation and the target allocation. The
original allocation is changed to the target allocation using the selected temporary registers. Specifically, one or more instructions are generated to change the original allocation to the target allocation using the selected temporary registers. The instructions are executed using one or more processors.

[0008] In one embodiment, a processor-implemented system for register allocation includes: one or more non-transitory machine-readable storage media for storing a computer database having a database schema that includes and interrelates original variable fields, target variable fields, original allocation fields, and target allocation fields. The original variable fields store one or more original variables associated with an original code block. The target variable fields store one or more target variables associated with a target code block, the original code block and the target code block being associated with a branch of an execution loop. The original allocation fields store original data representing an original allocation of a plurality of physical registers to the original variables. The target allocation fields store target data representing a target allocation of the plurality of physical registers to the target variables. The system further includes one or more processors configured to: process a database query that operates over data related to the original variable fields, the target variable fields, the original allocation fields, and the target allocation fields; select one or more temporary registers from the plurality of physical registers based at least in part on the original data and the target data; and change the original allocation of the plurality of physical registers to the target allocation of the plurality of physical registers using the temporary registers, the physical registers being included in the processors.

[0009] In another embodiment, a system for register allocation includes: one or more processors including a plurality of physical registers, and a computer-readable storage medium encoded with instructions for commanding the data processors to execute certain operations. An
original code block and a target code block associated with a branch of an execution loop are determined. An original allocation of a plurality of physical registers to one or more original variables associated with the original code block is detected. A target allocation of the plurality of physical registers to one or more target variables associated with the target code block is determined. One or more temporary registers are selected from the plurality of physical registers based at least in part on the original allocation and the target allocation. The original allocation is changed to the target allocation using the selected temporary registers. Specifically, one or more instructions are generated to change the original allocation to the target allocation using the selected temporary registers. The instructions are executed using one or more processors.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] FIG. 1 depicts an example diagram showing a flow chart involving a plurality of code blocks.

[0011] FIG. 2 depicts an example diagram showing conflict resolution of register allocation.

[0012] FIG. 3 depicts an example diagram for conflict resolution of register allocation.

[0013] FIG. 4 depicts an example flow chart for register allocation.

[0014] FIG. 5 depicts an example diagram showing a database.

DETAILED DESCRIPTION

[0015] Linear scan register allocation is widely applied to JIT compilers which are sensitive to compiling time. However, under some circumstances (e.g., execution loops), conflicts may occur during the register allocation process and need to be resolved for executing programs correctly.
FIG. 1 depicts an example diagram showing a flow chart involving a plurality of code blocks. As shown in FIG. 1, code blocks 202 and 204 are included in an execution loop. Register allocation (e.g., with a linear scan algorithm) is performed for executing the code blocks. Conflicts occur in the register allocation process for the code block 202 and the code block 204.

Specifically, a code block 206 is executed first. Then, a forward loop branch starts from the code block 202 and ends at the code block 204, and a backward loop branch starts from the code block 204 and ends at the code block 202. In the forward loop branch, the register allocation is performed for the code block 202, and one or more physical registers (e.g., hardware registers in one or more processors) are allocated to one or more variables, e.g., as shown in Table 1.

<table>
<thead>
<tr>
<th>Registers</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>v0</td>
</tr>
<tr>
<td>r1</td>
<td>v3</td>
</tr>
<tr>
<td>r2</td>
<td>v1</td>
</tr>
<tr>
<td>r3</td>
<td>v2</td>
</tr>
<tr>
<td>r6</td>
<td>v4</td>
</tr>
<tr>
<td>r7</td>
<td>v5</td>
</tr>
<tr>
<td>r8</td>
<td>v6</td>
</tr>
</tbody>
</table>

Then, for the code block 204, the register allocation is performed, and the physical registers are allocated to certain variables, e.g., as shown in Table 2.
Table 2

<table>
<thead>
<tr>
<th>Registers</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>v0</td>
</tr>
<tr>
<td>r1</td>
<td>v1</td>
</tr>
<tr>
<td>r4</td>
<td>v3</td>
</tr>
<tr>
<td>r6</td>
<td>v5</td>
</tr>
<tr>
<td>r7</td>
<td>v6</td>
</tr>
<tr>
<td>r8</td>
<td>v4</td>
</tr>
<tr>
<td>r9</td>
<td>v7</td>
</tr>
</tbody>
</table>

[0019] In the backward loop branch, i.e., from the code block 204 back to the code block 202, the physical registers, as shown in Table 2, do not contain proper variables for executing the code block 202 which needs the register allocation as shown in Table 1. Such conflicts need to be resolved for correct execution of code blocks.

