

METHOD FOR EXECUTING INSTRUCTIONS IN CPU

TECHNICAL FIELD

[01] One or more embodiments described by the present invention relate to the field of computer hardware chips, and particularly relate to a method for executing instructions in a Central Processing Unit (CPU).

BACKGROUND

[02] In the contemporary big data cloud environment, the storage and processing of massive data are necessary, imposing higher requirements on how fast data is computed. It is well known that CPU performance is one determinant factor of computation speed. In order to enable faster computation, CPU is being constantly improved in various aspects, from physical process to logical control.

[03] For example, to improve parallel processing power, the CPU hyper-threading technology has been proposed, in which hardware instructions of special characters are utilized to simulate a physical chip from two logical cores. In this way, a single processor is capable of thread-level parallel computation and thus compatible with multithreading parallel computation. In other words, a hyper-threaded CPU can run two or more threads in parallel on a single physical core, increasing the number of parallelizable instructions and exhibiting improved overall operating performance.

[04] On the other hand, instruction prediction schemes are used to perform pre-fetching and pre-execution of instructions, to utilize a CPU's clock cycles more efficiently and avoiding pipeline bubbles or stalls.

[05] These schemes have improved CPU execution efficiency to a certain

extent. However, instruction prediction is not always accurate, and prediction misses can significantly deteriorate CPU execution efficiency.

[06] Therefore, further improvements are desired to achieve higher CPU efficiency.

SUMMARY OF THE INVENTION

[07] In one or more embodiments of the present invention, a method for executing instructions in a CPU is described, which further improves CPU execution efficiency by leveraging the advantages of existing instruction prediction techniques while avoiding the execution of mis-predicted instructions.

[08] According to a first aspect, a method for executing instructions in a CPU is provided. The method includes:

[09] sequentially fetching, from a current thread queue, instructions to form an instruction block and transmitting the instruction block to a CPU execution unit for execution, wherein the instruction block contains one jump instruction, and the jump instruction is the last instruction in the instruction block;

[10] replenishing the current thread queue with at least one instruction to form a thread queue to be executed;

[11] determining a target instruction of the jump instruction according to an execution result of the CPU execution unit;

[12] determining whether the target instruction is contained in the thread queue to be executed; and

[13] if the target instruction is not contained in the thread queue to be executed, flushing the thread queue to be executed, obtaining the target instruction and adding the target instruction to the thread queue to be executed.

[14] According to one embodiment, the instruction block is formed by:

[15] reading a predetermined threshold number of instructions from the current thread queue, the predetermined threshold number depending on the number of CPU execution units; determining whether the jump instruction is present in the predetermined threshold number of instructions; and if the jump instruction is present, taking the jump instruction and preceding instruction(s) thereof as the instruction block .

[16] According to another embodiment, the instruction block is formed by:

[17] reading a first instruction from the current thread queue; if the number of instructions in a current instruction block does not reach a predetermined threshold number, adding the first instruction to the current instruction block, the predetermined threshold number depending on the number of CPU execution units; determining whether the first instruction is a jump instruction; and if the first instruction is a jump instruction, taking the current instruction block as the instruction block.

[18] In a possible design, the instruction(s) is/are replenished to the current thread queue by: according to a predicted branch of an instruction prediction, replenishing the current thread queue with at least one instruction corresponding to the predicted branch.

[19] In a possible scheme, the thread queue to be executed is formed by replenishing the current thread queue to form a thread queue to be executed with corresponding instruction(s) read from a decoded cache, wherein the decoded cache stores a plurality of pre-fetched and decoded instructions.

[20] In one possible embodiment, the jump instruction is a register operation instruction, and wherein the instruction block further includes at least one memory operation instruction.

[21] Further, in one embodiment, determining the target instruction of the jump instruction before execution of the at least one memory operation instruction is completed.

