value of another variable (e.g., B). The result of the comparison determines whether the associated task will be executed by the processor.

[038] FIG. 3 is a pseudocode that illustrates the basic structure of an if-then(-else) conditional statement 310. When an interpreter finds an If, it expects a Boolean condition 301 and evaluates that Boolean condition 301 to either of the values True or False. If the condition is True, the processor executes a consequent task 302. Otherwise, the execution continues in the following branch. If there is an alternative task 303 (e.g., in an Else block), the processor executes the alternative task 303. If there is no Else branch, the processor executes the statement following the End If. After either branch has been executed, control returns to the point after the end If.

[039] Referring again to FIG. 2, the system (e.g., transformation unit 121) transforms (220) the Boolean condition into an arithmetic and/or bitwise expression of the Boolean condition. The arithmetic and/or bitwise expression of the Boolean condition takes the variables of the conditional statement as variables. In some implementations, the arithmetic and/or bitwise expression of the Boolean condition is such that the output of the arithmetic and/or bitwise expression of the Boolean condition verifies the following two properties. If the values of the variables verify the Boolean condition, the arithmetic and/or bitwise expression of the Boolean condition is equal to 1. If the values of the variables do not verify the Boolean condition, the arithmetic and/or bitwise expression of the Boolean condition is equal to 0. As such, Boolean value TRUE is converted to 1, while Boolean value FALSE is converted to 0. In execution, the arithmetic and/or bitwise expression of the Boolean condition maintains the part of the executable code that is to be executed while nulling the parts that are not to be executed. The transformation of the Boolean condition statement into an arithmetic and/or bitwise expression is described in more details in relation to FIG. 4.

[040] The transformation can be performed at various levels of any suitable programming language. In particular, the method can be used in any suitable low-level language (e.g., assembly language, or at the level of the instruction set architecture (ISA)), as well as in any suitable high-level programming language (e.g., C++ code, or Java, etc.) of a particular software program. The process can be performed as a pre-processing step

0, the label is equal to the start statement of the alternative task, the jump statement points to the alternative task, and the alternative task is executed.

[089] FIG. 10 is a flowchart of an example process for providing (e.g., 240 of FIG. 2) the arithmetic and/or bitwise expression of the computer program and optionally executing (e.g., 250 of FIG. 2) the arithmetic and/or bitwise expression of the computer program. The arithmetic and/or bitwise expression is fragmented. The multiples fragments can then be executed by multiple arithmetic and/or bitwise blocks in parallel. As such, the execution speed of the processor can be improved by executing the computer program by multiple arithmetic blocks working in parallel. Further, multilevel conditional statements can be executed in parallel.

[090] The system (e.g., transformation unit 121) fragments (1010) the arithmetic and/or bitwise expression of the computer program into multiples fragments.

[091] The system (e.g., ABU 130) executes (1020) the fragments in parallel. In particular, the ABU 130 can perform multithreading. The ABU 130 execute at least two fragments in parallel. In particular, the fragments can be executed using Reverse Polish Notation.

[092] FIG. 11A is an example of a C++ computer program that comprises a conditional statement 1110A with arithmetic body.

[093] The Boolean condition 1101A of the conditional statement 1110A is whether x is equal to y. The consequent task 1102 is that z is equal to 1. The alternative task 1103 is that z is equal to 0. The conditional statement 1110A can be expressed as follows:

If x is equal to y, then z is equal to 1.

If x is not equal to y, then z is equal to 0.

The execution of the conditional statement 1110A thus requires the comparison of variables x and y.

[094] FIG. 11B is an example of a C++ computer program resulting from the transformation of the computer program of FIG. 11A.

[095] The conditional statement 1110A has been transformed into an arithmetic and/or bitwise expression 1110B comprising only arithmetic and/or bitwise operations. The arithmetic and/or bitwise expression 1101B of the Boolean condition is z=1<<(x-y))%2. As such, if x is equal to y, then z=1<<(x-y))%2=1, and if x is not equal to y, then

z=1<<(x-y))%2=0.

The execution of the arithmetic and/or bitwise expression 1101B of the conditional statement thus provides the same result as the execution of the conditional statement 1110A but does not require the comparison of variables x and y.

[096] FIG. 11C is an example of an assembly language computer program 1100C that comprises a conditional statement with arithmetic body. FIG. 11D is an example of an assembly language computer program 1100D resulting from the transformation of the computer program of FIG. 11C.

[097] FIG. 12A is an example of a C++ computer program that comprises a conditional statement 1210A with a non-arithmetic body

[098] The Boolean condition 1201A of the conditional statement 1210A is whether x is equal to y. The consequent task 1201 is to output “Equals.” The alternative task 1202 is to output “Not equals.” The conditional statement 1210A can be expressed as follows. If x is equal to y, the program outputs "Equals." If x is not equal to y, the program outputs "Not equals".

[099] FIG. 12B is an example of a C++ computer program resulting from the transformation of the computer program of FIG. 12A.

[0100] The conditional statement 1210A has been transformed into an arithmetic and/or bitwise expression 1210B comprising only arithmetic and/or bitwise operations. The arithmetic and/or bitwise expression of the Boolean condition 1201B is res=(1<<k)%2, where k=x-y. As such, if x=y, res=(1<<k)%2=1, while if x≠y, res=(1<<k)%2=0. An unconditional jump 1204 is initialized in an initialization statement 1205. The unconditional jump provides an unconditional jump to a label (“condition[res]”).

[0101] When the arithmetic and/or bitwise expression of the conditions statement is equal to 1 (res=1), the label (“condition[1]”) is equal to the start statement (“to_do”) of the consequent task 1201. The unconditional jump statement points to the consequent task 1201. During execution, when the processor finds the unconditional jump statement 1204, the processor executes the consequent task 1201.

[0102] When the arithmetic and/or bitwise expression of the conditions statement is equal to 0 (res=0), the label (“condition[0]”) is equal to the start statement (“elso_do”) of the alternative task 1202. The unconditional jump statement points to the alternative task