[0020] One way to resolve the conflicts is to store current values in all physical registers that are involved in the conflicts to a memory first, and then load desired values from the memory to the physical registers. However, the storing operations and the loading operations involving the memory are often time consuming, and are not optimal for dynamic compilation.

[0021] Instead, one or more temporary registers may be implemented for resolving the conflicts. For example, certain physical registers are selected as temporary registers for storing or moving data to reduce time consumption. FIG. 2 depicts an example diagram showing conflict resolution of register allocation. As shown in FIG. 2, an original allocation 250 for the
code block 204 is changed using one or more temporary registers to a target allocation 252 for the code block 202.

[0022] Specifically, the original allocation 250 corresponds to the allocation of the physical registers to the variables as shown in Table 2. The target allocation 252 corresponds to the allocation of the physical registers to the variables as shown in Table 1. To correctly execute the code block 202 in the backward loop branch (e.g., as shown in FIG. 1), temporary registers are selected from the physical registers. Current values stored in certain physical registers are moved to the temporary registers first, and then desired values are moved into the physical registers, so that conflicts in register allocation can be resolved.

[0023] FIG. 3 depicts an example diagram for conflict resolution of register allocation. As shown in FIG. 3, temporary registers are used to move data around for conflict resolution in register allocation. Specifically, at 402, certain data structures (e.g., EntryR2Vmap, EntryBitVector, and BackwardBitVector) are set up. Table 3 shows an example data structure EntryR2Vmap that corresponds to the target allocation 252 (e.g., as shown in Table 1). Particularly, the numerical values in the data structure EntryR2Vmap indicate the variables that correspond to one or more physical registers in the target allocation 252. For example, the physical register \( r_0 \) is allocated to the variable \( v_0 \), and the physical register \( r_1 \) is allocated to the variable \( v_3 \).

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryR2VMap</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[0024] A data structure EntryBitVector associated with the target allocation is set up as shown in Table 4. Specifically, the values in the data structure EntryBitVector indicate whether a
particular physical register is allocated to a variable in the target allocation. For example, if the value of a particular bit of the data structure EntryBitVector is 1, it indicates that a corresponding physical register is allocated to a variable in the target allocation. If the value of a particular bit of the data structure EntryBitVector is 0, it indicates that a corresponding physical register is not allocated to any variable in the target allocation.

Table 4

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryR2VMap</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EntryBitVector</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

[0025] A data structure BackwardBitVector is set up to indicate the statuses of the physical registers. For example, the data structure BackwardBitVector is set up according to the following algorithm:

for a physical register R associated with the original allocation 250

    if R is not allocated to any variable in the original allocation 250, mark R as available to be a temporary register;

    if R is allocated to a variable V in the original allocation 250,

        if V is not included in the target allocation 252 (i.e., V is not in EntryR2VMap)

            store a value of V back to a memory;

        free R; and

        mark R as available to be a temporary register;

    else (i.e., V is included in the target allocation 252)

        if EntryR2VMap[R] == V (i.e., no conflicts to resolve),
clearBit(EntryBitVector, R);
else
    setBit(BackwardBitVector, R);
endif
endif
endif
endfor

For example, in the original allocation 250 (e.g., as shown in Table 2), the physical register r9 is allocated to the variable v7. However, no physical register is allocated to the variable r7 in the target allocation 252 (e.g., as shown in Table 1). Thus, according to the above algorithm, the value of the variable v7 in the physical register r9 is stored back to a memory, and the physical register r9 is freed to be available as a temporary register.

After the data structure BackwardBitVector is set up according to the above-noted algorithm, Table 5 shows an example diagram of resulting data structures.