[22] According to a possible design, the target instruction is obtained by:

[23] determining whether the target instruction is present in a decoded cache, wherein the decoded cache stores a plurality of pre-fetched and decoded instructions;

[24] if present, obtaining the target instruction from the decoded cache; and

[25] if not present, obtaining the target instruction from memory.

[26] According to a second aspect, a CPU controller is provided, comprising:

[27] an instruction fetching logic configured to sequentially fetch, from a current thread queue, instructions to form an instruction block and transmit the instruction block to a CPU execution unit for execution, wherein the instruction block contains one jump instruction, and the jump instruction is the last instruction in the instruction block;

[28] an instruction replenishing logic configured to replenish the current thread queue with at least one instruction to form a thread queue to be executed;

[29] a target determination logic configured to determine a target instruction of the jump instruction according to an execution result of the CPU execution unit;

[30] a determination logic configured to determine whether the target instruction is contained in the thread queue to be executed; and

[31] a queue handling logic configured to, if the target instruction is not contained in the thread queue to be executed, flush the thread queue to be executed, obtain the target instruction and add the target instruction to the thread queue to be executed.

[32] According to a third aspect, a central processing unit (CPU) is provided,

comprising the controller as provided in the second aspect.

[33] In the scheme provided in embodiments described herein, while instructions are pre-fetched and put into a decoded cache and thread queues as conventionally practiced, each instruction block sent for execution is always guaranteed to contain at most one jump instruction which is the last instruction. In addition, before a target instruction of the jump instruction is determined, renaming and allocation of executable resources for subsequent instructions in the thread queue are suspended. After the target instruction is determined, instructions in the thread queue are compared with the target instruction to determine whether they match the target instruction. In this way, it is ensured that only correct target instructions are executed. Thus, overall CPU execution efficiency is enhanced by leveraging the advantages of existing instruction prediction schemes while avoiding time-consuming loop-back operations required when mis-predicted instructions are executed.

BRIEF DESCRIPTION OF THE DRAWINGS

[34] In order to more clearly explain the subject matter of embodiments of the present invention, the accompanying drawings, to which reference is to be made in connection with the following description of embodiments, will be briefed below. Apparently, these drawings show only some embodiments of the invention, and those of ordinary skill in the art can obtain other drawings in light of those contained herein, without making any creative effort.

[35] Fig. 1 shows an execution process in a CPU according to one embodiment.

[36] Fig. 2 shows a method for executing instructions in a CPU according to one embodiment.

[37] Fig. 3 shows a functional block diagram of a CPU controller according to one embodiment.

DETAILED DESCRIPTION

[38] The subject matter disclosed herein will be described with reference to the accompanying drawings.

[39] Fig. 1 shows an execution process in a CPU according to one embodiment. As shown in Fig. 1, the execution process is divided to multiple phases and begins with an Instruction Fetching phase. For current mainstream CPUs, up to 16 bytes, equivalent to about 4 instructions, can be fetched in each instruction cycle. The fetched instructions are then pre-decoded. The main purpose of this Pre-Decoding phase is to identify lengths of the instructions and mark any jump instruction therein. Generally, mainstream CPUs have a throughput of 5 instructions per cycle in this phase.

[40] The Pre-Decoding phase is followed by a Decoding phase primarily for transforming complex instructions into simpler (fixed-length) instructions and specifying types of operations to be performed thereon. Usually, this phase also provides a throughput of 5 instructions per cycle. The decoded instructions are stored in a decoded cache.

[41] As a cache pool for instructions, the decoded cache can store a plurality of decoded instructions to be read in the subsequent Thread Queue phase. A throughput for the decoded cache to feed instructions to the next phase can be up to 6 jump instructions per cycle.

[42] As described above, for a hyper-threaded CPU, multiple threads can be executed in parallel. During the execution, each thread reads instructions to be

executed next, forming its own thread cache queue, also referred to as thread queue hereinafter. If there is any instruction to be executed stored in the decoded cache, the queue is replenished with the instruction stored in the decoded cache. Otherwise, corresponding instructions are added to the queue from the front end (the memory). While Fig. 1 only illustrates queues of threads A and B, it will be appreciated that the hyper-threaded CPU may also support the parallel execution of more threads.