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryR2VMap</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Referring to FIG. 3, at 404, a data structure tempBitVector is determined based on the data structures EntryBitVector and BackwardBitVector. Specifically, a logic operation "~" (i.e., NOT) is performed on the data stmcture BackwardBitVector to generate an intermediate result. Another logic operation "&" (i.e., AND) is performed on the intermediate result and the data
structure EntryBitVector to generate the data structure tempBitVector. Table 6 shows an example of the data structure tempBitVector.

Table 6

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tempBitVector</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

[0029] At 406, it is determined whether the data structure tempBitVector is equal to 0 (i.e., whether every bit of tempBitVector is equal to 0). If the data structure tempBitVector is not equal to 0, at 412, one or more physical registers that correspond to the non-zero bits in the data structure tempBitVector are available to receive values of corresponding target variables according to the target allocation 252. For example, as shown in Table 6, a bit of the data structure tempBitVector corresponding to the physical register r2 is not zero. It indicates that the physical register r2 is available to receive a value of the corresponding variable v1 in the target allocation 252 as shown in Table 1. In the original allocation as shown in Table 2, the physical register r1 is allocated to the variable v1. Thus, the value in the physical register r1 can be moved directly to the physical register r2 as part of the process for changing the original allocation to the target allocation.

[0030] In addition, as shown in Table 6, a bit of the data structure tempBitVector corresponding to the physical register r3 is not zero. It indicates that the physical register r3 is available to receive a value of the corresponding variable v2 in the target allocation 252 as shown in Table 1. However, no physical register is allocated to the variable v2 in the original allocation as shown in Table 2. Thus, a value of the variable v2 may be loaded from a memory.
to the physical register r3 as part of the process for changing the original allocation to the target allocation.

[0031] An algorithm for the process at 412 is as follows, where Ri represents a physical register that is available to receive a value of a corresponding variable in the target allocation 252:

for each set bit in tempBitVector
    index = set bit index in tempBitVector;
    int src=moveRegTo(index);
    if Rsrc is available to be a temporary register, mark it as an available temporary register;
endfor

function moveRegTo(int i)
    V = EntryR2VMap(i);
    if Rs is allocated to V in the original allocation 250
        move Ri, Rs;
        clearBit(EntryBitVector, i);
        clearBit(BackwardBitVector, s);
        return s;
    else
        load V to Ri;
        clearBit(EntryBitVector, i);
    endif
endfunction
Table 7 shows an example of the resulting data structures after the process at 412. Then, the process at 404 is executed again. Table 8 shows an example of the resulting data structures after the process at 404. As shown in Table 8, the data structure tempBitVector is not equal to 0, and the process at 412 is executed again. Specifically, a bit of the data structure tempBitVector corresponding to the physical register r1 is not zero. It indicates that the physical register r1 is available to receive a value of the corresponding variable v3 in the target allocation 252 as shown in Table 1. In the original allocation as shown in Table 2, the physical register r4 is allocated to the variable v1. Thus, the value in the physical register r4 can be moved directly to the physical register r1 as part of the process for changing the original allocation to the target allocation.
According to the above algorithm for the process at 412, Table 9 shows an example of the resulting data structures after the process at 412.

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tempBitVector</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Then, the process at 404 is executed again. Table 10 shows an example of the resulting data structures after the process at 404.

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tempBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

As shown in Table 10, the data structure tempBitVector is equal to 0. At 408, it is determined that the data structure EntryBitVector is not equal to 0. At 414, a new data structure bitVector is determined based on the data structures EntryBitVector and BackwardBitVector. Specifically, a logic operation "&" (i.e., AND) is performed on the data structures EntryBitVector and BackwardBitVector to generate the data structure bitVector. Table 11 shows an example of the resulting data structures.
At 416, it is determined whether any temporary registers are available. If there is no temporary register available, at 418, values in one or more physical registers that correspond to the non-zero bits in the data structure bitVector are stored back to the memory and the physical registers are freed. If one or more temporary registers are available, at 420, the temporary registers are used to move data around in the physical registers as part of the process for changing the original allocation to the target allocation. An algorithm for the processes at 416, 418 and 420 is as follows:

if a temporary register Rt is available

move Rt, Ri;
clearBit(BackwardBitVector, i);
setBit(BackwardBitVector, i);
mark Rt to be used (i.e., Rt cannot be used as a temporary register);
else
store a variable value in Ri back to a memory;
free Ri;
clearBit(BackwardBitVector, i);
endif

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tempBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
For example, as shown in Table 11, a bit of the data structure bitVector corresponding to the physical register r6 is not zero. As described above, the physical register r9 is available as a temporary register. Thus, a value of the variable v5 stored in the physical register r6 according to the original allocation as shown in Table 2, is moved to the physical register r9. Table 12 shows an example of the resulting data structures.