[43] The Thread Queue phase is followed by the next phase: Renaming & Allocation of executable resources. A throughput from Thread Queue to this phase may be up to 5 jump instructions per cycle. The Renaming & Allocation phase is mainly for maximizing the number of instructions executable in parallel by addressing read and write dependencies of registers and removing unnecessary dependencies, as well as for allocating various resources necessary for execution.

[44] Following the allocation of necessary resources for execution, the instructions may be fed to the CPU execution units for execution. A modern CPU has multiple execution units. Currently, the most common CPUs has 8 parallel execution pipelines. For example, 8 micro-operations per each cycle may be executed. Although instructions can be executed in random order, they are finally submitted in the same order as that of the program.

[45] As described above, in order to avoid pipeline bubbles or stalls caused by missing instructions, almost all of the existing CPUs employ instruction prediction (also referred to as branch prediction) schemes for prediction and pre-fetching of instructions. At the end of each cycle, the prediction unit predicts, based on an execution-status history table it maintains, the instructions to be pre-fetched next. If there is no jump instruction, then a 16-byte instruction block starting at the current fetch address will be fetched in the Instruction Fetching phase. Otherwise, instructions

will be fetched according to instruction prediction result.

[46] After continuous improvement, existing instruction prediction schemes have achieved prediction accuracy over 90%, and some of them can even reach up to 98%. However, there remains possible misprediction that may introduce wrong instruction block into an execution unit.

[47] For example, assuming there are instructions L1, L2, L3, L4, L5, and L2 is a jump instruction specifying that, if some condition is satisfied, the control will jump to instruction L5 for execution; otherwise, the instructions L3 and L4 will be executed sequentially. If, when predicting the instructions, the target branch of the jump instruction L2 is predicted to be L3, then L3 and the subsequent instruction will be read in the Instruction Fetching phase, likely leading to feeding and execution of L1, L2, L3, L4 in the CPU execution units in subsequent phases. If executing L2 will actually result in a jump to L5, the execution of L3 and L4 will give rise to an error. In this case, the CPU may have to refresh the entire pipeline, loop back to the previous branch, hot restart again and select another branch for execution. Although such a prediction error is less likely to occur, it necessitates the above set of operations, which is very time-consuming and finally limits the CPU efficiency to only 75% or below.

[48] In view of this, embodiments disclosed herein provide an improved scheme, which maintains and leverages the high-accuracy advantages of the existing instruction prediction schemes as much as possible while avoiding the execution of mis-predicted instructions. According to one or more embodiments, instructions are fetched and put into a decoded cache and thread queues in existing manners, but before a valid target address is obtained for a jump instruction, i.e., prior to determining a target instruction, renaming and allocation of executable resources will not be carried out, thus ensuring correct subsequent execution and avoiding reduced

efficiency caused by misprediction. The implementation of this concept will be detailed below.

[49] Fig. 2 is a flowchart of a method for executing instructions in a CPU according to one embodiment. As shown in Fig. 2, the method includes: in step 21, fetching, from a current thread queue, instructions to form an instruction block and transmitting the instruction block to a CPU execution unit for execution, wherein the instruction block contains one jump instruction, and the jump instruction is the last instruction in the instruction block; in step 22, replenishing the current thread queue with at least one instruction to form a thread queue to be executed; in step 23, determining a target instruction of the jump instruction according to an execution result of the CPU execution unit; in step 24, determining whether the target instruction is contained in the thread queue to be executed; if the target instruction is not contained in the thread queue to be executed, in step 25, flushing the thread queue to be executed, obtaining the target instruction and adding the target instruction to the thread queue to be executed. These steps will be described in greater detail below.

[50] As described above, according to embodiments disclosed herein, in order to leverage the advantages of the existing instruction prediction schemes, instructions are fetched and put into a decoded cache and thread queues, as conventionally practiced. In other words, the Instruction Fetching, Pre-Decoding and Decoding phases shown in Fig. 1 are implemented using existing techniques, and the decoded instructions are stored in a decoded cache. In addition, various threads can read instructions from the decoded cache to form respective thread queues. Therefore, it is assumed herein that thread queues have been formed, as conventionally practiced, before step 21.

[51] In step 21, fetching, from a current thread queue, instructions to form an

instruction block and transmitting the instruction block to a CPU execution unit for execution.

[52] If the instructions fetched from the current thread queue do not contain any jump instruction, then an instruction block with a maximum length corresponding to maximum processing power of the CPU hardware is formed in the conventional way. Generally, such maximum processing power depends on the number of execution units that the CPU possesses, and the maximum length of the instruction block may be a predetermined threshold determined according to the number of execution units. For example, currently, the most commonly used CPUs each has 8 parallel pipelines. So, the predetermined threshold and the maximum instruction block length may be both set to 8 accordingly. When the 8 fetched instructions do not contain any jump instruction, they can be treated and processed as a normal instruction block.

[53] Differing from the conventional schemes, when the fetched instructions contain a jump instruction, it is ensured that each instruction block delivered to a respective one of the CPU execution units contains only one jump instruction and the jump instruction is the last instruction of the instruction block. In other words, during transfer of instructions from the thread queues to the CPU execution units, types of the instructions are ascertained, and instruction blocks are formed with jump instructions as their boundaries, so that in each set of instructions delivered to the respective CPU execution unit, the jump instruction is the last instruction.

[54] These instruction blocks can be formed in different ways. In one embodiment, a predetermined threshold number of instructions are read from the current thread queue at each time, the predetermined threshold number corresponding to the CPU's maximum processing power, i.e., depending upon the number of the CPU execution units. After that, a check is performed to find out if there is any jump

instruction in the read-out instructions. If not, as described above, these instructions are taken as an instruction block. Otherwise, if these instructions contain a jump instruction, then this jump instruction and all preceding instruction(s) are taken as an instruction block.

[55] For example, assuming the predetermined threshold is 8, 8 instructions are read out from the current thread queue at one time. If there is no jump instruction in these 8 instructions, these 8 instructions are treated as an instruction block and fed to one of the CPU execution units. If there is a jump instruction contained in the 8 instructions, then an instruction block is alternatively formed, including the jump instruction and the all preceding one(s). For example, if the fifth instruction is a jump instruction, then the instruction block is formed by the first to fifth instructions, with the fifth instruction as the last instruction.

[56] In this way, it is ensured that in each set of instructions delivered to the respective CPU execution unit for execution, the jump instruction is the last instruction. It could be appreciated that, before a jump instruction is executed, it is difficult to accurately predict the next instruction to be executed, i.e., the target instruction. However, in the instruction prediction schemes, the predicted target instruction is to be pre-fetched into the thread queue. In order to overcome this, if it is ensured that the jump instruction is the last instruction in each set of instructions fed to the respective CPU execution unit, an isolation or break is established between the jump instruction and any predicted subsequent target instruction, ensuring that any predicted target instruction is not to be delivered to one of the CPU execution units along with a jump instruction. This creates opportunities and possibilities for accurately determining target instructions and correcting mis-predicted target instructions.

[57] In the above example with instruction L2 out of instructions L1, L2, L3,

L4, L5 being a jump instruction, even when the target instruction of the jump instruction is mistakenly predicted as L3, according to the above embodiments, only L1 and L2 is to be fed to and executed in one of the CPU execution units as an instruction block, instead of feeding and executing L1, L2, L3, L4 as a block. As a result, when L1 and L2 are executed, an opportunity can be provided for determining the correct target branch of L2 and for correcting the previous target branch misprediction.