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>tempBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Then, the process at 404 is executed again. Table 13 shows an example of the resulting data structures after the process at 404.

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>tempBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Then, the process at 406 is executed. As shown in Table 13, a bit of the data structure tempBitVector corresponding to the physical register r6 is not zero. It indicates that the physical register r6 is available to receive a value of the corresponding variable v4 in the target allocation.
252 as shown in Table 1. In the original allocation as shown in Table 2, the physical register r8 is allocated to the variable v4. The process at 412 is executed. That is, the value in the physical register r8 can be moved directly to the physical register r6 as part of the process for changing the original allocation to the target allocation. Then, the process 404 is executed again. Table 14 shows an example of the resulting data structures.

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tempBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 15

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Then, the process at 406 is executed. As shown in Table 14, a bit of the data structure tempBitVector corresponding to the physical register r8 is not zero. It indicates that the physical register r8 is available to receive a value of the corresponding variable v6 in the target allocation 252 as shown in Table 1. In the original allocation as shown in Table 2, the physical register r7 is allocated to the variable v6. The process at 412 is executed. That is, the value in the physical register r7 can be moved directly to the physical register r8 as part of the process for changing the original allocation to the target allocation. Then, the process 404 is executed again. Table 15 shows an example of the resulting data structures.
Then, the process at 406 is executed. As shown in Table 15, a bit of the data structure tempBitVector corresponding to the physical register r7 is not zero. It indicates that the physical register r7 is available to receive a value of the corresponding variable v5 in the target allocation 252 as shown in Table 1. As described above, the physical register r9 includes a value of the variable v5. The process at 412 is executed. That is, the value in the physical register r9 can be moved directly to the physical register r7 as part of the process for changing the original allocation to the target allocation. Then, the process 404 is executed again. Table 16 shows an example of the resulting data structures.

Table 16

<table>
<thead>
<tr>
<th>Physical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BackwardBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tempBitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bitVector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

At 406, it is determined that the data structure tempBitVector is equal to 0. At 408, it is determined that the data structure EntryBitVector is equal to 0, and it indicates that there are no more conflicts to be resolved. Thus, at 410, the conflict resolution process ends. In some embodiments, an additional data structure may be set up for the original allocation 250. An algorithm for the entire conflict resolution process as shown in FIG. 3 is as follows:
while (true)
    
    tempBitVector = EntryBitVector & ~BackwardBitVector;
    
    if tempBitVector != 0
        
        for each set bit in tempBitVector
            
            index = set bit index in tempBitVector;
            
            int src = moveRegTo(index);
            
            if Rsrc is available to be a temporary register, mark it as an available temporary register;
        
        endfor
    
    else
        
        if EntryBitVector != 0
            
            bitVector = EntryBitVector & BackwardBitVector;
            
            i = first set bit index in bitVector;
            
            if a temporary register Rt is available
                
                move Rt, Ri;
                
                clearBit(BackwardBitVector, i);
                
                setBitfBackwardBitVector, t);
                
                mark Rt to be used (i.e., Rt cannot be used as a temporary register);
            
            else
                
                store a variable value in Ri back to a memory;
                
                free Ri;
                
                clearBit(BackwardBitVector, i);
            
        
    

endif
else
break;
endif
endwhile

function moveRegTo(int i)
    V = EntryR2VMap(i);
    if Rs is allocated to V in the original allocation 250
        move Ri, Rs;
        clearBit(EntryBitVector, i);
        clearBit(BackwardBitVector, s);
        return s;
    else
        load V to Ri;
        clearBit(EntryBitVector, i);
    endif
endfunction