[58] As described above, after at least some instructions are fetched from the thread queue and fed to the CPU execution units, there may be temporarily fewer instructions waiting for execution in the thread queue. Accordingly, during or after delivery of the resulting instruction block to the CPU execution units, the thread queue is replenished so that its length is maintained. In other words, in step 22, at least one instruction is replenished into the current thread queue to form a thread queue to be executed. It can be understood that the “thread queue to be executed” refers to a thread queue from which the next instruction block is to be formed and transmitted to the CPU execution units for execution.

[59] According to one embodiment, in this step, the thread queue to be executed may be formed, as conventionally practiced, i.e., by replenishing the thread queue with the instruction(s) according to the predicted branch of the instruction prediction. In one embodiment, according to an instruction prediction result, instruction(s) is/are read from a decoded cache storing multiple pre-fetched and decoded instructions and added to the current thread queue. In extremely rare situations, it is also possible to request instruction(s) from the front end (e.g., memory), which are decoded and replenished into the thread queue.

[60] On the other hand, after the instruction block formed in step 21 is

transmitted to the respective CPU execution unit, the CPU execution unit may add the instructions therein to a pipeline so that they can be executed. In particular, the last instruction in the instruction block is a jump instruction, and when the jump instruction is executed, the target address jumped to is accurately determined, that is, the target instruction is determined. Thus, in step 23, the target instruction of the jump instruction is determined according to an execution result of the CPU execution unit.

[61] After that, in step 24, determining whether the thread queue to be executed formed by replenishment in step 23 contains the target instruction. If yes, this means that the target instruction to be executed next has been added to the thread queue to be executed due to a correct instruction prediction result, and no additional operation is necessary. Thus, after the execution of the current instruction block is completed, a new instruction block can be fetched from the thread queue and executed in accordance with the method of Fig. 2.

[62] However, if the thread queue to be executed does not contain the target instruction, this means that the instruction to be executed next (target instruction) is absent from the thread queue. In other words, the instruction sequence contained in the thread queue to be executed is not the one to be executed next. This may be because instruction misprediction has occurred, leading to pre-fetching of wrong instructions in the thread queue. In this situation, in step 25, flushing the current thread queue to be executed, and obtaining the target instruction and adding the target instruction to the thread queue to be executed.

[63] Specifically, in step 25, since the current thread queue to be executed contains wrong instructions which should not be executed, the current thread queue to be executed is first flushed. Flushing is a CPU operation that can clear away all the data stored in an object. Flushing the current thread queue to be executed results in the

deletion of all the instructions therein so that the queue becomes empty.

[64] Following the flushing of the wrong instructions, the correct target instruction is obtained and added to the thread queue to be executed.

[65] In one embodiment, it is first determined whether the correct target instruction is present in the decoded cache. If the correct target instruction is present, the target instruction is obtained from the decoded cache. It can be understood that the wrong branch instructions previously added in the thread queue are often due to an ordinal error in instruction execution scheduling. As a large number of instructions are consecutively pre-fetched, decoded, and added to the decoded cache according to the instruction prediction schemes, in almost all situations, the correct target instruction may have been obtained from the decoded cache, and added to the thread queue to be executed. In addition, apart from adding the target instruction, one or more instructions succeeding it may also be added to the thread queue to be executed.

[66] Nevertheless, in extremely rare situations, the target instruction is not present in the decoded cache. In such situations, a request may be made to the memory for the target instruction. Once obtained, the target instruction is decoded and added to the thread queue to be executed.

[67] By the operation in step 25, it is ensured that the thread queue to be executed contains the correct branch instruction, and hence the correct instruction block is subsequently transmitted to a respective CPU execution unit, preventing the execution of any mis-predicted instruction by the execution unit.

[68] The above process is further explained with reference to a particular example as follows. Assuming there is the following sequence of instructions (text enclosed by each `/*...*/` explains a corresponding one of the instructions):

Loop: 1. `fld f0,0(x1)` `/* According to the address stored in x1, transfer content`

from memory to register f0 */

2. fadd.d f4,f0,f2 /* Save the result of f0+f2 to register f4*/

3. fsd f4, 0(x1) /* According to the address stored in x1, save data from f4 to memory */

4. addi x1,x1-8 /* Execute x1 minus “8” and save the result in register x1*/

5. bne x1, x2, Loop /*If x1 is not equal to x2, jump to Loop and execute instruction 1; otherwise proceed to instruction 6*/

6. addi x2, x2+1 /* Execute x2 plus “1” and save the result in register x2*/

7...

8...

[69] In this sequence, the instruction 5 is a jump instruction for dispatching control to different branches, depending on whether x1 is equal to x2. The target instruction of first branch is the instruction 1, while the target instruction of the second branch is the instruction 6. It is assumed that these instructions have been pre-fetched, decoded and stored in the decoded cache, as conventionally practiced. It is also assumed that, according to an instruction prediction result, when the instruction 5 is executed, control will jump to the second branch for execution, so the instructions 1 to 8 are sequentially put into a thread queue.

[70] In the conventional manner, the instructions 1 to 6 are likely to be regarded as an instruction block and transmitted to a respective CPU execution unit for execution. After instruction 5 is executed, the first branch, i.e., the instruction 1, is supposed to be executed. However, in the event of misprediction, the instruction 6 is mistakenly executed, necessitating the time-consuming flushing and loop-back operations and leading to a drastic reduction in CPU efficiency.

[71] According to the scheme provided in embodiments described herein, in

step 21, since the instruction 5 is a jump instruction, only the instructions 1 to 5 will be transmitted, as an instruction block, to a respective CPU execution unit for execution.

[72] In step 22, replenishing the thread queue with instruction(s) from the decoded cache to form a thread queue to be executed. As the target branch is the instruction 6 according to the instruction prediction result, at this time, the thread queue to be executed may include the instruction 6, instruction 7 and the replenished instruction(s).

[73] In step 23, the execution result of the CPU execution unit indicates that the target instruction of the jump instruction 5 is in fact the instruction 1 in the first branch.

[74] Subsequently, in step 24, it can be found that the target instruction 1 is not present in the thread queue to be executed. As a result, in step 25, the thread queue to be executed is flushed, and the instruction 1 is fetched from the decoded cache and added to the thread queue to be executed, thereby enabling the formation of a thread queue to be executed consistent with the correct branch.

[75] Therefore, in the above embodiments, while instructions are pre-fetched and loaded into a decoded cache and a thread queue as conventionally practiced, each instruction block sent for execution is guaranteed to contain at most one jump instruction which is the last instruction in the block. In addition, before a target instruction of the jump instruction is determined, renaming and allocation of executable resources for subsequent instructions in the thread queue are suspended. After the target instruction is determined, instructions in the thread queue are compared with the target instruction to determine whether they match the target instruction. In this way, it is ensured that only correct branch are executed, without suffering from deteriorated efficiency due to misprediction.

[76] It can be understood that increasing CPU execution efficiency requires maximizing the utilization of each execution cycle and minimizing pipeline bubbles and stalls as well as idle operations where no instruction is executed. In order to avoid decreases in efficiency caused by time-consuming loop-backs due to misprediction, the above-described scheme carries out determination and comparison on the thread queue once the target instruction of the jump instruction has been determined, to ensure the correctness of subsequent operations. However, whether such an “additional” operation will lead to a pipeline stall or idle operation that may harm the CPU’s execution efficiency remains an issue needing clarification. From research and analysis, the inventors have found that such an operation will not cause a waste of a CPU’s execution cycles or a decrease in its execution efficiency. This will be discussed below in greater detail.

[77] First of all, the optimized schemes described herein are immune from the issue of multiple idle cycles in which no instruction is executed. Generally, a number of instructions, e.g., 5 instructions, can be fetched in each CPU cycle. Statistics show that, on average, in the 5 instructions, there is 1 jump instruction, 1.4 memory read instructions, 1 memory write instruction and 1.6 instructions related to computing operations. Statistics also show that most jump instructions involve simple register operations, and the execution of each of them can be completed within a single cycle.