[0045] Correspondingly, instructions can be generated as follows to be executed by one or more processors:

    str  r9, [v7]
    mov  r2, r1
    ldr  r3, [v2]
    mov  rl, r4
FIG. 4 depicts an example flow chart for register allocation. As shown in FIG. 4, at 302, an original code block and a target code block associated with a branch of an execution loop are determined. At 304, an original allocation of a plurality of physical registers to one or more original variables associated with the original code block is detected. At 306, a target allocation of the plurality of physical registers to one or more target variables associated with the target code block is determined. At 308, one or more temporary registers are selected from the plurality of physical registers based at least in part on the original allocation and the target allocation. At 310, the original allocation is changed to the target allocation using the selected temporary registers. Specifically, one or more instructions are generated to change the original allocation to the target allocation using the selected temporary registers. The instructions are executed using one or more processors. For example, the plurality of physical registers are included in the processors.

FIG. 5 depicts an example diagram showing a database. As shown in FIG. 5, the database 500 includes a database schema that includes and interrelates original variable fields 502, target variable fields 504, original allocation fields 506, and target allocation fields 508. For example, the database 500 is stored in one or more non-transitory machine-readable storage media.

Specifically, the original variable fields 502 store one or more original variables associated with an original code block. The target variable fields 504 store one or more target
variables associated with a target code block. The original code block and the target code block are associated with a branch of an execution loop. The original allocation fields 506 store original data representing an original allocation of a plurality of physical registers to the original variables. The target allocation fields 508 store target data representing a target allocation of the plurality of physical registers to the target variables.

[0049] In some embodiments, one or more processors are configured to process a database query that operates over data related to the original variable fields, the target variable fields, the original allocation fields, and the target allocation fields. Particularly, the processors are configured to select one or more temporary registers from the plurality of physical registers based at least in part on the original data and the target data, and change the original allocation of the plurality of physical registers to the target allocation of the plurality of physical registers using the temporary registers, the physical registers being included in the processors.

[0050] This written description uses examples to disclose the invention, include the best mode, and also to enable a person skilled in the art to make and use the invention. The patentable scope of the invention may include other examples that occur to those skilled in the art. Other implementations may also be used, however, such as firmware or appropriately designed hardware configured to carry out the methods and systems described herein. For example, the systems and methods described herein may be implemented in an independent processing engine, as a co-processor, or as a hardware accelerator. In yet another example, the systems and methods described herein may be provided on many different types of computer-readable media including computer storage mechanisms (e.g., CD-ROM, diskette, RAM, flash memory, computer's hard drive, etc.) that contain instructions (e.g., software) for use in execution by one
or more processors to perform the methods’ operations and implement the systems described herein.
What is claimed is:

1. A method for register allocation, the method comprising:
   - determining an original code block and a target code block associated with a branch of an execution loop;
   - detecting an original allocation of a plurality of physical registers to one or more original variables associated with the original code block;
   - determining a target allocation of the plurality of physical registers to one or more target variables associated with the target code block;
   - selecting one or more temporary registers from the plurality of physical registers based at least in part on the original allocation and the target allocation;
   - generating one or more instructions to change the original allocation to the target allocation using the selected temporary registers; and
   - executing the instructions using one or more processors;
   - wherein the plurality of physical registers are included in the processors.

2. The method of claim 1, wherein:
   - the branch of the execution loop corresponds to a backward loop branch from the original code block to the target code block; and
   - the execution loop further includes a forward loop branch from the target code block to the original code block.
3. The method of claim 1, wherein the selecting one or more temporary registers from the plurality of physical registers based at least in part on the original allocation and the target allocation includes:

determining whether a first physical register corresponds to a first original variable in the original allocation; and

in response to the first physical register not corresponding to a first original variable in the original allocation, indicating the first physical register available to be a temporary register.

4. The method of claim 1, wherein the selecting one or more temporary registers from the plurality of physical registers based at least in part on the original allocation and the target allocation includes:

determining whether a first physical register corresponds to a first original variable in the original allocation;

in response to the first physical register corresponding to the first original variable in the original allocation,

determining whether the first original variable is included in the target variables; and

in response to the first original variable not being included in the target variables,

storing a value of the first original variable in the first physical register to a storage medium; and

indicating the first physical register available to be a temporary register.