[78] On the other hand, any memory operation may be accompanied by a long delay lasting for several cycles. Specifically, if a memory operation hits an L1 cache, a delay of 4 cycles will be required; if it hits an L2 cache, then a delay of 12 cycles will be required; and if it hits no cache, a delay of at least 50 cycles will be required for reading data from the RAM. Therefore, by first executing jump instructions, it is possible to determine target addresses as early as possible without

causing bubbles.

[79] For example, continuing the above example with the instructions 1 to 5 out of the instructions 1 to 6 being fed as an instruction block to the execution unit, as the instruction 1 requires reading of data from the memory, even if an L1 cache is hit, it will require 4 cycles. Due to the dependencies between the instructions, the instructions 2 and 3 must be executed in succession, and this requires additional 2 cycles. Further, the instructions 4 and 5 are both register operations, each of which can be finished in one cycle. Thus, before the instructions 1, 2, 3 are finished, it is possible to know whether a jump is required at the instruction 5.

[80] Additionally, current mainstream CPUs are multithreaded ones. For example, in the event of two threads, the CPU will fetch instructions alternately for these threads, at first for a thread A, then for a thread B, again for thread A, ... In this way, the thread A will periodically wait for one cycle, and target instructions of the jump instructions can be determined in these inherent waiting periods.

[81] In fact, in the current big data environment enabled by cloud servers, performance issues are concentrated in memory access delays. Such delays usually span up to hundreds of instruction cycles and sufficient for determining target instructions of jump instructions and ensuring the correctness of thread queues while not affecting CPU operations. Therefore, the above operations in the present schemes will not introduce issues to CPUs, such as idle operations without instructions being executed, stalls, etc.

[82] Another concern is the pre-fetching of instructions and data in the optimized and existing instruction prediction schemes. Conventional instruction prediction methods include an Instruction Pre-Execution phase, in which instructions that have not been ascertained can be run. From this point of view, the conventional

methods can help the CPU pre-fetch many codes to be executed soon and reduce delays caused by missing instructions. The present schemes can also adopt this approach, except that correct instructions are executed only after target addresses of jump instructions have been determined. As for data pre-fetching, as the conventional instruction prediction methods allow instructions that have not been ascertained to be run in the Instruction Pre-Execution phase, data to be used soon can be transferred from the memory into the CPU's caches. In the embodiments disclosed herein, this approach may also be adopted, except that necessary data are read only after target addresses have been ascertained and effective instructions have been executed.

[83] Therefore, the schemes provided in the embodiments described herein can make full use of the advantages of conventional methods and combine them with the various characteristics and primary operating conditions of existing CPUs, thereby resulting in significantly enhanced CPU utilization and greatly improved throughput of cloud computing clusters.

[84] As known to those skilled in the art, instructions are executed in a CPU under the control of a controller, which is a command and control center of the entire CPU and is configured to coordinate operations of various components therein. Such a controller generally includes instruction control logic, timing control logic, bus control logic, interruption control logic and other logics. The instruction control logic is configured for fetching, analysis and execution of instructions.

[85] A controller circuit, especially instruction control logic therein, may be adapted in the hardware level in accordance with the optimizations and modification made to the conventional instruction control schemes according to the above-described embodiments, so as to be able to effectuate the control process as described in the above embodiments.

[86] Fig. 3 shows a functional block diagram of a CPU controller according to one embodiment. As shown in Fig. 3, the controller 300 may include: an instruction fetching logic 31 configured to sequentially fetch, from a current thread queue, instructions to form an instruction block and transmit the instruction block to a CPU execution unit for execution, wherein the instruction block contains one jump instruction and the jump instruction is the last instruction in the instruction block; an instruction replenishing logic 32 configured to replenish the current thread queue with at least one instruction to form a thread queue to be executed; a target location logic 33 configured to determine a target instruction of the jump instruction according to an execution result of the CPU execution unit; a determination logic 34 configured to determine whether the target instruction is contained in the thread queue to be executed; and a queue handling logic 35 configured to, if the target instruction is not contained in the thread queue to be executed, flush the thread queue to be executed, obtain the target instruction and add the target instruction to the thread queue to be executed.