5. The method of claim 1, further comprising:
determining whether a first physical register corresponds to a first original variable in the original allocation;

in response to the first physical register corresponding to the first original variable in the original allocation,

determining whether the first original variable is included in the target variables;

and

in response to the first original variable being included in the target variables,

determining whether the first physical register corresponds to the first original variable in the target allocation; and

in response to the first physical register not corresponding to the first original variable in the target allocation, indicating that the first physical register is not available to receive a value of a first target variable, the first physical register corresponding to the first target variable in the target allocation.

6. The method of claim 1, wherein the generating one or more instructions to change the original allocation to the target allocation using the selected temporary registers includes:

determining whether a first physical register is available to receive a value of a first target variable, the first physical register corresponding to the first target variable in the target allocation;

in response to the first physical register being available to receive the value of the first target variable,

detecting a second physical register corresponding to the first target variable in the original allocation; and
moving the value of the first target variable in the second physical register to the
first physical register.

7. The method of claim 1, wherein the generating one or more instructions to change
the original allocation to the target allocation using the selected temporary registers includes:
determining whether a first physical register is available to receive a value of a first target
variable, the first physical register corresponding to the first target variable in the target
allocation;
in response to the first physical register being available to receive the value of the first
target variable,
locating the value of the first target variable in a storage medium; and
loading the value of the first target variable to the first physical register.

8. The method of claim 1, wherein the generating one or more instructions to change
the original allocation to the target allocation using the selected temporary registers includes:
determining whether a first physical register is available to receive a first value of a first
target variable, the first physical register corresponding to the first target variable in the target
allocation; and
in response to the first physical register not being available to receive the first value of
the first target variable,
moving a second value of the first physical register to a first temporary register;
and
indicating the first physical register available to receive the first value of the first target variable.

9. The method of claim 1, wherein the generating one or more instructions to change the original allocation to the target allocation using the selected temporary registers includes:

determining whether a first physical register is available to receive a first value of a first target variable, the first physical register corresponding to the first target variable in the target allocation; and

in response to the first physical register not being available to receive the first value of the first target variable,

storing a second value of the first physical register to a storage medium; and

indicating the first physical register available to receive the first value of the first target variable.

10. A processor-implemented system for register allocation, the system comprising:

one or more non-transitory machine-readable storage media for storing a computer database having a database schema that includes and interrelates original variable fields, target variable fields, original allocation fields, and target allocation fields;

the original variable fields storing one or more original variables associated with an original code block;

the target variable fields storing one or more target variables associated with a target code block, the original code block and the target code block being associated with a branch of an execution loop;
the original allocation fields storing original data representing an original allocation of a plurality of physical registers to the original variables;

the target allocation fields storing target data representing a target allocation of the plurality of physical registers to the target variables; and

one or more processors configured to:

process a database query that operates over data related to the original variable fields, the target variable fields, the original allocation fields, and the target allocation fields;

select one or more temporary registers from the plurality of physical registers based at least in part on the original data and the target data; and

change the original allocation of the plurality of physical registers to the target allocation of the plurality of physical registers using the temporary registers, the physical registers being included in the processors.

11. The system of claim 10, wherein:

the branch of the execution loop corresponds to a backward loop branch from the original code block to the target code block; and

the execution loop further includes a forward loop branch from the target code block to the original code block.

12. The system of claim 10, wherein:

the database schema further includes backward bit fields for indicating statuses of the plurality of physical registers; and
the processors are further configured to:

determine whether a first physical register corresponds to a first original variable in the original allocation; and

in response to the first physical register not corresponding to the first original variable in the original allocation, set a first backward bit field associated with the first physical register to a value that indicates the first physical register is available to be a temporary register.

13. The system of claim 10, wherein:

the database schema further includes backward bit fields for indicating statuses of the plurality of physical registers; and

the processors are further configured to:

determine whether a first physical register corresponds to a first original variable in the original allocation;

in response to the first physical register corresponding to the first original variable in the original allocation,

determine whether the first original variable is included in the target variables;

in response to the first original variable not being included in the target variables,

store a first value of the first physical register to the storage media;

and
set a first backward bit field corresponding the first physical
register to a second value that indicates the first physical register is
available to be a temporary register.