[87] These logics can be implemented with appropriate circuit elements as needed. For example, the determination logic may be implemented with a number of comparators.

[88] With this controller, the control process shown in Fig. 2 can be carried out to enhance the overall CPU execution efficiency by leveraging the advantages in instruction prediction and pre-fetching, while avoiding and preventing efficiency impairments arising from misprediction.

[89] Those skilled in the art should appreciate that the foregoing specific embodiments are presented to describe the objects, subject matter and benefits of the present invention in greater detail. It will be also appreciated that presented above are

merely particular embodiments of the present invention, which do not limit the scope of the invention in any sense. Accordingly, any and all changes, equivalent alternatives, modifications and the like made to the subject matter of the present invention are all intended to be embraced within the scope of the invention.

Claims

1. A method for executing instructions in a central processing unit (CPU), comprising:

fetching, from a current thread queue, instructions to form an instruction block and transmitting the instruction block to a CPU execution unit for execution, wherein the instruction block contains one jump instruction, and the jump instruction is the last instruction in the instruction block;

replenishing the current thread queue with at least one instruction to form a thread queue to be executed;

determining a target instruction of the jump instruction according to an execution result of the CPU execution unit;

determining whether the target instruction is contained in the thread queue to be executed; and

if the target instruction is not contained in the thread queue to be executed, flushing the thread queue to be executed, obtaining the target instruction and adding the target instruction to the thread queue to be executed.

2. The method according to claim 1, wherein fetching, from a current thread queue, instructions to form an instruction block comprises:

reading a predetermined threshold number of instructions from the current thread queue, the predetermined threshold number depending on the number of CPU execution units;

determining whether the jump instruction is present in the predetermined threshold number of instructions; and

if the jump instruction is present, taking the jump instruction and preceding instruction(s) thereof as the instruction block.

3. The method according to claim 1, wherein replenishing the current thread queue with at least one instruction to form a thread queue to be executed comprises:

according to a predicted branch of an instruction prediction, replenishing the current thread queue with at least one instruction corresponding to the predicted branch.

4. The method according to claim 1, wherein replenishing the current thread queue with at least one instruction to form a thread queue to be executed comprises:

replenishing the current thread queue with corresponding instruction(s) read from a decoded cache, wherein the decoded cache stores a plurality of pre-fetched and decoded instructions.

5. The method according to claim 1, wherein the jump instruction is a register operation instruction, and wherein the instruction block further includes at least one memory operation instruction.

6. The method according to claim 5, wherein determining a target instruction of the jump instruction according to an execution result of the CPU execution unit comprises:

determining the target instruction of the jump instruction before execution of the at least one memory operation instruction is completed.

7. The method according to claim 1, wherein obtaining the target instruction comprises:

determining whether the target instruction is present in a decoded cache,

wherein the decoded cache stores a plurality of pre-fetched and decoded instructions;

if present, obtaining the target instruction from the decoded cache; and

if not present, obtaining the target instruction from memory.

8. A central processing unit (CPU) controller, comprising:

an instruction fetching logic configured to sequentially fetch, from a current thread queue, instructions to form an instruction block and transmit the instruction block to a CPU execution unit for execution, wherein the instruction block contains one jump instruction, and the jump instruction is the last instruction in the instruction block;

an instruction replenishing logic configured to replenish the current thread queue with at least one instruction to form a thread queue to be executed;

a target determination logic configured to determine a target instruction of the jump instruction according to an execution result of the CPU execution unit;

a determination logic configured to determine whether the target instruction is contained in the thread queue to be executed; and

a queue handling logic configured to, if the target instruction is not contained in the thread queue to be executed, flush the thread queue to be executed, obtain the target instruction and add the target instruction to the thread queue to be executed.

9. A central processing unit (CPU), comprising the controller of claim 8.