14. The system of claim 10, wherein:

the database schema further includes backward bit fields for indicating statuses of the
plurality of physical registers; and

the processors are further configured to:

determine whether a first physical register corresponds to a first original variable
in the original allocation;

in response to the first physical register corresponding to the first original variable
in the original allocation,

determine whether the first original variable is included in the target
variables;

in response to the first original variable being included in the target
variables,

determine whether the first physical register corresponds to the
first original variable in the target allocation; and

in response to the first physical register not corresponding to the
first original variable in the target allocation, set a first backward bit field
associated with the first physical register to a first value that indicates the
first physical register is not available to receive a second value of a first
target variable, the first physical register corresponding to the first target variable in the target allocation.

15. The system of claim 10, wherein the processors are further configured to:
determine whether a first physical register is available to receive a value of a first target variable, the first physical register corresponding to the first target variable in the target allocation;
in response to the first physical register being available to receive the value of the first target variable,
detect a second physical register corresponding to the first target variable in the original allocation; and
move the value of the first target variable in the second physical register to the first physical register.

16. The system of claim 10, wherein the processors are further configured to:
determine whether a first physical register is available to receive a value of a first target variable, the first physical register corresponding to the first target variable in the target allocation;
in response to the first physical register being available to receive the value of the first target variable,
locate the value of the first target variable in the storage media; and
load the value of the first target variable to the first physical register.
17. The system of claim 10, wherein the processors are further configured to:

determine whether a first physical register is available to receive a first value of a first target variable, the first physical register corresponding to the first target variable in the target allocation; and

in response to the first physical register not being available to receive the first value of the first target variable,

move a second value of the first physical register to a first temporary register; and

indicate the first physical register available to receive the first value of the first target variable.

18. The system of claim 10, wherein the processors are further configured to:

determine whether a first physical register is available to receive a first value of a first target variable, the first physical register corresponding to the first target variable in the target allocation; and

in response to the first physical register not being available to receive the first value of the first target variable,

store a second value of the first physical register to the storage media; and

indicate the first physical register available to receive the first value of the first target variable.

19. A system for register allocation, the system comprising:

one or more processors including a plurality of physical registers; and
a computer-readable storage medium encoded with instructions for commanding the data processors to execute operations including:

determining an original code block and a target code block associated with a branch of an execution loop;

detecting an original allocation of the plurality of physical registers to one or more original variables associated with the original code block;

determining a target allocation of the plurality of physical registers to one or more target variables associated with the target code block;

selecting one or more temporary registers from the plurality of physical registers based at least in part on the original allocation and the target allocation; and

changing the original allocation of the plurality of physical registers to the target allocation of the plurality of physical registers using the selected temporary registers.

20. The system of claim 19, wherein:

the branch of the execution loop corresponds to a backward loop branch from the original code block to the target code block; and

the execution loop further includes a forward loop branch from the target code block to the original code block.
FIG. 1
Set up data structures (e.g., EntryBitVector and BackwardBitVector)

\[ \text{tempBitVector} = \text{EntryBitVector} \& \sim \text{BackwardBitVector} \]

\( \text{tempBitVector} = 0 ? \)

\( \text{No} \rightarrow \text{Move Ri to Rt} \)

\( \text{Yes} \rightarrow \text{EntryBitVector} = 0 ? \)

\( \text{No} \rightarrow \text{bitVector} = \text{EntryBitVector} \& \text{BackwardBitVector} \)

\( \text{Yes} \rightarrow \text{End} \)

Circular dependency resolved

Temp register(s) available?

\( \text{No} \rightarrow \)

\( \text{Yes} \rightarrow \text{Move Ri to Rt} \)

Clobber and flush Ri

FIG. 3
Determine an original code block and a target code block associated with a branch of a program loop.

Detect an original allocation of a plurality of physical registers to one or more original variables associated with the original code block.

Determine a target allocation of the plurality of physical registers to one or more target variables associated with the target code block.

Select temporary registers.

Change the original allocation to the target allocation using the selected temporary registers.

FIG. 4